FACULTAD 9

ESTRATEGIA PARA EL DESARROLLO DE SISTEMAS A TRAVÉS DE COMPONENTES EN EL POLO PETROSOFT

TRABAJO DE DIPLOMA PARA OPTAR POR EL TITULO DE INGENIERO EN CIENCIAS INFORMATICAS.

AUTOR: DANIEL SAMPEDRO BELLO

TUTOR: LIC. DAVID SILVA BARRERA

CONSULTOR: M.Sc. FEBE A. CIUDAD RICARDO

CIUDAD DE LA HABANA, MAYO DE 2009

“AÑO 50 DE LA REVOLUCION”

"Muchas veces la gente no sabe lo que quiere hasta que se lo enseñas."

"Tu tiempo está limitado, así que no lo desaproveches viviendo la vida de algún otro. No te dejes arrastrar por los dogmas, que es lo mismo que vivir con los resultados del pensamiento de otras personas. No dejes que el ruido de las opiniones de otros ahoguen completamente tu voz interior. Y más importante, ten el valor de seguir a tu corazón y a tu intuición. Ellos, de algún modo, ya saben en lo que verdaderamente te quieres convertir. Todo lo demás es secundario."

Steve Jobs

A mi padre, a quien le debo todo lo que soy, que aunque no encuentre presente, vivirá siempre en mí en cada uno de mis actos.

A mi abuela, por haberme entregado siempre todo su amor ilimitado, por guiarme y compartir mis momentos buenos y malos, mis logros y fracasos, por estar siempre a mi lado.

A mi novia Aliosmi, especialmente, magnifica compañera para la vida, por su amor incondicional y apoyo total en cada aspecto de mi vida.

A mi madre y a mi hermano Yuri, a los cuales siempre tengo presente.

A mi hermana y a Olga gracias por toda la ayuda que siempre me han dado.

A mi tutor David, del cual he aprendido mucho hace ya varios años de trabajo diario en proyectos productivos, cuyos consejos me han sido de gran valía y han tributado directamente a mi formación profesional.

A mi amigos Julio, Douglas y Luis, así como a todos aquellos con los que he compartido durante casi cinco años momentos inolvidables de mi vida, en especial gracias a mis amigos del otrora grupo 9107.

A los profesores de la facultad con los cuales he trabajado directamente y he aprendido en todo momento, especialmente gracias a Armando, a Yancy, Yesnier, a Puebla a Yordanys y a Febe, por sus consejos y ayuda incondicional en el desarrollo del presente trabajo diploma.

A todos los que me han ayudado directa o indirectamente, a todos gracias.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a La Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____	_____
Daniel Sampedro Bello	Lic. David Silva Barrera

Título: Estrategia para el Desarrollo de Sistemas a través de Componentes en el polo PetroSoft.

Autor: Daniel Sampedro Bello

_____ _____/____/_____
Firma Fecha
Lic. David Silva Barrera

RESUMEN

El principal objetivo del polo productivo PetroSoft de la Facultad 9, en la Universidad de las Ciencias Informáticas, es lograr alcanzar mayor producción y reutilización en el desarrollo de los proyectos de software que tenga asignados. El desarrollo de software basado en componentes (DSBC) figura como una de las soluciones a considerar para alcanzar tales propósitos. Entre los principales problemas que registra esta área se encuentran la definición de tareas a desarrollar y técnicas a aplicar para obtener productos de software de alta calidad.

Con el desarrollo de este trabajo se desea mostrar una propuesta consistente en una metodología basada en la solución del Rational Unified Process (RUP, Proceso Unificado de Desarrollo) para el DSBC, la cual ha sido modificada y adaptada para ajustarse a las condiciones actuales del polo. Dicha propuesta recoge las principales actividades y flujos de trabajo que se deben seguir para lograr el ensamblaje de sistemas basados en componentes de alta calidad.

El camino tomado por el autor para lograr cumplir lo anteriormente trazado está representado en cada una de las páginas de este documento conformado por tres (3) capítulos. En el Capítulo 1 se abordan características y conceptos relacionados con el DSBC, así como los principales temas de actualidad afines. En Capítulo 2 se realiza la propuesta de la metodología utilizando como basamento la comparación entre otras análogas y características circunstanciales de las mismas. La evaluación de los resultados y la evolución de los componentes son los temas tratados en el Capítulo 3 y final.

PALABRAS CLAVE

Metodología

Desarrollo de Software Basado en Componentes

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS Y FIGURAS

INTRODUCCIÓN

La creciente demanda de los sistemas computacionales actuales así como las exigencias en cuanto a factores como el aumento de la calidad y la reducción de los plazos de desarrollo de los mismos, son las causas principales de la búsqueda de la reutilización del software existente. El Desarrollo de Software Basado en Componentes (DSBC) trae aparejado diversos beneficios tales como las mejoras a la calidad, la reducción del ciclo de desarrollo y el mayor retorno sobre la inversión; debido a que se organiza el desarrollo a partir de elementos pre-elaborados que desarrollan alguna funcionalidad común y que pueden ser reutilizados una y otra vez.

Durante algunos años, el DSBC fue conocido como una aproximación del desarrollo de software basado en la filosofía "compre, y no construya" promulgada por Fred. Brooks en 1987 (FREDERICK P. BROOKS 1987), la cual abogaba por utilizar componentes previamente elaborados por terceros evitando así tener que desarrollarlos. Sin embargo en los años posteriores surgieron gran cantidad de estándares, procedimientos, prácticas de ingeniería y metodologías, los cuales han sido aceptados por muchos autores entre los que se encuentran (G. HEINEMAN 2001) o (KURT WALLNAU 2001) que han propiciado que se empiece a hablar con el término "Ingeniería de Software Basada en Componentes (ISBC) como una subdisciplina de la Ingeniería de Software".

El proceso de industrialización del desarrollo de software actual, ha dado ya sus primeros pasos con implementaciones como la plataforma .net la cual es partidaria y promueve la idea de industrializar la producción de software a través de un desarrollo basado en componentes. En el continuo desarrollo de esta plataforma, se han registrado avances muy superiores a las implementaciones iniciales como COM y CORBA, por lo que los componentes .net se han convertido en verdaderas piezas de ensamblaje. Nuevos paradigmas como las Fábricas de Software proveen toda una filosofía para realizar una acertada transición entre la manera antigua de producir software (a partir de productores independientes con baja reutilización) y la nueva, ensamblado a partir de componentes independientemente desarrollados.

Nuestro país ha experimentado un desarrollo acelerado en los últimos años en la esfera informática; así lo demuestra la creación de nuevos centros y planes de estudio como es el caso de la Universidad de las Ciencias Informáticas (UCI) y los Institutos Politécnicos de Informática (IPI), por solo mencionar algunos. Entre los principales objetivos que han propiciado esta iniciativa, se encuentran la informatización de la sociedad y la producción y exportación de software aportando así significativamente también al desarrollo de la economía.

Por los motivos antes expuestos, se hace muy importante la actualización constante de nuestras prácticas de ingeniería así como el estudio de los nuevos paradigmas que internacionalmente surgen y se aplican. El DSBC como una extensión de la Ingeniería de Software que se encuentra entre las más promisorias en los últimos tiempos, constituye una de las principales propuestas que debemos analizar para poderla adaptar y aplicar exitosamente en cualquier institución de nuestro país que lo requiera.

La estructura anterior al polo estaba enfocada a proyectos y semejante a la disposición del cliente, provocando duplicación de esfuerzos en cada uno de los proyectos.

Un análisis a fondo propició la reestructuración del polo, que comienza a regirse por las funcionalidades generales detectadas en el cliente, apostando por el desarrollo basado en componentes que sean reutilizables de un proyecto a otro, logrando así mismo una mejor infraestructura de producción de software.

La nueva estructura ocasiona la necesidad de trazar los lineamientos y metodologías que deben guiar el desarrollo por componentes para lograr alcanzar mayor producción y reutilización. De esta situación se derivó el **problema a resolver**, constituido por la incógnita de cómo ensamblar los componentes de software en proyectos productivos en el polo PetroSoft.

Mucho se ha trabajado hasta el momento y se han registrado avances significativos que han posibilitado la aplicación de nuevas filosofías y procedimientos relacionados con la producción de software basado en componentes que han dado grandes beneficios desde los primeros momentos. Autores reconocidos internacionalmente en esta esfera de la talla de A. Brown e Ivar Jacobson, han propuesto nuevas metodologías y prácticas, muchas de las cuales se aplican en la actualidad en prestigiosas empresas de software del mundo. Sin embargo, ninguno de estas propuestas puede utilizarse íntegramente en el marco productivo del polo, puesto que no están preparadas para ajustarse a condiciones propias del entorno productivo, tales como: (1) la mayor parte de nuestra fuerza laboral está constituida por estudiantes y (2) la Universidad tiene estructurado su proceso docente educativo dirigido a tres frentes fundamentales: docencia, investigación y producción, lo que limita el tiempo dedicado netamente al desarrollo de software, entre otras.

Dadas las problemáticas tratadas y debido a la necesidad que impera de darle solución a las mismas, se ha decidido cumplir con el siguiente **objetivo general:** Definir el proceso de ensamblaje de componentes de software en el polo PetroSoft. Como complemento para alcanzar el objetivo antes mencionado se han definido los siguientes **objetivos específicos**:

1. Evaluar las posibles metodologías de ensamblaje de componentes existentes.

2. Especificar una metodología de ensamblaje de componentes de software adecuada al polo PetroSoft.

Motivo de las características anteriormente expuestas, se ha enmarcado el **objeto de estudio** en el ensamblaje de componentes de software.

En consecuencia a la problemática en cuestión el **campo de acción** serán los proyectos productivos en el polo PetroSoft. Una vez analizadas con claridad las necesidades crecientes que surgían en el polo a raíz de la nueva restructuración aplicada, para lograr altos índices productivos se pudo llegar a la conclusión de que si se logran identificar las tareas a realizar y las técnicas a aplicar para el ensamblaje de componentes de software del polo, se logrará definir una estrategia de ensamblaje de dichos componentes para la construcción de soluciones informáticas del polo PetroSoft lo que constituye la **Hipótesis** de esta investigación.

Con el objetivo de darle solución a las cuestiones relacionadas, se han trazado un conjunto de tareas investigativas y orientaciones concretas que constituirán la guía para resolver los problemas tratados, en función de las necesidades prácticas y cognoscitivas existentes. Las mismas son:

1. Describir el estado del arte relacionado con el objeto de estudio.
2. Identificar las metodologías de desarrollo existentes relacionadas con el desarrollo de sistemas basados en componentes de software.
3. Generalizar los elementos identificados para obtener la propuesta de ensamblaje de sistemas basados en componentes de software para el polo productivo PetroSoft.
4. Definir la propuesta de ensamblaje de sistemas basados en componentes de software para el polo productivo PetroSoft.
5. Evaluar la propuesta de de ensamblaje de sistemas basados en componentes de software para el polo productivo PetroSoft.
6. Aplicar la propuesta de de ensamblaje de sistemas basados en componentes de software para el polo productivo PetroSoft en proyectos pilotos.

Resaltan entre los métodos científicos teóricos, el histórico lógico; debido a que resulta de extrema importancia para la comprensión del DSBC el estudio de su devenir histórico. La tecnología de componentes se ha basado en gran medida en otros procesos similares como el que ha tenido lugar con la evolución dividida en etapas lógicas de las metrópolis o el desarrollo científico técnico, los cuales en uno y otro caso han evolucionado hasta llegar finalmente a la creación e integración de componentes. Prestigiosos autores aseveran estas teorías y han publicado analogías de las mismas como es el caso

del Ingeniero en Sistemas Computacionales Julio Casal Terreros en su artículo “Desarrollo de Software Basado en Componentes” el cual referenciamos en el presente trabajo.

También se utilizó el método experimental, puesto que parte importante del desarrollo de la estrategia de ensamblaje, es la aplicación de las prácticas y técnicas de ingeniería definidas en cada momento, como parte de un proceso imprescindible de retroalimentación del cual se pudiera nutrir la propuesta final para corregir a tiempo posibles fallos o para introducir mejoras en la definición de la misma.

Una vez cumplimentadas todas las tareas reflejadas y las investigaciones necesarias se espera obtener entre los posibles resultados, la identificación de los métodos que favorecerán el ensamblaje de componentes de software en el polo. La realización de una metodología de esta envergadura supondrá una investigación profunda en todos los campos relacionados al DSBC pero brindará beneficios concretos; debido a que adaptada a las condiciones específicas del polo, no solo tornará mas cómodo y manejable el trabajo a los miembros de los equipos de desarrollo de los proyectos, sino que además servirá como guía rectora en el Proceso de Desarrollo e Ingeniería de Software Basada en Componentes proporcionando de esta manera todos los bienes que encierra la misma.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción:

El Desarrollo de Software Basado en Componentes (DSBC), obtuvo desde sus inicios un lugar privilegiado en el desarrollo. Aspectos implícitos en esta tecnología, tales como el aumento de la reutilización, la reducción de los costos y el esfuerzo de desarrollo, así como el crecimiento de la productividad, hicieron de esta propuesta una de las mas prometedoras de su tiempo. La misma fue adoptada rápidamente por las grandes empresas productoras de software a nivel internacional.

A lo largo de este capítulo se realiza un detallado recorrido de las principales características del DSBC, en donde se abordan aspectos muy significativos de esta tecnología tales como: su estado actual, su impacto económico, metodologías desarrolladas para su aplicación, entre otras. También se relacionan las características productivas actuales del polo productivo PetroSoft, y se hace referencia a las ventajas que introduciría la aplicación de la tecnología de componentes dentro del mismo.

1.2. Historia y definiciones de Componentes de Software.

La iniciativa de utilizar componentes en el desarrollo de software no es nueva, incluso para muchos autores no es más que una evolución de la metodología orientada a objetos, debido a que su concepción parte de las características del diseño de la última. Pero la historia del Desarrollo de Software Basado en Componentes (DSBC) es aún más antigua. A partir de la revolución industrial y de la necesidad de organizar la producción en base a componentes como en el caso de las producciones de automóviles, surgen novedosos conceptos como los de estandarización, adaptación y ensamblaje de componentes. De todos estos principios se nutrieron los desarrolladores de software para empezar a idear una estrategia similar que le brindará los beneficios que proporcionó en otras áreas.

En esencia, un componente es una pieza de código pre elaborado que encapsula alguna funcionalidad expuesta a través de interfaces estándar. Los componentes son los "ingredientes de las aplicaciones", que se juntan y combinan para llevar a cabo una tarea. Es algo muy similar a lo que podemos observar en el equipo de música que tenemos en nuestra sala. Cada componente de aquel aparato ha sido diseñado para acoplarse perfectamente con sus pares, las conexiones son estándar y el protocolo de comunicación está ya preestablecido. Al unirse las partes, obtenemos música para nuestros oídos. (TERREROS 2004)

Al definir el concepto de DSBC existen aspectos muy importantes que se deben tener en cuenta, entre ellos resalta la diferenciación de las características existentes entre el mismo y el paradigma de objetos. La Programación Orientada a Objetos (POO), se plantea cuando el software puede ser desarrollado a través de un modelo mental de un objeto real o imaginado. Por su parte el DSBC estipula que el software debe ser desarrollado sobre la base de componentes prefabricados. Connotados expertos han realizado varias definiciones de componentes de software, tomando como base la metodología de la programación orientada a objetos y el modelado a través de UML:

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. (SZYPERSKI, CLEMENS 2002)

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 (Componente IBM, (IBM-WEBSPHERE 2001; IRIBARNE MARTÍNEZ)).

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 (Componente SEI, (BROWN 1999; IRIBARNE MARTÍNEZ)).

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

Las definiciones ofrecidas internacionalmente para componente de software no son mutuamente excluyentes, entre sí se complementan y construyen. Existen además un grupo de características adicionales que son imprescindibles a la hora de catalogar un producto como componente de software o no. Seguidamente se muestran tales características, referidas por Maribel Ariza Rojas y Juan Carlos Molina García, en su artículo INTRODUCCIÓN Y PRINCIPIOS BÁSICOS DEL DESARROLLO DE SOFTWARE BASADO EN COMPONENTES.

– **Identificable**: Debe tener una identificación que permita acceder fácilmente a sus servicios y que permita su clasificación.

- **Auto contenido**: Un componente no debe requerir de la utilización de otros para finiquitar la función para la cual fue diseñado.
- **Puede ser remplazado por otro componente**: Se puede remplazar por nuevas versiones u otro componente que lo reemplace y mejore.
- **Con acceso solamente a través de su interfaz:** Debe asegurar que estas no cambiarán a lo largo de su implementación.
- **Sus servicios no varían**: Las funcionalidades ofrecidas en su interfaz no deben variar, pero su implementación sí.
- **Bien Documentado**: Un componente debe estar correctamente documentado para facilitar su búsqueda si se quiere actualizar, integrar con otros, adaptarlo, etc.
- **Es genérico:** Sus servicios debe servir para varias aplicaciones.
- **Reutilizado dinámicamente:** Puede ser cargado en tiempo de ejecución en una aplicación.
- **Independiente de la plataforma:** Hardware, Software, S.O. (ARIZA 2004)

Como conclusión de estas definiciones se pueden realizar algunas valoraciones. Los componentes son partes software que se pueden combinar con otros componentes para generar un conjunto aún mayor tales como otro componente, subsistema o sistema. Un componente juega el papel de una unidad software reutilizable que puede interoperar con otros módulos software mediante sus interfaces. Un componente define una o más interfaces desde donde se puede tener acceso a los servicios que éste ofrece a los demás componentes.

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

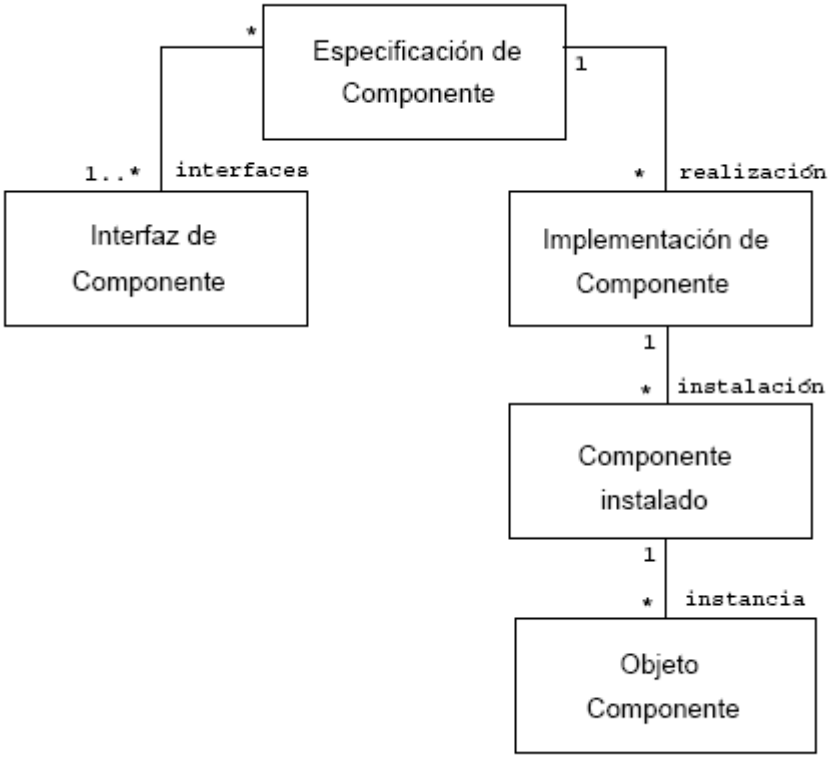

Fig. 1 Formas en las que puede presentarse el término componente.

Es altamente significativo el tiempo y los altos costos invertidos en el que incurren los desarrolladores tradicionales de aplicaciones. La baja reutilización de software, es decir, empezar el desarrollo desde cero es un gran reto, incluso para las grandes empresas productoras de software a nivel mundial. Si se mantuviera este tipo de práctica, sin duda decaería vertiginosamente la productividad total de las grandes empresas de software así como no tendrían cabida las pequeñas y medianas empresas que desean construir sus propias soluciones y adquirir tecnologías.

Las empresas pequeñas han estado siempre al margen del desarrollo de software; solo hasta la década pasada se les permitió la introducción a este campo. Esta introducción se originó ante la demanda de estas por buscar sus propios productos para dejar de depender de aquellos que las grandes empresas ponían en el mercado. Por otra parte, las empresas buscaban la reducción de costos en la tecnología (Hardware, Software e Infraestructura de Comunicación). (ARIZA 2004)

El DSBC busca, dentro de otros objetivos, reducir el tiempo de trabajo, el esfuerzo que requiere implementar una aplicación y los costos del proyecto, y, de esta forma, incrementar el nivel de

productividad de los grupos desarrolladores y minimizar los riesgos globales sin incurrir en gastos exorbitantes. De esta manera, las pequeñas empresas pueden tener una mayor confiabilidad a la hora de realizar una inversión tecnológica.

El hecho de poder integrar componentes de software para obtener aplicaciones funcionales, ofrece otra gran ventaja concerniente en poder personalizar los productos a la medida de las necesidades de los clientes. Esto permite a las empresas y a los desarrolladores adquirir según sus necesidades, las tecnologías que mas se adapten a las mismas y de esta manera no incurrir en gastos por cuenta de licenciamiento o soporte y actualización, ya que muchas de estas son gratis y se encuentran liberadas bajo la premisa de *Freeware (Tecnologías Libres)* y GPL (*General Public License*), por lo cual no se tendría que invertir en costosas tecnologías cuando se pueden incorporar estas alternativas de igual manera a nuestro sistema.

Debe tomarse en cuenta además que de el DSBC pertenece al paradigma de programación de sistemas abiertos, estos son extensibles y tienen una interacción con componentes heterogéneos que se pueden acoplar o separar de los sistemas de manera dinámica, es decir que los componentes pueden ser remplazados, por otros independientemente de su arquitectura y desarrollo. De esta forma se puede hacer una analogía entre DSBC y el mundo percibido por los seres humanos como la que nos muestran M. Ariza y Juan C. Molina a continuación, desde un punto de vista menos formal y con una perspectiva basada en ejemplos cotidianos donde se tienen sistemas abiertos y cerrados.

Por ejemplo, el ser humano es un sistema abierto y realiza interacciones con el medio (otro sistema), la mayoría de los sistemas en el mundo percibido por el hombre son abiertos y necesitan una interacción con el medio para poder sobrevivir como en el caso del hombre (alimentos, aire, relaciones, etc). Esta interacción es la que permite que crezca el sistema (extensión), dándole la posibilidad de mejorar con el tiempo y lograr una estabilidad (evolución para el caso del hombre). (ARIZA 2004)

De manera general podemos llegar a la conclusión de que los componentes de software son todos aquellos recursos especialmente desarrollados para cumplir con un fin concreto y que pueden formar solo o junto con otro u otros, un entorno funcional requerido por cualquier proceso predefinido. Son independientes entre si, y tienen su propia estructura e implementación.

1.3. DSBC a nivel internacional. Estado actual.

Actualmente el DSBC ha obtenido un lugar cimero a nivel Internacional, ocupando en los primeros países exportadores de software como India, Irlanda e Israel toda la atención de las empresas de

desarrollo de software. Una de las iniciativas más recientes, que clasifica entre las ideas más promisorias basada en esta tecnología es la de “Las Fabricas de Software”.

El concepto de “**Fábrica de Software**”, se ha convertido rápidamente en uno de los más novedosos y prometedores en la industria de software. El mismo, está directamente relacionado con la manufactura de software. Toma como base de la producción y ensamblaje de sistemas, precisamente a la tecnología de componentes; debido a que brinda características tales como la implementación de interfaces y envolturas que posibilitan la interacción entre los diferentes componentes. Una fábrica de software, presenta algunas ideas básicas que caracterizan su manera de producción, tal como refiere prestigioso Ingeniero de Sistemas Computacionales Julio Casal Terreros a continuación.

Una fábrica de software es una línea de producto que configura herramientas de desarrollo extensibles como Microsoft Visual Studio Team System con contenido empaquetado y guías, cuidadosamente diseñadas para construir tipos específicos de aplicaciones. Una fábrica de software contiene tres ideas básicas: (TERREROS 2004)

- Un esquema de fabricación. La analogía de esta idea 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.

El concepto de “Fábrica de Software”, así como los patrones industriales que pretende adaptar a la industria del software, introducen algunas interrogantes nuevas tales como: ¿es posible automatizar el desarrollo del software?, realmente ya se ha logrado. Los software destinados a la edición que utilizan WYSIWYG (What You See Is What You Get (en inglés, "lo que ves es lo que obtienes")) por ejemplo, realizan automáticamente y por ende de manera más fácil la construcción y mantenimiento de las

interfaces gráficas de los usuarios, derivándose de esta práctica beneficios tales como independencia de dispositivos y el ensamblaje visual.

Formas similares de automatización pueden encontrarse en los diseños de las Bases de Datos. El tema recurrente que podemos observar en la creación de este tipo de herramientas es el hecho de pensar en modelos, más que solamente en objetos, lo que constituye una gran ventaja en cuanto a la reducción de los tiempos y esfuerzos de los equipos de desarrollo.

Pero incluso cuando los modelos resultan muy importantes, no lo son todo. Para incrementar hasta los niveles más altos la productividad es necesario asentar las bases y crear habilidades relacionadas con la configuración, adaptación, y ensamblaje rápido de componentes desarrollados independientemente, auto descriptivos, e independientes de ubicación, con el objetivo de producir sistemas similares basados en los mismos que puedan ser desplegados con éxito en la mayor brevedad posible.

Para lograr lo anteriormente expresado, será necesaria la creación y utilización de patrones con dominio específico, tales como patrones arquitectónicos que puedan ser puestos en práctica frecuentemente. También será necesaria la utilización de frameworks y herramientas que se alineen tanto con la arquitectura del producto como con el ciclo de vida del producto. Además debemos analizar la manera en la que realizamos la captura de requisitos y la implementación posterior de los mismos, para implementar las funcionalidades del sistema como componentes que luego puedan ser reutilizados en sistemas que brinden prestaciones similares. Necesitamos disponer de las mejores prácticas, contenido reutilizable y distintos tipos de patrones.

Las fábricas de software son una iniciativa que se está desarrollando y demostrando su eficiencia en la práctica cada día más 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.

1.4. Programación Orientada a Componentes

A partir del surgimiento y evolución de la tecnología de componentes, nace el paradigma de la programación orientada a componentes (POC) (SZYPERSKI, C. PFISTER, C. 1997), como resultado de una extensión natural de la orientación a objetos, concebida para los sistemas abiertos principalmente. Este paradigma propugna el desarrollo y reutilización de los componentes como base del ensamblaje de sistemas de software en el marco internacional.

Sin embargo, contar con repositorios de componentes anteriormente desarrollados y probados, no es suficiente tampoco, es necesario garantizar su reutilización. La reutilización de un componente no se limita al hecho de ser utilizado más de una vez, sino que se debe garantizar la usabilidad y adaptabilidad del mismo en contextos ajenos a aquellos para el que fue diseñado. A raíz de esto inmediatamente surgieron otras ideas relacionadas con la adquisición de componentes de software a nivel mundial.

Uno de los sueños que siempre ha tenido la ingeniería del software es el de contar con un mercado global componentes, al igual que ocurre con otras ingenierías, o incluso con el hardware. De hecho, la analogía con los circuitos integrados (IC) llegó a poner de moda los términos software IC, software bus y software backplane, aunque nunca se haya podido llegar más allá de la definición de estos conceptos. (FUENTES 2002)

De todos los conceptos e iniciativas expuestas anteriormente, nace la idea a partir de los componentes reutilizables dentro de un mercado global de software de desarrollar sistemas donde todo es propietario, la misma recibió la denominación de componentes COTS (commercial off-the-shelf). Este concepto va a cambiar la idea tradicional en el desarrollo de software y propone el ensamblaje de sistemas a partir de componentes desarrollados por terceros y generalmente suministrados de manera binaria. Aparecen nuevas modificaciones en las fases de desarrollo concernientes a la búsqueda y adaptación de componentes que se necesitan.

1.5. Tipos de Componentes de Software

Componentes de interfaz de usuario (IU). Frecuentemente los sistemas de software necesitan ofrecer a los usuarios una manera cómoda y práctica de interactuar con el mismo. Tanto en las aplicaciones Web como Desktop, se han desarrollado históricamente todo tipos de componentes visuales que facilitan el trabajo a los desarrolladores, proporcionándoles prestaciones tales formularios de datos, implementaciones de eventos, validación de campos entre otras. El desarrollo de componentes IU, bien definidos y documentados, siempre garantizará la optimización de los procesos de desarrollo que sobrevengan en el futuro.

Componentes de proceso de usuario. En múltiples ocasiones se conoce de manera anticipada las posibles interacciones del usuario con el sistema; un ejemplo sería si se tiene una aplicación comercial. En dicha aplicación se podría implementar un procedimiento para conocer los datos referentes a los productos que se producen u ofertan. También se podría presentar una lista de productos de acuerdo a

diferentes categorías o parámetros de clasificación para que se seleccione y se conozcan los detalles del elemento seleccionado.

Otro ejemplo de un procedimiento predecible es el proceso de compra donde es sabido que se necesitan los datos del usuario, los detalles de el o los productos que desea comprar así como la forma de pago y por último si se desea o no el envío especificando los detalles del mismo. Como ya se puede adivinar en estas interacciones, es factible para facilitar la organización y sincronización de las interacciones con el sistema utilizar para el desarrollo del mismo componentes de procesos de usuarios individuales. De esta manera, varias interfaces podrían reutilizar el mismo proceso de lógica de negocio, aunque en el mismo no se incluya el código de los elementos de interfaz de usuario.

Flujos de trabajo empresariales. Una vez que el proceso de usuario ha recopilado los datos necesarios, estos se pueden utilizar para realizar un proceso empresarial. Por ejemplo, tras enviar los detalles del producto, el pago y el envío a la aplicación comercial, puede comenzar el proceso de cobro del pago y preparación del envío. Gran parte de los procesos empresariales conllevan la realización de varios pasos, los cuales se deben organizar y llevar a acabo en un orden determinado. Por ejemplo, el sistema empresarial necesita calcular el valor total del pedido, validar la información de la tarjeta de crédito, procesar el pago de la misma y preparar el envío del producto. El tiempo que este proceso puede tardar en completarse es indeterminado, por lo que sería preciso administrar las tareas necesarias, así como los datos requeridos para llevarlas a cabo. Los flujos de trabajo empresariales definen y coordinan los procesos empresariales de varios pasos de ejecución larga y se pueden implementar utilizando herramientas de administración de procesos empresariales.

Componentes empresariales. Independientemente de si el proceso empresarial consta de un único paso o de un flujo de trabajo organizado, la aplicación requerirá probablemente el uso de componentes que implementen reglas empresariales y realicen tareas empresariales. Por ejemplo, en la aplicación comercial, debería implementar una funcionalidad que calcule el precio total del pedido y agregue el costo adicional correspondiente por el envío del mismo. Los componentes empresariales implementan la lógica empresarial de la aplicación.

Agentes de servicios. Cuando un componente empresarial requiere el uso de la funcionalidad proporcionada por un servicio externo, tal vez sea necesario hacer uso de código para administrar la semántica de la comunicación con dicho servicio. Por ejemplo, los componentes empresariales de la aplicación comercial descrita anteriormente podrían utilizar un agente de servicios para administrar la comunicación con el servicio de autorización de tarjetas de crédito y utilizar un segundo agente de servicios para controlar las conversaciones con el servicio de mensajería.

Los agentes de servicios permiten aislar las idiosincrasias de las llamadas a varios servicios desde la aplicación y pueden proporcionar servicios adicionales, como la asignación básica del formato de los datos que expone el servicio al formato que requiere la aplicación.

Interfaces de servicios. Para exponer lógica empresarial como un servicio, es necesario crear interfaces de servicios que admitan los contratos de comunicación (comunicación basada en mensajes, formatos, protocolos, seguridad y excepciones, entre otros) que requieren los clientes. Por ejemplo, el servicio de autorización de tarjetas de crédito debe exponer una interfaz de servicios que describa la funcionalidad que ofrece el servicio, así como la semántica de comunicación requerida para llamar al mismo. Las interfaces de servicios también se denominan fachadas empresariales.

Componentes lógicos de acceso a datos. La mayoría de las aplicaciones y servicios necesitan obtener acceso a un almacén de datos en un momento determinado del proceso empresarial. Por ejemplo, la aplicación empresarial necesita recuperar los datos de los productos de una base de datos para mostrar al usuario los detalles de los mismos, así como insertar dicha información en la base de datos cuando un usuario realiza un pedido. Por tanto, es razonable abstraer la lógica necesaria para obtener acceso a los datos en una capa independiente de componentes lógicos de acceso a datos, ya que de este modo se centraliza la funcionalidad de acceso a datos y se facilita la configuración y el mantenimiento de la misma.

Componentes de entidad empresarial. Cuando se desarrolla un sistema de software basado en componentes, es usual que se requiera el paso de datos entre los mismos. Por ejemplo, en la aplicación comercial es necesario pasar una lista de productos de los componentes lógicos de acceso a datos a los componentes de la interfaz de usuario para que éste pueda visualizar dicha lista. Los datos se utilizan para representar entidades empresariales del mundo real, como productos o pedidos. Las entidades empresariales que se utilizan de forma interna en la aplicación suelen ser estructuras de datos, como conjuntos de datos, DataReader o secuencias de lenguaje de marcado extensibles (XML), aunque también se pueden implementar utilizando clases orientadas a objetos personalizadas que representan entidades del mundo real necesarias para la aplicación, como productos o pedidos.

Componentes de seguridad, administración operativa y comunicación. Seguramente la aplicación necesitará la implementación de funcionalidades que garanticen aspectos como la asignación de privilegios así como la gestión de trazas, administración de excepciones entre otros. Componentes que encapsulen las funcionalidades antes descritas, que se adapten a procesos genéricos relativos a la seguridad y que permitan la comunicación con otros servicios y aplicaciones, serán requeridos en la mayoría de los sistemas informáticos.

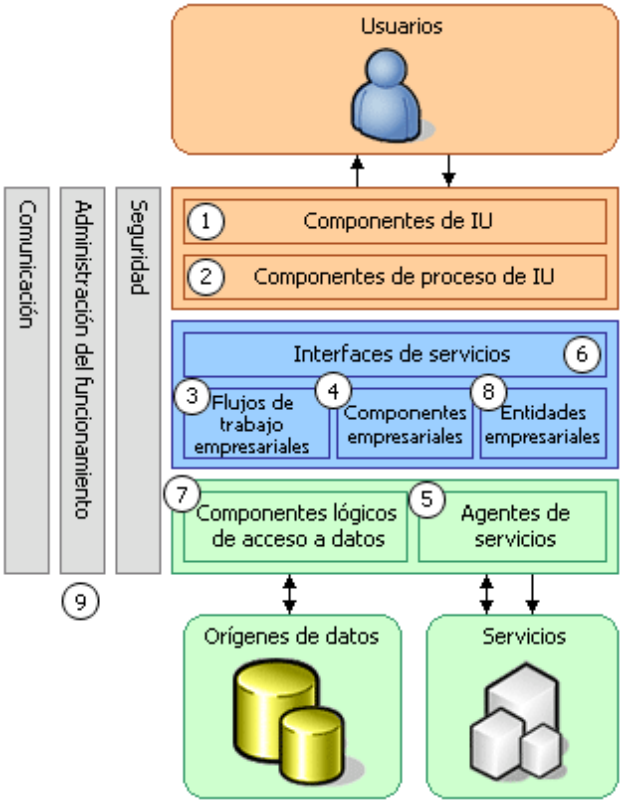

Fig. 2 Tipos de Componentes y sus relaciones.

1.6. ¿Por qué desarrollar software basado en componentes?

El construir software ensamblando componentes conlleva grandes promesas para la ingeniería de software de la nueva generación. Es necesario ir más allá y asegurar que no se puede hablar de ingeniería antes de haber dominado este paso en el desarrollo. Si no, se está hablando de hacer las cosas a mano, algo similar a la manufactura temprana previa a la Revolución Industrial. Así que, es claro que se debe comenzar a construir software basado en componentes. (SZYPERSKI, CLEMENS 2002)

El desarrollo de software basado en componentes (DSBC), es una tecnología que ha empezado a demostrar que ofrece ventajas en tiempo de desarrollo y reducción de costos en el proceso de desarrollo de software, estas ventajas se convierten en la principal razón de porque desarrollar aplicaciones computacionales sobre una arquitectura de componentes, a continuación se relacionan estas ventajas.

1.6.1. Beneficios del Desarrollo de Software basado en Componentes

La adopción de la tecnología de componentes, brinda un conjunto de beneficios que se encuentran directamente relacionados con aspectos tales como la reutilización de los activos de

software, así como la reducción del esfuerzo y el aumento de la productividad. A continuación se encuentran desglosadas las principales características que hacen del DSBC una de las iniciativas más prometedoras de estos tiempos.

Reutilización del software: La posibilidad de utilizar constantemente componentes de software que se encuentren previamente desarrollados por nuestra entidad o por terceros, aporta la ventaja de reducir notablemente los tiempos de desarrollo y el esfuerzo del equipo de desarrollo.

Simplifica las pruebas: El uso de componentes de software permite que los mismos puedan ser probados de manera unitaria para garantizar que cumplen con las funcionalidades para las que fue diseñado. Esta característica influye directamente en la reducción de los márgenes de error así como en la ubicación de manera más acelerada de posibles fallas.

Simplifica el mantenimiento del sistema: En un sistema desarrollado a partir de componentes en donde los mismos están débilmente acoplados entre ellos, los desarrolladores se encuentran en la libertad de agregar o separar componentes o cambiar los mismos para lograr cumplir con éxito con las funcionalidades del sistema.

Mayor calidad: Debido a que los componentes pueden ser constantemente mejorados por los desarrolladores de los mismos, los sistemas ensamblados a partir de estos incrementarán su calidad con el paso del tiempo debido a que nuevas versiones de los componentes podrán ser adheridas o sustituirán viejos componentes agregándole nuevas funcionalidades, mejor concebidas e implementadas.

Ciclos de desarrollo más cortos: En el DSBC, la adición de un nuevo componente de software, tomará días como mucho. En caso de que el equipo de desarrollo deba asumir la implementación de las funcionalidades dadas, podría tardarse meses o aun más tiempo en llevar la actividad a término e integrarla al sistema.

Mejor Retorno sobre la Inversión (*ROI*)[1]: Utilizando de manera correcta la tecnología de componentes, podrá garantizarse la reintegración de toda la inversión realizada en la fase de aprovisionamiento, donde pueden haberse efectuado gastos concernientes a compras de componentes o licenciamiento de los mismos.

Funcionalidad mejorada: Para usar un componente que contenga la implementación de un requisito dado, solo se necesita entender su naturaleza, más no sus detalles internos. De esta

1 Return Of the Investment

manera, funcionalidades que anteriormente podrían resultar imprácticas para la entidad se encuentre desarrollando el software, se vuelven ahora completamente asequibles.

1.6.2. Inconvenientes del Desarrollo de Software basado en Componentes

Si no existen los componentes, se hace necesario desarrollarlos y se puede perder mucho tiempo en la construcción de los mismos. Durante el ensamblaje de componentes, pueden surgir conflictos propiciados por incompatibilidades entre los diferentes estándares establecidos por las empresas desarrolladoras de dichos componentes. En mayor o menor medida, las funcionalidades siempre van atadas a los requisitos propios del negocio descrito, por lo que cuando se utilizan componentes anteriormente desarrollados, se corre el riesgo de que el software no cumpla las expectativas del cliente. Cuando se utilizan componentes desarrollados por terceros, las actualizaciones de los mismos no están en manos de los desarrolladores del sistema.

1.7. Significación del uso del DSBC.

Debido a problemas ingenieriles de la industria del software, tales como el atraso en las fechas de entrega de los productos y el incumplimiento de los tiempos acordados con los clientes, así como la escasez de personal lo suficientemente capacitado como para garantizar la calidad de los productos, el DSBC viene a solucionar y apoyar en gran medida estos contratiempos y a brindar beneficios de todo tipo que bien utilizados asegurarán el éxito en cualquier empresa. El polo Petrosoft no escapa a estos problemas y la implantación de una arquitectura basada en componentes, flexible y a la vez robusta vendría a mejorar significativamente todo el proceso de producción de software en el polo. Estos beneficios brindados se clasifican de dos maneras principalmente:

1.7.1. Tecnológico

Como mencionamos anteriormente el uso del paradigma de DSBC posee determinadas ventajas, la reutilización del software determinara el acortamiento de los plazos de entrega, así como del esfuerzo de los equipos de desarrollo.

La simplificación de las pruebas permitirá que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados,

además de garantizar la calidad del producto final, este beneficio incidirá también en la disminución de los tiempos de desarrollos .

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, a menudo la preparación y conocimiento del equipo de desarrollo no es la suficiente, esta característica del DSBC facilita la integración del sistema facilitando el ensamblaje de dichos componentes sin tener que tener altos conocimientos.

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, esta característica es la clave principal para garantizar una evolución exitosa de los productos.

1.7.2. Económico

La economía de la ingeniería del software basada en componentes tiene un atractivo evidente. En teoría, debería proporcionar a las empresas de software unas ventajas notables en lo referente a la calidad y a los tiempos de realización. Éstas a su vez deberían traducirse en ahorros de coste. ¿Existen datos reales que apoyen nuestra intuición?

A lo largo de los últimos años, existen notables evidencias procedentes de casos prácticos en la industria, las cuales indican que es posible obtener notables beneficios de negocios mediante la reutilización agresiva del software. Se mejora tanto la calidad del producto, como la productividad del desarrollador y los costes en general. (PRESSMAN 2005)

Económicamente el DSBC debido a todos sus beneficios, significa un crecimiento directo de los ingresos económicos por concepto de ventas de software. En el tiempo que se desarrollaba y se vendían los proyectos anteriormente, con este paradigma se podrán estar desarrollando y vendiendo varios, lo cual repercute directamente en los ingresos de el país. Entre las principales características que aseveran esto están:

La aplicación del paradigma del DSBC garantiza la disminución en los plazos de entrega de los productos, debido a que se empiezan a ensamblar componentes para satisfacer las necesidades de los clientes, en los casos ideales ya estos componentes debían estar elaborados previamente, lo que asegura un ahorro muy significativo del tiempo, cuando antes se tardarían meses

desarrollando alguna funcionalidad, ahora esta podría solucionarse en días. En el polo Petrosoft, esta ventaja significaría un aumento de la disponibilidad, la cual suele ser variable, y en un futuro inmediato la contratación de mayor cantidad de proyectos con los clientes.

Usando correctamente la estrategia del DSBC, el retorno sobre la inversión (ROI) puede ser más favorable que desarrollando los componentes uno mismo, el cálculo del costo de desarrollo de los proyectos o partes de ellos, atendiendo a aspectos como esfuerzo mental, recursos y otros, en algunas circunstancias puede ser mayor que el precio a pagar por el desarrollo de un tercero.

CALIDAD

En un entorno ideal, un componente de software que se desarrolle para su reutilización estará verificado en lo tocante a su corrección y no contendrá defectos. En realidad, la verificación formal no se lleva a cabo de forma rutinaria y los defectos pueden aparecer y aparecen. Sin embargo, con cada reutilización, los defectos se van hallando y eliminando, y como resultado, la calidad del componente mejora. Con el tiempo el componente llega a estar libre de defectos.

En un estudio realizado por Hewlet – Packard, se informa que la tasa de defectos para el código reutilizado es de 0.9 por KLDC, mientras que la tasa para el software recién desarrollado es de 4,1 defectos por KLDC. Para que una aplicación esté compuesta por un 68 por 100 de código reutilizado, la tasa de defectos será de 2,0. Aun cuando existen recortes anecdóticos que abarcan un espectro razonablemente amplio en porcentajes de mejoras de la calidad, es justo que la reutilización proporcione un beneficio no despreciable en función de la calidad y fiabilidad del software proporcionado. (PRESSMAN 2005)

PRODUCTIVIDAD

Cuando se aplican componentes reutilizables a lo largo del proceso del software, se invierte menos tiempo creando los planes, modelos, documentos, códigos y datos necesarios para crear un sistema fiable. Se sigue proporcionando un mismo nivel de funcionalidad al cliente con menos esfuerzo. Consiguientemente, mejora la productividad. Aun cuando los informes acerca de las mejoras de productividad son sumamente difíciles de interpretar, parece que una reutilización de entre el 30 y el 50 por 100 puede dar lugar a mejoras de productividad que se encuentren entre el intervalo que media entre el 25 y el 40 por 100. (PRESSMAN 2005)

Un componente de Software brinda la posibilidad de ser utilizado sin la necesidad de comprender su funcionamiento interno, por lo cual seria muy útil poderlo utilizar sin la inversión de tiempo y esfuerzo, que muchas veces se ve incrementado debido a factores como la escasez de

conocimientos y experiencia dentro del equipo de desarrollo, en conocerlo y adaptarlo para utilizarlo. Por tanto, soluciones que antes serian demasiado costosas de implementar en el ámbito del proyecto, serian ahora completamente asequibles, brindando así todas las ventajas que anteriormente se han relacionado.

COSTE

El ahorro neto de costes para la reutilización se estima proyectando el coste del proyecto como si se hubiera desarrollado este desde cero y restando después la suma de los costes asociados para la reutilización y el coste real del software cuando este finalmente se implanta. (PRESSMAN 2005)

Los costes asociados a la reutilización incluyen:

- Modelado y análisis del dominio.
- Desarrollo de la arquitectura del dominio.
- Incremento de la documentación para facilitar la reutilización.
- Mantenimiento y mejora de componentes de la reutilización.
- Licencias para componentes adquiridos externamente.
- Creación o adquisición y funcionamiento de un depósito para la reutilización.
- Formación del personal en el diseño y construcción para la reutilización.

Resumiendo, la Ingeniería de Software Basada en Componentes proporciona unos beneficios inherentes en lo tocante a la calidad del software, productividad del desarrollador y coste general del sistema. Sin embargo es preciso vencer muchas dificultades antes de que el modelo de proceso ISBC se utilice ampliamente en la industria.

1.8. Metodologías para el DSBC existentes

Históricamente, los desarrolladores de software han seguido un enfoque de desarrollo ascendente (top - down) basados en el clásico modelo en cascada (análisis – diseño -implementación), se establecen como aspectos fundamentales los requisitos y la definición de la arquitectura de la aplicación y luego se van diseñando e implementando cada parte de la misma. Este modelo ha dado paso a la ISBC, donde la idea central es desarrollar software a partir del ensamblado de componentes previamente elaborados, implementados no solo para integrarse fácilmente a otros sistemas sino además para se reutilizados en

el futuro. Consecuentemente, muchos prestigiosos autores han elaborado y publicado fuertes bases teóricas consistentes en diferentes metodologías y establecimientos de etapas para el DSBC algunas de las cuales relacionamos a continuación.

Primeramente citaremos la visión de John C. Dean y Mark Vigder los cuales identifican cuatro procesos de desarrollo de sistemas basados en componentes, que son: la búsqueda de componentes software (residentes en repositorios) que pueden satisfacer las necesidades del usuario o de la arquitectura de software de la aplicación; la evaluación de componentes según algunos criterios; la adaptación o extensión de los componentes seleccionados para que se ajusten a las necesidades de la arquitectura del sistema; y el ensamblaje o la integración de estos componentes. (DEAN 1997)

También se puede citar una propuesta del proceso de desarrollo de sistemas basados en componentes comerciales COTS (Commercial Off-The-Shelf), como la que ofrece (CRAIG M. 2001) la cual es una de las más extendidas en este campo. La misma describe las fases y lineamientos para desarrollar sistemas a partir de componentes desarrollados por terceros que se pueden realizar a pedido, comprar otros previamente elaborados o comprar licencias según convenga.

El adjetivo “COTS” hará referencia a una clase especial de componentes: entidades comerciales (es decir, que se pueden vender o licenciar) que permiten el empaquetado, distribución, almacenamiento, recuperación y particularización por parte del usuario, que generalmente son de grano grueso, y que residen en repositorios software. (BERTOA 2002)

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. (BOEHM 2001)

Vale destacar también la clasificación de Alan Brown en 1999. En este caso el proceso de desarrollo está compuesto de cuatro etapas: la selección de componentes; la adaptación de componentes; el ensamblaje de los componentes al sistema; y la evolución del sistema.

Por último describiremos la clasificación de las etapas del desarrollo de los sistemas basados en componentes, de RUP (Rational Unified Process, Proceso Unificado de Desarrollo) (JACOBSON 2000), un proceso de desarrollo adoptado por Rational (http://www.rational.com) y por los Componentes UML (CHEESMAN 2001). En este caso el proceso se divide en seis etapas principalmente, observando la metodología de manera general, debido a que cuenta con las fases tradicionales de RUP modificadas y adaptadas para el DSBC, estas son: fase de requisitos, donde se identifican los requisitos de los

componentes, de la arquitectura de software y del sistema en general; fase de especificación, donde se realiza la especificación de los componentes y de la arquitectura de software; fase de aprovisionamiento de componentes, donde se buscan y seleccionan componentes; fase de ensamblaje de los componentes encontrados; fase de pruebas; y fase de implantación de la aplicación del sistema. Posteriormente describiremos detalladamente esta metodología.

Si comparamos la descripción antes realizada, con la metodología RUP orientada a objetos, se comprueba que las actividades iniciales y finales de definición de requerimientos, prueba y despliegue coinciden, mientras que difieren en que las fases centrales del proceso RUP (análisis, diseño e implementación) se han reemplazado por otras que representan la especificación, aprovisionamiento y ensamblado de componentes.

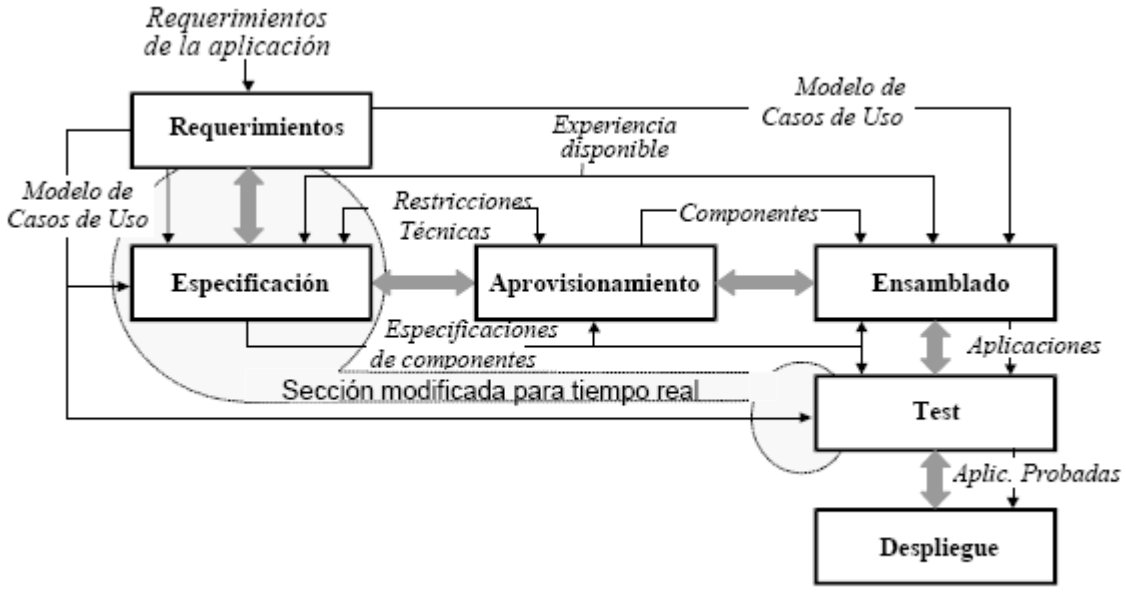

Fig. 3 "Rational Unified Process" modificado para componentes.

RUP admite el desarrollo basado en componentes de las siguientes maneras (JACOBSON 2000):

- El enfoque iterativo permite identificar componentes de forma progresiva y decidir cuáles desarrollar, cuáles reutilizar y cuáles comprar.
- La arquitectura de software propuesta y descrita en el Rational Unified Process (RUP), permite definir la estructura (los componentes y las formas de integración) que incluye los mecanismos fundamentales y los patrones de interacción. A su vez admite los aspectos de planificación de la

gestión de proyectos, ya que las dependencias de los componentes pueden ayudar a determinar qué componentes pueden desarrollarse de forma concurrente y cuáles de forma secuencial.

- Conceptos como paquetes, subsistemas y capas se utilizan durante el análisis y el diseño para organizar componentes y especificar interfaces.
- La realización de pruebas se organiza a partir de conceptos y poco a poco a partir de conjuntos mayores de componentes integrados.

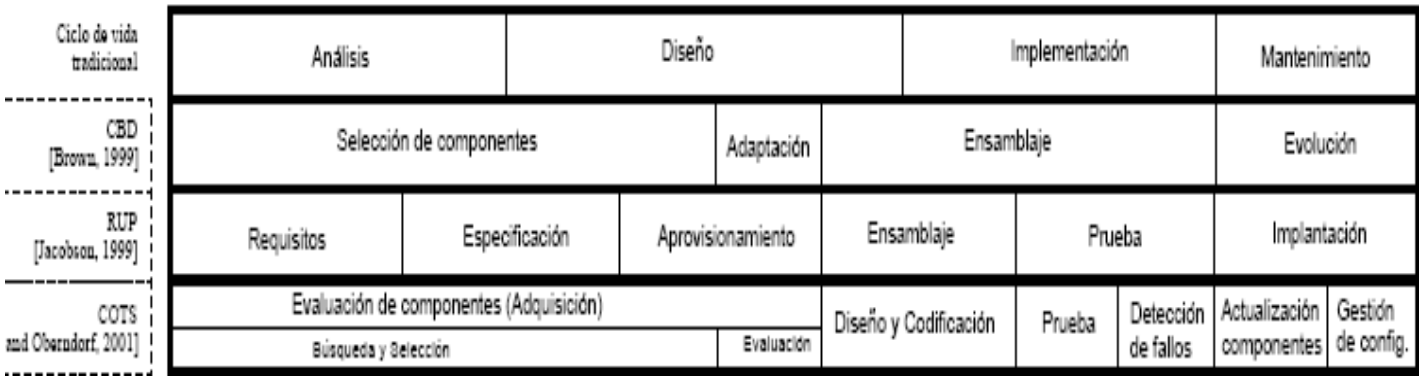

Fig. 4 Comparación entre Procesos de Desarrollo Basados en Componentes.

Como se puede observar, aunque no existe un consenso generalizado en cuanto a que Proceso de Desarrollo Basado en componentes se debe usar, sí se pueden observar que las etapas de algunos de ellos se encuentran solapadas. La fase de Selección de componentes de CBD abarca la de Requisitos, Especificación y la mitad de Aprovisionamiento de RUP, debido a que en la segunda parte de esta última se realizan las actividades relacionadas con la adaptación de los componentes. Resulta muy importante conocer cada una de estos procesos pues sería incorrecto afirmar que uno es superior a otro, lo ideal es que dadas las características de cada uno, decidir cual se adapta mejor a nuestras necesidades.

La elección de una metodología para el DSBC debe de estar directamente relacionada a las características propias del entorno donde será aplicada y a las necesidades que se espera que resuelva, estos deben de ser los principales factores que deben guiar a los especialistas del software a elegir la mejor opción.

1.9. ¿Como se desarrolla el software actualmente en el polo PetroSoft? Superioridad del DSBC.

La manera actual en la que se realiza el proceso de producción en el polo productivo PetroSoft, fue motivo de un análisis profundo en donde se determinó desechar la antigua forma. La misma se encontraba atada a un enfoque a proyectos y semejante a la estructura del cliente, esto provocaba la duplicación de los esfuerzos en cada proyecto productivo del polo; debido a que muchas veces se debían desarrollar aplicaciones que compartían muchas características similares. Sin embargo un mecanismo de información sobre los procesos de desarrollos entre los proyectos no estaba creado.

Luego de realizarse la reestructuración en el polo, se comienza a regirse por las funcionalidades generales detectadas en el cliente, sin embargo se hace necesario la implementación de mecanismos de documentación e información entre los proyectos productivos en su haber y la utilización de la tecnología de componentes por todos los beneficios que proporciona la misma.

El DSBC promueve la implementación de componentes que encapsulan funcionalidades de aplicación provenientes directamente de los requisitos funcionales del sistema. Mediante esta tecnología y la creación de un repositorio bien estructurado y documentado de componentes para el polo, se podría evitar la duplicación de esfuerzos de una buena vez, ya que los equipos de desarrollo buscarían primeramente la solución a los requisitos de sus aplicaciones como parte de un componente de software en dicho repositorio.

Mediante una encuesta realizada al jefe del polo PetroSoft, así como a estudiantes y profesores que se desempeñan en este momento laborando como integrantes de proyectos productivos o desarrollando soluciones para el polo como parte de sus tesis, se determinaron los siguientes datos preliminares.

- El 100% de los encuestados utiliza la metodología RUP tradicional o su variante ágil para el desarrollo.
- El 100% utiliza estilos arquitectónicos como: Modelo-Vista – Controlador (MVC) y N-Capas sin relacionarlas con la tecnología de componentes.
- El 0% de los encuestados utiliza Arquitectura de Componentes o Ingeniería de Software Basada en Componentes (ISBC).

La Arquitectura de Componentes es una tecnología que representa la evolución de todo un proceso informático tan antiguo como los primeros desarrollos. Los beneficios que brinda que fueron relacionados en este capítulo, propiciaron la transición completa de las empresas hacia esta nueva manera de hacer software. El prestigioso arquitecto de Microsoft, Carlos Billy Reynoso plantea la evolución de la arquitectura de Software de la siguiente manera (REYNOSO 2004):

Fig. 5 Evolución de la arquitectura de Software.

Donde además plantea que el nivel de abstracción de dichas arquitecturas va creciendo proporcionalmente a su evolución. Posteriormente han surgido implementaciones que han demostrado tanto en la academia como en la industria, superioridad frente a la tecnología de componentes en cuanto a aspectos como la reutilización, las dependencias y granularidad, como es el caso de SOA. Pero continúa siendo la tecnología de componentes la más usada por la industria debido a que sirve además como base para estas otras tecnologías más novedosas.

A continuación relacionamos las ventajas marcadas de los componentes ante otras tecnologías como la usada actualmente en nuestro polo como es el caso de los objetos (REYNOSO 2004).

	Programación Estructurada	Objetos	Componentes
Granularidad	Muy fina	Fina	Intermedia
Contrato	Definido	Privado/Publico	Publico
Reusabilidad	Baja	Baja	Intermedia
Acoplamiento	Fuerte	Fuerte	Débil
Dependencias	Tiempo de Compilación	Tiempo de Compilación	Tiempo de Compilación
Ámbito de Comunicación	Intra-Aplicación	Intra-Aplicación	Inter-Aplicaciones

Fig. 6 Comparación de la tecnología de componentes con otras.

Según Reynoso, el desarrollo basado en componentes presenta todas las ventajas contenidas en la Fig 6, vale destacar entre todas la Reusabilidad de la cual dependen otras características beneficiosas tales como la reducción de los tiempos y el esfuerzo de los equipos de desarrollo, así como el aumento en la facilidad de las pruebas entre otras. Como se ha podido observar resulta necesaria la aplicación de la tecnología de componentes en el polo PetroSoft, para garantizar todas las ventajas que trae aparejadas de las cuales no goza hoy en día el proceso productivo en función.

CONCLUSIONES PARCIALES

La tecnología de componentes es considerada actualmente como la solución más prometedora de la ingeniería software para acortar los tiempos de desarrollo y generar aplicaciones fiables dentro de la tendencia de incremento continuado de la complejidad. Tenemos la fortuna de presenciar el nacimiento de una nueva forma de hacer software, que traerá beneficios inmensos para todos.

El desarrollo de software basado en componentes desde siempre fue la idea revolucionaria que nos llevó a pensar que sí era posible el construir software de calidad en corto tiempo y con la misma calidad que la mayoría de las industrias de nuestro tiempo. Al mirar hacia atrás, vemos los increíbles avances que hemos logrado en la comprensión de la forma correcta de reutilizar el software y el conocimiento existente, y nos asombramos cada vez más al darnos cuenta de que este solo es el inicio.

El desarrollo de software basado de 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. Empresas como Microsoft entendieron el potencial de esta metodología hace años y hoy nos ofrecen nuevas iniciativas y herramientas que buscan llevar al proceso de construcción de software hacia el sitial privilegiado en el que debió colocarse desde un principio.

Las técnicas de análisis y diseño de componentes reutilizables se basan en los mismos principios y conceptos que forman parte de las buenas prácticas de ingeniería del software. Los componentes reutilizables deberían de diseñarse en el seno de un entorno que establezca unas estructuras de datos estándar, unos protocolos de interfaz y una arquitectura de programas para todos los dominios de aplicaciones.

CAPÍTULO II: Metodología para el DSBC en el polo productivo PetroSoft

En este capítulo abordaremos los aspectos necesarios para aplicar con éxito una metodología para el DSBC y además realizaremos una propuesta de Metodología de Ensamblaje de Componentes de Software en PetroSoft (MECSP).

2.1 Propuesta de Metodología de DSBC para PetroSoft

Antes de asumir y aplicar una nueva metodología en cualquier entorno de trabajo, lo primero que se debe hacer es conocer bien la misma y entender las ventajas y desventajas que ofrece sobre otras. La adopción de una metodología presupone un grupo de modificaciones y acondicionamientos para lograr adecuar las bases teóricas a nuevas ideas y maneras de hacer las actividades para lograr procesos más óptimos y por ende económicos.

Se propone establecer en el polo productivo PetroSoft una metodología basada en la solución descrita por RUP (Rational Unified Process) (I. JACOBSON 2000), para el DSBC, la cual será modificada y adaptada a las características propias del polo; Debido a dos factores principales, primeramente RUP es la metodología de desarrollo de software que se enseña en la universidad (constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos), y en segundo lugar, la variante de la misma para el DSBC ha demostrado ampliamente su eficiencia y calidad, es utilizada actualmente por Rational e IBM. La misma surge a partir del auge creciente que ha venido tomando el DSBC en el mundo así como la demanda de metodologías y procedimientos que la soporten. Las fases y tareas por flujos de trabajo de la **Metodología para el DSBC en el polo PetroSoft** las relacionamos seguidamente.

2.1.1 Actividades de la fase inicial

Se realizan los flujos de trabajos básicos para la fase inicial establecidos en la Metodología Tradicional RUP, agregándose las siguientes ampliaciones o modificaciones:

Gestión de proyectos

Actividad: Concebir un proyecto nuevo

- El foco de la Tarea: El desarrollo del caso de negocio se ajusta para que tenga en cuenta que la utilización de componentes cambia la estructura de costes del desarrollo. Específicamente, el coste de desarrollar componentes disminuye, pero se necesita más esfuerzo para identificar componentes y comprobar que los componentes seleccionados satisfacen los requisitos.

Actividad: Evaluar el riesgo y el ámbito del proyecto.

Al tomar el enfoque de un componente, cambia la naturaleza de determinados riesgos y se introducen riesgos nuevos. Específicamente:

- los componentes de origen externo aumentan el riesgo porque introducen elementos críticos que no están bajo el control directo del equipo de desarrollo.
- Los componentes de origen externos pueden introducir incompatibilidades como consecuencia de diferentes estándares propuestos por las entidades que desarrollan los mismos.
- los componentes de origen externo pueden reducir el tiempo de desarrollo y, de este modo, reducir el riesgo de recursos.
- los componentes de origen externo pueden distorsionar la arquitectura del sistema si imponen restricciones arquitectónicas propias.

Actividad: Planificar el proyecto

- En la Tarea: Planear fases e iteraciones, el plan de la fase de construcción tiene el potencial para mostrar la división del proyecto en dos seguimientos diferentes pero paralelos: uno que desarrolla componentes específicos del dominio y específicos de la aplicación (organizado en las capas superiores de la arquitectura) y otro que desarrolla los componentes que no son específicos del dominio ni de la aplicación y que se organizan en las capas inferiores. En algunos casos, los equipos de desarrollo gestionados independientemente desarrollarán componentes reutilizables. La decisión de introducir

seguimientos paralelos es un aspecto de personal y recursos conducido por un deseo de gestionar componentes reutilizables como activos independientes de las aplicaciones en que se despliegan.

Requisitos

Actividad: Concebir la Ingeniería de Software Basada en Componentes (ISBC).

La Ingeniería de Software Basada en Componentes, es un proceso que se centra en el diseño y construcción de sistemas basados en computadoras que utilizan componentes de software reutilizables (PRESSMAN 2005).

- Cuando el equipo de software establece los requisitos del sistema, y luego de establecer un diseño arquitectónico, en lugar de entrar en tareas de diseño detalladas, se deberán examinar los requisitos para determinar cual es el subsistema que esta dispuesto para la composición y no para la construcción. Para cada uno de los requisitos encontrados se deben plantearse las siguientes cuestiones:
    - ¿Se dispone de componentes reutilizables desarrollados internamente para implementar el requisito?
    - ¿Es posible disponer de componentes comerciales ya desarrollados para implementar el requisito?
    - ¿Son compatibles las interfaces de los componentes que están disponibles dentro de la arquitectura del sistema a construir?
- El equipo intentará modificar o eliminar aquellos requisitos del sistema que no se pueden implementar con componentes adquiridos o de desarrollo propio. Si los requisitos no se pueden cambiar o eliminar, se aplicarán métodos convencionales u orientados a objetos para desarrollar los componentes nuevos que se deben diseñar para cumplir los requisitos.

Actividad: Perfeccionar la definición del sistema

- Cuando se perfeccionan los requisitos del sistema, deben capturarse las restricciones que impone la infraestructura del componente seleccionado. Las infraestructuras de

componentes aumentan la productividad de desarrollo, en parte restringiendo los grados de libertad que se ofrecen al arquitecto y el diseñador de software. La Tarea: Detallar los requisitos de software debe centrarse en documentar estas restricciones.

Prueba

Actividad: Definir la misión de evaluación

- Debería crearse un plan de prueba que identifique las pruebas globales previstas para el proyecto, denominado “Plan de prueba maestro”.

Entorno

Actividad: Preparar el entorno para el proyecto

Cuando recopile y prepare directrices para el proyecto, tenga en cuenta la infraestructura de componentes específica y las restricciones que impone. Las directrices del proyecto deben incluir cómo diseñar y codificar utilizando la infraestructura. También deberían proporcionar instrucciones de prueba sobre cómo comprobar la conformidad con la infraestructura de componentes y las interfaces definidas entre componentes.

2.1.2 Actividades de la fase de elaboración

Se aplica el flujo de trabajo básico para la fase de elaboración, con las siguientes ampliaciones o variaciones:

Requisitos

Actividad: Perfeccionar la definición del sistema

- La Tarea: Detallar los requisitos de software también se centra en las restricciones y los requisitos técnicos y no funcionales que imponen los componentes construidos o

adquiridos. Los requisitos no funcionales específicos que se deben considerar son el tamaño, el rendimiento, la impresión en la memoria o el disco, los aspectos de la licencia, el tiempo de ejecución y otras restricciones similares que influirán en la construcción o la selección del componente.

Análisis y diseño

Actividad: Realizar el proceso de análisis del dominio.

El proceso de análisis del dominio de aplicación define un estudio de las funcionalidades que deberá poseer nuestro sistema así como las características comunes entre ellas para garantizar la posterior identificación de componentes que cumplan con las mismas y sean a su vez reutilizables.

Los pasos del proceso se definen de la siguiente manera (PRESSMAN 2005):

- Definir el dominio que hay que investigar.
- Categorizar los elementos extraídos del dominio.
- Recoger una muestra representativa de las aplicaciones del dominio.
- Analizar cada aplicación de la muestra.
- Desarrollar un modelo de análisis de los objetos.

El análisis del dominio se puede aplicar a cualquier paradigma de la ingeniería del software, siendo posible aplicarlo tanto para el desarrollo convencional como para el desarrollo orientado a objetos (PRESSMAN 2005).

Actividad: Definir una arquitectura candidata.

- La Tarea: Análisis de la arquitectura, utiliza la infraestructura de componentes y los requisitos técnicos y no funcionales para definir una arquitectura inicial, incluido un esquema de creación de capas inicial y un conjunto por omisión de componentes y servicios (representados como mecanismos de análisis y diseño). La Tarea: Análisis de

caso de uso, se centra en identificar componentes significativos arquitectónicamente de casos de uso arquitectónicamente significativos.

Actividad: Perfeccionar la arquitectura

- La Tarea: Estructurar el modelo de implementación establece un modelo de implementación compatible con la infraestructura de componentes y la estructura y las responsabilidades del equipo o los equipos de desarrollo.
- La Tarea: Identificar los mecanismos de diseño perfecciona los mecanismos de diseño iniciales para que tengan en cuenta componentes y servicios específicos de la infraestructura.
- La Tarea: Identificar elementos de diseño, identificará los principales componentes significativos arquitectónicamente del sistema. Las responsabilidades potencialmente reutilizables deberían agruparse para mejorar la capacidad de reutilización; las funciones específicas de la aplicación deben separarse de las funciones específicas del dominio e independientes del dominio y la aplicación. Para el diseño, los componentes se pueden representar como el Producto de trabajo: Subsistemas de diseño. Debería identificarse el Producto de trabajo: Interfaces para estos componentes/subsistemas.
- La Tarea: Incorporar elementos de diseño existentes garantiza que los componentes identificados son coherentes y compatibles con los componentes existentes identificados en iteraciones anteriores, en la infraestructura o en orígenes externos.
- La Tarea: Describir la arquitectura del tiempo de ejecución describe el proceso básico y la arquitectura de hebras de la infraestructura de componentes, mientras que la Tarea: Describir la distribución describe el entorno informático distribuido en el que se ejecutará la aplicación de componentes.

Actividad: Diseñar componentes

- La Tarea: Diseño del subsistema perfecciona aún más el diseño de los componentes, identificando las clases del componente que proporcionan el comportamiento real del

componente. Al principio de la fase de elaboración, es probable que haya una sola clase, un tipo de 'proxy de componente/subsistema' que actúa como un fragmento para simular el comportamiento del componente para crear prototipos arquitectónicos. Más adelante, el comportamiento de esta clase se distribuye en una colaboración de clases contenidas en el subsistema. Estas clases contenidas se perfeccionan en la Tarea: Diseño de clase, propuesta en el desarrollo tradicional de RUP.

Se debe realizar especial énfasis en el diseño de los componentes arquitectónicamente significativos; primeramente porque serán los que garantizarán el correcto funcionamiento del sistema, y además porque serán los que primeramente se implementaran e integrarán de manera incremental probando en cada momento las funcionalidades ganadas.

- La Tarea: Diseñar Componente de Integración, genera el diseño de un componente que se utilizará para realizar la integración organizada de todos los demás. Debe de tenerse en cuanta el diseño de interfaces genéricas bien definidas y documentadas para garantizar el futuro acoplamiento con éxito de los componentes que se deseen integrar. Esta tarea debe realizarse cuando el tamaño y/o la complejidad del sistema lo requieran.

Actividad: Diseñar la base de datos

- El aspecto más importante de la elaboración es garantizar que la estrategia de permanencia es escalable y que el mecanismo de permanencia y el diseño de bases de datos soportarán los requisitos de rendimiento del sistema. Las clases permanentes se identifican y se correlacionan con el mecanismo de permanencia. Los casos de uso de datos intensivos se analizan para garantizar que los mecanismos serán escalables. Junto con las actividades de prueba, se valora y se valida el mecanismo de permanencia y el diseño de bases de datos.

Implementación

Actividad: Implementar componentes

- La Tarea: Implementar los elementos de diseño debe cumplir las restricciones impuestas por la infraestructura de componentes, como se describe en las directrices de programación, que se proporcionan como parte del Producto resultante del trabajo: Directrices específicas del proyecto, definido en el desarrollo tradicional de RUP. En la fase de elaboración, la mayoría de los componentes contienen una gran cantidad de código de 'fragmento para simulación', ya que la implementación se centra en validar la arquitectura, y no producen código de calidad de producción.

Prueba

Actividades: Definir la misión de evaluación, Verificar el enfoque de prueba, Probar y evaluar, Conseguir una misión aceptable, Aumentar los activos de prueba

Las actividades de prueba de la fase elaboración se centran en validar la arquitectura. Para un sistema basado en componentes, se centran en:

- ejercitar las interfaces entre componentes para garantizar que los límites del componente son adecuados
- una atención mayor en la prueba de rendimiento, especialmente en las pruebas de escala de rendimiento, para garantizar que se pueden sostener los volúmenes de transacción anticipados

También es necesario valorar las suposiciones inherentes de la infraestructura de componentes. Estas suposiciones suelen ser la escalabilidad y el rendimiento de los mecanismos de gestión de la transacción y la permanencia, donde el diseñador de mecanismos efectúa suposiciones ocultas que a menudo socavan eficazmente la arquitectura de la aplicación si no conoce la suposición.

Gestión de proyectos

Actividad: Planear la siguiente iteración

Si se utilizan los subsistemas de implementación como 'unidades lógicas de responsabilidad', el trabajo de construcción se puede particionar en dos o más "seguimientos" paralelos: uno que se

centra en las funciones específicas de la aplicación y uno o varios que se centran en componentes genéricos reutilizables. Por supuesto, esto depende de si se tienen los recursos humanos suficientes para esfuerzos de desarrollo paralelo. La capacidad de dividir los equipos de desarrollo y trabajar en paralelo depende totalmente de la estabilidad de la arquitectura y, más específicamente, de la calidad y la estabilidad de las interfaces entre los componentes. Esta división será posible si se realiza un esfuerzo enorme en la fase de elaboración.

2.1.3 Actividades de la fase de construcción

Se aplica el flujo de trabajo básico para la fase de construcción, con las siguientes ampliaciones o variaciones:

Gestión de proyectos

Actividad: Planear la siguiente iteración

Anteriormente se describió la planificación de la primera iteración de construcción, igual que sucede hacia el final de la elaboración. Igual que la planificación de la iteración, la capacidad de dividir los equipos de desarrollo y trabajar en paralelo sigue dependiendo de la estabilidad de la arquitectura y la calidad y la estabilidad de las interfaces entre los componentes.

Análisis y diseño

La Actividad: Perfeccionar la arquitectura y la Actividad: Diseñar componentes

El aspecto principal de la construcción es el análisis del recordatorio de los casos de uso y la identificación de las colaboraciones de componentes y los componentes adecuados que realizan los casos de uso. La arquitectura existente se amplía y se completa y los “comportamientos internos” del componente se diseñan y se implementan completamente.

La Actividad: Introducir un enfoque basado en Patrones de Software:

Esta actividad, asegurará la adaptación de posibles patrones de software en el diseño e implementación de las funcionalidades y las interfaces de los componentes a utilizar. Cuando se utilizan patrones de diseño en la implementación de componentes de software no solo se agiliza el desarrollo del mismo sino que se facilita y se propicia la reutilización del mismo. Se propone principalmente el uso de patrones estructurales y creacionales, más específicamente los que a continuación se describen:

Abstract Factory (Fábrica Abstracta): Este patrón describe el trabajo con objetos de diferentes familias sin que los mismos se mezclen entre sí, de igual modo realiza la utilización transparente de la familia que se esté usando.

Singleton (Instancia Única): La utilización de este patrón garantiza la creación de una instancia única de una clase y su acceso global a esta instancia en donde sea requerida.

Adapter (Adaptador): Este patrón se encarga de adaptar una interfaz para que pueda ser accedida y utilizada por una clase que de otra manera no podría usarla.

Facade (Fachada): La implementación de la solución propuesta por este patrón, provee de una interfaz unificada simple para acceder a otra interfaz – grupo de interfaces de un subsistema.

Actividad: Diseñar la base de datos

El aspecto principal de la construcción es terminar el diseño de la base de datos y asegurarse de que todas las clases permanentes son soportadas tanto por la base de datos como por el mecanismo de permanencia. Este trabajo se realiza en paralelo y de forma repetitiva con el trabajo que se realiza en la Actividad: Perfeccionar la arquitectura y la Actividad: Diseñar componentes. La organización ideal consiste en contar con equipos de componentes integrados, y con coordinación entre equipos de los asuntos de permanencia para garantizar un buen diseño de base de datos.

Implementación

Actividad: Implementar los elementos de diseño

- El trabajo aquí es similar al de elaboración, aunque los demás detalles se completan en incrementos, a medida que avanza la fase.
- La Actividad: Integrar los subsistemas y la Actividad: Integrar el sistema.

  La integración de los componentes debe de realizarse de manera ordenada, siguiendo un camino consecuente de acuerdo a la prioridad de los requisitos funcionales descritos en cada caso. Esto permitirá que las funcionalidades operacionales del sistema sean mostradas tempranamente y aumentará la confianza del equipo de desarrollo en que el proyecto está progresando de manera satisfactoria.

  Se deben implementar e integrar primeramente los componentes arquitectónicamente significativos e incluir una interfaz de prueba para garantizar la correcta funcionalidad de los mismos así como las interacciones con otros componentes.
- El sistema se construye progresivamente a medida que la fase continúa.

Prueba

Actividades: Definir la misión de evaluación, Verificar el enfoque de prueba, Validar la estabilidad de la compilación, Probar y evaluar, Conseguir una misión aceptable, Aumentar los activos de prueba.

Las pruebas de rendimiento siguen siendo importantes, aunque cada vez se presta más atención a las pruebas funcionales. Deben tenerse en cuenta la completitud de la funcionalidad, las pruebas de revisión de la funcionalidad existente y la satisfacción de las expectativas de rendimiento.

2.1.4 Actividades de la fase de transición

- El “release” de un sistema de software suele ser incremental y continuo, y se centra menos en la distribución tradicional de medios de soporte. La planificación del release debe ajustarse en consecuencia.
- El soporte de producción es cada vez más importante en la fase.

- Normalmente cuando un sistema, en su entorno, debe interactuar con aplicaciones diferentes, se realizan actividades de conversión de datos para lograr la interoperabilidad entre las mismas.

2.2 Aspectos previos a la adopción de la Metodología de DSBC para PetroSoft.

Para de empezar a utilizar la Metodología de DSBC en el polo PetroSoft, es preciso definir un grupo de elementos y condiciones necesarios, ya sea entre los equipos de proyectos que deben utilizar la misma o en la condiciones de desarrollo propias del polo, para garantizar la adopción y utilización exitosa de la nueva metodología de la cual dependerá su futuro uso en el desarrollo y termino de los proyectos productivos venideros. Tales elementos servirán de guía a los líderes de proyectos así como a los demás miembros de los grupos de desarrollo para la transición una vez que se haya decidido la utilización de la propuesta metodológica de DSBC para PetroSoft. A continuación relacionamos dividido en dos epígrafes los aspectos necesarios a tener en cuenta para garantizar lo antes mencionado.

2.2.1 Composición de componentes.

La composición de componentes se encarga del ensamblado de componentes cualificados, adaptados y diseñados para ajustarse a la arquitectura establecida para una aplicación. Para poder llevar a cabo esta tarea debe de establecerse una infraestructura donde los componentes estén unidos a un sistema operacional, también descrito como un componente integrador en la metodología propuesta anteriormente. La infraestructura, que será el conjunto de componentes candidatos a ensamblar para construir el sistema, proporciona un modelo para la coordinación de componentes y servicios específicos que hacen posible la coordinación de unos componentes con otros y llevar a cabo tareas comunes.

Para crear una infraestructura eficaz, existe un conjunto de cuatro elementos arquitectónicos que deberían estar presentes para lograr la composición de componentes. Según Roger Pressman (PRESSMAN 2005), tales elementos son:

- **Modelo de intercambio de datos:** Se trata de mecanismos que capacitan a los usuarios y aplicaciones para interactuar y transferir datos, y que deberían estar definidos para todos los componentes reutilizables. Los mecanismos de intercambio de datos no solamente permiten la transferencia de estos entre el hombre y el programa, y entre componentes, sino que también hacen posible la transferencia entre los recursos del sistema.
- **Automatización:** Para facilitar la interacción entre componentes reutilizables, se deberían de implementar herramientas, macros y guiones.
- **Almacenamiento estructurado:** Los datos heterogéneos (por ejemplo, los datos gráficos, de voz, texto, video y datos numéricos) dentro de un documento compuesto, deben de estar organizados para poder acceder a ellos como si de una sola estructura de datos se tratase, en lugar de comportarse como si fuera toda una colección de archivos por separados. Los datos estructurados mantienen un índice descriptivo de estructuras de anidamiento, que las aplicaciones pueden recorrer libremente para localizar, crear o editar el contenido de datos individuales según lo indique el usuario final.
- **Modelo de objetos subyacente:** El modelo de objetos asegura, que los componentes desarrollados en diferentes lenguajes de programación que residan en distintas plataformas puedan ser interoperables. Los objetos deben ser capaces de comunicarse en una red. Por este motivo, el modelo de objetos, define un estándar para la interoperabilidad de componentes.

2.2.2 Elementos necesarios para la adopción de la Metodología de DSBC para PetroSoft.

Posteriormente a tomar la decisión de adoptar la propuesta metodológica para el DSBC en PetroSoft, es imprescindible el aprendizaje y difusión de la misma. Los lideres del polo y de los equipos de desarrollo, luego de darla a conocer, deberán orientar la enseñanza de la misma a través de la organización de cursos de capacitación hasta asegurar el dominio generalizado de la misma. La aplicación de esta metodología en el polo, dependerá también de otros cambios en el plan educacional de las Asignaturas: Ingeniería de Software I e Ingeniería de Software II, en las

cuales de deberá comenzar a introducir las nuevas actividades por flujos de trabajo que se han definido previamente en esta propuesta.

Una de las prácticas centrales detrás de RUP, es el desarrollo iterativo e incremental. Cuando empiece con la Metodología para el DSBC en PetroSoft, recuerde la filosofía de esta práctica: no intente todo el proceso de una vez. Adopte un enfoque a la implementación, el aprendizaje y la utilización de la misma que sea a su vez iterativo e incremental. Empiece por valorar su proceso existente y seleccione una o dos áreas clave que desee mejorar. Utilice RUP para mejorar primero estas áreas y después, en iteraciones posteriores o ciclos de desarrollo, realice mejoras incrementales en otras áreas.

El flujo de trabajo "entorno" de Rational Unified Process, provee un conjunto de actividades, para garantizar los procesos y las herramientas que dan soporte al equipo de desarrollo. Luego del estudio de este flujo de trabajo en la Metodología para el DSBC para PetroSoft, Decida cómo llevar a cabo el flujo de trabajo observando las actividades del mismo. Estudie el diagrama con sus condiciones y directrices. Decida las actividades que se van a realizar y su orden.

2.2.3 Principios fundamentales del desarrollo dirigido por la empresa.

El desarrollo de software es un deporte de equipo. Lo ideal sería que la actividad implicara a equipos bien coordinados que trabajasen en diversas disciplinas durante todo el ciclo vital del software. Pero no es una ciencia y tampoco es exactamente ingeniería, al menos desde el punto de vista de los principios cuantificables basados en datos irrefutables. Los esfuerzos de desarrollo de software que asumen que pueden planificarse y crearse partes sueltas y después ensamblarlas, como se hace en la construcción de puentes o de naves, fracasan constantemente a causa de las fechas de entrega, los presupuestos y el nivel de satisfacción del usuario.

A falta de datos irrefutables, debe confiarse en las técnicas de desarrollo de software que denominamos recomendaciones, cuyo valor ha sido demostrado repetidamente en otros compromisos con clientes. En lugar de prescribir una secuencia de actividades del tipo planificar-compilar-ensamblar para un proyecto de software, describen un proceso iterativo, incremental que dirige a los equipos de desarrollo hacia la consecución de resultados.

Las seis recomendaciones de Rational Unified Process que son completamente fiables han sido la base de la evolución de las herramientas y procesos de Rational durante más de una década. Actualmente, a medida que el desarrollo de software se está convirtiendo en una habilidad empresarial clave, nuestras recomendaciones están madurando en el contexto ampliado del desarrollo dirigido por la empresa. Los principios siguientes vuelven a articular nuestras recomendaciones para un ciclo vital más amplio, el de los sistemas en continua evolución, en los que el principal elemento de la evolución es el software. Esos principios son:

Adaptar el proceso

Este principio expresa que es fundamental ajustar el tamaño del proceso de desarrollo a las necesidades del proyecto. Más no significa mejor, menos no significa mejor: por el contrario, la cantidad de ceremonia, precisión y control de un proyecto debe ajustarse a diversos factores que incluyen el tamaño y distribución de los equipos, la cantidad de restricciones impuestas desde fuera y la fase en la que se encuentra el proyecto.

Equilibrar las prioridades de los interesados que están enfrentadas

Este principio articula la importancia del equilibrio de las necesidades empresariales y de los interesados, que suelen entrar en conflicto, así como del equilibrio del desarrollo personalizado frente a la reutilización de activos en la satisfacción de esas necesidades.

Colaborar con los otros equipos

Este principio enfatiza la importancia de fomentar una óptima comunicación en todo el proyecto. Esto se logra mediante una adecuada organización del equipo y la configuración de entornos de colaboración eficaces.

Demostrar el valor de forma iterativa

Este principio explica por qué el desarrollo de software se beneficia enormemente del proceso iterativo. Un proceso iterativo posibilita una incorporación sencilla de los cambios, la obtención de información de retorno y su factorización en el proyecto, la reducción temprana del riesgo y el ajuste dinámico del proceso.

Elevar el nivel de abstracción

La complejidad es una cuestión central en el desarrollo de software. Elevar el nivel de abstracción ayuda a reducir la complejidad y la cantidad de documentación que necesita el proyecto. Esto puede lograrse mediante la reutilización, el uso de herramientas de modelado de alto nivel, y la estabilización temprana de la arquitectura.

Centrarse continuamente en la calidad

Este principio enfatiza que, para obtener calidad, debe abordarse en la totalidad del ciclo vital del proyecto. Un proceso iterativo está particularmente adaptado para conseguir calidad ya que ofrece muchas oportunidades para la medición y las correcciones.

2.3 Desarrollo de Software Basado en Líneas de Productos.

Cuando se ha logrado un desarrollar software basado en componentes durante mucho tiempo y se ha empezado a orientar muchas de las soluciones que hemos realizado (que nombraremos activos) a elementos comunes de lógica de negocio o procesos similares que existen en nuestro entorno, entonces sin saberlo estaremos desarrollando Software Basado en Líneas de Productos.

Esta idea constituye el punto máximo en la evolución de la reutilización del software, se propone, como elemento ideal para el desarrollo de software en el polo PetroSoft debido al alto grado de elementos comunes que presentan las soluciones creadas para un entorno petrolero.

La idea básica:

- Ensamblaje de partes de software previamente elaboradas
- Inspirada en los procesos de producción de sistemas físicos:
    - Producción de aviones, vehículos, computadores, aparatos electrónicos, etc.
- Fundamentada en la Reutilización de Software.
- Asume la existencia de una industria de partes.

Conceptos de LPS:

Las líneas de productos de software es un cúmulo de sistemas de software desarrollados bajo un conjunto de funcionalidades comunes para todos que tienen que ver con elementos reutilizables de la lógica de los negocios que tienen que ver con la línea que se este tratando.

"...se refieren a técnicas de ingeniería para crear un portafolio de sistemas de software similares, a partir de un conjunto compartido de activos de software, usando un medio común de producción" (Krueger, 2006).

"... es un conjunto de sistemas de software que comparten un conjunto común y gestionado de aspectos que satisfacen las necesidades específicas de un segmento de mercado o misión y que son desarrollados a partir de un conjunto común de activos fundamentales [de software] de una manera preescrita" (Clements and Northrop, 2002).

"...consiste de una familia de sistemas de software que tienen una funcionalidad común y alguna funcionalidad variable" (Gomma, 2004)

- La funcionalidad común descansa en el uso recurrente de un conjunto común de activos reutilizables (requisitos, diseños, componentes, servicios web, etc.)
- Los activos son reutilizados por todos los miembros de la familia. (MONTILVA 2006)

Las líneas de productos de Software constituyen la prácticas de Ingeniería de Software que mas se ajusta a las necesidades productivas de polo PetroSoft, debido a que garantizara la reutilización de los sistemas construidos en forma de componentes y además la orientación de estos a líneas productivas en este caso relacionadas con todos los procesos de la industria del petróleo que podrán ser encapsulado en cada componente y rehusados continuamente.

Evolución de la Reutilización de Software (Jonás A. Montilva C., Ph.D. IEEE Member)

Fig. 7 Evolución de la reutilización del Software.

2.3.1 Beneficios del DSBLP

La entrega de productos de software de una manera

- más rápida,
- económica y
- con una mejor calidad

Las LPS producen mejoras en:

- Tiempo de entrega del producto (time to market)
- Costos de ingeniería
- Tamaño del portafolio de productos

- Reducción de las tasas de defectos
- Calidad de los productos

CONCLUSIONES PARCIALES

En este capítulo hemos visto de manera general las principales propuestas metodológicas a nivel mundial que han sido propuestas por connotados autores internacionales. Si bien es cierto que no se ha llegado a un consenso sobre cual de todas es superior, si se puede apreciar que las fases y etapas de cada una de ellas se solapan unas con otras. Es preciso conocerlas todas y realizar un estudio a fondo de las características propias del negocio y de los requerimientos propios del sistema que estamos construyendo para escoger la mejor opción.

Existen aspectos claves que no se pueden pasar por alto a la hora de utilizar una metodología para el DSBC y que de cierta forma forman parte de estas, de manera general la búsqueda y selección de componentes, ya sea en repositorios propios, u obtenidos a través de terceros es un primer paso vital para garantizar la calidad de nuestra futura aplicación. La evaluación y adaptación de los mismos representan las fases en donde se garantiza que el componente previamente seleccionado cumple con las características necesarias para acoplarse a nuestro sistema. Finalmente las estepas de ensamblaje y evolución concretarán la construcción rápida y eficiente de nuestro sistema así como también garantizaran el mantenimiento y desarrollo futuro del mismo.

La propuesta para en DSBC en el polo PetroSoft, está basada en la variación de la metodología Rational Unified Process para el desarrollo basado en componentes. El objetivo principal de la misma es introducir mejoras potenciando en algunos flujos de trabajo aspectos como Ingeniería de Dominio e Ingeniaría de Software, los cuales prepararían mas marcadamente a nuestros desarrolladores en elementos cruciales como la identificación de componentes a partir de los requerimientos de los sistemas.

El Desarrollo de Software Basado en Líneas de Productos (DSBLP) constituye la cima en la evolución del DSBC y una propuesta muy interesante a aplicar al polo puesto que se trata de la agrupación y organización de todo el desarrollo en componentes orientado a uno o mas productos en específico, en nuestro caso el petróleo. El DSBLP propicia la creación y desarrollo de activos de software que mantienen

elementos comunes basados en la línea de productos y que son altamente demandados por entidades nacionales y extranjeras que están relacionados con dicho producto.

CÁPITULO III: Evaluación y Aplicación de la Propuesta.

3.1 Introducción

Después de haber estudiado las metodologías que existen internacionalmente para el DSBC y de definirse la propuesta metodológica para el polo PetroSoft, se validará el trabajo tomando como base el uso de los criterios de un panel de expertos y el empleo de técnicas propuestas en el Método Delphi.

Se hace necesario validar el trabajo para una posterior utilización por los equipos de proyectos productivos que estén a cargo del desarrollo de los mismos en el polo. También se ha realizado una aplicación inicial en proyectos no productivo a modo de pruebas piloto, para garantizar la funcionalidad y usabilidad de la metodología.

3.2 Evaluación de la propuesta. Método de Delphi

El método Delphi, creado por la compañía Rand Corporation y más específicamente por Olaf Helmer, Dalkey y Gordon; es considerado como una de las técnicas subjetivas de pronosticación más confiable. Su nombre proviene de la historia griega del antiguo oráculo de Delfos como también se conoce a dicho método. Fue creado con el objetivo de elaborar pronósticos a largo plazo referentes a posibles acontecimientos que podrían tener lugar en varias ramas de la ciencia, la técnica y la política. De esta manera, se define como la utilización sistémica del juicio intuitivo de un grupo de expertos, para obtener un consenso de opiniones informadas.

Este método, consiste en el dialogo anónimo de un grupo de expertos que deben ser consultados independientemente para que sus planteamientos no influyan en los ajenos. Las consultas se realizarán mediante cuestionarios, con vista a tener un consenso general sobre el tema que se está tratando.

3.2.1 Proceso de selección de los expertos

Este grupo de expertos se conformó con especialistas que poseen un vasto conocimiento de los temas relacionados con las metodologías de desarrollo como son líderes de algunos proyectos y profesores de Ingeniería de Software que ya tienen varios años de experiencia en estas competencias. La correcta elección de los expertos propicia obtener resultados con calidad y una opinión grupal con un alto grado de consenso.

Seguidamente se hará la descripción de los pasos utilizados para la selección de los Expertos y los resultados obtenidos de los mismos.

Un experto debe ser una persona experimentada en el tema que se estudia la cual posee una gran experiencia o habilidad en la misma, capaz de ofrecer valoraciones conclusivas de un problema en cuestión y hacer recomendaciones al respecto.

Esta selección se realizó atendiendo a factores tales como la experiencia en proyectos productivos y asignaturas docentes relacionadas con la propuesta, que deben poseer los candidatos a expertos, los mismos laboran en la Facultad 9 pues es donde se piensa poner en práctica inicialmente la estrategia propuesta. Los expertos propuestos poseen además, amplios conocimientos en temas relacionados con el proceso a evaluar, pues de una forma u otra han estado relacionados con el mismo, estos temas se relacionan a continuación:

1. Metodologías de desarrollo.
2. Ingeniería de Software
3. Gestión de Proyectos.
4. Planificación de Proyectos.

El conocimiento sobre estos temas ha permitido que las opiniones brindadas sean confiables y válidas para el objetivo propuesto.

3.2.2 Elaboración del Objetivo a Valorar

Objetivo a evaluar por los especialistas: Valorar la estrategia para el desarrollo de software basado en componentes (DSBC) para el polo Productivo PetroSoft, concerniente a la metodología propuesta y la integración con el Desarrollo de Software Basado en Líneas de Productos (DSBLP).

3.2.3 Cantidad de Expertos seleccionados

No existe una norma generalizada que determine un valor óptimo respecto al número de expertos. Los investigadores de Rand Corporation, por ejemplo Landeta (ASTIGARRAGA) indican que es necesario como mínimo de siete expertos y un máximo de 30.

En este trabajo se decidió contar con un número de 7 expertos, teniendo en cuenta nivel de complejidad y profundidad del contenido.

3.2.4 Guía para la validación de la propuesta

Para llevar a cabo el desarrollo de la validación se efectuaron un conjunto de pasos, los cuales se detallan a continuación:

1) Elaboración de los criterios de evaluación que fueron utilizado en el desarrollo de la validación y se agruparon por categorías.

CRITERIOS PARA METODO DE EXPERTOS

Rango de evaluación: 1-10

Donde 1 es la evaluación mínima y 10 es la evaluación máxima.

Criterios de mérito científico.

Calidad de la investigación.

Novedad científica.

Aporte científico.

Criterios de implantación

Satisfacción de las necesidades de la producción.

Garantía de principios básicos del DSBC.

Criterios de generalización

Facilidades de comprensión.

Facilidades de uso.

Adaptabilidad a diferentes entornos de producción de software.

Criterios de impacto

Contribución al Proceso de Desarrollo de Software.

Contribución al proceso de desarrollo en proyectos productivos.

2) Determinar el peso relativo de cada grupo de criterios de acuerdo al por ciento que representa cada grupo del total y los intereses a evaluar.

Grupo No.1…… 30

Grupo No.2…… 20

Grupo No.3…… 30

Grupo No.4…… 20

3) Solicitud a los expertos seleccionados la evaluación de cada uno de los criterios en una escala del 1 al 10, teniendo en cuenta que la suma del valor dado por parte de los expertos a cada criterio de un grupo no exceda del peso relativo asignado a este.

4) Construcción de una tabla que guarda el resultado del trabajo de los expertos.

G	C/E	E1	E2	E3	E4	E5	E6	E7	Ep
30	C1	8	10	10	10	10	10	9	9,57143
	C2	10	9	9	10	10	10	10	9,71429
	C3	10	7	8	10	9	5	4	7,57143
20	C4	9	8	7	10	8	6	4	7,42857
	C5	10	9	10	10	10	8	10	9,57143
30	C6	8	9	9	10	10	9	10	9,28571
	C7	9	8	10	10	9	9	6	8,71429
	C8	9	10	8	10	8	6	5	8
20	C9	10	9	10	10	10	10	8	9,57143
	C10	10	10	10	10	9	10	10	9,85714
Total		93	89	91	100	93	83	76	89,2857

Fig. 8 Resultado del trabajo de los expertos.

Conociendo los criterios que obtuvieron mayor puntuación luego de concluida la evaluación, se podrán predecir las fortalezas de la propuesta realizada. En la figura 9 se representan los criterios evaluados así como el valor obtenido por cada uno de ellos expresado en porciento:

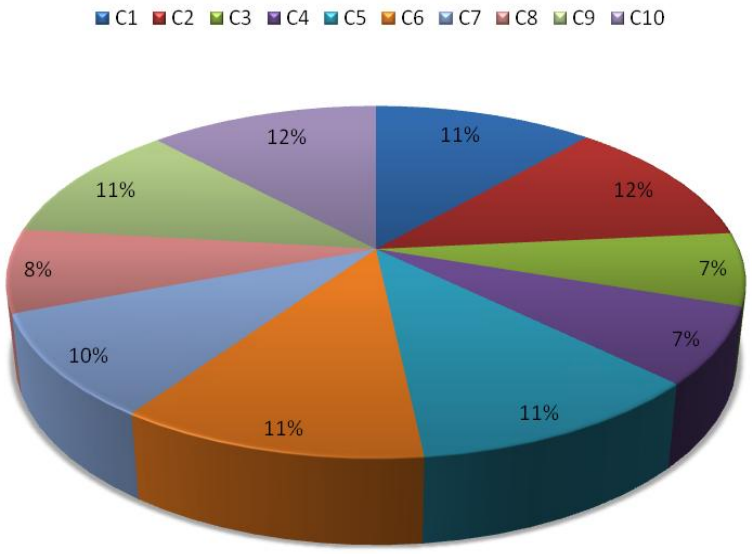

Fig. 9 Representación del valor de los criterios en porciento.

5) Verificación de la consistencia en el trabajo de los expertos, para lo que se utiliza el coeficiente de concordancia de **Kendall y el estadígrafo Chi cuadrado (X^2).**

Para esto se sigue el procedimiento siguiente:

Sea C el número de criterios que van a evaluarse y (E) el número de expertos que realizan la evaluación.

Para cada criterio se determina:

- ΣE: Sumatoria del peso dado por cada experto.
- Ep: Puntuación promedio del peso dado por cada experto.
- MΣE: media de los ΣE.
- ΔC: Diferencia entre ΣE y MΣE.

Se determina la desviación de la media, que posteriormente se eleva al cuadrado para obtener la dispersión (S) por la expresión:

$S = \Sigma (\Sigma E - \Sigma \Sigma E / C)^2$

Conociendo la dispersión se puede calcular el coeficiente de concordancia de Kendall (W):

$W = 12 * S / E^2 (C^3 – C)$

El coeficiente de concordancia de Kendall permite calcular el Chi cuadrado real:

$X^2 = E (C-1) W$

Los valores obtenidos se muestran en la siguiente tabla.

C/E	E1	E2	E3	E4	E5	E6	E7	∑E	Ep	ΔC
C1	8	10	10	10	10	10	9	67	9,57143	5,16039
C2	10	9	9	10	10	10	10	68	9,71429	6,16039
C3	10	7	8	10	9	5	4	53	7,57143	-8,8396
C4	9	8	7	10	8	6	4	52	7,42857	-9,8396
C5	10	9	10	10	10	8	10	67	9,57143	5,16039
C6	8	9	9	10	10	9	10	65	9,28571	3,16039
C7	9	8	10	10	9	9	6	61	8,71429	-0,8396
C8	9	10	8	10	8	6	5	56	8	-5,8396
C9	10	9	10	10	10	10	8	67	9,57143	5,16039
C10	10	10	10	10	9	10	10	69	9,85714	7,16039
M∑E	61,8396									
S	2601									
W	0,64341									
X^2	40,5351									
X^2 (α, c-1)	9,414									

Fig. 10 Tabla para el cálculo de concordancia de Kendall

Determinado el coeficiente de Kendall, es necesario realizar la prueba de hipótesis de que los expertos no tienen comunidad de preferencia. Con este criterio se intenta verificar la hipótesis fundamental:

H_0: No hay concordancia entre los expertos.

Contra la hipótesis alternativa

H_1: Hay una concordancia no casual entre los expertos.

El cálculo del coeficiente de concordancia de Kendall resultó ser de 0.64341, si w > 0.5 se presupone con este valor la convergencia de criterios entre los expertos por tanto estos criterios alcanzados convergen. Para validar este resultado estadísticamente se utilizó el estadígrafo X^2 lo que se hace necesario por ser la cantidad de criterios mayor que siete.

El Chi cuadrado calculado se compara con el obtenido de las tablas estadísticas (Ver Anexo 3).

Se considera que es válida la Hipótesis H_0 si se cumple:

$X^2_{real} < X^2_{(\alpha, c-1)}$

40.5351 > 9.414 Por tanto se rechaza H_0 y se acepta H_1.

Lo que demuestra que existe concordancia en el trabajo de expertos.

6) Si no existe concordancia se hace necesario repetir el trabajo de expertos.

7) Después de comprobar la consistencia del trabajo de expertos se puede definir el peso relativo de cada criterio (P).

$$P_i = E_i / \sum_{1}^{10} E$$

8) Conociendo el peso de cada criterio y la calificación dada por los evaluadores en una escala de 1-5 se puede construir la Tabla 5 calificación de cada criterio, para obtener el valor de de P × c., donde (c), es el criterio promedio concebido por los expertos.

Criterios	Calificación(C)					P	P x c
	1	2	3	4	5		
C1					x	0,1072	0,536
C2					x	0,1088	0,544
C3				x		0,0848	0,3392
C4				x		0,0832	0,3328
C5					x	0,1072	0,536
C6					x	0,104	0,52
C7				x		0,0976	0,3904
C8				x		0,0896	0,3584
C9					x	0,1072	0,536
C10					x	0,1104	0,552

Fig. 11 Tabla de calificación de cada criterio.

9) Se calcula el Índice de aceptación del proyecto (IA).

$IA = \Sigma (P \times c) / 5$

$IA = 0.928$

10) Por último se determina la probabilidad de éxito de la propuesta

3.2.5 Rangos predefinidos de Índice de Aceptación

$IA > 0,7$ Existe alta probabilidad de éxito.

$0,7 > IA > 0,5$ Existe probabilidad media de éxito.

0,5 > IA > 0,3 Probabilidad de éxito baja.

0,3 > IA Fracaso seguro.

Por lo que se puede concluir que la propuesta tiene una alta probabilidad de **éxito**.

3.2 Aplicación inicial de la propuesta.

La metodología para el Desarrollo de Software Basado en Componentes, presentada y descrita en el capítulo anterior, se aplicó en el desarrollo de pequeños proyectos de prueba realizados por los miembros del equipo de ensamblaje del Polo de PetroSoft de la facultad 9 con el propósito de examinar como lograr una adecuada implantación de la misma en los proyectos productivos en desarrollo.

Se utilizaron inicialmente dos pequeños proyectos y una fuerza de trabajo integrada por seis estudiantes para la construcción de los mismos. Los miembros mencionados pertenecen a diferentes años docentes, en este caso a tercero y cuarto. Se utilizaron puestos de trabajo contándose con tres máquinas asignados por la dirección del polo y se realizaron revisiones y reuniones de chequeo con una periodicidad semanal.

Inicialmente se realizó un trabajo de capacitación de todo el personal como parte de las actividades del Flujo de trabajo de Ambiente (*Environment*), para lograr así nivelar el conocimiento y comprensión de los temas relacionados con el Desarrollo de Software Basado en Componentes y también la tecnología aplicable a los mismos. Para lograr este objetivo se impartieron conferencias y exposiciones dando los primeros pasos hacia la comprensión inicial de los aspectos fundamentales del DSBC.

Además se orientaron tareas relacionadas con la búsqueda e investigación acerca de la tecnología de componentes, así como la exposición y defensa de los resultados obtenidos cumpliendo las mismas. En el taller que sirvió de punto de encuentro para todos los temas investigados se defendió también la propuesta de metodología a aplicar en el Polo de PetroSoft para el conocimiento y aprobación de todos.

En el desarrollo de los dos proyectos pilotos, se siguieron paso a paso las fases y flujos de trabajo definidos en la metodología a aplicar, poniéndose énfasis en los aspectos que diferencian la misma de la metodología definida por Jacobson (RUP adaptado para el DSBC), la cual se utilizó como base para la definición de la propuesta a implantar en el Polo. La aplicación de la misma se realizó de la siguiente manera.

En los pasos iniciales se dividió la fuerza de trabajo en dos equipos equitativos asignándosele un problema a cada uno. Durante el Modelamiento del Negocio se realizaron todas las actividades y se construyeron los artefactos que define este flujo de trabajo. Al cumplir con el desarrollo de los artefactos por cada uno de los miembros, se realizó la entrega y exposición de los mismos al resto del equipo de

ensamblaje durante varios encuentros donde se discutieron las tareas realizadas. Estas exposiciones y debates estuvieron enfocados fundamentalmente a cumplir con el propósito de garantizar la comprensión por parte de ambos equipos de los negocios correspondientes a uno y otro problema.

Durante el flujo de trabajo de requerimientos, se realizaron todas las tareas y artefactos que propone la misma. Se introdujo la actividad de análisis de dominio, para hacer un análisis profundo de los requisitos encontrados y encapsular las funcionalidades derivadas de los mismos en componentes de software. Se realizaron también exposiciones y debates de los trabajos realizados con la participación de los miembros de ambos proyectos.

También fue objetivo de las exposiciones antes mencionada la definición de componentes que encapsularan funcionalidades de los dos proyectos y que pudieran utilizarse en ambos, para garantizar de esta manera la reutilización y evitar la duplicación del esfuerzo y costo general en el desarrollo de las soluciones. Además para ir dando paso a la necesidad de contar con un repositorio de componentes debidamente documentados que permitan la adecuada identificación de los mismos para su futura reutilización.

En el flujo de trabajo de análisis y diseño de la fase de elaboración, llamado en una visión a alto nivel de esta propuesta metodológica “aprovisionamiento”, se definieron de conjunto los componentes finales a implementar descritas sus arquitecturas, interfaces, y estructura. También se describieron las funcionalidades del sistema que no alcanzaban a ser implementadas como componentes.

Basados en las actividades realizadas en las fases antes mencionadas, se procedió a la implementación de ambos sistemas, compartiendo los componentes que podías ser utilizados por ambas aplicaciones que habían sido definidos con antelación. Se utilizó para el desarrollo de las aplicaciones el lenguaje de Programación Java[2] en el Entorno Integrado de desarrollo NetBeans[3] 6.0, cumpliendo de esta manera con las políticas determinadas por el polo PetroSoft para el desarrollo con tecnologías libres. Actualmente el desarrollo de ambos proyectos se encuentra en las fases de transición, determinando y corrigiendo posibles fallos de funcionamiento e implementación.

CONCLUSIONES PARCIALES

La evaluación de la propuesta para el DSBC en el polo productivo PetroSoft por el método de los Expertos, valida y garantiza la aplicación y utilización de la misma con éxito en los proyectos productivos

2 Lenguaje de programación de alto nivel, multiplataforma y orientado a objetos desarrollado por Sun Microsystems.

3 Entorno de desarrollo integrado (IDE) empleando la Plataforma NetBeans para el desarrollo de aplicaciones usando Java.

del polo donde se adopte. La misma fue evaluada por expertos seleccionados basándose en sus competencias y otros aspectos como experiencia acumulada en el desarrollo de proyectos productivos, y relación de la asignatura que imparte con el tema que rige este trabajo.

El éxito de la aplicación de una metodología de software depende de diversos factores tales como: (1) disponibilidad de recursos, (2) experiencia de los desarrolladores, (3) Motivación del equipo de desarrollo entre otros muchos aspectos. La aplicación inicial de la metodología propuesta para el DSBC en el polo productivo PetroSoft en los proyectos pilotos realizados, ha confirmado los beneficios que brinda esta tecnología tales como una disminución significativa en los tiempos de desarrollo y menor tasa de errores en los proyectos.

CONCLUSIONES

En el presente trabajo se ha realizado un estudio del estado del arte de las metodologías para el desarrollo de software para el DSBC en el mundo así como una análisis de las características principales de las mismas, modo en que estas operan, roles que definen, sus ventajas y desventajas con el fin de ver cuál metodología es más óptima para ser aplicada a los diferentes proyectos de la Facultad 9.

El uso de la propuesta realizada en este trabajo basada en la solución de RUP para el DSBC, aplicada en los proyectos productivos del polo, aumentará sin dudas la productividad y calidad de los mismos permitiendo entregar los productos en fecha y lo que es igualmente importante, se podrá asumir una mayor demanda de soluciones informáticas en el polo. La misma fue evaluada por el método de los expertos e inicialmente probada en proyectos pilotos que se efectuaron en por parte del grupo de ensamblaje de Componentes de Software del polo PetroSoft de la Facultad 9.

Además forma parte de la estrategia para el ensamblaje y DSBC en el polo que es el objetivo de este trabajo la propuesta de el Desarrollo de Software Basado en Líneas de Productos (DSBLP) el cual constituye la cima en la evolución del DSBC y una propuesta muy interesante a aplicar al polo puesto que se trata de la agrupación y organización de todo el desarrollo en componentes orientado a uno o mas productos en específico, en nuestro caso el petróleo.

RECOMENDACIONES

Debido al estudio de las tendencias actuales se evidencia la convergencia internacional hacia el DSBC introduciéndose conceptos basados en esta idea como es el caso de las “Fabricas de Software”. El perfeccionamiento de los procesos de software internacionales exige la constante actualización y desarrollo de los nuestros para volver nuestra producción de software más productiva y competente.

Por la importancia que reviste la necesidad de reducir los tiempos de desarrollo, aumentar la productividad en el polo productivo PetroSoft, se recomienda:

- Aplicar esta metodología en los proyectos productivos del polo PetroSoft de la facultad 9.
- Realizar un análisis de la propuesta para propiciar su aplicación en el resto de los polos productivos de la facultad.
- Realizar un estudio profundo de la propuesta, para considerar posibles adaptaciones y fusiones con otras metodologías de desarrollo altamente utilizadas internacionalmente como XP (Xtreme Programming, Programación Extrema) y MSF (Microsoft Solution Framework, Marco de Soluciones de Microsoft).

REFERENCIAS BIBLIOGRÁFICAS:

ARIZA, M. M., J. C. . *INTRODUCCIÓN Y PRINCIPIOS BÁSICOS DEL DESARROLLO DE SOFTWARE BASADO EN COMPONENTES*, 2004.
[Disponible en: http://pegasus.javeriana.edu.co/~jcpymes/Docs/DSBC.pdf

ASTIGARRAGA, E. *EL MÉTODO DELPHI*.
Disponible en: http://www.prospectiva.eu/zaharra/Metodo_delphi.pdf

BERTOA, M. T., J. VALLECILLO,A. . *Atributos de Calidad para Componentes COTS: Una valoración de la información ofrecida por los vendedores.*, 2002.
[Disponible en: http://www.lcc.uma.es/~av/Publicaciones/03/TICS03.pdf

BOEHM, B. B., VICTOR R. *COTS-Based Systems Top 10 List*, 2001.
[Disponible en: http://sunset.usc.edu/csse/TECHRPTS/2001/usccse2001-504/usccse2001-504.pdf

BROWN, A. *Constructing Superior Software, capítulo Building Systems from Pieces: Principles and Practice of Component-based Software Engineering*, 1999.
[Disponible en: https://pearsoned.com.au/Schools/search5.asp?Isbn=9780130887207

CRAIG M., B. O., PATRICIA *Managing Software Acquisition: Open Systems and COTS Products*, 2001.
[Disponible en: http://www.sei.cmu.edu/publications/books/engineering/managing-sw-acquisition.html

CHEESMAN, J. D., J. *Book Review - UML Components A Simple Process for Specifying Component-Based Software*, 2001. [Disponible en: http://www.syntropy.co.uk/umlcomponents/cbdiforum.htm

DEAN, J. V., M. *System Implementation Using Commercial off-the-shelf (COTS) Software*, 1997.
[Disponible en: http://iit-iti.nrc-cnrc.gc.ca/iit-publications-iti/docs/NRC-40173.pdf

FREDERICK P. BROOKS, J. *No Silver Bullet Essence and Accidents of Software Engineering*, 1987.
[Disponible en: http://www.virtualschool.edu/mon/SoftwareEngineering/BrooksNoSilverBullet.html

FUENTES, L. T., J. M. VALLECILLO,A. *Desarrollo de Software Basado en Componentes*, 2002.
[Disponible en: http://www.lcc.uma.es/~av/Docencia/Doctorado/tema1.pdf

G. HEINEMAN, W. T. C. *Component-based Software Engineering: Putting the Pieces Together*. 2001. p. 0-201-70485-4

I. JACOBSON, G. B., J. RUMBAUGH. *El Proceso Unificado de Desarrollo de Software*. 2000. p. 84-7829-036-2

IBM-WEBSPHERE. 2001.
[Disponible en: http://netbuzos.spaces.live.com/blog/cns!994BD929B80714FB!155.entry

IRIBARNE MARTÍNEZ, L. F. *Un Modelo de Mediación para el Desarrollo de Software basado en Componentes COTS*. Disponible en: http://www.cotstrader.com/thesis/memoria-completa.pdf

JACOBSON, I. B., G. RUMBAUGH,J. *El Proceso Unificado de Desarrollo de Software*. 2000. p. 84-7829-036-2

KURT WALLNAU, R. S., SCOTT A. HISSAM. *Building Systems From Commercial Components*. 2001. p. 0-201-70064-6

MONTILVA, J. A. *Desarrollo de Software Basado en Líneas de Productos de Software*, 2006. [Disponible en: http://www.ieee.org.ar/downloads/2006-montilva-productos.pdf

PRESSMAN, R. S. *Ingeniería de Software, un enfoque práctico*. 2005. p. 9701054733

REYNOSO, C. B. *Arquitectura de Software - Arquitecturas Orientada a Servicios (SOA)*, 2004. [Disponible en: http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/introarq.pdf

SZYPERSKI, C. *Component Software: Beyond Object-Oriented Programming*, 2002. [Disponible en: http://msdn.microsoft.com/es-es/library/bb972268.aspx#ref06

SZYPERSKI, C. P., C. *Components vs. Objects vs. Component Objects*, 1997. [Disponible en: http://www.oberon2005.ru/paper/cs1999.pdf

TERREROS, J. C. *Desarrollo de Software basado en Componentes*, 2004. [Disponible en: http://msdn.microsoft.com/es-es/library/bb972268.aspx#EEAA

BIBLIOGRAFÍA

ARIZA, M. M., J. C. . INTRODUCCIÓN Y PRINCIPIOS BÁSICOS DEL DESARROLLO DE SOFTWARE BASADO EN COMPONENTES, 2004.
[Disponible en: http://pegasus.javeriana.edu.co/~jcpymes/Docs/DSBC.pdf

ASTIGARRAGA, E. EL MÉTODO DELPHI.
Disponible en: http://www.prospectiva.eu/zaharra/Metodo_delphi.pdf

BERTOA, M. T., J. VALLECILLO,A. . Atributos de Calidad para Componentes COTS: Una valoración de la información ofrecida por los vendedores., 2002.
[Disponible en: http://www.lcc.uma.es/~av/Publicaciones/03/TICS03.pdf

BOEHM, B. B., VICTOR R. COTS-Based Systems Top 10 List, 2001.
[Disponible en: http://sunset.usc.edu/csse/TECHRPTS/2001/usccse2001-504/usccse2001-504.pdf

BROOKS, J. F. P. No Silver Bullet Essence and Accidents of Software Engineering, 1987.
[Disponible en: http://www.virtualschool.edu/mon/SoftwareEngineering/BrooksNoSilverBullet.html

BROWN, A. Constructing Superior Software, capítulo Building Systems from Pieces: Principles and Practice of Component-based Software Engineering, 1999.
[Disponible en: https://pearsoned.com.au/Schools/search5.asp?Isbn=9780130887207

CHEESMAN, J. D., J. Book Review - UML Components A Simple Process for Specifying Component-Based Software, 2001.
[Disponible en: http://www.syntropy.co.uk/umlcomponents/cbdiforum.htm

CLEMENTS, P. N., L. . Software Product Lines: Practices and Patterns, 2001. [Disponible en: http://www.alibris.com/booksearch?qsort=&page=1&matches=9&browse=1&qwork=6176498&full=1

COUNCILL, W. T. H., G. . Component-based Software Engineering: Putting the Pieces Together, 2001. [Disponible en: http://www.amazon.co.uk/Component-based-Software-Engineering-Putting-Together/dp/0201704854

CRAIG M., B. O., PATRICIA Managing Software Acquisition: Open Systems and COTS Products, 2001. [Disponible en: http://www.sei.cmu.edu/publications/books/engineering/managing-sw-acquisition.html

DEAN, J. V., M. System Implementation Using Commercial off-the-shelf (COTS) Software, 1997. [Disponible en: http://iit-iti.nrc-cnrc.gc.ca/iit-publications-iti/docs/NRC-40173.pdf

FUENTES, L. T., J. M. VALLECILLO,A. Desarrollo de Software Basado en Componentes, 2002. [Disponible en: http://www.lcc.uma.es/~av/Docencia/Doctorado/tema1.pdf

GAO, J. Z., E. Y. SHIM, S. . Monitoring Software Components and Component-Based Software. Disponible en: http://www.engr.sjsu.edu/gaojerry/report/compsac2000.pdf

GOMMA, H. Disponible en: www.ieee.org.ar/downloads/2006-montilva-productos.pdf

GRADY BOOCH, J. R., IVAR JACOBSON. The Unified Modeling Language User Guide, October 20, 1998. Addison Wesley, 1998. [Disponible en: www.unap.cl/metadot/index.pl?id=25450&isa=Item&field_name=item_attachment_file&op=download_file

IBM-WEBSPHERE. 2001.
[Disponible en: http://netbuzos.spaces.live.com/blog/cns!994BD929B80714FB!155.entry

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology. IEEE td610.12, IEEE Computer Society Press.

IRIBARNE MARTÍNEZ, L. F. Un Modelo de Mediación para el Desarrollo de Software basado en Componentes COTS. Disponible en: http://www.cotstrader.com/thesis/memoria-completa.pdf

JACOBSON, I. B., G. RUMBAUGH,J. El Proceso Unificado de Desarrollo de Software. 2000. p. 84-7829-036-2

KRUEGER, C. W. Introduction to Software Product Lines, 2006.
[Disponible en: http://www.softwareproductlines.com/introduction/introduction.html

MONTILVA, J. A. Desarrollo de Software Basado en Líneas de Productos de Software, 2006.
[Disponible en: http://www.ieee.org.ar/downloads/2006-montilva-productos.pdf

PRESSMAN, R. S. Ingeniería de Software, un enfoque práctico. 2005. p. 9701054733

Prof. Stafford (2003) Software Maintenance As Part of the Software Life Cycle.

REYNOSO, C. B. Arquitectura de Software - Arquitecturas Orientada a Servicios (SOA), 2004. [Disponible en: http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/introarq.pdf

SEACORD, R. C. Building Systems from Commercial Components: Classroom Experiences, 2002. [Disponible en: http://www.sei.cmu.edu/news-at-sei/columns/the_cots_spot/2002/1q02/cots-spot-1q02.pdf

SZYPERSKI, C. Component Software: Beyond Object-Oriented Programming, 2002.
[Disponible en: http://msdn.microsoft.com/es-es/library/bb972268.aspx#ref06

SZYPERSKI, C. P., C. Components vs. Objects vs. Component Objects, 1997.
[Disponible en: http://www.oberon2005.ru/paper/cs1999.pdf

TERREROS, J. C. Desarrollo de Software basado en Componentes, 2004.
[Disponible en: http://msdn.microsoft.com/es-es/library/bb972268.aspx#EEAA

WALLNAU, K. S., R. HISSAM, S. A. . Building Systems From Commercial Components. 2001. p. 0-201-70064-6

ANEXOS

Anexo 1: Solución metodológica de RUP para el DSBC (Ayuda de Rational Unified Process).

Actividades de la fase inicial

Se aplica el flujo de trabajo básico para la fase inicial, con las siguientes ampliaciones o variaciones:

Gestión de proyectos

Actividad: Concebir un proyecto nuevo

- El foco de la Tarea: Desarrollar caso de negocio se ajusta para que tenga en cuenta que la utilización de componentes cambia la estructura de costes del desarrollo. Específicamente, el coste de desarrollar componentes disminuye, pero se necesita más esfuerzo para identificar componentes y comprobar que los componentes seleccionados satisfacen los requisitos.

Actividad: Evaluar el riesgo y el ámbito del proyecto

- Al tomar el enfoque de un componente, cambia la naturaleza de determinados riesgos y se introducen riesgos nuevos. Específicamente:
  - los componentes de origen externo aumentan el riesgo porque introducen elementos críticos que no están bajo el control directo del equipo del proyecto
  - los componentes de origen externo pueden reducir el tiempo de desarrollo y, de este modo, reducir el riesgo de recursos
  - los componentes de origen externo pueden distorsionar la arquitectura del sistema si imponen restricciones arquitectónicas propias

Actividad: Planificar el proyecto

- En la Tarea: Planear fases e iteraciones, el plan de la fase de construcción tiene el potencial para mostrar la división del proyecto en dos seguimientos diferentes pero paralelos: uno que desarrolla componentes específicos del dominio y específicos de la aplicación (organizado en las capas superiores de la arquitectura; consulte el apartado Concepto: Creación de capas) y otro que desarrolla los componentes que no son específicos del dominio ni de la aplicación y que se organizan en las capas inferiores. En algunos casos, los equipos de desarrollo gestionados independientemente desarrollarán componentes reutilizables. La decisión de introducir seguimientos paralelos es un aspecto de personal y recursos conducido por un deseo de gestionar componentes reutilizables como activos independientes de las aplicaciones en que se despliegan.

Requisitos

Actividad: Perfeccionar la definición del sistema

- Cuando se perfeccionan los requisitos del sistema, deben capturarse las restricciones que impone la infraestructura del componente seleccionado. Las infraestructuras de componentes aumentan la productividad de desarrollo, en parte restringiendo los grados de libertad que se ofrecen al arquitecto y el diseñador de software. La Tarea: Detallar los requisitos de software debe centrarse en documentar estas restricciones.

Prueba

Actividad: Definir la misión de evaluación

- Debería crearse un plan de prueba que identifique las pruebas globales previstas para el proyecto, denominado "Plan de prueba maestro".

Entorno

Actividad: Preparar el entorno para el proyecto

- Cuando recopile y prepare directrices para el proyecto, consulte la Tarea: Preparar directrices (RUP, desarrollo tradicional) para el proyecto para obtener detalles, tenga en cuenta la infraestructura de componentes específica y las restricciones que impone. Las directrices deben incluir cómo diseñar y codificar utilizando la infraestructura. También deberían proporcionar instrucciones de prueba sobre cómo comprobar la conformidad con la infraestructura de componentes y las interfaces definidas entre componentes.

Actividades de la fase de elaboración

Se aplica el flujo de trabajo básico para la fase de elaboración, con las siguientes ampliaciones o variaciones:

Requisitos

Actividad: Perfeccionar la definición del sistema

- La Tarea: Detallar los requisitos de software también se centra en las restricciones y los requisitos técnicos y no funcionales que imponen los componentes construidos o adquiridos. Los requisitos no funcionales específicos que se deben considerar son el tamaño, el rendimiento, la impresión en la memoria o el disco, los aspectos de la licencia de tiempo de ejecución y otras restricciones similares que influirán en la construcción o la selección del componente.

Análisis y diseño

Actividad: Definir una arquitectura candidata

- La Tarea: Análisis de la arquitectura utilice la infraestructura de componentes y los requisitos técnicos y no funcionales para definir una arquitectura inicial, incluido un esquema de creación de capas inicial y un conjunto por omisión de componentes y servicios (representados como mecanismos de análisis y diseño). La Tarea: Análisis de caso de uso se centra en identificar componentes significativos arquitectónicamente de casos de uso significativos arquitectónicamente.

Actividad: Perfeccionar la arquitectura

- La Tarea: Estructurar el modelo de implementación establece un modelo de implementación compatible con la infraestructura de componentes y la estructura y las responsabilidades del equipo o los equipos de desarrollo.

  La Tarea: Identificar los mecanismos de diseño perfecciona los mecanismos de diseño iniciales para que tengan en cuenta componentes y servicios específicos de la infraestructura.

  La Tarea: Identificar elementos de diseño identificará los principales componentes significativos arquitectónicamente del sistema. Las responsabilidades potencialmente reutilizables deberían agruparse para mejorar la capacidad de reutilización; las funciones específicas de la aplicación deben separarse de las funciones específicas del dominio e independientes del dominio y la aplicación. Para el diseño, los componentes se pueden representar como el Producto de trabajo: Subsistemas de diseño. Debería identificarse el Producto de trabajo: Interfaces para estos componentes/subsistemas.

  La Tarea: Incorporar elementos de diseño existentes garantiza que los componentes identificados son coherentes y compatibles con los componentes existentes identificados en iteraciones anteriores, en la infraestructura o en orígenes externos.

  La Tarea: Describir la arquitectura del tiempo de ejecución describe el proceso básico y la arquitectura de hebras de la infraestructura de componentes, mientras que la Tarea: Describir la distribución describe el entorno informático distribuido en el que se ejecutará la aplicación de componentes.

Actividad: Diseñar componentes

- La Tarea: Diseño del subsistema perfecciona aún más el diseño de los componentes, identificando las clases del componente que proporcionan el comportamiento real del componente. Al principio de la fase de elaboración, es probable que haya una sola clase, un tipo de 'proxy de componente/subsistema' que actúa como un fragmento para simular el comportamiento del componente para crear prototipos arquitectónicos. Más adelante, el comportamiento de esta clase se distribuye en una colaboración de clases contenidas en el subsistema. Estas clases contenidas se perfeccionan en la Tarea: Diseño de clase (RUP, Desarrollo Tradicional).

Actividad: Diseñar la base de datos

- El aspecto más importante de la elaboración es garantizar que la estrategia de permanencia es escalable y que el mecanismo de permanencia y el diseño de bases de datos soportarán los requisitos de rendimiento del sistema. Las clases permanentes se identifican y se correlacionan con el mecanismo de permanencia. Los casos de uso de datos intensivos se analizan para garantizar

que los mecanismos serán escalables. Junto con las actividades de prueba, se valora y se valida el mecanismo de permanencia y el diseño de bases de datos.

Implementación

Actividad: Implementar componentes

La Tarea: Implementar los elementos de diseño debe cumplir las restricciones impuestas por la infraestructura de componentes, como se describe en las directrices de programación, que se proporcionan como parte del Producto de trabajo: Directrices específicas del proyecto. En la fase de elaboración, la mayoría de los componentes contienen una gran cantidad de código de 'fragmento para simulación', ya que la implementación se centra en validar la arquitectura, y no producen código de calidad de producción.

Prueba

Actividades: Definir la misión de evaluación, Verificar el enfoque de prueba, Probar y evaluar, Conseguir una misión aceptable, Aumentar los activos de prueba

Las actividades de prueba de la fase elaboración se centran en validar la arquitectura. Para un sistema basado en componentes, se centran en:

- ejercitar las interfaces entre componentes para garantizar que los límites del componente son adecuados
- una atención mayor en la prueba de rendimiento, especialmente en las pruebas de escala de rendimiento, para garantizar que se pueden sostener los volúmenes de transacción anticipados

También es necesario valorar las suposiciones inherentes de la infraestructura de componentes. Estas suposiciones suelen ser la escalabilidad y el rendimiento de los mecanismos de gestión de la transacción y la permanencia, donde el diseñador de mecanismos efectúa suposiciones ocultas que a menudo socavan eficazmente la arquitectura de la aplicación si no conoce la suposición.

Gestión de proyectos

Actividad: Planear la siguiente iteración

Si se utilizan los subsistemas de implementación como 'unidades lógicas de responsabilidad', el trabajo de construcción se puede particionar en dos o más "seguimientos" paralelos: uno que se centra en las funciones específicas de la aplicación y uno o varios que se centran en componentes genéricos reutilizables. Por supuesto, esto depende de si se tienen los recursos humanos suficientes para esfuerzos de desarrollo paralelo. La capacidad de dividir los equipos de desarrollo y trabajar en paralelo depende totalmente de la estabilidad de la arquitectura y, más específicamente, de la calidad y la

estabilidad de las interfaces entre los componentes. Esta división será posible si se realiza un esfuerzo enorme en la fase de elaboración.

Actividades de la fase de construcción

Se aplica el flujo de trabajo básico para la fase de construcción, con las siguientes ampliaciones o variaciones:

Gestión de proyectos

Actividad: Planear la siguiente iteración

Anteriormente se describió la planificación de la primera iteración de construcción, igual que sucede hacia el final de la elaboración. Igual que la planificación de la iteración, la capacidad de dividir los equipos de desarrollo y trabajar en paralelo sigue dependiendo de la estabilidad de la arquitectura y la calidad y la estabilidad de las interfaces entre los componentes.

Análisis y diseño

La Actividad: Perfeccionar la arquitectura y la Actividad: Diseñar componentes

El aspecto principal de la construcción es el análisis del recordatorio de los casos de uso y la identificación de las colaboraciones de componentes y los componentes adecuados que realizan los casos de uso. La arquitectura existente se amplía y se completa y los "comportamientos internos" del componente se diseñan y se implementan completamente.

Actividad: Diseñar la base de datos

El aspecto principal de la construcción es terminar el diseño de la base de datos y asegurarse de que todas las clases permanentes son soportadas tanto por la base de datos como por el mecanismo de permanencia. Este trabajo se realiza en paralelo y de forma repetitiva con el trabajo que se realiza en la Actividad: Perfeccionar la arquitectura y la Actividad: Diseñar componentes. La organización ideal consiste en contar con equipos de componentes integrados, y con coordinación entre equipos de los asuntos de permanencia para garantizar un buen diseño de base de datos.

Implementación

Actividad: Implementar los elementos de diseño

- El trabajo aquí es similar al de elaboración, aunque los demás detalles se completan en incrementos, a medida que avanza la fase.
- La Actividad: Integrar los subsistemas y la Actividad: Integrar el sistema
- El sistema se construye progresivamente a medida que la fase continúa.

Prueba

Actividades: Definir la misión de evaluación, Verificar el enfoque de prueba, Validar la estabilidad de la compilación, Probar y evaluar, Conseguir una misión aceptable, Aumentar los activos de prueba.

Las pruebas de rendimiento siguen siendo importantes, aunque cada vez se presta más atención a las pruebas funcionales. Deben tenerse en cuenta la completitud de la funcionalidad, las pruebas de revisión de la funcionalidad existente y la satisfacción de las expectativas de rendimiento.

Actividades de la fase de transición

- El "**release" del producto** en el entorno web suele ser incremental y continuo, y se centra menos en la distribución tradicional de medios de soporte. La planificación del release debe ajustarse en consecuencia.
- El **soporte de producción** es cada vez más importante en la fase.
- Se realizan actividades de conversión de datos.

Anexo 2: Analogía entre la evolución de las tecnologías de la información y las ciudades. (TERREROS 2004)]

Al comprender la evolución de las ciudades actuales, podemos darnos una idea del futuro promisorio que tiene el desarrollo de software basado en componentes.

Por muchos años, las casas de software han desarrollado aisladamente, con aplicaciones creadas independientemente y sin mayor necesidad de interacción entre ellas. Dado que estas aplicaciones no estaban conectadas, no había mayores consecuencias para aquello. Pero recientemente se ha vuelto muy práctico el interconectar tanto las aplicaciones dentro de una casa de software, así como entre múltiples casas de software alrededor del mundo. Ahora la gente ya puede navegar y visitar aplicaciones muy distantes. Cantidades cada vez más grandes de información son fácilmente transmitidas entre aplicaciones. Pero lo que aún es difícil es hacer que estos datos trabajen entre distintas aplicaciones.

Para entender entonces lo que está ocurriendo en este momento con el ambiente Tecnologías de la Información (TI) hay que explorar 6 facetas de esta analogía:

Ciudades - Casas de Software

Las ciudades evolucionaron gradualmente como lugares para hacer comercio y manufactura. En estas ciudades existían edificios con poca o ninguna conexión entre ellos. Las ciudades tenían un contacto muy limitado con sus ciudades aledañas y desarrollaron su propia cultura, estilo y forma de hacer cosas. De la misma forma, las casas de software evolucionaron gradualmente mientras nuevas aplicaciones fueron construidas y luego extendidas. Cada aplicación separada e independiente de sus similares en la misma casa de software. Cada casa de software tenía su propia cultura, estilo y forma de hacer las cosas.

Las presiones económicas cambiaron nuestras ciudades, ya que fue la oportunidad económica la que realmente llevó a las ciudades a la modernización, a compartir servicios y a pensar en medios creativos para alcanzar eficiencias. Asimismo, las presiones económicas están cambiando nuestras casas de software. Mientras se construyen nuevas aplicaciones y se renuevan las más antiguas, hay que considerar cómo enlazarlas a la infraestructura compartida, cómo conectarlas y cómo dividirlas de manera que se maximice la reusabilidad de las mismas. Este es el reto al que todos nos vemos enfrentados en la actualidad.

Fábricas y Edificios – Aplicaciones

En los primeros años del siglo XIX, la manufactura típicamente era simple e independiente. Los bienes producidos estaban limitados tanto por las necesidades del mercado local, como por la sofisticación del proceso de manufactura. Las fábricas producían todas las partes del ensamblado final, armaban el ensamblado e incluso lo vendían. Si uno quería un par de zapatos, tenía que irse a la fábrica de zapatos. Esta no era la forma más eficiente de fabricar bienes y los ítems fabricados eran caros y usualmente no de la mejor calidad. Muchas de nuestras aplicaciones actuales son como aquellas fábricas. Producen datos procesados independientemente unos de otros, los cuales se entregan en 'mercados' limitados. Están integradas verticalmente y usualmente no aceptan el trabajo de otras aplicaciones como entrada.

El ferrocarril alteró profundamente la manufactura. Al bajar los costos de transportar las partes fabricadas, el mismo permitió que los fabricantes locales produjeran bienes más sofisticados y de mayor calidad. La componentización les permitió a los artesanos enfocarse en sus principales competencias, en lugar de tener que entender los diversos procesos necesarios para producir todo un ensamblado sofisticado. Los negocios se especializaron e independizaron. Tanto para las fábricas como para las aplicaciones, la independencia es esencial. Uno no puede terminar su trabajo si se necesita hacer que todo trabaje en conjunto perfectamente. Pero, aunque la independencia es esencial, no se pueden olvidar las ventajas de la interconexión, ya que es a través de la reutilización del trabajo de los demás que uno logra cumplir su trabajo con éxito y es justamente la demanda que ejercen los demás sobre nuestro trabajo lo que nos da el estímulo económico para seguir existiendo.

Transporte – Comunicaciones

A mediados del siglo XIX llegó el ferrocarril. Se hicieron enormes cantidades de dinero moviendo personas, carbón y trigo de un lugar a otro. Con una demanda arrolladora de transporte, eventualmente todo Estados Unidos estaba interconectado por ferrocarril. No solo que la gente empezó a viajar a nuevos lugares para conocer nuevas culturas, sino que ahora los comerciantes podían vender artículos de formas nunca antes pensadas. Pero más importante aún, el movimiento de los artículos despertó la expectativa de que las cosas funcionen en conjunto. Antes del ferrocarril simplemente no importaba si los bienes de un fabricante eran incompatibles con los de otro fabricante.

Al final del siglo XX llegó Internet. Se invirtieron montos enormes en navegación, correo electrónico, JPEGs, MP3s y chat. Se tendieron los hilos para permitir la navegación y el movimiento de las formas más simples de datos. El navegador le permitió a la persona el transportarse para interactuar directamente con una aplicación distante. Sin embargo, el movimiento de los datos aún no funciona bien en conjunto, haciendo muy limitados los procesos de negocios a través del Internet. Las nuevas

conexiones implicaron nuevos cambios en la estandarización de los artículos y los datos. Pronto esto implicaría cambios en los procesos de negocios.

Bienes Fabricados - Datos Estructurados

A inicios del siglo XVIII los bienes se hacían a mano. Los ensamblados se creaban "haciendo ajustes". Si una llave no encajaba en la cerradura, se ajustaba la cerradura de alguna manera para que permita el paso de la llave. Pioneros como **Honore LeBlanc** y **Eli Whitney** propusieron la idea de crear partes estandarizadas en el proceso de manufactura. Al establecer controles severos sobre la especificación y producción de las partes componentes, se pudieron realizar masivas producciones de todo tipo de artículos. Sin embargo, esta era una estandarización hacia dentro de las empresas. Pero para finales del siglo XVIII la idea ya se había expandido entre los fabricantes y se produjeron todo tipo de estándares para las partes comunes. Había tamaños y medidas para los artículos, con la expectativa de que aquellos producidos por una fábrica serían intercambiables e interoperables con componentes similares y complementarios producidos por otra. Las compañías que produjeron las partes con un alto grado de precisión tuvieron éxito; lo que tenían un proceso menos consistente, fracasaron.

Hoy en día aún tenemos estructuras de datos no estandarizadas. Cada aplicación modela la información a su propia manera y dependemos de operadores humanos para 'ajustar' las aplicaciones y así lograr integrarlas. Es necesario agregar semántica para hacer que las aplicaciones se entiendan. De la misma forma como el mercado demandó que se pudieran intercambiar artículos a finales de los '80, el mismo demandará el intercambio de datos en un futuro cercano. Esto significa estandarizar la funcionalidad de conceptos de negocio como un 'cliente' o una 'orden de compra'. Las organizaciones que no se percaten de las eficiencias de la integración-por-diseño perderán a la larga frente a los que persigan estas eficiencias. El resultado de este cambio será un boom económico para las compañías que sobrevivan a él y un dramático mejoramiento para el diario vivir de las personas.

Ensamblados Fabricados - Empresas Virtuales

La mayoría de los fabricantes de bicicletas no producen llantas, de la misma forma como quienes hacen camisas no producen sus propios botones. Al crear ensamblados con los mejores componentes disponibles, los fabricantes de bicicletas pueden crear productos más sofisticados y de mayor calidad. La competencia entre fabricantes de componentes conlleva eficiencias y mejoras en la calidad. Para lograr esto, necesitan especificaciones detalladas de las partes componentes, así como deben también considerar el contexto en el que cada parte será usada. Las compañías de hoy

en día están 'creando ensamblados' de su funcionalidad de negocios. En lugar de crear un departamento de distribución y entrega, el trabajo se entrega como **outsourcing**. En lugar de fabricar el producto, se delega la construcción del mismo a una compañía que se especialice en producción de bajo costo y alta calidad. La definición de 'compañía' evoluciona.

Las comunicaciones de alta velocidad e información estructurada ofrecen beneficios similares, produciendo la virtualización de las organizaciones. Se puede crear un modelo de componentes de negocio definiendo claramente la semántica y requerimientos operacionales de nuestras capacidades de negocios. Al definir interfaces claras, se puede encapsular los detalles de cómo estas capacidades son implementadas y cada componente puede ser orquestado como miembro de cualquier número de procesos. Se pueden incluso crear proveedores especializados que ofrezcan servicios de marketing, ventas, producción, recursos humanos, etc. Para lograr esto, se necesitan especificaciones detalladas de las capacidades de los componentes. Así, con estándares, se puede lograr la composición de cualquier cosa, porque los proveedores de componentes pueden manejar el costo de optimización a través de mercados amplios y la competencia conlleva cada vez mayores eficiencias.

Comercialización y Distribución - Procesos de Negocio

A finales del siglo XIX los centros de comercio urbanos se habían desarrollado. Los bienes se habían vuelto más sofisticados y las opciones del consumidor habían aumentado. Sin embargo, el ir de compras era algo tedioso. Un día de compras podía implicar el tomar el tren hacia la ciudad, luego ir donde el carnicero y luego donde el panadero, pasar comprando las verduras y por último una vuelta por la farmacia. Sin embargo, muy pronto la estandarización de los tamaños redujo significativamente el costo de muchos bienes permitiendo la producción masiva y la habilidad de transportar bienes a ubicaciones centrales para la venta y dieron origen a los centros comerciales y el supermercado. Así por ejemplo, **Wal-Mart** logró nuevas eficiencias al hacer primar el poder del comercio sobre los fabricantes. Wal-Mart logró proveer de una experiencia de compras placentera, de bajo costo y centralizada en un solo punto.

Examinando la situación actual de los procesos de negocios, podemos observar dos tipos de integración. Una de ellas se basa simplemente en enviar un fax y esperar que el mismo haya sido recibido y que tal vez nos respondan. Otra técnica que reduce errores es la conocida integración ALT-TAB, la cual permite el uso del porta papeles para copiar datos entre aplicaciones. Pero si se quiere hacer algo realmente mejor, se necesita lograr el intercambio y estandarización de datos y operaciones. Los procesos de negocios aún son hechos a mano y los estándares son muy pobres. En lo futuro, los procesos de negocios crecerán para ser la fuerza que le dé la forma y defina los

estándares para las nuevas aplicaciones, de la misma forma como Wal-Mart impone los estándares para cientos y miles de artículos.

Anexo 3: Tabla distribución chi-cuadrado inversa.

Tabla 1 Distribución chi-cuadrado inversa

k \ P	0,01	0,05	0,10	0,20	0,25	0,30	0,40	0,50	0,60	0,70	0,75	0,80	0,90	0,95	0,99
1	0,000	0,004	0,016	0,064	0,102	0,148	0,275	0,455	0,708	1,074	1,323	1,642	2,706	3,841	6,635
2	0,020	0,103	0,211	0,446	0,575	0,713	1,022	1,386	1,833	2,408	2,773	3,219	4,605	5,991	9,210
3	0,115	0,352	0,584	1,005	1,213	1,424	1,869	2,366	2,946	3,665	4,108	4,642	6,251	7,815	11,34
4	0,297	0,711	1,064	1,649	1,923	2,195	2,753	3,357	4,045	4,878	5,385	5,989	7,779	9,488	13,28
5	0,554	1,145	1,610	2,343	2,675	3,000	3,656	4,351	5,132	6,064	6,626	7,289	9,236	11,07	15,09
6	0,872	1,635	2,204	3,070	3,455	3,828	4,570	5,348	6,211	7,231	7,841	8,558	10,64	12,59	16,81
7	1,239	2,167	2,833	3,822	4,255	4,671	5,493	6,346	7,283	8,383	9,037	9,803	12,02	14,07	18,48
8	1,647	2,733	3,490	4,594	5,071	5,527	6,423	7,344	8,351	9,524	10,22	11,03	13,36	15,51	20,09
9	2,088	3,325	4,168	5,380	5,899	6,393	7,357	8,343	9,414	10,66	11,39	12,24	14,68	16,92	21,67
10	2,558	3,940	4,865	6,179	6,737	7,267	8,295	9,342	10,47	11,78	12,55	13,44	15,99	18,31	23,21