Estrategia de Prototipado y Ensamblado de Componentes

Tabla de Contenidos

Estrategia de Prototipado y Ensamblado de Componentes

Introducción

El presente material tiene el propósito de presentar en términos generales conceptos vinculados con algunos aspectos del desarrollo de software cómo lo son las Estrategias de Prototipado y de Ensamblado de Componentes.

Es importante destacar que este material es una recopilación que destaca los conceptos fundamentales para el desarrollo de la temática en el aula, siendo los objetivos de cada una de las estrategias los siguientes:

La **estrategia de Prototipado** es una elección de modelo de proceso que se recomienda elegir a la hora de implementar un proyecto complejo, con dominio no familiar, que utilizará una tecnología desconocida; de ahí que surge la necesidad de requerirse el uso de prototipos en el diseño y la implementación, además de utilizarlos durante la validación de requerimientos.

La **estrategia de Ensamblado de Componentes** es una decisión arquitectónica y de diseño de la solución final del software a construir, que implica desde decidir implementar por componentes, definir la granularidad del componente hasta su ensamblando final y prueba de integración.

Estrategia de Prototipado

- Es un modo de desarrollo de software.
- El prototipo es una aplicación que funciona.
- La finalidad del prototipo es probar varias suposiciones formuladas por analistas y usuarios con respecto a las características requeridas del sistema.
- Los prototipos se crean con rapidez, evolucionan a través de un proceso interactivo y tienen un bajo costo de desarrollo.

Concepto:

- ✓ Primera versión de un nuevo tipo de producto, en el que se han incorporado sólo algunas características del sistema final, o no se han realizado completamente.
- ✓ Modelo o maqueta del sistema que se construye para comprender mejor el problema y sus posibles soluciones:
  - evaluar mejor los requisitos.
  - probar opciones de diseño.

Características de los prototipos:

- ➢ Funcionalidad limitada.
- ➢ Poca fiabilidad.
- ➢ Características de operación pobres.
- ➢ Prototipo ≈ 10% presupuesto del proyecto.
  - ✓ normalmente pocos días de desarrollo.

Objetivos de los Prototipos

Los objetivos de los prototipos son:

a) aclarar los requerimientos de los usuarios

b) verificar la factibilidad del diseño del sistema

¿Cuándo es conveniente usar prototipos?

- ➢ *Siempre*, pero especialmente cuando...
  - ✓ El Área de aplicación no está bien definida (bien por su dificultad o por falta de tradición en su aplicación).

- ✓ El costo del rechazo por parte de los usuarios, por no cumplir sus expectativas, es muy alto.
- ✓ Es necesario evaluar previamente el impacto del sistema en los usuarios y en la organización.
- ✓ Se usan nuevos métodos, técnicas, tecnología.
- ✓ No se conocen los requerimientos o es necesario realizar una evaluación de los requerimientos
- ✓ Cuando los costos de inversión son altos.
- ✓ Cuando hay factores de riesgo alto asociados al proyecto.

Uso de los Prototipos:

- Se presenta al cliente un prototipo para su experimentación.
  - ✓ Ayuda al cliente a establecer claramente los requisitos.
- Ayuda a los desarrolladores a:
  - ✓ Validar corrección de la especificación.
  - ✓ Aprender sobre problemas que se presentarán durante el diseño e implementación del sistema.
  - ✓ Mejorar el producto.
  - ✓ Examinar viabilidad y utilidad de la aplicación.

Beneficios

Las razones para emplear los prototipos son:

a) aumentar la productividad

b) desarrollo planificado

c) entusiasmo de los usuarios respecto a los prototipos

Tipos de Prototipos

- **Prototipado de interfaz de usuario:** modelos de pantallas.
- **Prototipado funcional (operacional):** implementa algunas funciones, y a medida que se comprueba que son las apropiadas, se corrigen, refinan, y se añaden otras.
- **Prototipos Arquitectónicos**: sirven para evaluar decisiones arquitectónicas de infraestructura, tecnología e integración.
- **Modelos de rendimiento:** evalúan el rendimiento de una aplicación crítica (no sirven al análisis de requisitos).

Desde el punto de vista de la utilidad, los prototipos pueden construirse:

Rápido o desechable:

- Sirve al análisis y validación de los requisitos.
- Después se redacta la especificación del sistema y se desecha el prototipo.
- La aplicación se desarrolla siguiendo un paradigma diferente.
- Problema: cuando el prototipo no se desecha, y termina convirtiéndose en el sistema final.

Evolutivos:

- Comienza con un sistema relativamente simple que implementa los requisitos más importantes o mejor conocidos.
- El prototipo se aumenta o cambia en cuanto se descubren nuevos requisitos.
- Finalmente, se convierte en el sistema requerido.
- Actualmente se usa en el desarrollo de sitios Webs y en aplicaciones de comercio electrónico.

Con respecto al alcance, los prototipos puede clasificarse en:

- **Vertical:** desarrolla completamente alguna de las funciones.
- **Horizontal:** desarrolla parcialmente todas las funciones.

Etapas del Modelo de Prototipos

Las etapas del modelo de prototipos son:

1- Identificación de requerimientos conocidos

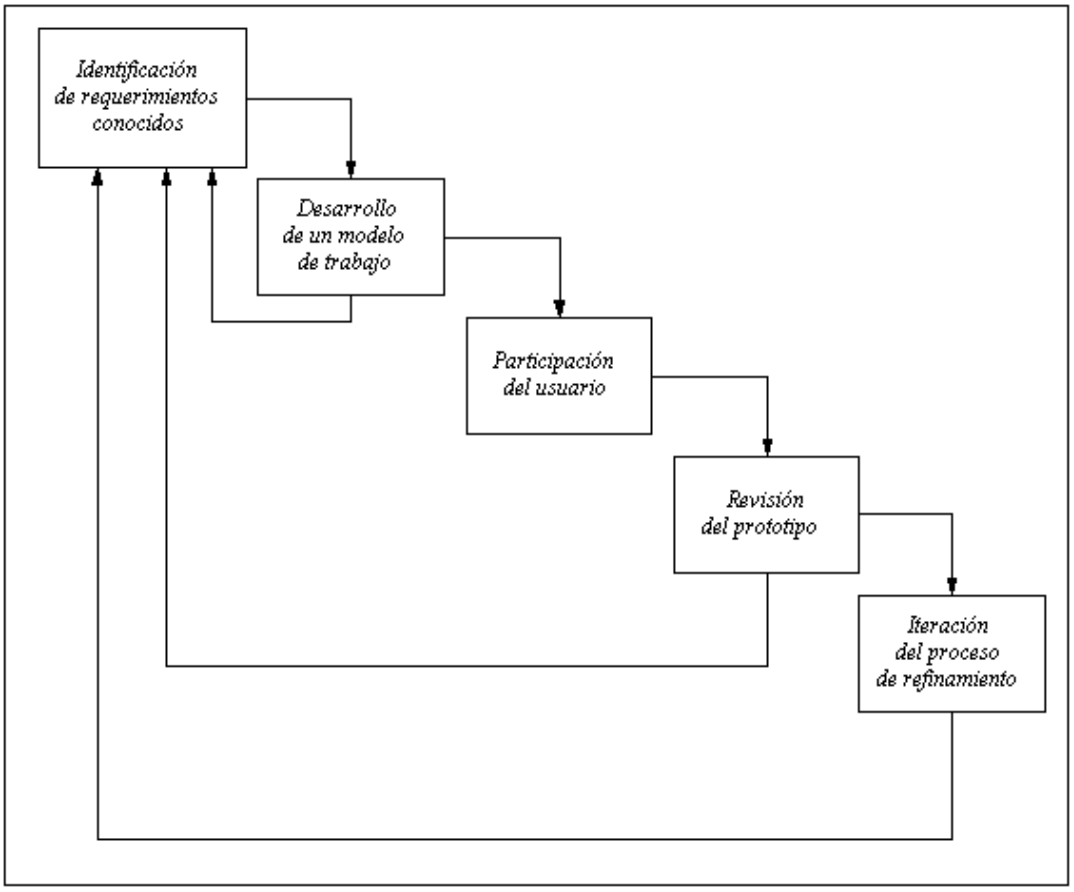
2- Desarrollo de un modelo de trabajo

3- Participación del usuario

4- Revisión del prototipo

5- Iteración del proceso de refinamiento

El modelo de prototipos o construcción de los mismos se puede graficar de la siguiente manera:



Estrategias para el Desarrollo de Prototipos

Las estrategias para el desarrollo de prototipos son:

1- Prototipos para pantallas: El elemento clave es el intercambio de información con el usuario.

2- Prototipos para procedimientos de procesamiento: El prototipo incluye solo procesos sin considerar errores.

3- Prototipos para funciones básicas: Solo se desarrolla el núcleo de la aplicación, es decir solo los procesos básicos.

Tareas de los usuarios

Las tareas que son responsabilidad de los usuarios son:

1- Identificar la finalidad del sistema

2- Describir la salida del sistema

3- Describir los requerimientos de datos

4- Utilizar y evaluar el prototipo

5- Identificar las mejoras necesarias

6- Documentar las características no deseables

Herramientas para el Desarrollo de Prototipos

Las herramientas para el desarrollo de prototipos serían:

- Lenguajes dinámicos de alto nivel.
- Lenguajes de cuarta generación (4GLs) (programación de BBDD).
- Ensamblaje de componentes y aplicaciones.
- Lenguajes no Orientados a Procedimientos
- Lenguajes de Consulta y Recuperación
- Generadores de reporte
- Generadores de aplicaciones
- Generadores de pantallas

Errores sobre el tema Prototipos

Los errores sobre el tema de prototipos son:

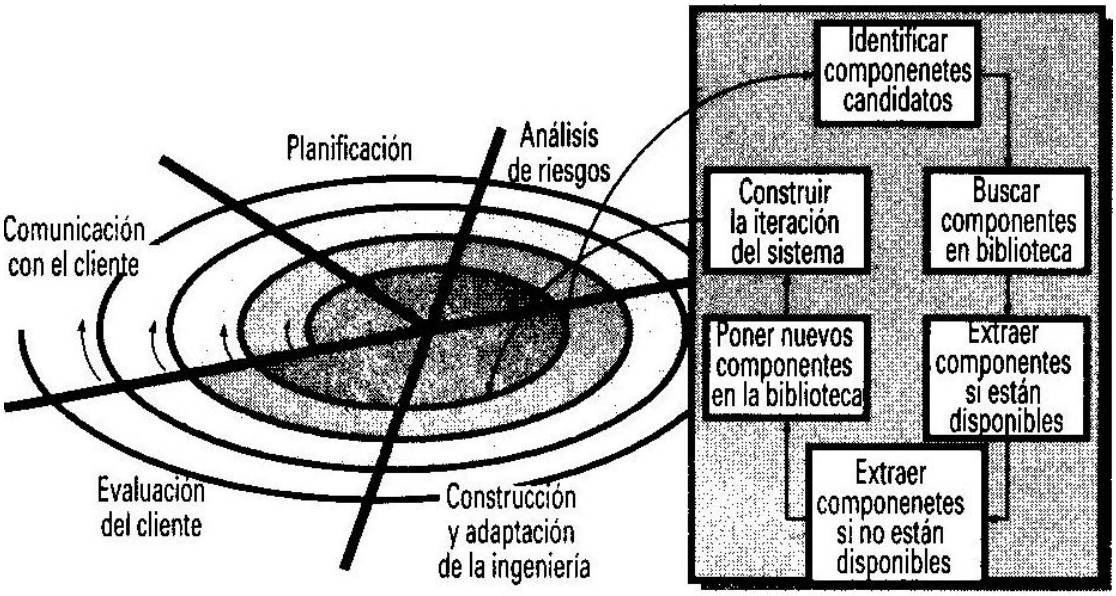
- El desarrollo del prototipo es trivial
- Es sólo para aplicaciones pequeñas
- Es sólo para aplicaciones sencillas
- La participación del usuario es simbólica

Críticas:

- El cliente cree que es el sistema funcional.
- Peligro de familiarización con malas elecciones iníciales.
- Difícil de administrar: se necesita mucha experiencia.
- Alto costo.

Estrategia de Ensamblaje de componentes

- Es un Modelo Evolutivo de Desarrollo de Software.
- El modelo de ensamblaje de componentes incorpora muchas de las características del modelo en espiral.
- Es evolutivo por naturaleza, y exige un enfoque iterativo para la creación del software.
- Sin embargo, el modelo ensamblador de componentes configura aplicaciones desde componentes preparados de software (algunas veces llamados clases o COTS)



¿En qué consiste la estrategia de ensamblaje de componentes?

- La actividad de la ingeniería comienza con la identificación de clases candidatas. Esto se lleva a cabo examinando los datos que se van a manejar por parte de la aplicación y el algoritmo que se va a aplicar para conseguir el tratamiento. Los datos y los algoritmos correspondientes se empaquetan en una clase.
- Las clases (llamadas componentes) creadas en los proyectos de ingeniería del software anteriores se almacenan en una biblioteca de clases.
- Una vez identificadas las clases candidatas, la biblioteca de clases se examina para determinar si estas clases ya existen.

Ensamblaje de componentes y aplicaciones

- El desarrollo de prototipos con reutilización comprende dos niveles:

(a) El nivel de aplicación, en el que una aplicación completa se integra con el prototipo

  - P.ej., si el prototipo requiere procesamiento de textos, se puede integrar un sistema estándar de procesamiento de textos (MS Office).

(b) El nivel de componente, en el que los componentes se integran en un marco de trabajo estándar.

  - Visual Basic, TCL/TK, Python, Perl...
  - Lenguajes de alto nivel sin tipos, con kits de herramientas gráficas.
  - Desarrollo rápido de aplicaciones pequeñas y relativamente sencillas, construidas por una persona o conjunto de personas.
  - No existe una arquitectura explícita del sistema.
  - CORBA, DCOM, JavaBeans
  - Junto con un marco arquitectónico, es más apropiado para sistemas grandes.

Beneficios

Según estudios de reutilización, se informa que el ensamblaje de componentes lleva a una reducción del 70 por ciento de tiempo de ciclo de desarrollo, un 84 por ciento del coste del proyecto y un índice de productividad del 26,2, comparado con la norma de industria del 16,9.

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

Desarrollo de Software Basado en Componentes

Introducción

La complejidad de los sistemas computacionales actuales nos ha llevado a buscar la reutilización del software existente. El desarrollo de software basado en componentes permite reutilizar piezas de código preelaborado que permiten realizar diversas tareas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y el mayor retorno sobre la inversión. Al comparar la evolución del ambiente de IT con el crecimiento de las metrópolis actuales, podemos entender el origen de muchos problemas que se han presentado históricamente en la construcción de software y vislumbrar las posibles y probables soluciones que nos llevarán hacia la industrialización del software moderno.

Este proceso de industrialización ha dado ya sus inicios con implementaciones como la plataforma .net, la cual impulsa la idea de industrializar el software utilizando tecnologías de componentes. Los avances y mejoras presentados en esta plataforma van mucho más allá de las implementaciones iniciales como COM y CORBA, convirtiendo a los componentes .net en verdaderas piezas de ensamblaje, en un estilo muy similar a las líneas de ensamblaje modernas. Asimismo, los nuevos paradigmas como las Fábricas de Software proveen de los medios para hacer la transición desde el 'hacer a mano' hacia la fabricación o manufactura de software.

Si hay algo que ha aprendido el ser humano desde tiempos muy antiguos es a reutilizar el conocimiento existente para sus cada vez más ambiciosas empresas. En efecto, al reutilizar trozos de experiencias, ideas y artefactos, no solo nos aseguramos de no cometer los mismos errores del pasado, sino que logramos construir cosas cada vez más grandes y maravillosas, con bases firmes y calidad incomparable. Este concepto de la reutilización, uno de los primeros que se nos enseñan a quienes entramos al mundo del desarrollo de software, habremos de utilizarlo desde el mismo instante en que escribamos nuestra primera línea de código.

Los sistemas de hoy en día son cada vez más complejos, deben ser construidos en tiempo récord y deben cumplir con los estándares más altos de calidad. Para hacer frente a esto, se concibió y perfeccionó lo que hoy conocemos como Ingeniería de Software Basada en Componentes (ISBC), la cual se centra en el diseño y construcción de sistemas computacionales que utilizan componentes de software reutilizables. Esta ciencia trabaja bajo la filosofía de "comprar, no construir", una idea que ya es común en casi todas las industrias existentes, pero relativamente nueva en lo que a la construcción de software se refiere.

Para este momento, ya muchos conocen las ventajas de este enfoque de desarrollo y, de la misma forma, muchos se preguntan día a día el por qué son tan pocos los que realmente alcanzan el éxito siguiendo esta filosofía. En realidad, hasta ahora solo hemos tanteado un poco con las posibilidades del software basado en componentes, y es justo hora, en la presente década, que la industria del software despegará y se perfeccionará para estar a la par de cualquier otra industria del medio. Las analogías que nos han llevado a estudiar a los sistemas comparándolos con las complejas metrópolis de la actualidad, así como las iniciativas más innovadoras como las Fábricas de Software de Microsoft, son la clara representación de que estamos a punto de presenciar un nuevo gran cambio en la forma como pensamos en software.

Beneficios del Desarrollo de Software basado en Componentes

En esencia, un componente es una pieza de código preelaborado 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.

El paradigma de ensamblar componentes y escribir código para hacer que estos componentes funcionen se conoce como Desarrollo de Software Basado en Componentes. El uso de este paradigma posee algunas ventajas:

1. **Reutilización del software.** Nos lleva a alcanzar un mayor nivel de reutilización de software.
2. **Simplifica las pruebas.** Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
3. **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.
4. **Mayor calidad.** Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

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

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

Anexo I: Metrópolis: Analogía de la Evolución del Software basado en Componentes

Pat Helland , uno de los arquitectos con mayor experiencia en Microsoft, ha desarrollado recientemente una metáfora llamada **Metrópolis**, en la cual compara la evolución de las tecnologías de la información con la forma como las ciudades de Estados Unidos han evolucionado durante los 2 últimos siglos. Al comprender la evolución de las ciudades actuales, podemos darnos una idea del futuro promisorio que tiene el desarrollo de software basado en componentes.

Como lo menciona Helland, las ciudades y las casas de software comparten un pasado muy similar, pues ambas empezaron en ambientes que se desarrollaron aisladamente. Este desarrollo independiente conllevó a diferencias de cultura con respecto a la forma como se hacían las cosas. A mediados del siglo XIX, las vías de ferrocarril conectaron la mayor parte de las ciudades en los Estados Unidos. Esto permitió que las personas y distintos artículos empezaran a trasladarse de una forma que no era posible anteriormente. Se inició entonces una carrera por asegurar que los artículos trabajen entre sí y cumplan estándares de compatibilidad. Estos cambios en expectativas y capacidades alimentaron la explosión del comercio, la fabricación y llevaron a la urbanización de la vida, tal y como la conocemos hoy.

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 IT 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 (Ver Figura 1). 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 (Ver Figura 2):

Figura 1: Evolución de las ciudades.

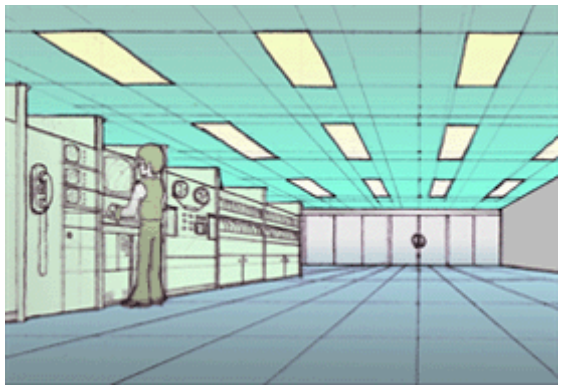


Figura 2: Evolución de las casas de software.

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 (Ver Figura 3). 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 (Ver Figura 4):

Figura 3: Fábricas y edificios.

Figura 4: Aplicaciones.

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.

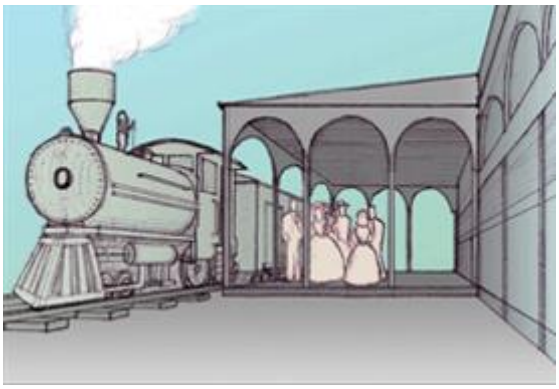


Figura 5: Transporte. Volver al texto.

Figura 6: Comunicaciones. Volver al texto.

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 (Ver Figura 7 y Figura 8):

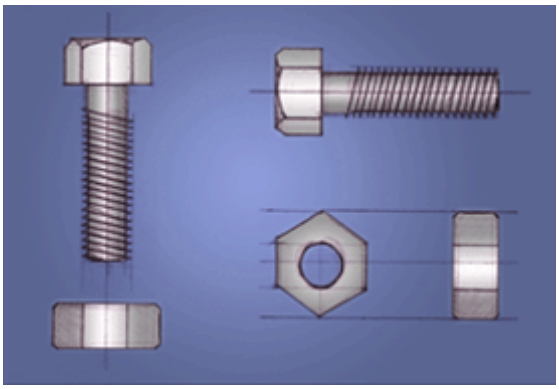


Figura 7: Bienes fabricados. .

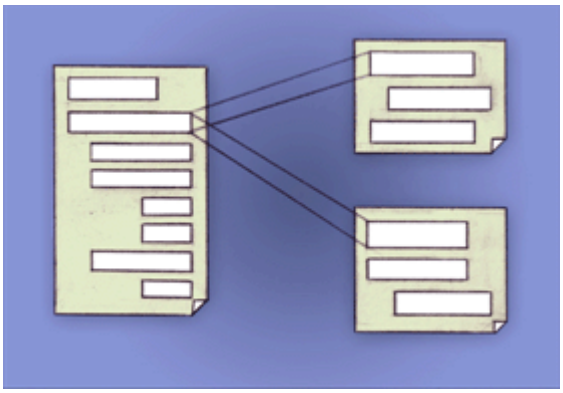


Figura 8: Datos estructurados. .

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 (Ver Figura 9). 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 (Ver Figura 10):

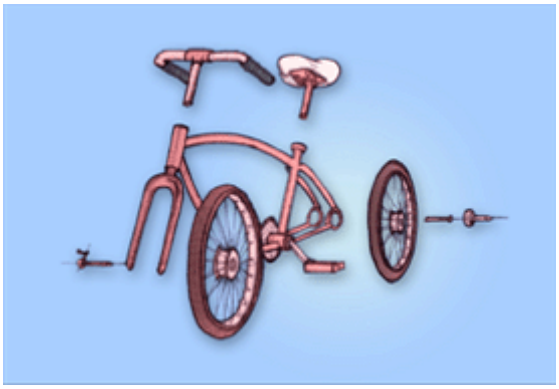


Figura 9: Ensamblados fabricados.

Figura 10: Empresas virtuales.

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 (Ver Figura 11):

Figura 11: Comercialización y distribución.

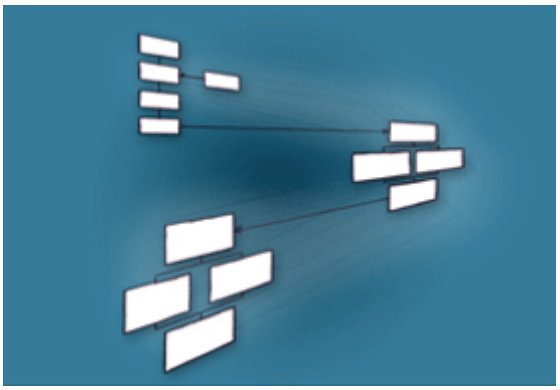


Figura 12: Procesos de negocios.

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 (Ver Figura 12).

Anexo 2: Fábricas de Software, el Nuevo Paradigma

Las Fábricas de Software son una iniciativa propuesta por Microsoft que plasma la necesidad y provee de los medios para hacer la transición desde el 'hacer a mano' hacia la fabricación o manufactura. En sí, una Fábrica de Software es un ambiente de desarrollo configurado para soportar el desarrollo acelerado de un tipo específico de aplicación. Las Fábricas de Software no son más que el siguiente paso lógico en la evolución continua de los métodos y prácticas de desarrollo de software; sin embargo, prometen cambiar el carácter de la industria de software introduciendo patrones de industrialización

¿Es posible automatizar el desarrollo del software? En realidad, ya lo hemos hecho. Los editores WYSIWYG por ejemplo, hacen mucho más fácil el construir y mantener interfaces gráficas de usuario, proveyendo beneficios como independencia de dispositivos y el ensamblaje visual. El diseño de base de datos ofrece formas similares de automatización. 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

Uno de los elementos clave de este patrón es el elevar el nivel de abstracción de los desarrolladores. Actualmente usamos UML para la documentación. Sin embargo, lo que buscamos hoy en día es la productividad. Mientras los estereotipos y los tags pueden ser usados para decorar modelos UML, la experiencia muestra que se necesitan características más precisas del lenguaje para soportar compilación, depuración, pruebas y otros tipos de tareas de desarrollo.

Pero aún cuando los modelos juegan un rol importante, no lo son todo. Para llegar a niveles más altos de productividad necesitamos la habilidad de configurar, adaptar y ensamblar rápidamente componentes desarrollados independientemente, autodescriptivos e independientes de ubicación, para producir así familias de sistemas similares, pero diferentes. Para ello, será necesaria la creación de patrones con dominio específico, frameworks y herramientas que se alineen tanto con la arquitectura del producto como con el ciclo de vida del producto. Más aún, debemos considerar los procesos que usamos para analizar requerimientos, desarrollar software y ponerlo en producción. Necesitamos disponer de las mejores prácticas, contenido reutilizable y distintos tipos de patrones. La aplicación de todo esto en conjunto se conoce como Fábrica de Software.

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:

- **Un esquema de fabricación.** La analogía de esto es una receta. En ella se listan ingredientes, como proyectos, código fuente, directorios, archivos SQL y archivos de configuración, y explica cómo deberían ser combinados para crear el producto. Especifica qué Lenguajes de Dominio Específico (DSL) pueden ser usados y describe cómo los modelos basados en estos DSLs pueden transformarse en código y otros artefactos, o en otros modelos. Describe la arquitectura de la línea de producto, y las relaciones clave entre componentes y frameworks que la componen.
- **Una plantilla de fábrica de software.** Esta es una gran bolsa de supermercado que contiene los ingredientes listados en la receta. Provee los patrones, guías, plantillas, frameworks, ejemplos, herramientas personalizadas como herramientas para edición visual de DSLs, scripts, XSDs, hojas de estilos, y otros ingredientes para construir el producto.
- **Un ambiente de desarrollo extensible.** Un ambiente como Visual Studio Team System es la cocina en la cual la comida es cocinada. Cuando se configura con una plantilla de fábrica de software, Visual Studio Team System se convierte en una fábrica de software para la familia de productos.

Si llevamos más allá la analogía, los productos son como comidas servidas en un restaurante. Los clientes de la fábrica de software son como los clientes que ordenan comidas de un menú. Una especificación de producto es como una orden de comida específica. Los desarrolladores del producto son como cocineros que preparan las comidas descritas en las órdenes, los cuales podrían modificar las definiciones de las comidas, o preparar comidas fuera del menú. Los desarrolladores de la línea de producto son como chefs que deciden qué aparecerá en el menú, y qué ingredientes, procesos, y equipo de cocina se usará para prepararlos.

Un ejemplo más concreto del uso de una fábrica de software sería el diseño de un esquema para la construcción de aplicaciones de clientes livianos para comercio electrónico usando el Microsoft .net Framework, C#, el Microsoft Business Framework, Microsoft SQL Server, Microsoft Biztalk Server y el Microsoft Host Integration Server - una familia amplia pero muy útil de aplicaciones. Podríamos usar este esquema de fábrica de software para configurar Visual Studio Team System para convertirse en una fábrica de software que construya miembros de esta familia.

Para este ejemplo, el esquema de software podría contener un DSL que represente requerimientos configurables, un DSL para describir procesos de negocios, un DSL para describir modelos lógicos y físicos, un DSL para describir la navegación Web, y un DSL para describir entidades de negocio. Podríamos usar las características de extensibilidad de Visual Studio Team System para hostear una plantilla de fábrica de software basada en este esquema. En esa plantilla incluiríamos algunos bloques de aplicación liberados por el grupo de Microsoft Patterns and Practices, políticas de check-in de Team Foundation Server, estructuras de proyecto, la guía de proceso de Microsoft Solution Framework (MSF), entre otros. Con esta fábrica de software, el equipo de desarrollo podría rápidamente lanzar una variedad de aplicaciones de comercio electrónico, cada una conteniendo características únicas basadas en requerimientos únicos de clientes específicos.

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

Conclusión

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.

Bibliografía

- Ingeniería del Software Un Enfoque Práctico. Roger S. Pressman. Sexta Edición, Mc Graw Hill
- Ingeniería de software, Ian Sommerville, 6ª. Edición, Adison Wesley
- http://msdn.microsoft.com/es-es/library/bb972268.aspx por Julio Casal Terreros

Historia de Cambios

Fecha	Versión	Descripción	Autor
11/05/2011	1.0	Versión Inicial	Judith Meles Gerardo Boiero
30/07/2011	1.1	Correcciones de redacción	Judith Meles