COMPARATIVA DE LOS MODELOS DE PROCESO DE SOFTWARE

I. DESCRIPCIÓN DE LOS MODELOS DE PROCESO DE SOFTWARE

EL MODELO LINEAL (O MODELO EN CASCADA).

Es el más antiguo de todos los modelos de Ingeniería del Software. El modelo lineal presenta una estructura secuencial (de ahí el nombre de Modelo en cascada) formada por seis fases o etapas:

− Análisis del Sistema

− Análisis de Requisitos de Software

− Diseño

− Codificación

− Prueba

− Mantenimiento

Las fases incluyen dentro de sí determinadas tareas que clasifican de una forma clara el trabajo a realizar.

El desarrollo de las fases, como he mencionado antes, se produce de manera secuencial. Una vez se produce el análisis tanto del Sistema como de los requisitos del software demandado por el cliente, (fases en las que la intervención del cliente es absolutamente necesaria), se procede a la fase de diseño de la arquitectura global del software. Un diseño elaborado de forma cuidadosa llevará a una rápida codificación. Tras haber traducido el programa a un lenguaje comprensible para el ordenador, se comprueban los elementos de forma individual y más tarde de manera homogénea (todos los sistemas a la vez). Una vez entregado el software al cliente, la fase de Mantenimiento comprenderá las actualizaciones y las correcciones de errores que sean necesarias en el programa.

El Modelo en cascada no permite retroceder (más tarde analizaremos las ventajas e inconvenientes de todos los modelos en común), por lo que se hace estrictamente necesario que al final de cada fase el analista de sistemas o, en su caso, el programador, verifique y valide todo el trabajo realizado, ya que un error no detectado a tiempo podría perjudicar gravemente la fecha de entrega del software a nuestro cliente.

EL MODELO INCREMENTAL.

El modelo incremental es una evolución del modelo de cascada; viene a suplir el problema de no poder retroceder en las fases de desarrollo del software. Es, por tanto, un modelo no secuencial.

El funcionamiento es sencillo. Comienza con el análisis de los requisitos, tras el cual se prepara un primer diseño. La novedad de este modelo respecto del anterior, es la introducción de iteraciones para bifurcar diseños. Es decir, este modelo ofrece la posibilidad de comenzar un diseño, arquitectura, estructura, etc del software, que de no convencer al cliente (o al propio programador) es rechazado y se

comienza con una segunda iteración (o un segundo diseño), sin necesidad de realizar un nuevo análisis de requisitos. Pueden realizarse tantas iteraciones (también llamadas incrementos) como sean necesarias.

EL MODELO DE CONSTRUCCIÓN DE PROTOTIPOS.

Este modelo no secuencial, basado en la construcción de simulaciones o modelos ejecutables de aplicaciones más extensos, persigue un objetivo principal: la participación directa del cliente en la construcción del software requerido. Las fases son similares a las del modelo en cascada: es necesario un análisis previo de los requisitos tanto del sistema como del cliente, se concibe la arquitectura del sistema y se realiza el diseño del software. Sin embargo, se incluye un elemento hasta ahora no utilizado, que consiste en el diseño rápido de un prototipo que se mostrará al cliente para que evalúe el trabajo realizado.

El prototipo es una versión reducida del programa completo; es una fachada virtual que mostramos al cliente (que carece de la posibilidad de ser utilizada de la forma en que lo haríamos con el software final. Tras recoger los requisitos tanto del cliente como del sistema, se comienza con el diseño rápido del prototipo; el diseño completo obedece al previo diseño de pequeños prototipos específicos para funciones individuales. Más tarde, estos diseños serán unidos en uno sólo.

Después, se procede a la construcción del mismo. Éste prototipo es el que mostraremos al cliente para que lo evalúe y considere cambios en él, aunque no se trate de una versión definitiva.

EL MODELO EN ESPIRAL.

Este modelo, también no secuencial, es algo más complejo que los anteriores, aunque incluye un elemento muy útil e importante en el desarrollo del software: análisis de riesgos. El modelo en espiral concreta cuatro fases:

− Planificación

− Análisis de Riesgos

− Ingeniería (Construcción del prototipo)

− Evaluación por el cliente

Si ésta última fase es afirmativa, el modelo continúa con la estructura del Ciclo de vida Clásico. Si el cliente no está satisfecho con el resultado, se cubre otra banda de la espiral y se vuelve a la primera fase (de planificación).

II. VENTAJAS E INCONVENIENTES DE LOS MODELOS

Podemos considerar al modelo en cascada como el más sencillo de utilizar, aunque también podríamos dudar de su eficacia dado el alto número de inconvenientes que presenta, siendo el principal el que se trate de un modelo secuencial; por otra parte, este modelo exige tareas de profundización en el análisis de requisitos del sistema. Si este sistema no es bien conocido, o es difícil de analizar, esta fase puede alargarse demasiado.

Ninguno de los modelos es perfecto; el modelo incremental añade la posibilidad de utilizar iteraciones para doblegar el diseño y contemplar varias posibilidades hasta elegir una. Es un modelo completamente interactivo, que permite trabajar con él en situaciones en las que los cambios de opinión

estén a la orden del día. Cada incremento es un paso más en el desarrollo del software final, lo que nos permite cambiar entre iteraciones (o bien proceder a la entrega del software a nuestro cliente como si se tratara de fascículos semanales).

Esta ventaja es también el principal inconveniente; no en todas las situaciones de desarrollo de software podemos permitirnos la división del trabajo en incrementos, ni tampoco periodificar la entrega de los mismos. Además, aunque la mayoría de las veces el software se puede fragmentar y podemos trabajar con un conjunto de programas determinado, pueden darse situaciones en las que sea imposible ejecutar una iteración sin la existencia de otra que cumpla una función complementaria.

Los prototipos (cambiando de modelo), son una herramienta muy eficaz para imaginar el software completo de una forma rápida y sencilla. De esta forma, incluso observando el prototipo podemos descubrir requerimientos del software en los que antes no habíamos reparado. Mejora también el proceso de introducción de cambios en el desarrollo de los programas. En el modelo incremental podíamos recurrir a las iteraciones, pero resultará más sencillo (y sobre todo, más visual) realizar éstas modificaciones sobre el prototipo en cuestión. Además, ésta operación puede realizarse bajo la supervisión del cliente, lo que hace a éste modelo más interactivo aún que su predecesor. Sin embargo, los prototipos tienen un gran problema en contraposición a sus ventajas: la rapidez con la que se diseñan y construyen pueden llevar a errores que no sean detectados en la fase de prueba y acaben integrándose en el producto final. Además, un prototipo es una representación casi exacta del programa final, carente de contenido real. Pero esto es algo que el cliente desconoce; si tiene prisa, puede creer que nuestro trabajo está mucho más avanzado de lo que creía (a pesar de que el prototipo sea tan sólo la fachada de un edificio sin paredes ni escaleras) y puede optar por adelantar la fecha de entrega; al final, el pobre programador es el que paga las consecuencias haciendo horas extras y, además, si se acelera demasiado la construcción del sistema final volvemos al problema de la inclusión de errores no detectados a tiempo.

Por raro que sea, o difícil de entender, el modelo en espiral parece entender los problemas de los anteriores e intentar subsanarlos. Si en el modelo anterior utilizábamos prototipos para hacernos una idea del software final, en éste modelo los utilizaremos también para hacernos una idea detallada de cuáles son los errores que tiene, o podría tener el programa durante su funcionamiento (lo que antes llamábamos análisis de riesgos). Esta manera de enfocar el diseño del software permite al cliente evaluar los factores de riesgo que le comunica el prototipo de análisis de riesgo, y según esta información tomar una decisión u otra. Esto hace que el modelo en espiral sea todavía más interactivo que los anteriores.

En cada fase se evalúa el trabajo terminado y, si nos dan el visto bueno, continuamos girando en la espiral hasta que llegamos a la evaluación del cliente, la cual decidirá si continuamos en el modelo clásico o volvemos a la primera fase del modelo en espiral. Sin embargo, todo éste análisis de riesgos (que tan útil parece ser) no parece fácil de utilizar; un análisis de riegos detallado utilizado sin experiencia podría sobre valorar (o subestimar) los errores que se presenten, haciendo imposible en paso a la siguiente fase (y entonces sí que nos meteríamos en una verdadera espiral sin fin, cosa que al cliente no debe hacerle mucha gracia). Éste problema genera otro adicional, y es que viendo estas situaciones, será difícil convencer al cliente para que acepte un proyecto realizado bajo las condiciones de éste modelo.

III. ¿DEBEMOS UTILIZAR UNA METODOLOGÍA DE DESARROLLO?

Es obvio; imaginemos que el analista de sistemas está desarrollando un plan de requisitos del software y el programador hace una simple imagen mental del programa y comienza a codificar la información recibida. Sin una arquitectura previa, es difícil hacer una casa. Más difícil debe ser hacer un programa o un conjunto de ellos. Utilizar una metodología de desarrollo puede conllevar algunos inconvenientes,

pero éstos son sencillos de subsanar. Además, establecer el trabajo en fases distribuye el desarrollo de una forma ordenada, lo que hace que cada uno se ocupe de su trabajo y no de aquel por el cual no le pagan. El orden escalonado, la posibilidad de retroceder (excepto en el modelo en cascada) en nuestro diseño o codificación, la interactividad con el cliente, la presentación de proyectos preliminares (prototipos) y las exigentes fases de prueba y mantenimiento, vienen a ser las principales ventajas de la metodología de desarrollo del software.

IV. TIPOS DE PROYECTOS PARA CADA MODELO

Es difícil hacer una discriminación exacta de para qué tipo de proyecto sirve cada modelo. Sin embargo, es más fácil decir para que tipo de proyectos o desarrollos NO sirve un modelo.

Por ejemplo; supongamos que realizamos un proyecto de sistemas informáticos de gestión para una empresa que acaba de establecerse en el Nuevo Mercado de Valores. Esta empresa se dedica a invertir el dinero de sus clientes en bolsa, valiéndose para ello de prestigiosos inversores financieros. Pues bien, ésta empresa sabe que acaba de introducir su actividad en uno de los negocios más inestables del mundo, como también inestables serán las exigencias de sus clientes y del mercado, las leyes que cambian casi una vez a la semana, las caídas repentinas de mercados extranjeros en los que la empresa tiene invertida gran parte de su capital, etc. Evidentemente, el uso del modelo en espiral, o el de cascada para informatizar la actividad de la empresa será como tratar de hablar en binario. Pasar de un análisis preliminar a un diseño de la arquitectura del sistema llevaría consigo la condición de haber revisado y validado este análisis, pero hay una gran posibilidad de que a los dos o tres días la empresa necesite hacer una corrección de sus requisitos. Jamás llegaríamos a la cuarta fase (en el caso del modelo en espiral) o simplemente nunca daríamos con el resultado necesario para la empresa. Ésta actividad es, sin embargo, más concordante con el modelo incremental. Podemos entregar piezas del software que sean necesarias para cubrir una actividad relativamente estable, mientras que continuamos perfeccionando el resto de iteraciones. Si se producen modificaciones en los requisitos, procedemos analizarlos y catalogarlos y a diseñar una nueva iteración que será incluida en el producto final una vez esté terminada.

Si un cliente nos pide un paquete de software de control de la asignación de números de tarjetas de crédito a titulares de cuentas bancarias, protección de códigos de seguridad, etc, una de nuestras principales prioridades será la de detectar el más mínimo fallo en la elaboración del software, porque cualquier rendija acaba por convertirse en una autopista para los amigos de lo ajeno. La creación de un prototipo de control de errores (análisis de riesgo) exigente puede facilitar y acelerar esta tarea. Estamos hablando del modelo en espiral.

Hemos dicho que el uso de prototipos genera una forma de trabajar mucho más visual que todo lo visto anteriormente. El uso de este modelo podría responder a un cliente obsesionado con la presentación de su software, calidad visual que lo haga mucho más fácil de utilizar (tal y como si se tratara de un sistema operativo adaptado a su empresa). Las empresas dedicadas a la compraventa de productos de primera necesidad (como pueden ser supermercados o tiendas de ropa) no necesitan complejos sistemas informáticos que lleven la contabilidad, o que visualicen de forma rápida en pantalla la relación unidades, precio unidad, total. Sin embargo, una empresa de telefonía móvil exigirá paquetes de software avanzados, concretos y , sobre todo, lo más visuales posibles. La creación de prototipos ayuda a que nuestro cliente imagine su propio software tal y como si él estuviera elaborándolo.

Por último, y para terminar, que ya está bien, el modelo en cascada obedece a las necesidades de un desarrollo de software sencillo, corto, y sin posibles obstáculos que detengan su diseño o construcción. No aconsejaría a nadie el uso de este modelo en empresas que necesiten un paquete de software compuesto de muchas piezas, porque no es precisamente el mejor modelo para desarrollar interactividad, ni tampoco para crear estructuras de software complejas.

V. EVALUACIÓN PERSONAL DE LOS MODELOS

Me quedo con los prototipos. La espiral es demasiado compleja; además, con el modelo de prototipos podemos también hacer una evaluación de riesgos (aunque un poco más sencilla) antes de ponernos a estructurar y codificar el sistema completo. El modelo incremental es bueno, pero las iteraciones pueden llevar un poco a la confusión (podemos incluir en el diseño de una iteración lo que ya estaba diseñado en otra). Sin duda, el Ciclo Clásico es el peor. Una metodología de desarrollo secuencial es muy difícil de llevar y presenta muchos problemas añadidos a los que ya nos encontremos.