Ciclo de Vida del Software

Un modelo de ciclo de vida define el estado de las fases a través de las cuales se mueve un proyecto de desarrollo de software.

El primer ciclo de vida del software, "Cascada", fue definido por Winston Royce a fines del 70. Desde entonces muchos equipos de desarrollo han seguido este modelo. Sin embargo, ya desde 10 a 15 años atrás, el modelo cascada ha sido sujeto a numerosas críticas, debido a que es restrictivo y rígido, lo cual dificulta el desarrollo de proyectos de software moderno. En su lugar, muchos modelos nuevos de ciclo de vida han sido propuestos, incluyendo modelos que pretenden desarrollar software más rápidamente, o más incrementalmente o de una forma más evolutiva, o precediendo el desarrollo a escala total con algún conjunto de prototipos rápidos.

Definición de un Modelo de Ciclo de Vida

Un modelo de ciclo de vida de software es una vista de las actividades que ocurren durante el desarrollo de software, intenta determinar el orden de las etapas involucradas y los criterios de transición asociadas entre estas etapas.

Un modelo de ciclo de vida del software:

- Describe las fases principales de desarrollo de software.
- Define las fases primarias esperadas de ser ejecutadas durante esas fases.
- Ayuda a administrar el progreso del desarrollo, y
- Provee un espacio de trabajo para la definición de un detallado proceso de desarrollo de software.

Así, los modelos por una parte suministran una guía para los ingenieros de software con el fin de ordenar las diversas actividades técnicas en el proyecto, por otra parte suministran un marco para la administración del desarrollo y el mantenimiento, en el sentido en que permiten estimar recursos, definir puntos de control intermedios, monitorear el avance, etc.

Alternativas de Modelos de Ciclo de Vida

Modelo Cascada

Este es el más básico de todos los modelos, y sirve como bloque de construcción para los demás modelos de ciclo de vida. La visión del modelo cascada del desarrollo de software es muy simple; dice que el desarrollo de software puede ser a través de una secuencia simple de fases. Cada fase tiene un conjunto de metas bien definidas, y las actividades dentro de una fase contribuye a la satisfacción de metas de esa fase o quizás a una subsecuencia de metas de la fase. Las flechas muestran el flujo de información entre las fases. La flecha de avance muestra el flujo normal. Las flechas hacia atrás representan la retroalimentación.

El modelo de ciclo de vida cascada, captura algunos principios básicos:

- Planear un proyecto antes de embarcarse en él.
- Definir el comportamiento externo deseado del sistema antes de diseñar su arquitectura interna.
- Documentar los resultados de cada actividad.
- Diseñar un sistema antes de codificarlo.
- Testear un sistema después de construirlo.

Una de las contribuciones más importantes del modelo cascada es para los administradores, posibilitándoles avanzar en el desarrollo, aunque en una escala muy bruta.

Modelo De Desarrollo Incremental

Los riesgos asociados con el desarrollo de sistemas largos y complejos son enormes. Una forma de reducir los riesgos es construir sólo una parte del sistema, reservando otros aspectos para niveles posteriores. El desarrollo incremental es el proceso de construcción siempre incrementando subconjuntos de requerimientos del sistema. Típicamente, un documento de requerimientos es escrito al capturar todos los requerimientos para el sistema completo.

Note que el desarrollo incremental es 100% compatible con el modelo cascada. El desarrollo incremental no demanda una forma específica de observar el desarrollo de algún otro incremento. Así, el modelo cascada puede ser usado para administrar cada esfuerzo de desarrollo, como se muestra en la figura.

El modelo de desarrollo incremental provee algunos beneficios significativos para los proyectos:

- Construir un sistema pequeño es siempre menos riesgoso que construir un sistema grande.
- Al ir desarrollando parte de las funcionalidades, es más fácil determinar si los requerimientos planeados para los niveles subsiguientes son correctos.
- Si un error importante es realizado, sólo la última iteración necesita ser descartada.
- Reduciendo el tiempo de desarrollo de un sistema (en este caso en incremento del sistema) decrecen las probabilidades que esos requerimientos de usuarios puedan cambiar durante el desarrollo.
- Si un error importante es realizado, el incremento previo puede ser usado.
- Los errores de desarrollo realizados en un incremento, pueden ser arreglados antes del comienzo del próximo incremento.

Modelo De Desarrollo Evolutivo

Como el modelo de desarrollo incremental, el modelo de desarrollo evolutivo (algunas veces denominado como prototipado evolutivo) construye una serie de grandes versiones sucesivas de un producto. Sin embargo, mientras que la aproximación incremental presupone que el conjunto completo de requerimientos es conocido al comenzar, el modelo evolutivo asume que los requerimientos no son completamente conocidos al inicio del proyecto.

En el modelo evolutivo, los requerimientos son cuidadosamente examinados, y sólo esos que son bien comprendidos son seleccionados para el primer incremento. Los desarrolladores construyen una implementación parcial del sistema que recibe sólo estos requerimientos.

El sistema es entonces desarrollado, los usuarios lo usan, y proveen retroalimentación a los desarrolladores. Basada en esta retroalimentación, la especificación de requerimientos es actualizada, y una segunda versión del producto es desarrollada y desplegada. El proceso se repite indefinidamente.

Note que el desarrollo evolutivo es 100% compatible con el modelo cascada. El desarrollo evolutivo no demanda una forma específica de observar el desarrollo de algún incremento. Así, el modelo cascada puede ser usado para administrar cada esfuerzo de desarrollo. Obviamente, el desarrollo incremental y evolutivo puede ser combinado también.

Todo lo que uno tiene que hacer es construir un subconjunto de requerimientos conocidos (incremental), y comprender al principio que muchos nuevos requerimientos es probable que aparezcan cuando el sistema sea desplegado o desarrollado.

El desarrollo de software en forma evolutiva requiere un especial cuidado en la manipulación de documentos, programas, datos de test, etc. desarrollados para distintas versiones del software. Cada paso debe ser registrado, la documentación debe ser recuperada con facilidad, los cambios deben ser efectuados de una manera controlada.

Modelo de Prototipado de Requerimientos.−

El prototipado de requerimientos es la creación de una implementación parcial de un sistema, para el propósito explícito de aprender sobre los requerimientos del sistema. Un prototipo es construido de una manera rápida tal como sea posible. Esto es dado a los usuarios, clientes o representantes de ellos, posibilitando que ellos experimenten con el prototipo. Estos individuos luego proveen la retroalimentación sobre lo que a ellos les gustó y no les gustó acerca del prototipo proporcionado, quienes capturan en la documentación actual de la especificación de requerimientos la información entregada por los usuarios para el desarrollo del sistema real. El prototipado puede ser usado como parte de la fase de requerimientos (determinar requerimientos) o justo antes de la fase de requerimientos (como predecesor de requerimientos). En otro caso, el prototipado puede servir su papel inmediatamente antes de algún o todo el desarrollo incremental en modelos incremental o evolutivo.

El Prototipado ha sido usado frecuentemente en los 90, porque la especificación de requerimientos para sistemas complejos tienden a ser relativamente dificultoso de cursar. Muchos usuarios y clientes encuentran que es mucho más fácil proveer retroalimentación convenientemente basado en la manipulación, desde un prototipo, en vez de leer una especificación de requerimientos potencialmente ambigua y extensa.

Diferente del modelo evolutivo donde los requerimientos mejor entendidos están incorporados, un prototipo generalmente se construye con los requerimientos entendidos más pobremente.

En caso que ustedes construyan requerimientos bien entendidos, el cliente podría responder con "sí, así es", y nada podría ser aprendido de la experiencia.

Modelo Espiral

El modelo espiral de los procesos software es un modelo del ciclo de *meta−vida*. En este modelo, el esfuerzo de desarrollo es iterativo. Tan pronto como uno completa un esfuerzo de desarrollo, otro comienza. Además, en cada desarrollo ejecutado, puedes seguir estos cuatros pasos:

- Determinar qué quieres lograr.
- Determinar las rutas alternativas que puedes tomar para lograr estas metas. Por cada una, analizar los riesgos y resultados finales, y seleccionar la mejor.
- Seguir la alternativa seleccionada en el paso 2.
- Establecer qué tienes terminado.

La dimensión radial en la figura refleja costos acumulativos incurridos en el proyecto.

Observemos un escenario particular. Digamos que en este proyecto, nosotros viajaremos a resolver un conjunto particular de problemas del cliente. Durante el primer viaje alrededor de la espiral, analizamos la situación y determinamos que los mayores riesgos son la interfaz del usuario. Después de un cuidadoso análisis de las formas alternativas de direccionar esto (por ejemplo, construir un sistema y esperar lo mejor, escribir una especificación de requerimientos y esperar que el cliente lo entienda, y construir un prototipo), determinamos que el mejor curso de acción es construir un prototipo.

Lo realizamos. Luego proveemos el prototipo al cliente quien nos provee con retroalimentación útil. Ahora, comenzamos el segundo viaje alrededor de la espiral. Este tiempo decidimos que el mayor riesgo es ese miedo a que muchos nuevos requerimientos comiencen a aparecer sólo después de que el sistema sea desplegado. Analicemos las rutas alternativas, y decidimos que la mejor aproximación es construir un incremento del sistema que satisfaga sólo los requerimientos mejor entendidos. Hagámoslo ya. Después del despliegue, el cliente nos provee de retroalimentación que dirá si estamos correctos con esos requerimientos, pero 50 nuevos requerimientos ahora se originarán en las cabezas de los clientes. Y el tercer viaje alrededor de la espiral comienza.

El modelo espiral captura algunos principios básicos:

- Decidir qué problema se quiere resolver antes de viajar a resolverlo.
- Examinar tus múltiples alternativas de acción y elegir una de las más convenientes.
- Evaluar qué tienes hecho y qué tienes que haber aprendido después de hacer algo.
- No ser tan ingenuo para pensar que el sistema que estás construyendo será "EL" sistema que el cliente necesita, y
- Conocer (comprender) los niveles de riesgo, que tendrás que tolerar.

El modelo espiral no es una alternativa del modelo cascada, ellos son completamente compatible.

Modelo Concurrente

Como el modelo espiral, el modelo concurrente provee una meta−descripción del proceso software. Mientras que la contribución primaria del modelo espiral es en realidad que esas actividades del software ocurran repetidamente, la contribución del modelo concurrente es su capacidad de describir las múltiples actividades del software ocurriendo simultáneamente.

Esto no sorprende a nadie que ha estado involucrado con las diversas actividades que ocurren en algún tiempo del proceso de desarrollo de software. Discutamos un poco tales casos:

Los requerimientos son usualmente "líneas de base", cuando una mayoría de los requerimientos comienzan a ser bien entendidos, en este tiempo se dedica un esfuerzo considerable al diseño. Sin embargo, una vez que comienza el diseño, cambios a los requerimientos son comunes y frecuentes (después de todo, los problemas reales cambian, y nuestro entendimiento de los problemas desarrollados también). Es desaconsejado detener el diseño en este camino cuando los requerimientos cambian; en su lugar, existe una necesidad de modificar y rehacer líneas de base de los requerimientos mientras progresa el diseño. Por supuesto, dependiendo del impacto de los cambios de los requerimientos el diseño puede no ser afectado, medianamente afectado o se requerirá comenzar todo de nuevo.

Durante el diseño de arquitectura, es posible que algunos componentes comiencen a ser bien definidos antes que la arquitectura completa sea estabilizada. En tales casos, puede ser posible comenzar el diseño detallado en esos componentes estables. Similarmente, durante el diseño detallado, puede ser posible proceder con la codificación y quizás regular testeando en forma unitaria o realizando testeo de integración previo a llevar a cabo el diseño detallado de todos los componentes.

En algunos proyectos, múltiples etapas de un producto se han desarrollado concurrentemente. Por ejemplo, no es inusual estar haciendo mantención de la etapa 1 de un producto, y al mismo tiempo estar haciendo mantención sobre un componente 2, mientras que se está haciendo codificación sobre un componente 3, mientras se realiza diseño sobre una etapa 4, y especificación de requisitos sobre un componente 5.

En todos estos casos, diversas actividades están ocurriendo simultáneamente. Eligiendo seguir un proyecto usando técnicas de modelación concurrente, se posibilita el conocimiento del estado verdadero en el que se

encuentra el proyecto.