Unidad II: Metodologías de Desarrollo

2.1 Metodologías clásicas

2.1.1 Cascada

El modelo de cascada original se desarrolló entre las décadas de los años 60 y 706 y se define como una secuencia de actividades, donde la estrategia principal es seguir el progreso del desarrollo de software hacia puntos de revisión bien definidos (en inglés, milestones o checkpoints) mediante entregas programadas con fechas precisas (en inglés, schedule). El modelo original planteaba que cada actividad debía completarse antes de poder continuar con la siguiente actividad. Sin embargo, en una revisión posterior se extendió el modelo, permitiendo regresar a actividades anteriores.

Algunas de las creencias del modelo de cascada son que las metas se logran mejor cuando se tienen puntos de revisión bien preestablecidos y documentados (dividiendo el desarrollo del software en actividades secuenciales bien definidas), los documentos técnicos son comprensibles para usuarios y administradores no técnicos, cada detalle de los requisitos se conoce de antemano antes de desarrollar el software (dichos detalles son estables durante el desarrollo) y las pruebas y evaluaciones se realizan de manera eficiente al final del desarrollo.

El modelo de cascada fue inicialmente bien recibido, dado que las actividades de las etapas eran razonables y lógicas. Lamentablemente, el modelo no explicaba cómo modificar un resultado, en especial considerando lo difícil que es definir todos los requisitos de un sistema desde el inicio y que éstos se mantengan estables y sin cambios durante el desarrollo.

Además, toma demasiado tiempo en obtener resultados, retrasando la detección de errores hasta el final. El modelo también hace difícil rastrear (en otras palabras, ver la dependencia entre los requisitos iniciales y el código final). Esta rigidez trajo dudas sobre la utilidad del modelo, resultando en que éste dejase de utilizarse de

acuerdo con su definición original, llevando a los desarrolladores a utilizar variantes del modelo básico, incluyendo el uso de prototipos y la reutilización de software.

El modelo de cascada llamado también lineal secuencial. Proporciona una simple visión del desarrollo del software. A los procesos los representa como fases separadas y secuenciales en tiempo.

Antes de codificar debemos diseñar el software, además probarlo antes de construirlo y ponerlo en operación.

2.1.2 Incremental

El modelo incremental consiste de un desarrollo inicial de la arquitectura completa del sistema, seguida de incrementos y versiones parciales de éste.9 Cada incremento tiene su propio ciclo de vida, típicamente siguiendo el modelo de cascada. Los incrementos pueden construirse de manera serial o paralela dependiendo de la naturaleza de la dependencia entre versiones y recursos. Cada incremento agrega funcionalidad adicional o mejorada sobre el sistema. Conforme se completa cada etapa, se verifica e integra la última versión con las demás versiones ya completadas del sistema.

Durante cada incremento, el sistema se evalúa con respecto al desarrollo de versiones futuras. Las actividades se dividen en procesos y subprocesos, dando lugar al término fábrica de software (en inglés, software factory). Para que la secuencia de desarrollo sea exitosa, es esencial definir etapas que norequieran cambiar los resultados anteriores al agregar nuevas. Por lo tanto, es importante comprender al inicio los requisitos completos del sistema, algo que normalmente es muy difícil de lograr. El desarrollo incremental evita la teoría del “big bang” para el desarrollo de software, donde una gran explosión de desarrollo se transforma repentinamente en el sistema final.

Algunas de las creencias del modelo de cascada son que la administración de proyectos es más fácil de lograr en incrementos más pequeños, es más fácil comprender y probar incrementos de funcionalidad más pequeños, la funcionalidad inicial se desarrolla más temprano (logrando resultados de inversión en menor tiempo) y hay más probabilidad de satisfacer el cambio en los requisitos de usuario mediante incrementos del software en el tiempo que si fueran planeados todos a la vez en un mismo periodo.

2.1.3 Evolutivo

El modelo evolutivo es una extensión al modelo incremental en la que los incrementos se hacen de manera secuencial, en lugar de en paralelo.10 Desde el punto de vista del cliente, el sistema evoluciona según vayan siendo entregados los incrementos. Desde el punto de vista del desarrollador, los requerimientos que son claros al principio del proyecto dictan el incremento inicial, mientras que los incrementos para cada uno de los siguientes ciclos de desarrollo se clarifican a través de la experiencia de los incrementos anteriores.

Este modelo considera que el desarrollo de sistemas es un proceso de cambios progresivos mediante deltas (cambios) de especificación de requerimientos. El modelo evolutivo es también conocido como desarrollo rápido de aplicaciones (en inglés, RAD—rapid application development), que se basa tradicionalmente en el uso de prototipos (en inglés, rapid prototyping).

Un prototipo de software se considera como un medio para especificar los requisitos y un enlace de comunicación entre el usuario final y el diseñador, ayudando a reducir el riesgo de carecer de requerimientos iniciales completos y estables.

Algunas de las creencias del modelo de cascada son la entrega temprana de parte del sistema, aunque no estén completos todos los requerimientos, ya que esto permite utilizarlo como herramienta para la generación de requerimientos faltantes, obteniendo beneficios para el sistema mediante entregas iniciales mientras las entregas posteriores se encuentran en desarrollo.

2.1.4 Espiral

El modelo espiral, desarrollado durante la década de los años 80, es una extensión del modelo de cascada. A diferencia del modelo de cascada, que es dirigido por documentos, el modelo espiral se basa en una estrategia para reducir el riesgo del proyecto en áreas de incertidumbre, como tener requerimientos iniciales incompletos e inestables. El modelo enfatiza ciclos de trabajo, cada uno de los cuales estudia el riesgo antes de proceder al siguiente ciclo. Cada ciclo comienza con la identificación de los objetivos, soluciones alternas y restricciones asociadas con cada alternativa, y finalmente se procede a su evaluación. Cuando se encuentra que existe cierta incertidumbre, se utilizan diversas técnicas para reducir el riesgo de las distintas alternativas.

Cada ciclo del modelo espiral termina con una revisión que discute los logros actuales y los planes para el siguiente ciclo. Al igual que el modelo evolutivo, el modelo espiral incorpora una estrategia de uso de prototipos como parte del manejo del riesgo.

Las creencias del modelo de espiral son que una actividad comienza cuando se entienden los objetivos y riesgos involucrados, se usan las herramientas que mejor reduzcan los riesgos basándose en la evaluación de soluciones alternas, todo el personal relacionado debe involucrarse en una revisión que determine cada

actividad (planeando y comprometiéndose con las siguientes actividades) y el desarrollo se incrementa en cada etapa, permitiendo prototipos sucesivos del producto. Con algunas variantes, éste es el modelo de proceso más importante en la actualidad.

2.1.6 Prototipos

Un prototipo es una versión preliminar, intencionalmente incompleta o reducida de un sistema. El uso de prototipos es una estrategia que puede aplicarse en casi todas las actividades del proceso de software. El propósito de los prototipos es obtener de manera rápida la información necesaria como ayuda en la toma de decisiones.

Los siguientes son algunos tipos de prototipo:

Los prototipos de requisitos permiten que los usuarios perciban la funcionalidad del producto final a través del diseño de interfaces o pantallas del sistema. El objetivo es ayudar a aclarar los requisitos y solicitar nuevas ideas.

Los prototipos de análisis permiten generar rápidamente una arquitectura general que considere las características principales del sistema de acuerdo con la especificación de requisitos.

Los prototipos de diseño permiten explorar y comprender la arquitectura particular de un sistema para poder evaluar aspectos como cuellos de botella (rendimiento y uso de memoria) o incoherencias en el diseño.

Los prototipos verticales permiten comprender parte de un problema y desarrollar su solución completa. Esto se hace generalmente cuando los conceptos básicos no están bien comprendidos (por ejemplo, el seguimiento de cierta metodología).

Los prototipos de factibilidad permiten demostrar si es posible lograr ciertos objetivos del proyecto (por ejemplo, aplicar una arquitectura particular, conectarse a una base de datos bajo ciertas restricciones de rendimiento, aprender a programar en cierto lenguaje en un tiempo determinado o predecir los costos de desarrollo de un proyecto). Un prototipo no es necesariamente un producto de calidad, que deba mantenerse a largo plazo. Por el contrario, los prototipos a menudo son creados y probados rápidamente, para luego ser descartados. Sin embargo, es común que, por presiones de tiempo, se trate de enviar un prototipo al mercado (venderlo) como si éste fuera el producto final.

En general, siempre existirá un conflicto entre un desarrollo rápido y un producto de calidad. Las siguientes dos listas muestran algunas consideraciones para el éxito o fracaso de los prototipos.

Los prototipos tienen éxito cuando se comprende su propósito y se usan de manera adecuada, se comprende la tecnología que va a utilizarse y su relación con el proceso de prototipos, se integra un grupo técnico apropiado para hacer el prototipo (líder de proyecto, documentador, elaborador de prototipos de requisitos y análisis, y elaborador de prototipos de diseño), se evalúa al grupo y las entregas finales, se involucra temprano en el proceso de software a los usuarios finales, se está dispuesto a repetir el proceso de prototipos para comprender mejor la arquitectura básica, se establecen criterios de evaluación apropiados al comienzo de cada etapa de prototipos y se basa uno firmemente en estos criterios para su terminación y/o se construyen prototipos basados en una biblioteca de código reutilizable, controlada por el bibliotecario asignado.

Los prototipos fallan cuando no se comprende qué es un prototipo y cómo debe usarse, no se comprende el proceso lo suficientemente bien como para organizar al grupo de manera adecuada, no se sabe hasta cuándo dejar de evolucionar el prototipo y comenzar de cero (así extendiendo demasiado el proceso), no se sabe hasta cuándo continuar para tratar de lograr los criterios de evaluación deseados (así terminando prematuramente el proceso), no se utilizan ambientes o

herramientas de apoyo adecuados para el desarrollo particular, se cree que un prototipo razonable es un producto aceptable y/o los prototipos nunca terminan.

2.1.6 Desarrollo basado en componentes

Ingeniería de Software Basada en Componentes (ISBC)

Tradicionalmente los ingenieros del software han seguido un enfoque de desarrollo descendente (top-Down) basado en el ciclo de vida en cascada (análisis-diseño implementación) para la construcción de sus sistemas, donde se establecen los requisitos y la arquitectura de la aplicación y luego se va desarrollando, diseñando e implementando cada parte software hasta obtener la aplicación completa implementada.

La ISBC parte de la idea de la integración de componentes software ya existente

(Desarrollo ascendente o bottom-up). Las tecnologías de objetos proporcionan el marco de trabajo técnico, para la ingeniería de software, para un modelo de proceso basado en componentes. El paradigma orientado a objetos enfatiza la creación de clases que encapsulan tanto los datos como los algoritmos para manejar esos datos. Si se diseñan y se implementan adecuadamente, las clases

Orientadas a objetos son reutilizables por diferentes aplicaciones.

Es la reutilización la que permite a los desarrolladores construir aplicaciones sin partir desde cero, sino acercándose más al modelo de construcción de otras ingenierías, donde los productos se construyen en base al ensamblaje y adaptación de distintos componentes desarrollados por terceros (como lo es la industria del hardware para computadoras). Por ello requiere un conjunto de

estándares, guías, procesos y prácticas, para que se la considere como una ingeniería tal, y como una sub-disciplina de la Ingeniería de Software.

Según Pressman, La metodología que propone entonces la “Ingeniería de Software Basada en Componentes” (ISBC) (Figura 1) incorpora muchas de las características del Modelo en Espiral. Es evolutivo2 por naturaleza, y por ello exige también un enfoque iterativo para la creación del software.