Técnico en

<DESARROLLO DE SOFTWARE>

Análisis de Requisitos del Sistema y del Software

Análisis de Requisitos del Sistema y del Software

Unidad III

Administración de requerimientos y modelos de ciclos de vida.

1. Administración de requerimientos

La ingeniería de software contiene tres elementos básicos: *i)* la *metodología* (o conjunto de *métodos*) los cuales establecen como construir el software. *ii)* Los *paradigmas o modelos* que definen la secuencia en la que se aplican los métodos y *iii)* Las *herramientas* utilizadas para dar soporte a los métodos.

Los *métodos* de la ingeniería de software abarcan las siguientes tareas: Planeación y estimación del proyecto, Recolección de los requerimientos, Análisis de los requerimientos del problema, Diseño de las estructuras de datos, arquitectura de los programas y procedimientos algorítmicos, Codificación, Prueba, Implantación, y Mantenimiento. Un *modelo* es una secuencia de pasos a seguir para alcanzar el final de un proyecto. Al modelo o proceso de desarrollo de software se le conoce como *ciclo de vida del software*, porque describe la vida de un producto de software desde su concepción hasta su implantación, entrega, utilización y mantenimiento. Se dice que los

procesos son importantes porque imponen consistencia y estructura sobre un conjunto de actividades. Estas características son útiles cuando se sabe cómo hacer algo bien y se desea asegurar que otros lo hagan de la misma manera. Un proceso es más que un procedimiento. Un procedimiento es como una receta: una manera estructurada de combinar herramientas y técnicas para generar un producto. Sin embargo, un proceso es un conjunto de procedimientos organizado de tal modo que los productos se construyen para satisfacer un conjunto de metas o estándares. El proceso puede sugerir que se seleccione entre varios procedimientos, con tal de que se cumpla con la meta propuesta. Cuando se desarrolla software a gran escala, el ingeniero asume alguno de los roles del proceso de desarrollo (ver figura 3.1), sin embargo, a pequeña escala, el ingeniero asume cada uno de los roles, conforme se va necesitando.

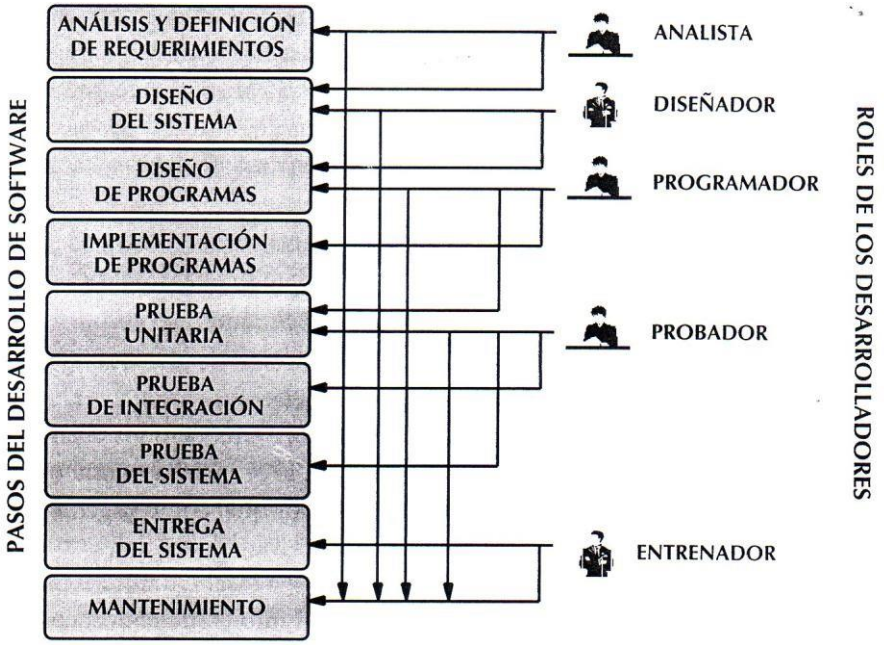

Figura 1: Roles de los desarrolladores de software (Pleeger 2002)

Un **proceso de desarrollo de software** es el conjunto estructurado de las actividades requeridas para elaborar un sistema de software, estas actividades son: especificación de requerimientos, diseño, codificación, validación (pruebas) y mantenimiento. Al proceso de desarrollo de software también se le conoce como ciclo de vida del software porque describe la vida de un producto de software; primero nace con la especificación de los requerimientos, luego se lleva a cabo su implantación, que consiste en su diseño, codificación y pruebas, posteriormente el producto se entrega, y sigue viviendo durante su utilización y mantenimiento. Cuando el producto evoluciona se le hacen modificaciones que generan nuevas versiones. La vida del sistema de software termina cuando éste se deja de utilizar.

Por otra parte, un **modelo de desarrollo de software** es una representación abstracta de este proceso. Al modelo de desarrollo también se le llama paradigma del proceso. Se clasifican todos los procesos de desarrollo de software en tres modelos o paradigmas generales que no son descripciones definitivas de los procesos del software sino más bien, son abstracciones de los modelos que se pueden utilizar para desarrollar software:

- *El modelo en cascada*. Representa a las actividades fundamentales del proceso de desarrollo de software como fases separadas y consecutivas. Estas actividades son: especificación, implantación (diseño, codificación, validación) y mantenimiento.
- *Modelo evolutivo*. Entrelaza las actividades de especificación, desarrollo y validación. Un sistema inicial se desarrolla rápidamente a partir de especificaciones abstractas. Éste se refina basándose en las peticiones del cliente para producir un sistema que satisfaga sus necesidades.

- *Modelo de componentes reutilizables*. Se basa en la existencia de un número significativo de componentes reutilizables. El proceso de desarrollo del sistema se enfoca en integrar estos componentes en el sistema en lugar de desarrollarlos desde cero.

Estos tres paradigmas o modelos de procesos genéricos se utilizan ampliamente en la práctica actual de la ingeniería del software. No se excluyen mutuamente y a menudo se utilizan juntos, especialmente para el desarrollo de sistemas grandes. Independientemente del modelo que se elija, siempre se presentará un reto fundamental: *el análisis de los requerimientos y la elaboración de la Especificación del sistema de software a desarrollar*.

2. El modelo en cascada.

El modelo en cascada (*figura 3.2*), presenta una visión muy clara de cómo se suceden las etapas durante el desarrollo, y sugiere a los desarrolladores cuál es la secuencia de eventos que podrán encontrar.

También conocido como ciclo de vida del software. Consta de 5 etapas, que son las actividades fundamentales en cualquier desarrollo de software:

- *Análisis y definición de requerimientos*. Se definen los servicios, metas y restricciones del sistema a partir de consultas con los clientes y usuarios. Con esta información se produce el documento de "Especificación del Sistema".
- *Diseño del sistema y del software*. El proceso de diseño del sistema divide los requerimientos en software o hardware. Establece una arquitectura completa del

sistema. El diseño de software identifica y describe las abstracciones fundamentales del sistema software y sus relaciones.

- *Implementación y validación de unidades*. Durante esta etapa, el diseño del software se lleva a cabo como un conjunto de unidades de programas. La prueba de unidades implica verificar que cada una cumpla su especificación.
- *Integración y validación del sistema*. Los programas o las unidades individuales de programas se integran y prueban como un sistema completo para asegurar que se cumplan los requerimientos del software. Después de las pruebas, el sistema de software se entrega al cliente.
- *Funcionamiento y mantenimiento*. Por lo general (aunque no necesariamente), ésta es la fase más larga del ciclo de vida. El sistema se instala y se pone en funcionamiento práctico. El mantenimiento implica corregir errores no descubiertos en las etapas anteriores del ciclo de vida y mejorar la implantación de las unidades del sistema.

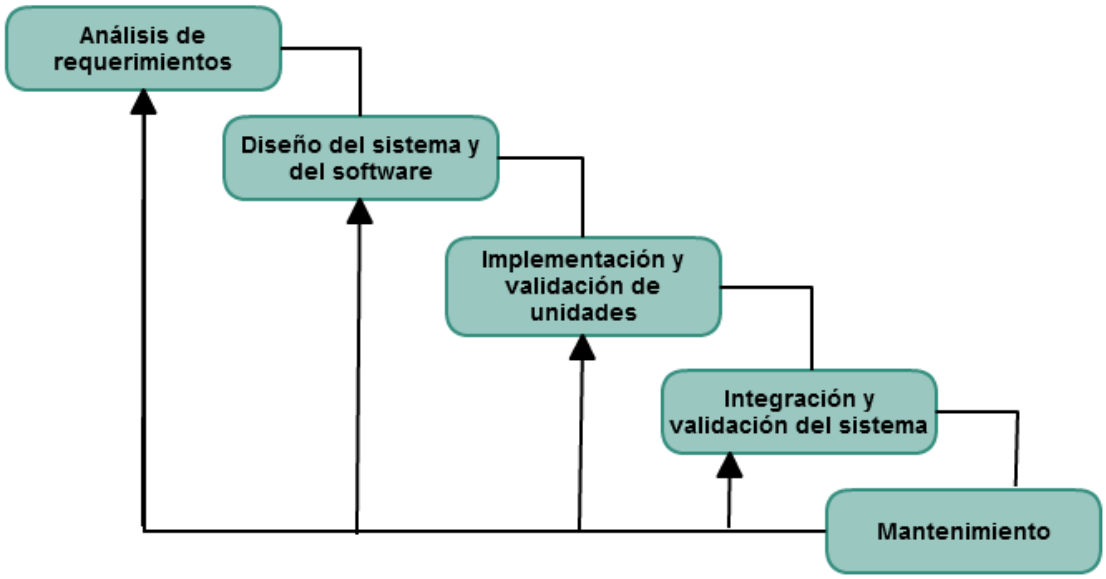

Figura 2: El modelo en cascada (Sommerville, 2005)

El resultado de cada fase es uno o más documentos aprobados (firmados). La siguiente fase no debe empezar hasta que la fase previa haya finalizado. En la práctica, estas etapas se superponen y proporcionan información a las otras. Durante el diseño se identifican problemas con los requerimientos, durante el diseño de código se encuentran problemas y así sucesivamente. El principal problema de este modelo es su inflexibilidad al dividir el proyecto en distintas etapas. Se deben hacer compromisos en las etapas iniciales, lo que hace difícil responder a los cambios en los requerimientos del cliente. Por lo tanto, el modelo en cascada solo se debe utilizar cuando los requerimientos se comprenden bien y sea improbable que cambien radicalmente durante el desarrollo del sistema. Sin embargo este modelo este modelo es muy importante porque define las etapas que se siguen en los procesos de software.

La limitación principal del modelo en cascada reside en que no trata al software como un proceso de resolución de problemas. El modelo en cascada deriva del mundo del hardware y presenta una visión de manufactura sobre el desarrollo del software. Pero la manufactura produce un artículo en particular y lo reproduce muchas veces. El software no se desarrolla de la misma manera; en cambio, evoluciona a medida que el problema se comprende y se evalúan las alternativas. Así el software es un proceso de creación, no de fabricación. La creación implica intentar un poco de esto y de aquello, como desarrollar y evaluar prototipos, valorar la factibilidad de los requerimientos, comparar varios diseños, aprender a partir de los errores, eventualmente, establecer una solución satisfactoria del problema en cuestión.

3. Modelos evolutivos

Los modelos evolutivos son iterativos. Se caracterizan por la forma en que permiten a los ingenieros del software desarrollar versiones cada vez más completas del software.

El desarrollo evolutivo se basa en la idea de desarrollar una implementación inicial, exponiéndola a los comentarios del usuario y refinándola a través de las diferentes versiones hasta que se desarrolla un sistema adecuado. Las actividades de especificación, desarrollo y validación se entrelazan en vez de separarse, con una rápida retroalimentación entre éstas. Existen dos tipos de desarrollo evolutivo:

1. *Desarrollo exploratorio*. Donde el objetivo del proceso es trabajar con el cliente para explorar sus requerimientos y entregar un sistema final. El desarrollo empieza con las partes del sistema que se comprenden mejor. El sistema evoluciona agregando nuevos atributos propuestos por el cliente.
2. *Prototipos desechables.* Donde el objetivo del proceso de desarrollo evolutivo es comprender los requerimientos del cliente y entonces desarrollar una definición mejorada de los requerimientos para el sistema. El prototipo se centra en experimentar con los requerimientos del cliente que no se comprenden del todo.

Los modelos evolutivos (o de prototipos) tienen como objetivo principal reducir el riesgo y la incertidumbre en el desarrollo, los requerimiento y/o el diseño requieren la investigación repetida para asegurar que el desarrollador, el usuario y el cliente tengan una comprensión unificada tanto de lo que se necesita como de lo que se propone como solución.

El modelo evolutivo suele ser más efectivo que el modelo en cascada para la producción de sistemas, ya que satisface las necesidades inmediatas de los clientes. Tiene la ventaja de que la especificación del sistema se puede desarrollar de forma creciente. Sin embargo, tiene dos problemas principales:

1. *El proceso no es visible*. Los administradores tienen que hacer entregas regulares para medir el progreso. Si los sistemas se desarrollan rápidamente, no es rentable producir documentos que reflejen cada versión del sistema.
2. *A menudo los sistemas tienen una estructura deficiente*. Los cambios continuos tienden a corromper la estructura del software. Incorporar cambios en él se convierte cada vez más en una tarea difícil y costosa.

Para sistemas pequeños y de tamaño medio (hasta 500,000 líneas de código), el modelo evolutivo de desarrollo es el mejor. Los problemas del desarrollo evolutivo se hacen particularmente agudos para sistemas grandes y complejos con un período de vida largo, donde diferentes equipos desarrollan distintas partes del sistema. Es difícil establecer una arquitectura del sistema estable, porque se hace difícil integrar las contribuciones de los equipos.

Para sistemas grandes, se recomienda un proceso mixto que incorpore las mejores características del modelo en cascada (como son que la documentación se produce en cada fase y que este cuadra con otros modelos del proceso de ingeniería) y del desarrollo evolutivo. Las partes bien comprendidas se pueden especificar y desarrollar utilizando un proceso basado en el modelo en cascada. Las otras partes del sistema, como la

interfaz de usuario, que son difíciles de especificar por adelantado, se deben desarrollar siempre utilizando un enfoque de programación exploratoria.

3.1. Ejemplos de modelos evolutivos

El modelo incremental.- Entrega el software en partes pequeñas, pero utilizables, llamadas *incrementos*. En general, cada incremento se construye sobre aquel que ya ha sido entregado.

El modelo iterativo.- Se entrega el esqueleto de un sistema completo desde el principio, y luego cambia la funcionalidad de cada subsistema con cada versión nueva.

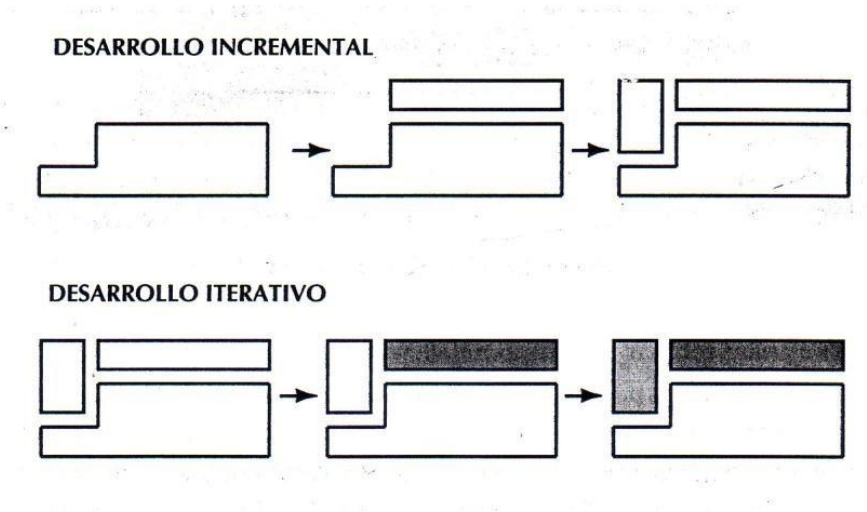

Figura 3.3 Modelos incremental e iterativo (Pleeger 2002)

El modelo en espiral.- Es un modelo de proceso de software evolutivo que conjuga la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial. Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo. Durante las últimas

iteraciones, se producen versiones cada vez más completas del sistema diseñado. Este 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 alternativas, restricciones asociadas con cada alternativa y, finalmente, se procede a su evaluación.

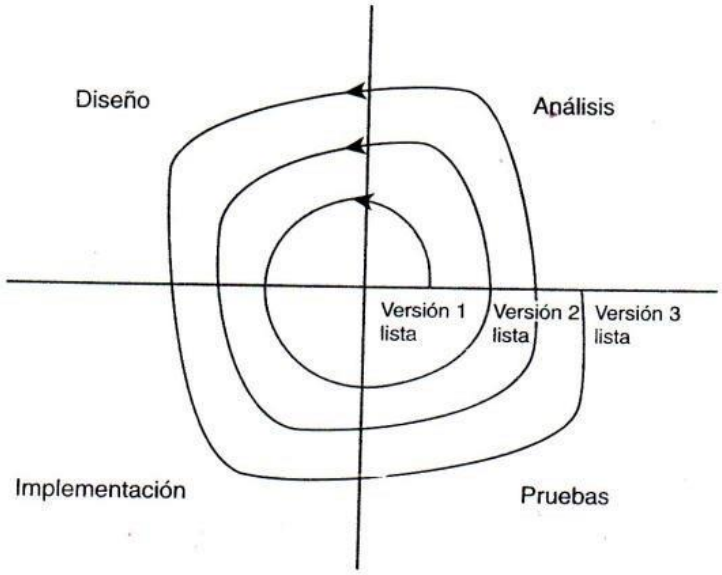

Figura 4 Modelos en espiral (Weitzenfeld, 2004)

Otros ejemplos de modelos evolutivos son: *Entrega por Etapas, Prototipado Evolutivo* y *Entrega Evolutiva* [Mc Connell, 1997].

4. Modelo de componentes reutilizables

En el desarrollo y mantenimiento del software, a menudo se saca ventaja de los aspectos comunes de las aplicaciones, reutilizando elementos de desarrollo previos. Por ejemplo, se usa el mismo sistema operativo o el mismo sistema de gestión de base de datos de

un proyecto de desarrollo a otro, en lugar de construir uno nuevo cada vez. Del mismo modo, cuando se construye un sistema similar, pero no igual a lo que se ha hecho antes, se reutilizan conjuntos de requerimientos, partes de diseños y grupos de guiones de prueba o de datos.

En la mayoría de los proyectos existe algo de reutilización de software. Por lo general, esto sucede informalmente cuando las personas que trabajan en el proyecto conocen diseños de código similares al requerido. Los buscan, los modifican según lo creen necesario y los incorporan en el sistema. Las etapas de especificación de requerimientos y de validación son comparables con los otros procesos, sin embargo, las etapas intermedias en el proceso orientado a la reutilización son diferentes, estas etapas son:

- *Análisis de componentes*. Consiste en encontrar componentes que sirvan para implementar la especificación de requerimientos. En general los componentes que se utilizan solo proporcionan parte de la funcionalidad requerida por lo que se necesita modificarlos.
- *Modificación de requerimientos*. Con la información que se tiene de los componentes ya identificados, se analizan los requerimientos. Si es posible, se modifican los requerimientos para que concuerden con los componentes disponibles. Si las modificaciones no son posibles entonces se lleva a cabo nuevamente el análisis de componentes para buscar soluciones alternativas.
- *Diseño del sistema con reutilización*. Se diseña o se reutiliza un marco de trabajo para el nuevo sistema teniendo en cuenta los componentes que se reutilizan y los componentes que serán completamente nuevos.

- *Desarrollo e integración*. El software que no se puede adquirir externamente se desarrolla y los componentes reutilizables se integran. En este modelo, la integración de los sistemas es parte del desarrollo más que una actividad separada.

El modelo de componentes reutilizables tiene la ventaja obvia de reducir la cantidad de software a desarrollarse y así reduce los costos y los riesgos, sin embargo, los compromisos en los requerimientos son inevitables, y esto puede dar lugar a un sistema que no cumpla las necesidades reales de los usuarios. Más aún, si las nuevas versiones de los componentes reutilizables no están bajo el control de la organización que los utiliza, se pierde el control sobre la evolución del sistema.

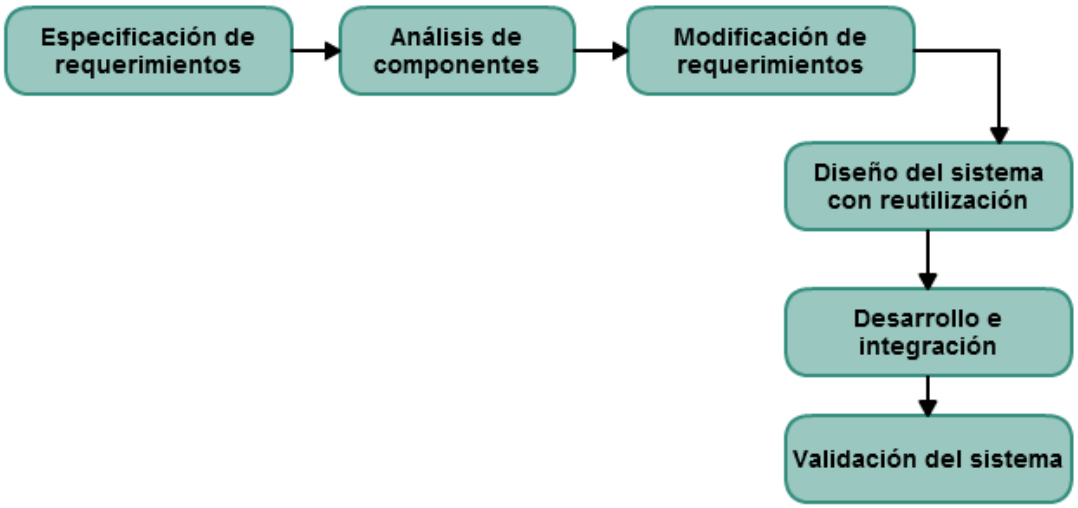

Figura 5 Modelo de componentes reutilizables (Sommerville, 2005)

Conclusión del capítulo: Los requerimientos están estrechamente relacionados con el modelo de ciclo de vida con el que se desarrolle el proyecto. En el modelo en cascada, los requerimientos tienen que estar bien definidos desde el inicio del proyecto y la probabilidad de que cambien debe ser mínima. Si se trabaja con los modelos evolutivos,

los requerimientos se trabajan al inicio de cada iteración para aumentarlos, corregirlos o redefinirlos. Cuando el paradigma a utilizar es el modelo basado en componentes, es necesario cuidar que la modificación de requerimientos no produzca un sistema que no cumple con las necesidades reales de los usuarios. El alcance de la recolección de los requerimientos cambia considerablemente con el ciclo de vida.