UNIVERSIDAD DE LA SIERRA SUR

MODELOS DEL PROCESO

INFORMÁTICA 506-A

Índice

1. Modelo en cascada **3**

2. Modelos evolutivos. **4**
2.0.1. Hacer prototipos. 4
2.0.2. El modelo espiral. 5
2.0.3. Modelos concurrentes. 6

3. Modelos incrementales. **7**
3.1. Ventajas. 7
3.2. Inconvenientes. 7

4. Modelos formales. **8**

5. Modelo basado en componentes reutilizables. **8**
5.1. Características y utilidades. 9
5.2. Beneficios SBDC. 9

6. Proceso unificado. **10**
6.1. Fases del proceso unificado 10

1. Modelo en cascada

Ocurre en cierto número limitado de nuevos esfuerzos de desarrollo, sólo cuando los requerimientos están bien definidos y tienen una estabilidad razonable.

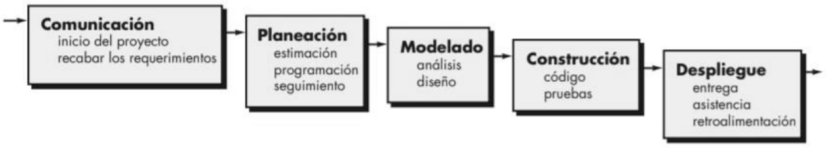

Ilustración 1 Modelo en cascada

Modelo de la cascada, o también llamado ciclo de vida clásico Sugiere un enfoque sistemático y secuencial para el desarrollo del software, que comienza con la especificación de los requerimientos por parte del cliente y avanza a través de planeación, modelado, construcción y despliegue.

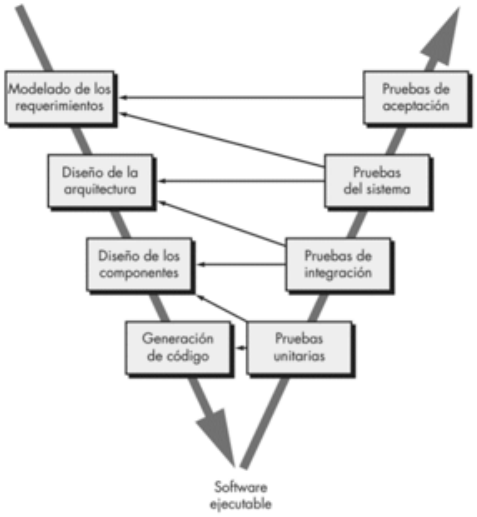

Ilustración 2 Modelo en V

A medida que el equipo de software avanza hacia abajo desde el lado izquierdo de la V, los requerimientos básicos del problema mejoran hacia representaciones técnicas cada vez más detalladas del problema y de su solución. Una vez que se ha generado el código, el equipo sube por el lado derecho de la V, y en esencia ejecuta una serie de pruebas (acciones para asegurar la calidad) que validan cada uno de los modelos creados cuando el equipo fue hacia abajo por el lado izquierdo. El modelo de la cascada es el paradigma más antiguo de la ingeniería de software. Problemas que en ocasiones surgen al aplicar el modelo de la cascada:

- Los proyectos reales no siempre siguen el flujo secuencial propuesto por el modelo.
- Es difícil para el cliente enunciar en forma explícita todos los requerimientos.
- El cliente debe tener paciencia.

2. Modelos evolutivos.

Los modelos evolutivos son iterativos, Se caracterizan por la manera en la que permiten desarrollar versiones cada vez más completas del software, los dos modelos comunes de proceso evolutivo son los siguientes:

2.0.1. Hacer prototipos.

El paradigma de hacer prototipos ofrece un mejor enfoque, ayudara a los equipos a mejorar la comprensión de que se va a elaborar cuando los requerimientos no estén claros.
El paradigma de hacer prototipos tiene un proceso, el cual comienza con la comunicación, donde se reúne los participantes para definir los objetivo generales de software, se identifican los requerimientos que se conozca y detecta las áreas en las que es imprescindible una mayor definición, después de esto se plantea rápidamente una iteración para hacer el prototipo, y se lleva a cabo el modelado(diseño rápido), este se centra en la representación de los aspectos del software que serán visibles para los usuarios finales. El diseño rápido lleva a la construcción de un prototipo donde se entrega y es evaluado por los participantes, que dan retroalimentación para mejorar los requerimientos. Lo ideal es que el prototipo sirva como mecanismo para identificar los requerimientos del software, el prototipo sirve como "el primer sistema", algunos prototipos se construyen para ser "desechables", otros son evolutivos ya que poco a poco se transforman en el sistema real, a los ingenieros de software les gusta el paradigma de hacer prototipos porque ellos siente que así los usuarios adquieren la sensación del sistema real, y los desarrolladores logran construir algo de inmediato.
Algunas razones por las cual el hacer prototipos llega a ser algo problemático son las siguientes:

1. Los participantes ven lo que parece ser una versión funcional del software, sin darse cuenta de que el prototipo se obtuvo de manera caprichosa; no perciben que en la prisa por hacer que funcionara, usted no consideró la calidad general del software o la facilidad de darle mantenimiento a largo plazo.
2. Es frecuente que llegue a compromisos respecto de la implementación a fin de hacer que el prototipo funcione rápido. Quizá utilice un sistema operativo inapropiado.

Hacer prototipos es un paradigma eficaz para la ingeniería de software. La clave es definir desde el principio las reglas del juego, es decir, todos los participantes deben estar de acuerdo en que el prototipo sirva como el mecanismo para definir los requerimientos esto hará que la ingeniería del software real con la mirada puesta en la calidad.

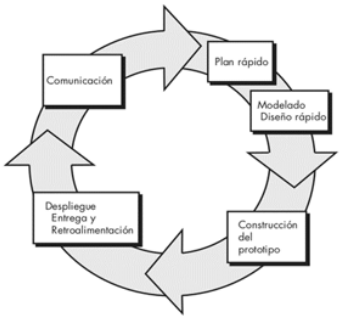

Ilustración 3 Hacer prototipos

2.0.2. El modelo espiral.

El modelo espiral es un modelo evolutivo del proceso del software y se acopla con la naturaleza iterativa de hacer prototipos con los aspectos controlados y sistémicos del modelo de cascada. Un modelo en espiral es dividido por el equipo de software en un conjunto de actividades estructurales.
En el proceso evolutivo el equipo de software realiza actividades implícitas en un circuito alrededor de la espiral en el sentido horario, partiendo del centro.
El primer circuito alrededor de la espiral da como resultado el desarrollo de una especificación del producto; las vueltas sucesivas se usan para desarrollar un prototipo y, luego, versiones cada vez más sofisticadas del software, cada paso por la región de planeación da como resultado ajustes en el plan del proyecto, el costo y la programación de actividades se ajustan con base en la retroalimentación obtenida del cliente después de la entrega, con esto el el gerente del proyecto ajusta el número planeado de iteraciones que se requieren para terminar el software.
El modelo espiral es un enfoque realista para el desarrollo de sistemas y de software a gran escala, el software evoluciona a medida que el proceso avanza, el desarrollador y cliente comprenden y reaccionan mejor ante los riesgos en cada nivel de evolución. El modelo espiral usa los prototipos como mecanismo de reducción de riesgos, permite aplicar el enfoque de hacer prototipos en cualquier etapa de la evolución del producto.

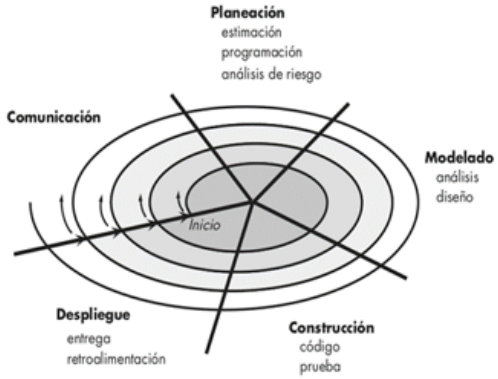

Ilustración 4 Modelo en espiral

2.0.3. Modelos concurrentes.

El modelo de desarrollo concurrente, en ocasiones llamado ingeniería concurrente, permite que un equipo de software represente elementos iterativos y concurrentes de cualquiera de los modelos de proceso antes mencionados. Por ejemplo, la actividad de modelado definida para el modelo espiral se logra por medio de invocar una o más de las siguientes acciones de software: hacer prototipos, análisis y diseño.
La figura muestra la representación esquemática de una actividad de ingeniería de software dentro de la actividad de modelado con el uso del enfoque de modelado concurrente. Cabe destacar que todas las actividades de ingeniería de software existen de manera concurrente, pero se hallan en diferentes estados.

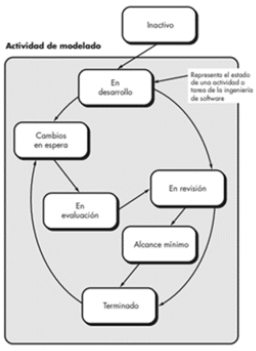

Ilustración 5 Modelo concurrente

El modelado concurrente define una serie de eventos que desencadenan transicio-

nes de un estado a otro para cada una de las actividades, acciones o tareas de la ingeniería de software. Por ejemplo, durante las primeras etapas del diseño (acción importante de la ingeniería de software que ocurre durante la actividad de modelado), no se detecta una inconsistencia en el modelo de requerimientos. Esto genera el evento corrección del modelo de análisis, que disparará la acción de análisis de requerimientos del estado terminado al de cambios en espera.
El modelado concurrente es aplicable a todos los tipos de desarrollo de software y proporciona un panorama apropiado del estado actual del proyecto.

3. Modelos incrementales.

El modelo incremental combina elementos del modelo en cascada con la filosofía interactiva de construcción de prototipos. Se basa en la filosofía de construir incrementando las funcionalidades del programa. Este modelo aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software.

3.1. Ventajas.

Entre las ventajas que puede proporcionar un modelo de este tipo encontramos las siguientes:

- Mediante este modelo se genera software operativo de forma rápida y en etapas tempranas del ciclo de vida del software.
- Es un modelo más flexible, por lo que se reduce el coste en el cambio de alcance y requisitos.
- Es más fácil probar y depurar en una iteración más pequeña.
- Es más fácil gestionar riesgos.
- Cada iteración es un hito gestionado fácilmente.

3.2. Inconvenientes.

Para el uso de este modelo se requiere una experiencia importante para definir los incrementos y distribuir en ellos las tareas de forma proporcionada. Entre los inconvenientes que aparecen en el uso de este modelo podemos destacar los siguientes:

- Cada fase de una iteración es rígida y no se superponen con otras.
- Pueden surgir problemas referidos a la arquitectura del sistema porque no todos los requisitos se han reunido, ya que se supone que todos ellos se han definido al inicio.

El modelo incremental de gestión de proyectos tiene como objetivo un crecimiento progresivo de la funcionalidad. Es decir, el producto va evolucionando con cada una de las entregas previstas hasta que se amolda a lo requerido por el cliente o destinatario.
Este enfoque, que se usó inicialmente para proyectos de software aunque más tarde se aplicó a otros sectores, establece entregas parciales mediante un calendario de plazos. En cada una de ellas, el producto debe mostrar una evolución con respecto a la fecha anterior; nunca puede ser igual.
La principal diferencia del modelo incremental con los modelos tradicionales es que las tareas están divididas en iteraciones, es decir, pequeños lapsos en los cuales se trabaja para conseguir objetivos específicos. Con los modelos tradicionales no pasaba esto; era necesario esperar hasta el final del proceso. Sin embargo, no se trata de iteraciones independientes. Por el contrario, están vinculadas de forma que cada una suponga un avance con respecto a la anterior.

4. Modelos formales.

En este tipo de modelo se agrupan actividades que llevan a la especificación matemática formal del software de cómputo. Los métodos formales permiten realizar la especificación, el desarrollo y la verificación de un sistema basado en computadora por medio del empleo de una notación matemática rigurosa.
Existen algunas organizaciones que se encargan del desarrollo de software y aplican una variante de este modelo, dicha variante es denominada como Ingeniería de software de quirófano.
Durante el desarrollo se usan métodos formales, se obtiene un mecanismo para eliminar problemas con otros paradigmas de la ingeniería de software. Lo ambiguo, incompleto e inconsistente se corrige con el análisis matemático, implementando métodos formales, esto sirve para verificar el programa, permitiendo descubrir y corregir errores, prometiendo un software libre de defectos y de alta calidad. El desarrollo de modelos formales consume mucho tiempo muy pocos desarrolladores tienen formación para aplicar los métodos formales ya que son difíciles de utilizar cuando se tratan como medio de comunicación entre los clientes, esto debido a su complejidad.
El enfoque que nos muestran los métodos formales gana partidarios entre los desarrolladores que tienen la necesidad y el enfoque de construir un software de alta calidad, evitando al mismo tiempo las pérdidas económicas.

5. Modelo basado en componentes reutilizables.

La programación orientada a componentes, es un paradigma que programa la construcción de componentes reutilizables en entornos abiertos y distribuidos. Con el objetivo de lograr un método global de software. Uno de los enfoques en los que

actualmente se trabaja constituye lo que se conoce como desarrollo de software basado en componentes (DSBC), trate de sentar las base para el diseño y desarrollo de aplicaciones distribuidas basada en componentes de software reutilizables. Beneficios como:

- Las mejoras de calidad
- La reducción del ciclo de desarrollo
- El mayor entorno sobre la inversión

"La reutilización de software es un proceso de la ingeniería de software que conlleva al uso recurrente de activos de software en la especificación, análisis, diseño, implementación y pruebas de una aplicación o sistema de software" El desarrollo basado en componentes constituye una aproximación del desarrollo de software que describe, constituye y emplea técnicas de software para elaborar sistemas abiertos y distribuidos mediante el ensamblaje de partes de software reutilizables.

5.1. Características y utilidades.

Es utilizado para reducir costos tiempo y esfuerzos de desarrollo del software, y de esta manera incrementar el nivel de productividad de los grupos desarrolladores a si mismo minimizar los riesgos a su vez ayuda a optimizar la factibilidad, flexibilidad y la reutilización de la aplicación final.
La modularidad, la reusabilidad y compatibilidad son características muy relevantes de la tecnología de programación basada en componentes, en las cuales coincide con la tecnología orientada a objetos que puede considerarse una evolución.
Sin embargo, esta tecnología también requiere robustez debido a que los componentes deben operar en entornos más heterogéneos. El DSBC corresponde al paradigma de programación de sistemas abiertos, los cuales son extensibles u tienen componentes diferentes que se integran o abandonan el sistema de manera dinámica, los componentes pueden ser sustituidos por otros componentes, independientemente de su arquitectura y desarrollo.

5.2. Beneficios SBDC.

Simplifica las pruebas: permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
Simplifica el mantenimiento del sistema: cuando existe un débil acoplamiento entre sus componentes el desarrollo puede actualizar componentes según sea requerido, sin afectar otras partes del sistema.
Mayor calidad: dado que un componente puede ser construido y luego optimizado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorara con el paso del tiempo.

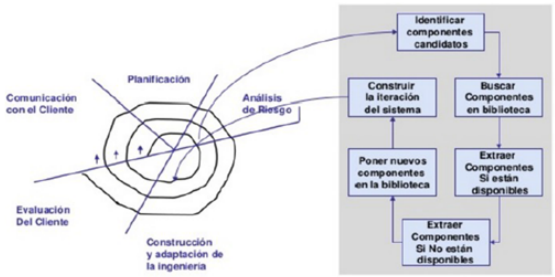

Ilustración 5 Modelo de componentes reutilizables

El modelo de desarrollo basado en componentes incorporará muchas de las características del modelo en espiral. Es evolutivo por naturaleza y exige un enfoque iterativo para la creación del software.

6. Proceso unificado.

El proceso unificado es un intento por obtener los mejores rasgos y características de los modelos tradicionales del proceso del software, pero en forma que implemente muchos de los mejores principios del desarrollo ágil de software.

- El proceso unificado reconoce la importancia de la comunicación con el cliente y los métodos directos para describir su punto de vista respecto de un sistema (el caso de uso).
- Hace énfasis en la importancia de la arquitectura del software y "ayuda a que el arquitecto se centre en las metas correctas, tales como que sea comprensible, permita cambios futuros y la reutilización"
- Sugiere un flujo del proceso iterativo e incremental, lo que da la sensación evolutiva que resulta esencial en el desarrollo moderno del software.

6.1. Fases del proceso unificado

Concepción: agrupa actividades tanto de comunicación con el cliente como de planeación. Al colaborar con los participantes, se identifican los requerimientos del negocio, se propone una arquitectura aproximada para el sistema y se desarrolla un plan para la naturaleza iterativa e incremental del proyecto en cuestión. Los requerimientos fundamentales del negocio se describen por medio de un conjunto de casos de uso preliminares que detallan las características y funciones que desea cada clase principal de usuarios.
En este punto, la arquitectura no es más que un lineamiento tentativo de subsistemas principales y la función y rasgos que tienen. La arquitectura se mejorará después y se expandirá en un conjunto de modelos que representarán distintos puntos de

vista del sistema. La planeación identifica los recursos, evalúa los riesgos principales, define un programa de actividades y establece una base para las fases que se van a aplicar a medida que avanza el incremento del software.

Elaboración: incluye las actividades de comunicación y modelado del modelo general del proceso. La elaboración mejora y amplía los casos de uso preliminares desarrollados como parte de la fase de concepción y aumenta la representación de la arquitectura para incluir cinco puntos de vista distintos del software: los modelos del caso de uso, de requerimientos, del diseño, de la implementación y del despliegue. En ciertos casos, la elaboración crea una "línea de base de la arquitectura ejecutable" que representa un sistema ejecutable de "primer corte". La línea de base de la arquitectura demuestra la viabilidad de ésta, pero no proporciona todas las características y funciones que se requieren para usar el sistema.

Además, al terminar la fase de elaboración se revisa con cuidado el plan a fin de asegurar que el alcance, riesgos y fechas de entrega siguen siendo razonables. Es frecuente que en este momento se hagan modificaciones al plan.

Construcción: es idéntica a la actividad de construcción definida para el proceso general del software. Con el uso del modelo de arquitectura como entrada, la fase de construcción desarrolla o adquiere los componentes del software que harán que cada caso de uso sea operativo para los usuarios finales. Para lograrlo, se completan los modelos de requerimientos y diseño que se comenzaron durante la fase de elaboración, a fin de que reflejen la versión final del incremento de software.

Después se implementan en código fuente todas las características y funciones necesarias para el incremento de software (por ejemplo, el lanzamiento). A medida de que se implementan los componentes, se diseñan y efectúan pruebas unitarias para cada uno. Además, se realizan actividades de integración (ensamble de componentes y pruebas de integración). Se emplean casos de uso para obtener un grupo de pruebas de aceptación que se ejecutan antes de comenzar la siguiente fase del PU.

Transición: incluye las últimas etapas de la actividad general de construcción y la primera parte de la actividad de despliegue general (entrega y retroalimentación). Se da el software a los usuarios finales para las pruebas beta, quienes reportan tanto los defectos como los cambios necesarios. Además, el equipo de software genera la información de apoyo necesaria (por ejemplo, manuales de usuario, guías de solución de problemas, procedimientos de instalación, etc.) que se requiere para el lanzamiento. Al finalizar la fase de transición, el software incrementado se convierte en un producto utilizable que se lanza.

Producción: coincide con la actividad de despliegue del proceso general. Durante esta fase, se vigila el uso que se da al software, se brinda apoyo para el ambiente de operación (infraestructura) y se reportan defectos y solicitudes de cambio para su evaluación.

Es probable que al mismo tiempo que se llevan a cabo las fases de construcción, transición y producción, comience el trabajo sobre el siguiente incremento del software. Esto significa que las cinco fases del PU no ocurren en secuencia sino que concurren en forma escalonada.

El flujo de trabajo de la ingeniería de software está distribuido a través de todas las fases del PU. En el contexto de éste, un flujo de trabajo es análogo al conjunto de tareas (que ya se describió en este capítulo). Es decir, un flujo de trabajo iden-

tifica las tareas necesarias para completar una acción importante de la ingeniería de software y los productos de trabajo que se generan como consecuencia de la terminación exitosa de aquéllas. Debe notarse que no toda tarea identificada para el flujo de trabajo del PU es realizada en todos los proyectos de software. El equipo adapta el proceso (acciones, tareas, subtareas y productos del trabajo) a fin de que cumpla sus necesidades.

Referencias

Pressman, R. S. (2010). *Ingeniería del software.* New York: y McGraw-Hill.