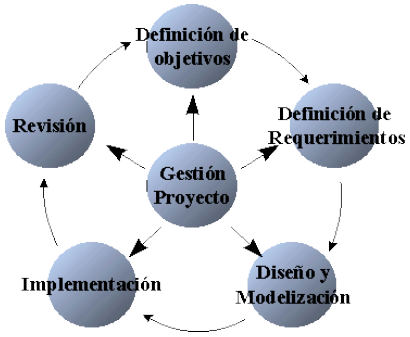
METODOLOGÍA DE ROGER PRESSMAN

De acuerdo con Roger Pressman, las etapas metodológicas a llevar a cabo para el desarrollo de Sistemas de Información, se establecen de la siguiente manera:

Etapas o Fases:

- **Análisis**
- **Diseño**
- **Codificación**
- **Prueba**
- **Mantenimiento**



Etapa I - Análisis de los requisitos del software: El proceso de reunión de requisitos se intensifica y se centra especialmente en el software. Dentro del proceso de análisis, es fundamental que a través de una colección de requerimientos funcionales y no funcionales, el desarrollador o desarrolladores del software comprendan completamente la naturaleza de los programas que deben construirse para desarrollar la aplicación, la función requerida, comportamiento, rendimiento e interconexión. Es de suma importancia que antes de empezar a codificar los programas, se tenga una completa y plena comprensión de los requisitos del software.

Pressman establece que la tarea del análisis de requisitos es un proceso de descubrimiento, refinamiento, modelado y especificación. Se refina en detalle el ámbito del software, y se crean modelos de los requisitos de datos, flujo de información y control, y del comportamiento operativo. Se analizan soluciones alternativas y se asignan a diferentes elementos del software. El análisis de requisitos permite al desarrollador o desarrolladores especificar la función y el rendimiento del software, indica la interfaz del software con otros elementos del sistema y establece las restricciones que debe cumplir el software.

El análisis de requisitos del software puede dividirse en cinco áreas de esfuerzo, que son:

1. **Reconocimiento del problema.** Reconocer los elementos básicos del problema tal y como los perciben los usuarios finales.
2. **Evaluación y síntesis.** Definir todos los objetos de datos observables externamente, evaluar el flujo y contenido de la información, definir y elaborar todas las funciones del software, entender el comportamiento del software en el contexto de acontecimientos que afectan al sistema.
3. **Modelado.** Crear modelos del sistema con el fin de entender mejor el flujo de datos y control, el tratamiento funcional y el comportamiento operativo y el contenido de la información.
4. **Especificación.** Realizar la especificación formal del software.
5. **Revisión.** Un último chequeo general de todo el proceso.

Etapa II - Diseño: Según Pressman, el diseño del software es realmente un proceso de muchos pasos pero que se clasifican dentro de uno mismo. En general, la actividad del diseño se refiere al establecimiento de las estructuras de datos, la arquitectura general del software, representaciones de interfaz y algoritmos. El proceso de diseño traduce requisitos en una representación de software.

El diseño es el primer paso en la fase de desarrollo de cualquier producto o sistema de ingeniería. De acuerdo con Pressman, el objetivo del diseño es producir un modelo o representación de una entidad que se va a construir posteriormente.

El diseño de datos esencialmente se encarga de transformar el modelo de dominio de la información creado durante el análisis. En el diseño arquitectónico se definen las relaciones entre los principales elementos estructurales del programa. Para una herramienta de software basada en el desarrollo e implementación de ambientes virtuales éste es un aspecto fundamental dado que en esta representación del diseño se establece la estructura modular del software que se desarrolla.

El diseño de interfaz describe cómo se comunica el software consigo mismo, con los sistemas que operan con él, y con los operadores que lo emplean.

Etapa III - Generación de Código: Esta actividad consiste en traducir el diseño, en una forma legible por la máquina. La generación de código se refiere tanto a la parte de generación de los ambientes virtuales, como a la parte en la cual se añadirá comportamiento a estos ambientes. Por ejemplo, el lenguaje de programación VRML 2.0 es un lenguaje de modelado en 3D en el cuál se dibuja por medio de generar código de programación de formato y marcado para especificar las características del objeto u objetos que se van agregando a un mundo o entorno virtual. El comportamiento de las escenas virtuales es decir, su funcionalidad, se puede construir a través de algún otro lenguaje de programación, como clases Java o scripts especificados en JavaScript. Todas estas actividades implican generar código.

Etapa IV - Pruebas: Una vez que se ha generado código, comienzan las pruebas del software o sistema que se ha desarrollado. De acuerdo con Pressman, el proceso de pruebas se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, y en los procesos externos funcionales, es decir, la realización de las prue ba para la detección de errores. En el caso de una herramienta de software, es necesario tener etapas de pruebas tanto para la parte funcional del software, como para la parte aplicativa del mismo.

Etapa V – Mantenimiento: El software indudablemente sufrirá cambios, y habrá que hacer algunas modificaciones a su funcionalidad. Es de suma importancia que el software de calidad pueda adaptarse con fines de acoplarse a los cambios de su entorno externo. Por medio de la documentación apropiada y atinada del software se pueden presentar las vías para el mantenimiento y modificaciones al mismo.

METODOLOGÍA DE IAN SOMMERVILLE

Sommerville define modelo de proceso de software como "Una representación simplificada de un proceso de software, representada desde una perspectiva específica. Por su naturaleza los modelos son simplificados, por lo tanto un modelo de procesos del software es una abstracción de un proceso real."

Los modelos genéricos no son descripciones definitivas de procesos de software; sin embargo, son abstracciones útiles que pueden ser utilizadas para explicar diferentes enfoques del desarrollo de software.

Modelos que se van a discutir a continuación:

- **Codificar y corregir**
- **Modelo en cascada**
- **Desarrollo evolutivo**
- **Desarrollo formal de sistemas**
- **Desarrollo basado en reutilización**
- **Desarrollo incremental**
- **Desarrollo en espiral**

Codificar y corregir

Este es el modelo básico utilizado en los inicios del desarrollo de software. Contiene dos pasos:

- Escribir código.
- Corregir problemas en el código.

Se trata de primero implementar algo de código y luego pensar acerca de requisitos, diseño, validación, y mantenimiento.

Este modelo tiene tres problemas principales:

- Después de un número de correcciones, el código puede tener una muy mala estructura, hace que los arreglos sean muy costosos.
- Frecuentemente, aún el software bien diseñado, no se ajusta a las necesidades del usuario, por lo que es rechazado o su reconstrucción es muy cara.
- El código es difícil de reparar por su pobre preparación para probar y modificar.

Modelo en cascada

El primer modelo de desarrollo de software que se publicó se derivó de otros procesos de ingeniería. Éste toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución y las representa como fases separadas del proceso.

El modelo en cascada consta de las siguientes fases:

1. Definición de los requisitos: Los servicios, restricciones y objetivos son establecidos con los usuarios del sistema. Se busca hacer esta definición en detalle.
2. Diseño de software: Se particiona el sistema en sistemas de software o hardware. Se establece la arquitectura total del sistema. Se identifican y describen las abstracciones y relaciones de los componentes del sistema.
3. Implementación y pruebas unitarias: Construcción de los módulos y unidades de software. Se realizan pruebas de cada unidad.
4. Integración y pruebas del sistema: Se integran todas las unidades. Se prueban en conjunto. Se entrega el conjunto probado al cliente.
5. Operación y mantenimiento: Generalmente es la fase más larga. El sistema es puesto en marcha y se realiza la corrección de errores descubiertos. Se realizan mejoras de implementación. Se identifican nuevos requisitos.

La interacción entre fases puede observarse en la Figura 5. Cada fase tiene como resultado documentos que deben ser aprobados por el usuario.

Una fase no comienza hasta que termine la fase anterior y generalmente se incluye la corrección de los problemas encontrados en fases previas.

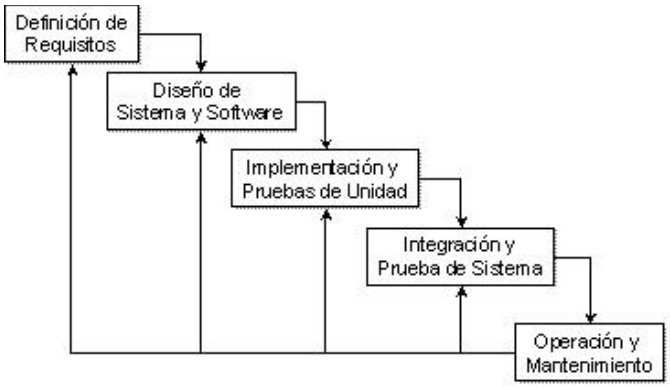


Figura 1: Modelo de desarrollo en cascada.

En la práctica, este modelo no es lineal, e involucra varias iteraciones e interacción entre las distintas fases de desarrollo. Algunos problemas que se observan en el modelo de cascada son:

- Las iteraciones son costosas e implican rehacer trabajo debido a la producción y aprobación de documentos.
- Los problemas se dejan para su posterior resolución, lo que lleva a que estos sean ignorados o corregidos de una forma poco elegante.
- Existe una alta probabilidad de que el software no cumpla con los requisitos del usuario por el largo tiempo de entrega del producto.

Este modelo sólo debe usarse si se entienden a plenitud los requisitos. Aún se utiliza como parte de proyectos grandes.

Desarrollo evolutivo

La idea detrás de este modelo es el desarrollo de una implantación del sistema inicial, exponerla a los comentarios del usuario, refinarla en N versiones hasta que se desarrolle el sistema adecuado. En la Figura 6 se observa cómo las actividades concurrentes: especificación, desarrollo y validación, se realizan durante el desarrollo de las versiones hasta llegar al producto final.

Una ventaja de este modelo es que se obtiene una rápida realimentación del usuario, ya que las actividades de especificación, desarrollo y pruebas se ejecutan en cada iteración.

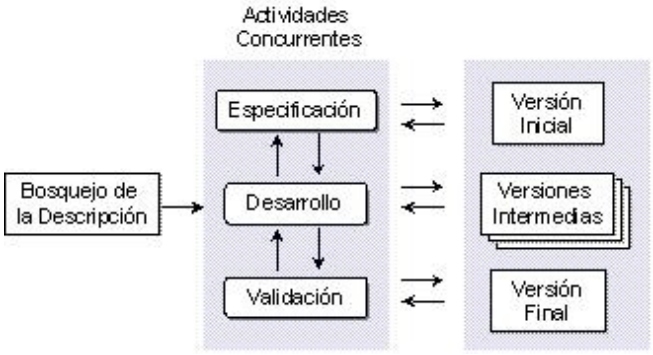


Figura 2: Modelo de desarrollo evolutivo.

Existen dos tipos de desarrollo evolutivo:

- Desarrollo Exploratorio: El objetivo de este enfoque es explorar con el usuario los requisitos hasta llegar a un sistema final. El desarrollo comienza con las partes que se tiene más claras. El sistema evoluciona conforme se añaden nuevas características propuestas por el usuario.
- Enfoque utilizando prototipos: El objetivo es entender los requisitos del usuario y trabajar para mejorar la calidad de los requisitos. A diferencia del desarrollo exploratorio, se comienza por definir los requisitos que no están claros para el usuario y se utiliza un prototipo para experimentar con ellos. El prototipo ayuda a terminar de definir estos requisitos.

Entre los puntos favorables de este modelo están:

- La especificación puede desarrollarse de forma creciente.
- Los usuarios y desarrolladores logran un mejor entendimiento del sistema. Esto se refleja en una mejora de la calidad del software.
- Es más efectivo que el modelo de cascada, ya que cumple con las necesidades inmediatas del cliente.

Desde una perspectiva de ingeniería y administración se identifican los siguientes problemas:

- Proceso no Visible: Los administradores necesitan entregas para medir el progreso. Si el sistema se necesita desarrollar rápido, no es efectivo producir documentos que reflejen cada versión del sistema.
- Sistemas pobremente estructurados: Los cambios continuos pueden ser perjudiciales para la estructura del software haciendo costoso el mantenimiento.

- Se requieren técnicas y herramientas: Para el rápido desarrollo se necesitan herramientas que pueden ser incompatibles con otras o que poca gente sabe utilizar.

Este modelo es efectivo en proyectos pequeños (menos de 100.000 líneas de código) o medianos (hasta 500.000 líneas de código) con poco tiempo para su desarrollo y sin generar documentación para cada versión.

Para proyectos largos es mejor combinar lo mejor del modelo de cascada y evolutivo: se puede hacer un prototipo global del sistema y posteriormente reimplementarlo con un acercamiento más estructurado. Los subsistemas con requisitos bien definidos y estables se pueden programar utilizando cascada y la interfaz de usuario se puede especificar utilizando un enfoque exploratorio.

Desarrollo formal de sistemas

Este modelo se basa en transformaciones formales de los requisitos hasta llegar a un programa ejecutable.

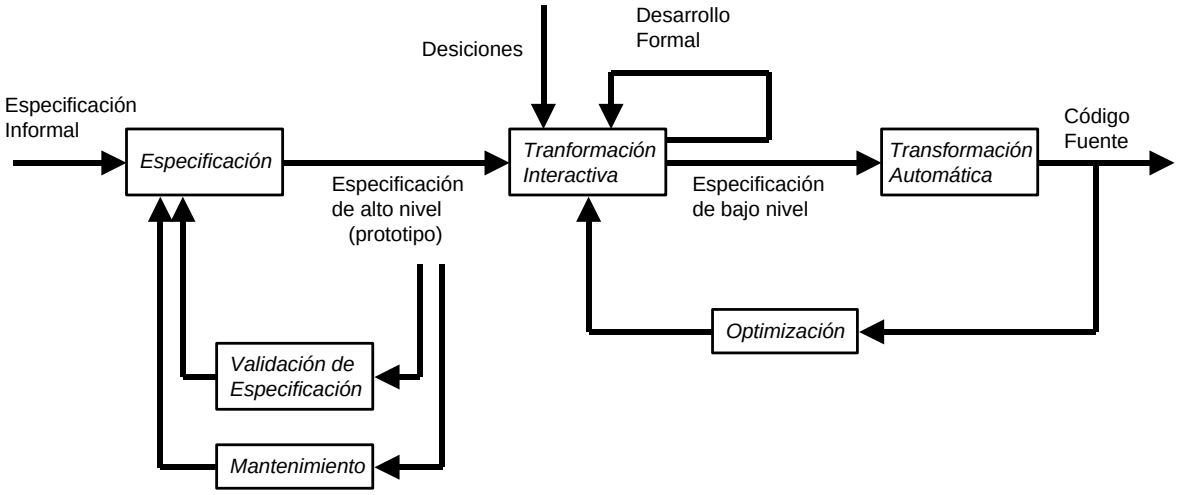


Figura 3: Paradigma de programación automática.

La Figura 7 ilustra un paradigma ideal de programación automática. Se distinguen dos fases globales: especificación (incluyendo validación) y transformación. Las características principales de este paradigma son: la especificación es formal y ejecutable constituye el primer prototipo del sistema), la especificación es validada mediante prototipación. Posteriormente, a través de transformaciones formales la especificación se convierte en la implementación del sistema, en el último paso de transformación se obtiene una implementación en un lenguaje de programación determinado. El mantenimiento se realiza sobre la especificación, la documentación es generada automáticamente y el mantenimiento es realizado por repetición del proceso.

Observaciones sobre el desarrollo formal de sistemas:

- Permite demostrar la corrección del sistema durante el proceso de transformación. Así, las pruebas que verifican la correspondencia con la especificación no son necesarias.
- Es atractivo sobre todo para sistemas donde hay requisitos de seguridad y confiabilidad importantes.
- Requiere desarrolladores especializados y experimentados en este proceso para llevarse a cabo.

Desarrollo basado en reutilización

Como su nombre lo indica, es un modelo fuertemente orientado a la reutilización. Este modelo consta de 4 fases ilustradas en la Figura 9. A continuación se describe cada fase:

1. Análisis de componentes: Se determina qué componentes pueden ser utilizados para el sistema en cuestión. Casi siempre hay que hacer ajustes para adecuarlos.
2. Modificación de requisitos: Se adaptan (en lo posible) los requisitos para concordar con los componentes de la etapa anterior. Si no se puede realizar modificaciones en los requisitos, hay que seguir buscando componentes más adecuados (fase 1).
3. Diseño del sistema con reutilización: Se diseña o reutiliza el marco de trabajo para el sistema. Se debe tener en cuenta los componentes localizados en la fase 2 para diseñar o determinar este marco.
4. Desarrollo e integración: El software que no puede comprarse, se desarrolla. Se integran los componentes y subsistemas. La integración es parte del desarrollo en lugar de una actividad separada.

Las ventajas de este modelo son:

- Disminuye el costo y esfuerzo de desarrollo.
- Reduce el tiempo de entrega.
- Disminuye los riesgos durante el desarrollo.

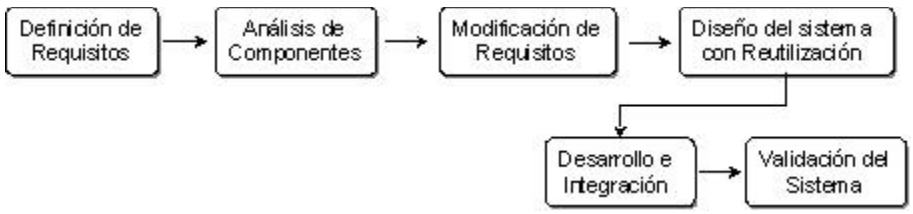


Figura 4: Desarrollo basado en reutilización de componentes

Desventajas de este modelo:

- Los "compromisos" en los requisitos son inevitables, por lo cual puede que el software no cumpla las expectativas del cliente.
- Las actualizaciones de los componentes adquiridos no están en manos de los desarrolladores del sistema.

Procesos iterativos

A continuación se expondrán dos enfoques híbridos, especialmente diseñados para el soporte de las iteraciones:

- Desarrollo Incremental.
- Desarrollo en Espiral.

Desarrollo incremental

Mills [9] sugirió el enfoque incremental de desarrollo como una forma de reducir la repetición del trabajo en el proceso de desarrollo y dar oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema (ver Figura 10). Es una combinación del Modelo de Cascada y Modelo Evolutivo.

Reduce el rehacer trabajo durante el proceso de desarrollo y da oportunidad para retrasar las decisiones hasta tener experiencia en el sistema.

Durante el desarrollo de cada incremento se puede utilizar el modelo de cascada o evolutivo, dependiendo del conocimiento que se tenga sobre los requisitos a implementar. Si se tiene un buen conocimiento, se puede optar por cascada, si es dudoso, evolutivo.

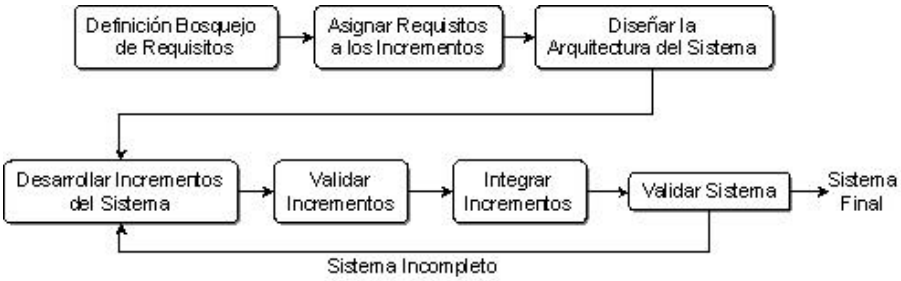


Figura 5: Modelo de desarrollo iterativo incremental.

Entre las ventajas del modelo incremental se encuentran:

- Los clientes no esperan hasta el fin del desarrollo para utilizar el sistema. Pueden empezar a usarlo desde el primer incremento.
- Los clientes pueden aclarar los requisitos que no tengan claros conforme ven las entregas del sistema.
- Se disminuye el riesgo de fracaso de todo el proyecto, ya que se puede distribuir en cada incremento.
- Las partes más importantes del sistema son entregadas primero, por lo cual se realizan más pruebas en estos módulos y se disminuye el riesgo de fallos.

Algunas de las desventajas identificadas para este modelo son:

- Cada incremento debe ser pequeño para limitar el riesgo (menos de 20.000 líneas).
- Cada incremento debe aumentar la funcionalidad.
- Es difícil establecer las correspondencias de los requisitos contra los incrementos.
- Es difícil detectar las unidades o servicios genéricos para todo el sistema.

Desarrollo en espiral

El modelo de desarrollo en espiral es actualmente uno de los más conocidos y fue propuesto por Boehm. El ciclo de desarrollo se representa como una espiral, en lugar de una serie de actividades sucesivas con retrospectiva de una actividad a otra.

Cada ciclo de desarrollo se divide en cuatro fases:

1. Definición de objetivos: Se definen los objetivos. Se definen las restricciones del proceso y del producto. Se realiza un diseño detallado del plan administrativo. Se identifican los riesgos y se elaboran estrategias alternativas dependiendo de estos.
2. Evaluación y reducción de riesgos: Se realiza un análisis detallado de cada riesgo identificado. Pueden desarrollarse prototipos para disminuir el riesgo de requisitos dudosos. Se llevan a cabo los pasos para reducir los riesgos.
3. Desarrollo y validación: Se escoge el modelo de desarrollo después de la evaluación del riesgo. El modelo que se utilizará (cascada, sistemas formales, evolutivo, etc.) depende del riesgo identificado para esa fase.
4. Planificación: Se determina si continuar con otro ciclo. Se planea la siguiente fase del proyecto.

Este modelo a diferencia de los otros toma en consideración explícitamente el riesgo, esta es una actividad importante en la administración del proyecto.

El ciclo de vida inicia con la definición de los objetivos. De acuerdo a las restricciones se determinan distintas alternativas. Se identifican los riesgos al sopesar los objetivos contra las alternativas. Se evalúan los riesgos con actividades como análisis detallado, simulación, prototipos, etc. Se desarrolla un poco el sistema y se planifica la siguiente fase.

BIBLIOGRAFIA

- Royce, W., Managing the development of large software systems: concepts and technique, IEEE Westcon, 1970.
- Pressman, R, Ingeniería del Software: Un enfoque práctico, McGraw Hill 1997.
- Sommerville, I., Ingeniería de Software, Pearson Educación, 2002.
- Boehm, B. W., A Spiral Model of Software Development and Enhancement, IEEE Computer, 1988.
- Mills, H., O´Neill, D. The Management of Software Engineering, IBM Systems, 1980.