7- Procesos de desarrollo

La distribución del esfuerzo en el proceso de desarrollo es como sigue:

- Análisis de requerimientos: 10-20%
- Diseño: 10-30%
- Codificación: 20-30%
- Testing: 30-50%

Notemos que la codificación NO es la fase más cara, sino el testing.

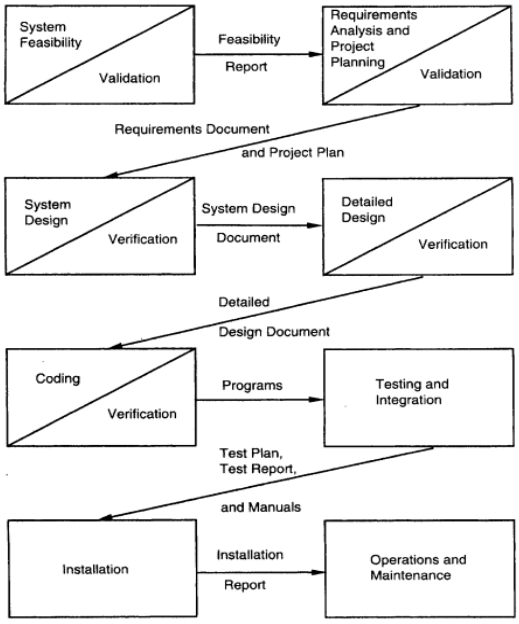
Un modelo de proceso especifica un proceso general, usualmente como un conjunto de etapas, el cual es adecuado para una clase de proyectos. Provee una estructura genérica de los procesos que puede seguirse en algunos proyectos para alcanzar sus objetivos.

Si se elige un modelo para un proyecto, usualmente será necesario **adecuarlo al proyecto**. Esta adecuación produce la especificación del proceso del proyecto, indicando cuál será el proceso a seguir. Es decir:

- **Modelo del proceso** es una especificación genérica del proceso.
- **Especificación del proceso** es un plan de lo que debe ejecutarse.
- **Proceso** es lo que realmente se ejecuta.

Se han propuesto muchos modelos para el proceso de desarrollo. Los más comunes son:

1. Cascada
2. Prototipado
3. Iterativo
4. Timeboxing



Modelo de cascada

Es el modelo más viejo, y es ampliamente utilizado. Secuencia inicial de las distintas fases:

1. Análisis de requerimientos
2. Diseño de alto nivel
3. Diseño detallado
4. Codificación
5. Testing
6. Instalación

Una fase comienza sólo cuando la anterior finaliza (en principio, no hay feedback). Las fases dividen al proyecto; cada una de ellas se encarga de distintas incumbencias.

Fase	Producto
Análisis y especificación de requerimientos	Documento de requisitos/ SRS
Diseño de alto nivel	Plan del proyecto
Diseño detallado	Documentos de diseño (arquitectura, sistema, diseño detallado)
Codificación	Plan de test y reportes de test
Testing e integración	Código final
Instalación	Manuales del software (usuario, instalación, etc)

Notar que:

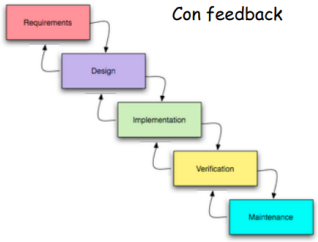
- En la práctica no se necesitan requerimientos detallados para el planeamiento, sino que el planeamiento y análisis de requisitos se superponen.
- El final de una fase y el inicio de la siguiente está claramente identificado por un **mecanismo de certificación** (validación y verificación)
- Todas las fases deben producir una salida (producto de trabajo) definida.

Ventajas:

- **Conceptualmente simple:** divide claramente el problema en distintas fases que pueden realizarse de manera independiente.
- Enfoque natural a la solución del problema.
- **Fácil de administrar en un contexto contractual**: existen fronteras bien definidas entre cada fase.

Desventajas:

- Asume que **los requerimientos de un sistema pueden congelarse** antes de que empiece el diseño. Es casi imposible tener requerimientos que no cambien nunca.
- Congelar los requerimientos suele requerir elegir hardware. En procesos que llevan mucho tiempo esto es una desventaja pues pueden ir surgiendo nuevas tecnologías que sean mejor opción que las tecnologías elegidas.
- El cliente conoce el producto que va a recibir recién cuando finaliza todo el proceso. **No puede ir dando feedback** a lo largo del proceso.
- Requiere documentos formales al final de cada fase.



El modelo en cascada con feedback ataca esas falencias.

¿Cuándo sirve?

- Para proyectos donde los **requerimientos son bien comprendidos** y las decisiones sobre la tecnología son tempranas (electrodomésticos por ejemplo).
- Para tipos de proyectos en los cuales los desarrolladores están muy familiarizados con el problema a atacar y el proceso a seguir. Por ejemplo, para la automatización de procesos manuales existentes.
- Sirve para proyectos de corta duración.

Prototipado

El prototipado aborda las limitaciones del modelo de cascada en la especificación de los requerimientos. En lugar de congelar los requerimientos sólo basado en charlas, se construye un **prototipo que permita comprender los requerimientos**, esto permite que el cliente tenga una idea de lo que sería el SW y así **conseguir mejor feedback**. Ayuda a disminuir los riesgos de requerimientos. La etapa de análisis de requerimientos es reemplazada por una mini-cascada. Atención: **el prototipo debe descartarse.**

Desarrollo del prototipo:

1. Comienza con una **versión preliminar de los requerimientos**.
2. Se desarrolla el **prototipo** que solo incluye las características claves que necesitan mejor comprensión. Es inútil incluir características bien entendidas.
3. El **cliente** “juega” con el prototipo y **provee feedback** importante que mejora la comprensión de los requerimientos.
4. Luego del feedback el **prototipo se modifica** y se repite el proceso hasta que los costos y el tiempo superen los beneficios de este proceso.
5. Teniendo en cuenta el feedback, los requerimientos iniciales se modifican para producir la especificación final de los requerimientos.

El **costo** de prototipado debe mantenerse **bajo**: Hay que construir sólo aspectos que se necesiten aclarar. "*Quick & dirty*": la calidad no importa, sólo poder desarrollar el prototipo rápidamente. Se omite el manejo de excepciones, recuperación, estándares.

Ventajas:

- Mayor estabilidad en los requerimientos.
- Los requerimientos se congelan más tarde.
- La experiencia en la construcción del prototipo ayuda al desarrollo principal.

Desventajas:

- Potencial impacto en costo y en tiempo.
- No permite cambios tardíos.

Aplicación:

- Cuando los requerimientos son difíciles de determinar y la confianza en ellos es baja (es decir, no se han comprendido).
- Cuando se desarrollan sistemas con usuarios novatos.
- Cuando las interfaces con el usuario son muy importantes.

Desarrollo iterativo

Combina beneficios del prototipado y del cascada: Aborda el problema de "todo o nada" del modelo de cascada y desarrolla y entrega el SW incrementalmente.

Cada incremento es completo en sí mismo. Provee un marco para facilitar el testing (el testing de cada incremento es más fácil que el testing del sistema completo). Puede verse como una **"secuencia de cascadas"**. El feedback de una iteración puede usarse en iteraciones futuras.

Beneficios:

- Entregas regulares y rápidas, por lo tanto **Buen feedback**.
- Prioriza requisitos.
- Acepta cambios naturalmente.

Inconvenientes:

- **Sobrecarga de planeamiento** en cada iteración.
- Posible incremento del costo total.
- La arquitectura y diseño pueden ser afectados con tantos cambios. Por lo tanto, muchas veces se necesitará **refactorización**.

Aplicación:

- cuando el tiempo de respuesta es importante;
- cuando no se puede tomar el riesgo de proyectos largos;
- cuando no se conocen todos los requerimientos.
- Muy efectivo en desarrollo de productos:
    - los desarrolladores mismos proveen la especificación.
    - los usuarios proveen el feedback en cada release.
    - basado en esto y experiencia previa => nueva versión.
- Desarrollo "a gusto del comprador" (customized):
    - Las empresas requieren respuestas rápidas.
    - No se puede arriesgar el "todo o nada".

Modelo con mejora iterativa

Primer paso:

- **Implementación** simple para un **subconjunto** del problema completo (sólo aspectos claves y fáciles de entender).
- **Crear lista de control del proyecto** (LCP) que contiene (en orden) las tareas que se deben realizar para lograr la implementación final.

Luego:

- Cada paso consiste en eliminar la siguiente tarea de la lista haciendo diseño, implementación y análisis del sistema parcial, y actualizar la LCP.

Este proceso se repite hasta vaciar la lista. Así, **la LCP guía los pasos de iteración** y lleva las tareas a realizar, cada entrada en LCP es una tarea a realizarse en un paso de iteración y debe ser lo suficientemente simple como para comprenderla completamente.

Nuevos enfoques

Extreme programming:

- Programación de a pares/mucho code reviews/unit test
- No programar nuevas características, hasta necesitarlas
- Simplicidad/Claridad del código
- Esperar cambios de requerimientos
- Fluida comunicación cliente/ programador

Test Driven Development

Desarrollo ágil

- Esfuerzo colaborativo
- Auto-organización
- Equipos multidisciplinar
- Adaptive planning
- Evolutionary development
- Early delivery
- Continual improvement
- Flexibilidad hacia cambios

Proceso unificado

- Proceso dirigido por casos de uso
- Iterativo e incremental
- Enfocado en los riesgos

Modelo de timeboxing

El modelo iterativo es una secuencia lineal de iteraciones donde cada iteración es una "mini cascada" pues se decide la especificación, luego se planea, etc. Así, la secuencia en un modelo iterativo sería: requerimientos -> construcción -> entrega -> requerimientos -> construcción -> entrega y así…

Con timeboxing primero se fija la **duración de las iteraciones** y luego se determina la especificación. Divide la iteración en partes iguales y usa **pipelining** para ejecutar iteraciones en paralelo. Así, la seCuencia quedaría

requer -> constr -> entrega
        requer -> constr -> entrega
                requer -> constr -> entrega

El desarrollo se realiza iterativamente en “**cajas temporizadas**” (time boxes) de **igual duración**. Cada time box se divide en etapas fijas, y cada etapa realiza un tarea bien definida que puede desarrollarse independientemente.

Todas las etapas tienen aproximadamente igual duración, y **hay un equipo dedicado a cada etapa**. Cuando el equipo de una etapa finaliza, se lo pasa al equipo de la siguiente etapa. En este modelo existe un **alto compromiso con el cronograma**: éste no es negociable.

Ventajas:

- Todas las ventajas del modelo iterativo
- Planeamiento y negociación un poco más fácil.
- Ciclo de entrega muy corto.
- Muy preciso a la hora de las entregas

Desventajas:

- Necesita muchas personas en cada equipo. Por ende, puede ser caro.
- La administración del proyecto es compleja.
- Es posible el incremento de los costos.
- Se necesita mucha sincronización.

Aplicación:

- Cuando es necesario tiempos de entrega muy cortos y hay flexibilidad en agrupar características (features).