Técnico en
DESARROLLO DE SOFTWARE

Aseguramiento de Calidad del Desarrollo de Software

Instituto Von Neumann

Aseguramiento de Calidad del Desarrollo de Software

Unidad II

1. Modelo de ciclo de vida de software

Se 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 (Code-and-Fix)

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:

- Definición de los requisitos: Los servicios, restricciones y objetivos son establecidos con los usuarios del sistema. Se busca hacer esta definición en detalle.
- 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.
- Implementación y pruebas unitarias: Construcción de los módulos y unidades de software. Se realizan pruebas de cada unidad.
- Integración y pruebas del sistema: Se integran todas las unidades. Se prueban en conjunto. Se entrega el conjunto probado al cliente.
- 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 1. 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.

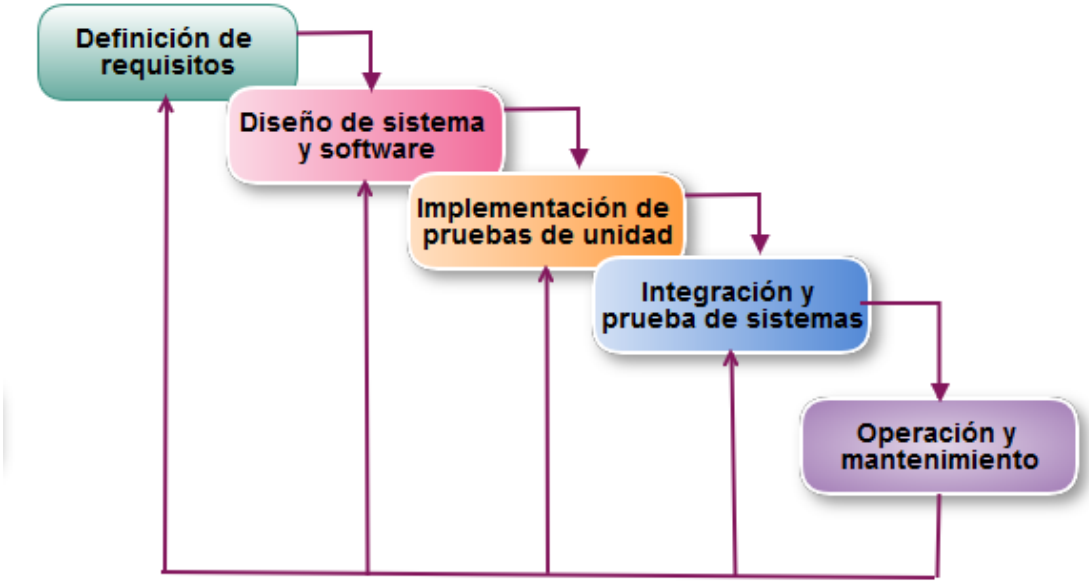

Figura 1 - Modelo de Desarrollo en Cascada

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.
- Aunque son pocas iteraciones, es normal congelar parte del desarrollo y continuar con las siguientes fases.
- 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.
- Es inflexible a la hora de evolucionar para incorporar nuevos requisitos. Es difícil responder a cambios en los requisitos.

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 2 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 retroalimentación 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 re implementarlo 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.

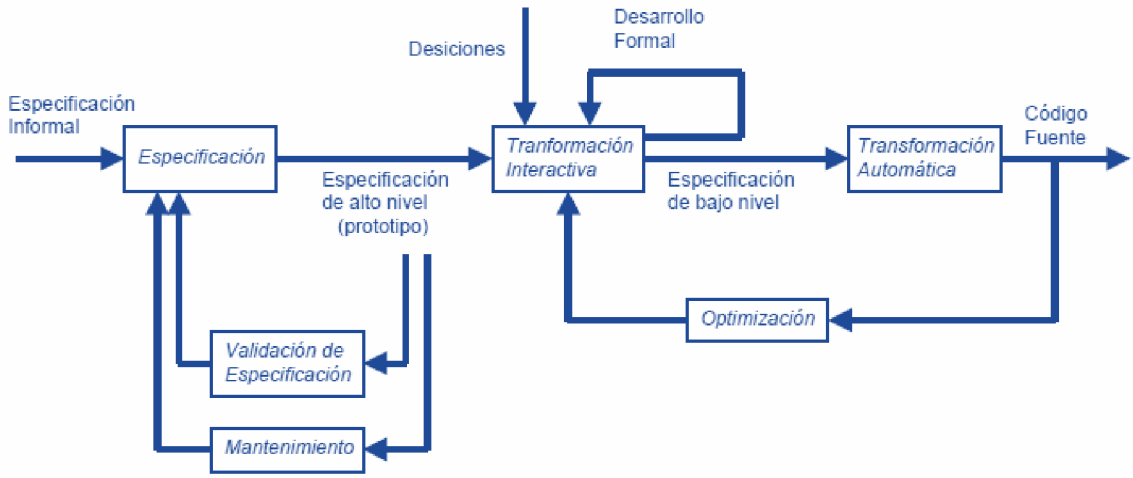

Figura 3 - Paradigma de Programación Automática

La Figura 3 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 prototipado.
- 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 (no sobre el código fuente), la documentación es generada automáticamente y el mantenimiento es realizado por repetición del proceso (no mediante parches sobre la implementación).

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 4.

A continuación, se describe cada fase:

- **Análisis de componentes**: Se determina qué componentes pueden ser utilizados para el sistema en cuestión. Casi siempre hay que hacer ajustes para adecuarlos.
- **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).
- **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.

- **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.

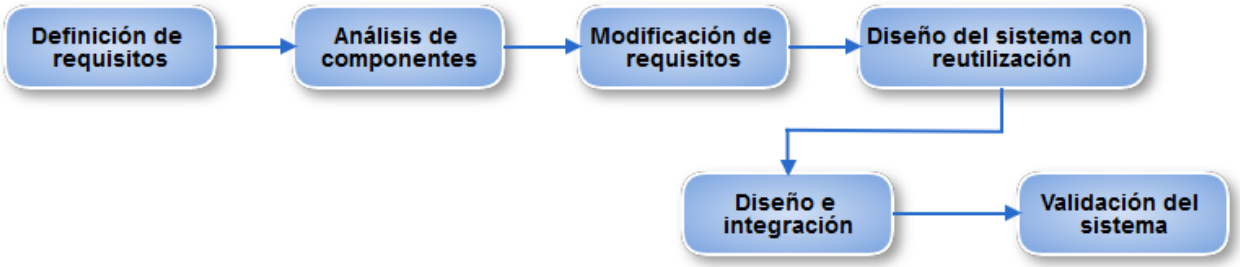

Figura 4. Desarrollo Basado en Reutilización de Componentes

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 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 5). 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.

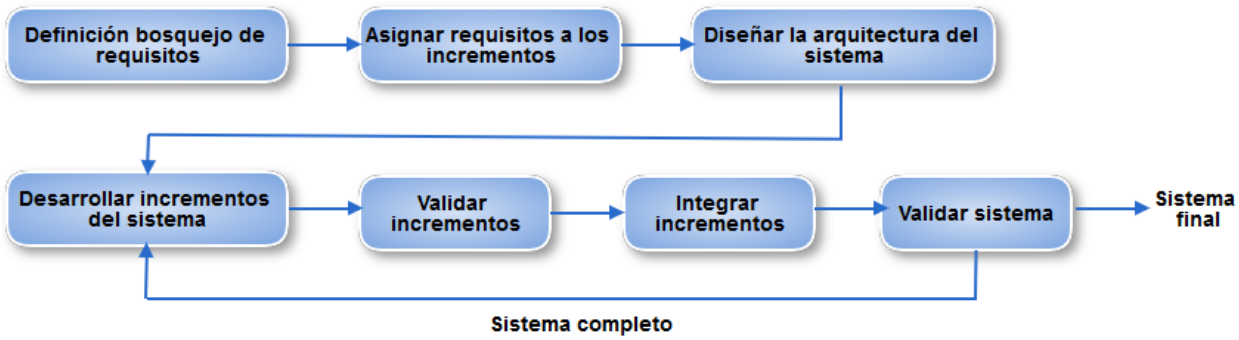

Figura 5 - Modelo de Desarrollo Iterativo Incremental

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 (ver Figura 2.6) 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:

- 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.

- 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.
- 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.
- 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. Se planifica la siguiente fase.

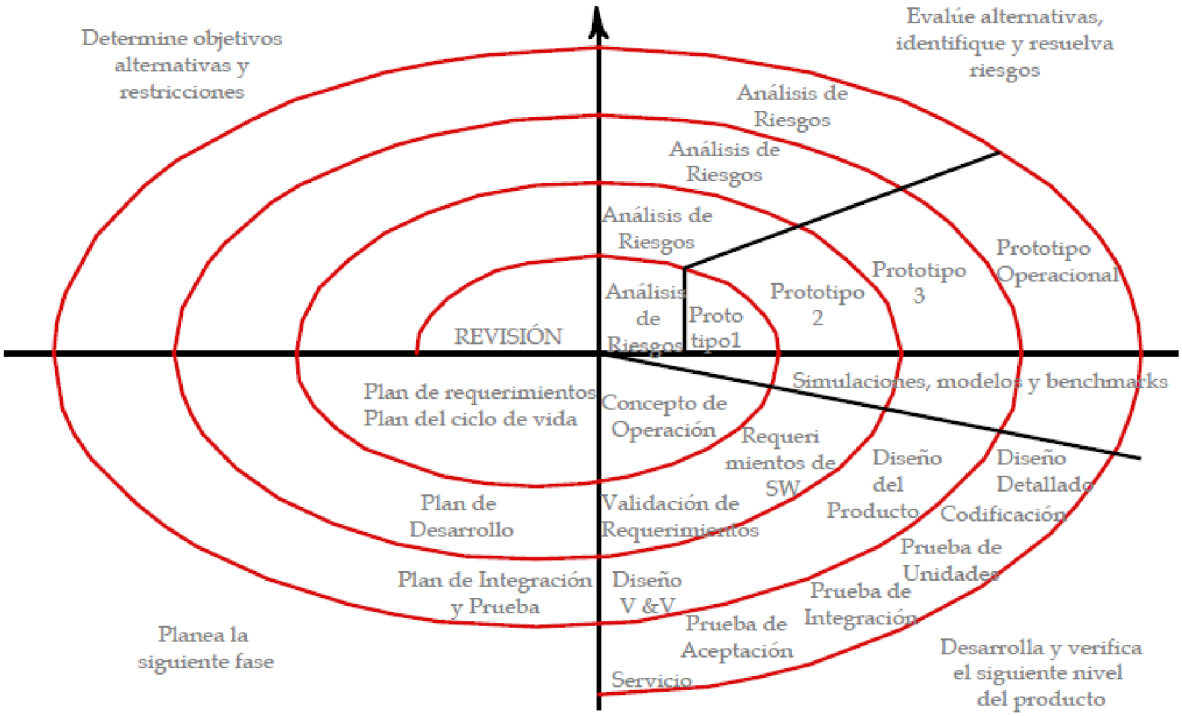

Figura 6 - Modelo de Desarrollo en Espiral

¿Cuál es el modelo de proceso más adecuado?

Cada proyecto de software requiere de una forma de particular de abordar el problema. Las propuestas comerciales y académicas actuales promueven procesos iterativos, donde en cada iteración puede utilizarse uno u otro modelo de proceso, considerando un conjunto de criterios (Por ejemplo: grado de definición de requisitos, tamaño del proyecto, riesgos identificados, entre otros).

En la Tabla 2 se expone un cuadro comparativo de acuerdo con algunos criterios básicos para la selección de un modelo de proceso, la medida utilizada indica el nivel de efectividad del modelo de proceso de acuerdo al criterio (Por ejemplo: El modelo Cascada responde con un nivel de efectividad Bajo cuando los Requisitos y arquitectura no están predefinidos):

Modelo de Proceso	Funciona con requisitos y arquitectura no predefinidos	Produce software altamente fiable	Gestión de riesgos	Permite correcciones sobre la marcha	Visión del progreso por el Cliente y el Jefe de Proyecto
Codificar y Corregir	Bajo	Bajo	Bajo	Alto	Medio
Cascada	Bajo	Alto	Bajo	Bajo	Bajo
Evolutivo exploratorio	Medio o Alto	Alto	Bajo	Bajo	Bajo
Evolutivo prototipado	Alto	Medio	Medio	Alto	Alto
Desarrollo formal de sistemas	Bajo	Alto	Bajo a Medio	Bajo	Bajo
Desarrollo orientado a la reutilización	Medio	Bajo a Alto	Bajo a Medio	Alto	Alto
Incremental	Bajo	Alto	Medio	Bajo	Bajo
Espiral	Alto	Alto	Alto	Medio	Medio

Tabla 2 – Comparación entre modelos de proceso de software

2. Verificación y validación de software

Los procesos de verificación y validación determinan si un producto software satisface las necesidades del negocio y si se está construyendo acorde a las especificaciones.

Según el SWEBOK 2004, existen dos grupos de técnicas para la comprobación de software:

- **Técnica Estática**: Técnica para evaluación del software que no requiere la ejecución del mismo.

- **Técnica Dinámica**: Técnica para evaluación del software que sí requiere la ejecución del mismo.

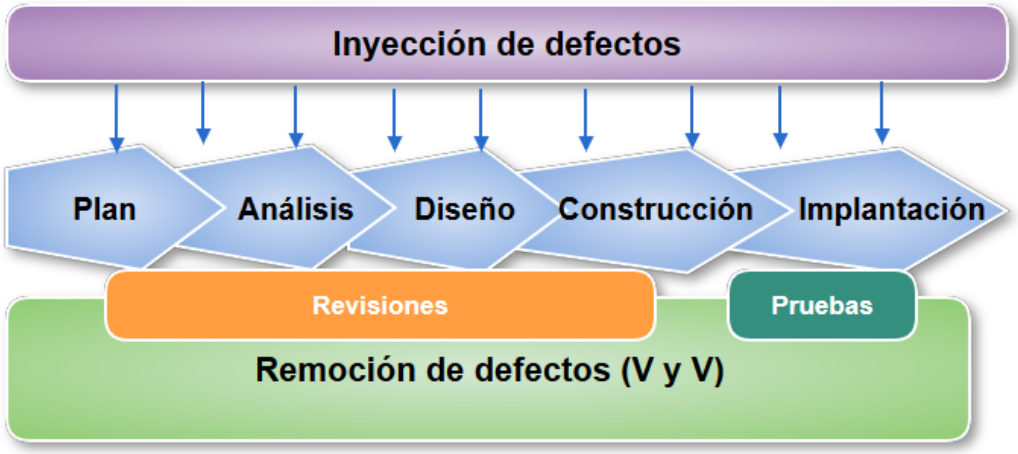

Figura 7 – Revisiones y Pruebas

En la Figura 7 se puede apreciar que cuando se aplica la técnica dinámica, normalmente se usan pruebas; y cuando se aplica técnica estática se utilizan revisiones.

Verificación

Según la ISO-IEC 12207, la Verificación es el proceso para determinar si los productos software de una actividad cumplen con los requerimientos o condiciones que tienen impuestas por las actividades precedentes.

La verificación permite comprobar que el artefacto producido en un proceso corresponde con el que se utilizó a la entrada del proceso.

La verificación permite responder la pregunta: ¿Estamos construyendo el producto en forma correcta?

La verificación se orienta al proceso.

Validación

Según la ISO-IEC 12207, la Validación es el proceso para determinar si los requerimientos y el sistema o producto software, tal como se ha construido, cumple con su uso específico previsto. La validación se puede llevar a cabo en etapas tempranas.

La validación permite comprobar si el artefacto producido es lo que el usuario necesita.

La validación permite responder la pregunta: ¿Estamos construyendo el producto correcto?

La validación se orienta al producto.

Las actividades de verificación y validación deben desarrollarse a lo largo de todo el ciclo de vida de software (ver Figura 8).

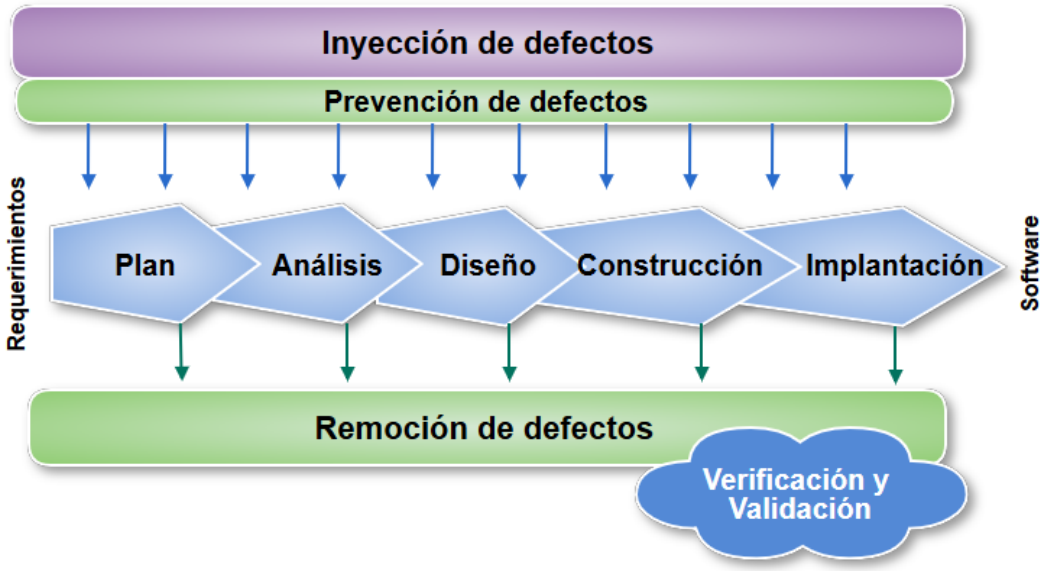

Figura 8 – Verificación y Validación

Ventajas de las Revisiones de Software

- No requiere de código ejecutable, por lo que puede ser realizada desde el inicio (por lo tanto es menos costosa).
- Se encuentran varios defectos a la vez.
- Encuentra hasta un 85% de los defectos.
- Se localiza la posición exacta del defecto.
- Refuerza el uso de estándares.
- Mejora la capacitación.

Desventajas de las Revisiones de Software

- Requiere del tiempo de expertos.
- No se puede verificar características no-funcionales (ejemplo rendimiento).
- Validan cumplimiento de lo que se especificó, en vez de lo que realmente desea el cliente.
- Difícil de implementar (es vista como "improductiva" por los desarrolladores).

Revisiones Informales y Formales

1. **Informales:**
   - No hay proceso definido.
   - No existen roles.
   - Usualmente no planeadas.
2. **Formales:**
   - Objetivos definidos.
   - Proceso documentado.

- Roles definidos y personas entrenados en ellos.
- Check-list, reglas y métodos para encontrar defectos.
- Reporte del resultado.
- Recolección de datos para el control del proceso.

Figura 9 – Tipos de Revisiones de Software