Universidad Intercontinental de la Empresa

Asignatura

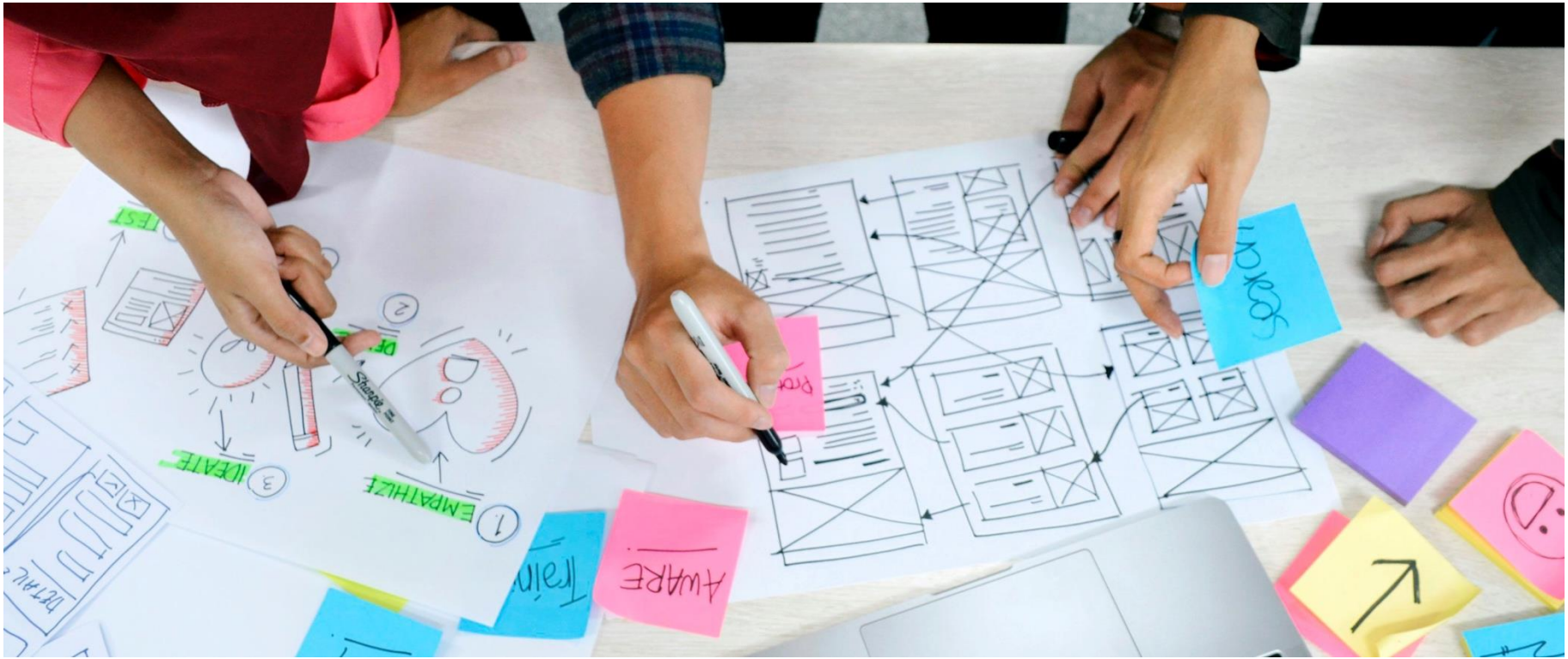
Arquitectura del Software 

Profesor

Yago Fontenla Seco

{*yago.fontenla1@uie.edu*}

El Proceso del Software



Proceso del Software

Un proceso de software es una serie de actividades relacionadas que conduce a la elaboración de un producto de software. El ciclo de vida del software abarca todas las fases por las que pasa un sistema de software, desde su concepción hasta su retiro.

Existen distintas **metodologías o modelos de proceso** de software, pero **todos deben incluir las actividades fundamentales**:

1. **Especificación**: Definir qué debe hacer el software.
2. **Diseño e implementación**: Diseñar y codificar el software.
3. **Validación**: Verificar que cumple con los requisitos.
4. **Evolución**: Modificarlo para responder a nuevas necesidades.

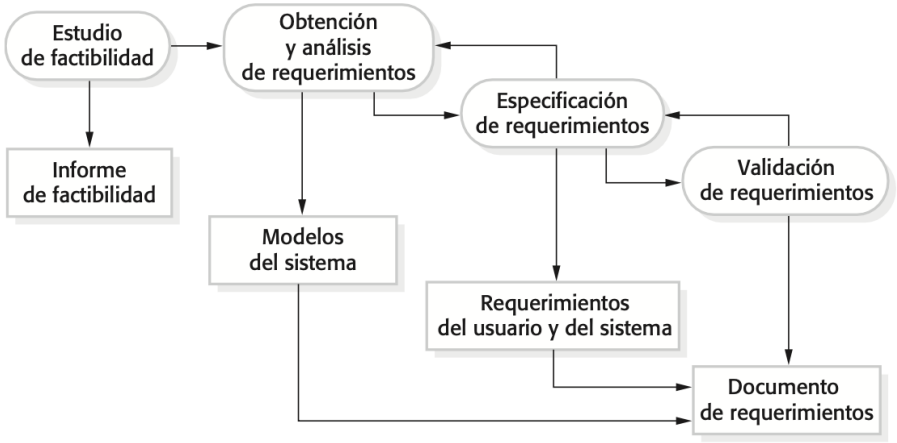
Un **modelo de proceso** de software es una representación simplificada de este proceso. Cada modelo del proceso representa a otro desde una particular perspectiva.

Actividades del proceso– Especificación

La **especificación del software** implica definir lo que debe hacer el sistema y las limitaciones bajo las cuales debe operar. Se documentan los **requerimientos funcionales** y **no funcionales** que guiarán el desarrollo y posterior validación del sistema. Esto incluye:

- **Requerimientos funcionales**: Describen el comportamiento que el sistema debe exhibir.
- **Requerimientos no funcionales**: Definen las restricciones en cuanto a rendimiento, seguridad, fiabilidad, etc.

El objetivo es crear una base sólida para el diseño y desarrollo del software, minimizando ambigüedades y asegurando que se cumplan las expectativas del cliente.

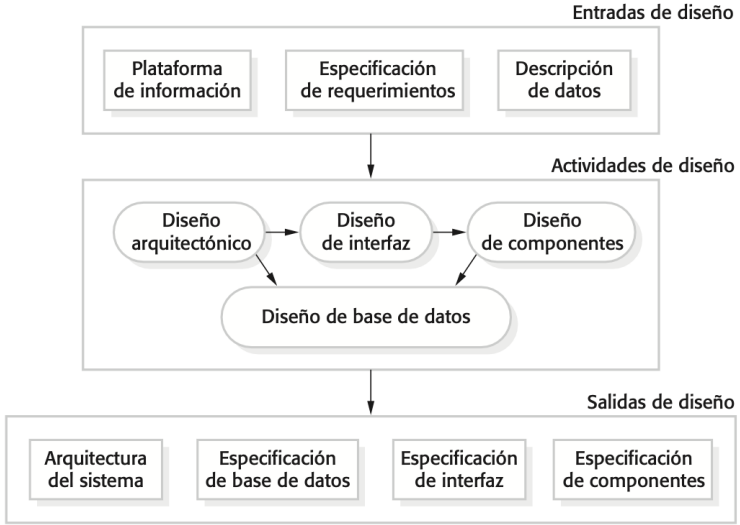


Actividades del proceso– Diseño e implementación

Un **diseño de software** se entiende como una descripción de la estructura del software que se va a implementar, los modelos y las estructuras de datos utilizados por el sistema. También abarca la **implementación** de los requerimientos en código funcional, siguiendo los principios de diseño definidos. En esta fase se construyen los componentes del software y se integran en un sistema operativo funcional. Las principales actividades incluyen:

- **Codificación**: Traducción de los diseños en lenguaje de programación.
- **Integración**: Ensamblaje de los diferentes componentes en un sistema cohesivo.

El desarrollo incluye la creación de versiones iniciales del sistema que podrán ser probadas y revisadas en fases posteriores.

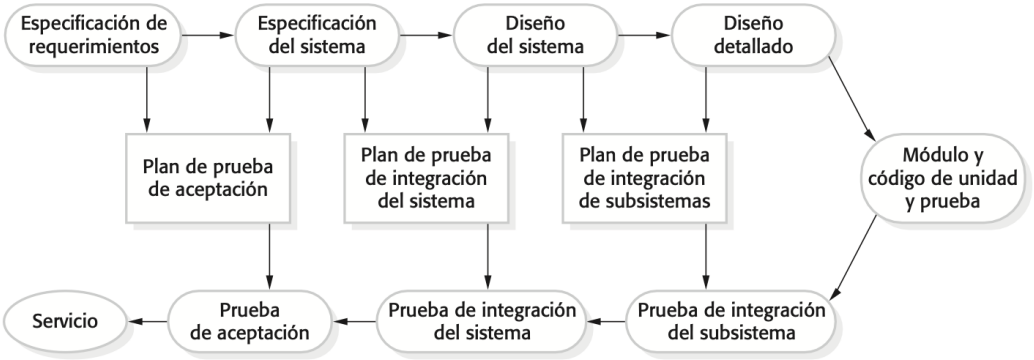


Actividades del proceso– Validación

La **validación** tiene como objetivo asegurarse de que el software cumple con los requerimientos establecidos y que funciona como se espera. Esto se logra mediante:

- **Pruebas funcionales**: Verificación de que el sistema realiza las tareas correctas.
- **Pruebas de aceptación**: Asegura que el sistema cumple las expectativas del cliente.

La validación es crucial para identificar errores o desviaciones respecto a los requerimientos antes de que el software sea implementado.

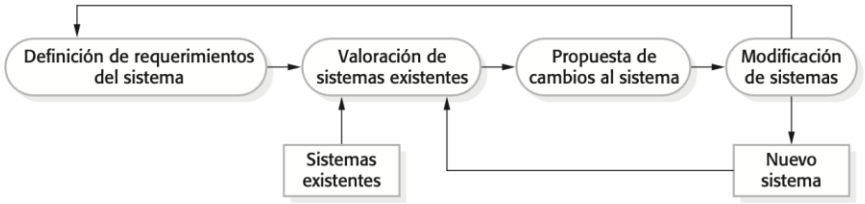


Actividades del proceso– Evolución y cambio

La **evolución del software** se refiere a las modificaciones que se realizan al sistema después de su entrega inicial para corregir errores, adaptarse a nuevas plataformas o mejorar sus características. Los sistemas deben evolucionar para mantenerse útiles en un entorno que cambia constantemente. Esta etapa incluye:

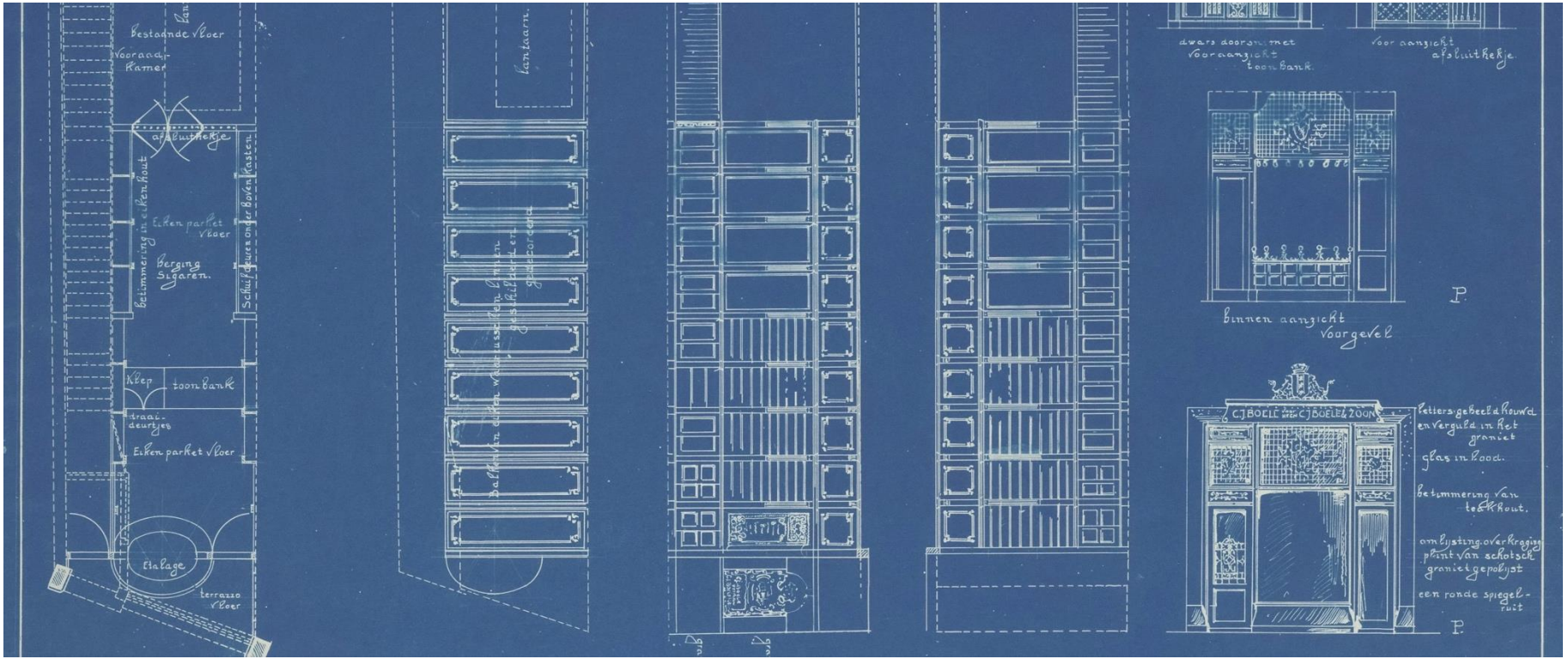
- **Actualizaciones**: Incorporación de nuevas funcionalidades o mejoras.
- **Corrección de errores**: Modificación de partes del sistema que no funcionan según lo esperado.

El software puede necesitar evolucionar incluso antes de su entrega oficial, adaptándose a nuevos requerimientos.



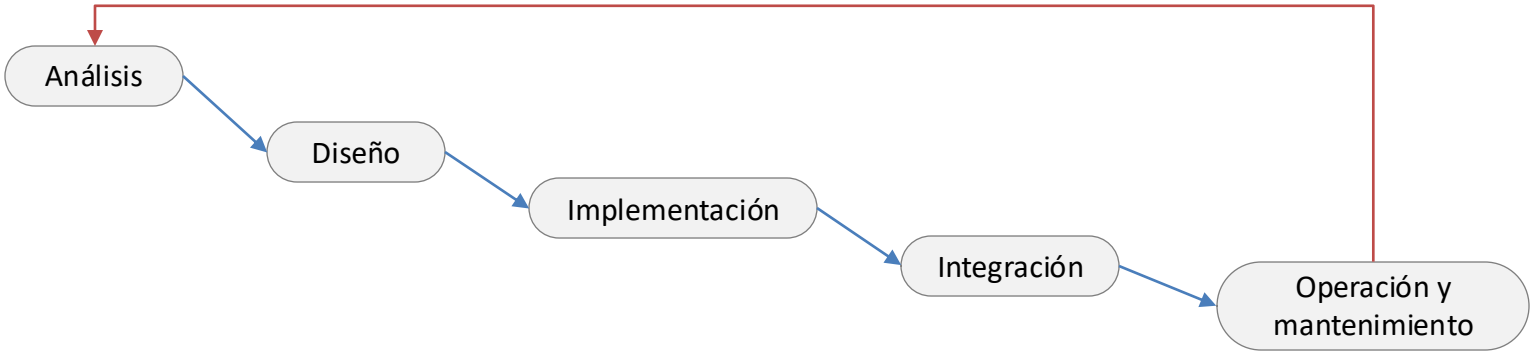
Existen dos alternativas principales para gestionar el cambio: **Prototipos del sistema** y **Entrega incremental**.

Modelos de Proceso del Software



Modelo en cascada

El primer modelo publicado sobre el proceso de desarrollo de software se derivó a partir de procesos más generales de ingeniería de sistemas (Royce, 1970). Este modelo se ilustra en la figura 2.1. Debido al paso de una fase en cascada a otra, este modelo se conoce como “modelo en cascada” o ciclo de vida del software.

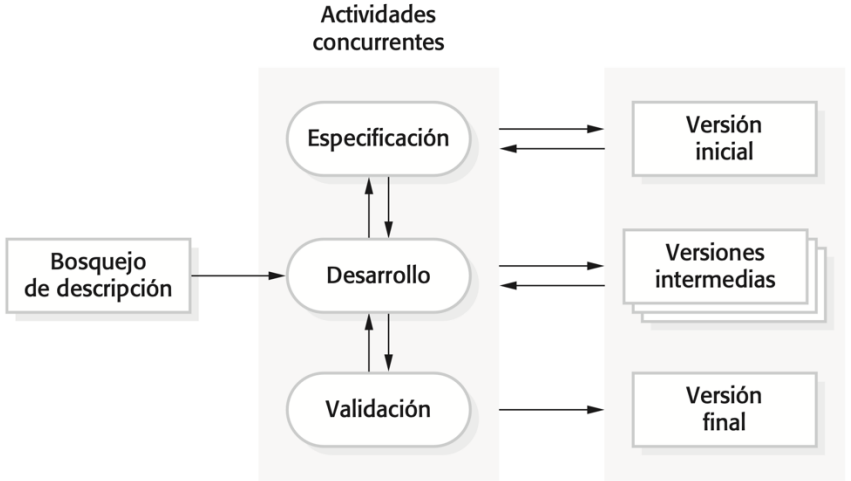


El modelo en cascada es un ejemplo de un proceso dirigido por un plan; se debe planear y programar todas las actividades del proceso, antes de comenzar a trabajar con ellas.

Modelo incremental

El **modelo incremental** es una metodología de desarrollo de software que se basa en la creación de software a través de múltiples **iteraciones** o **versiones**. En lugar de desarrollar todo el sistema en una sola fase, el software se construye por partes, permitiendo la entrega de versiones funcionales desde el inicio.

Bosquejo de Descripción:
El desarrollo comienza con una **descripción inicial** del sistema. Este bosquejo incluye los requisitos clave y proporciona una visión general del software que se va a construir.

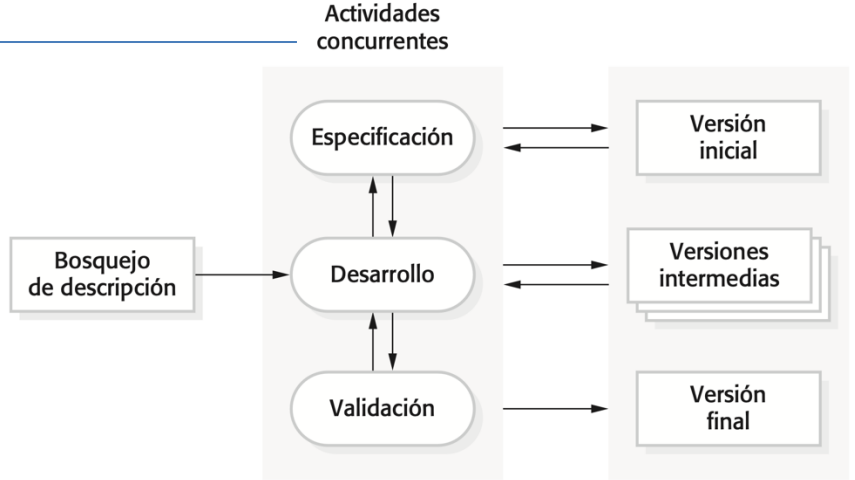


Modelo incremental

El **modelo incremental** es una metodología de desarrollo de software que se basa en la creación de software a través de múltiples **iteraciones** o **versiones**. En lugar de desarrollar todo el sistema en una sola fase, el software se construye por partes, permitiendo la entrega de versiones funcionales desde el inicio.

Actividades Concurrentes:
Las actividades de **especificación**, **desarrollo** y **validación** no son secuenciales, sino que se ejecutan de manera **concurrente** y cíclica. A medida que se avanza en cada ciclo, el sistema se amplía y mejora:

Especificación: Se identifican los requisitos del sistema y los objetivos para cada iteración.
Desarrollo: Se implementa una versión funcional del software basada en los requisitos especificados.
Validación: La versión desarrollada se prueba para asegurar que cumple con los requisitos.

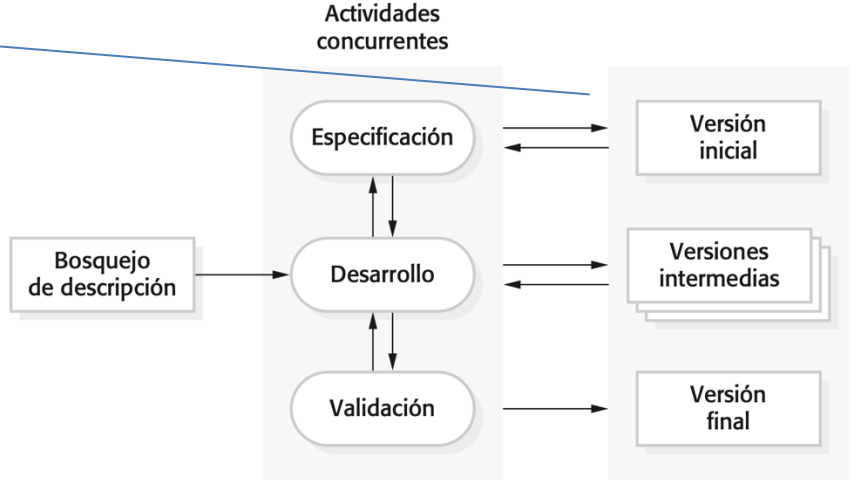


Modelo incremental

El **modelo incremental** es una metodología de desarrollo de software que se basa en la creación de software a través de múltiples **iteraciones** o **versiones**. En lugar de desarrollar todo el sistema en una sola fase, el software se construye por partes, permitiendo la entrega de versiones funcionales desde el inicio.

Iteraciones y Versiones:

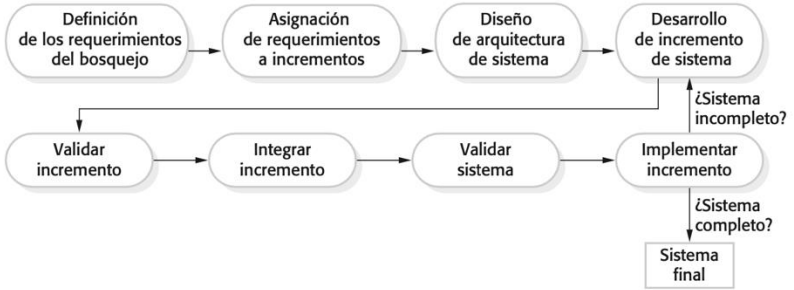
Versión inicial: La primera entrega es un prototipo funcional que cubre las funciones más básicas o críticas del sistema.
Versiones intermedias: Cada nueva iteración mejora o añade nuevas funcionalidades. Estas versiones pueden ser usadas y revisadas por los usuarios finales.
Versión final: Después de varias iteraciones, se llega a la versión final del sistema, que incluye todas las características y funcionalidades planificadas.



Modelo incremental

La **entrega incremental** consiste en desarrollar y entregar el sistema por fases o incrementos, donde cada incremento ofrece un subconjunto funcional del sistema. Este enfoque se caracteriza por:

• **Priorización de funcionalidades**: Los servicios más importantes para el cliente se desarrollan y entregan primero.
• **Iteraciones cortas**: Cada incremento proporciona valor inmediato al cliente y permite ajustar requisitos en base a retroalimentación temprana.



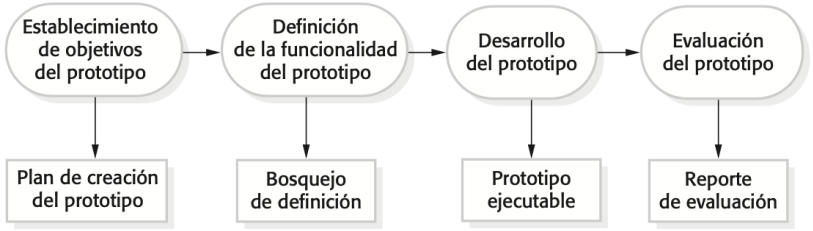
Ventajas:
• Los clientes pueden usar incrementos como prototipos y experimentar con el sistema.
• Se reduce el riesgo y se facilita la integración de cambios.

Ejemplo: Un sistema en el que primero se entrega la funcionalidad básica y, en fases posteriores, se añaden características adicionales.

Creación de prototipos

La **creación de prototipos** es un proceso iterativo que se utiliza para demostrar conceptos, explorar soluciones de diseño y obtener retroalimentación rápida. Un prototipo es una versión inicial del sistema, no necesariamente completa o robusta, que sirve para:

- **Validar requisitos**: Ayudar a los usuarios a visualizar funcionalidades y descubrir problemas.
- **Explorar opciones de diseño**: Probar alternativas de diseño de interfaces o estructura del sistema.



Ventajas:

- Reduce riesgos al validar ideas antes de la implementación
- Permite cambios tempranos en los requisitos.

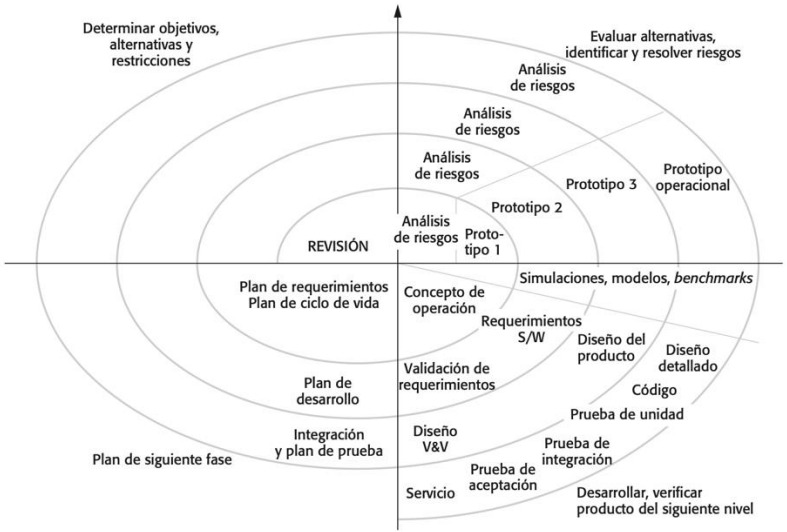
Limitaciones:

- La calidad del prototipo puede no reflejar la del sistema final.
- Los usuarios podrían malinterpretar el prototipo como el producto final.

Espiral de Boehm

El **modelo en espiral de Boehm** es un proceso de desarrollo de software basado en la gestión del riesgo, que combina elementos de desarrollo incremental y prototipos. Cada ciclo en la espiral representa una fase del desarrollo, y las decisiones se toman en función de la identificación y mitigación de riesgos.

Fases de la espiral:
1. **Establecimiento de objetivos**: Se definen los objetivos y se identifican los riesgos.
2. **Valoración y reducción de riesgos**: Se analizan y mitigan los riesgos mediante prototipos o simulaciones.
3. **Desarrollo y validación**: Se elige el modelo de desarrollo más adecuado, basado en los riesgos.
4. **Planeación**: Se decide si continuar con otra fase o ajustar el proyecto.



Ventajas:
- Proporciona un enfoque flexible y controlado para gestionar proyectos complejos.
- Se ajusta continuamente en base a riesgos identificados.

Modelo ágil

Las **metodologías ágiles** son un enfoque de desarrollo de software que se centra en la **flexibilidad**, **colaboración** y **entregas rápidas**. Permiten adaptarse a los cambios y ofrecen valor de manera continua a través de iteraciones cortas y retroalimentación constante. Algunas de las metodologías ágiles más populares incluyen:

- **Scrum**: Organiza el trabajo en ciclos cortos llamados **sprints** (generalmente de 2-4 semanas) para entregar incrementos funcionales del producto.
- **Kanban**: Utiliza un tablero visual para gestionar tareas y optimizar el flujo continuo de trabajo sin iteraciones fijas.
- **Extreme Programming (XP)**: Enfoca en **prácticas técnicas rigurosas** como **TDD** y **programación en pares** para mejorar la calidad y la capacidad de respuesta.
- **Lean Software Development**: Busca **eliminar desperdicios** y maximizar el valor entregado optimizando los recursos.
- **Crystal**: Adapta los procesos ágiles según el tamaño y criticidad del proyecto, priorizando la **comunicación y colaboración**.

Proceso Unificado Racional (RUP)

El **Proceso Unificado Racional (RUP, por las siglas de *Rational Unified Process*)** es un ejemplo de un modelo de proceso moderno que se derivó del trabajo sobre el UML y el proceso asociado de desarrollo de software unificado.

Conjunta elementos de todos los modelos de proceso genéricos, ilustra la buena práctica en especificación y diseño, y apoya la creación de prototipos y entrega incremental.

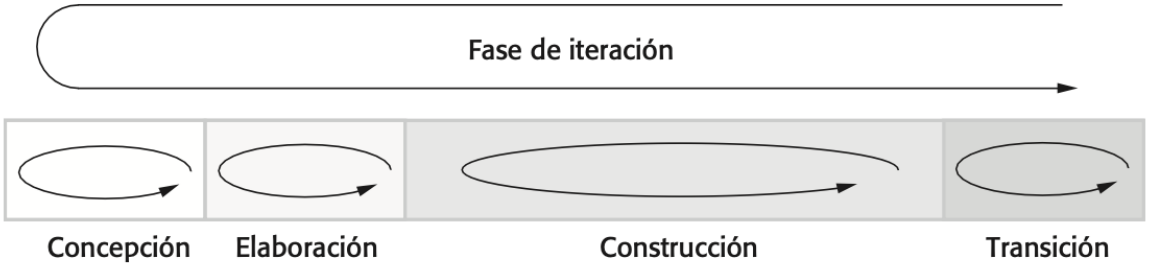
El RUP reconoce que los modelos de proceso convencionales presentan una sola visión del proceso. En contraste, el RUP por lo general se describe desde tres perspectivas:

- Una perspectiva **dinámica** que muestra las fases del modelo a través del tiempo.
- Una perspectiva **estática** que presenta las actividades del proceso que se establecen.
- Una perspectiva **práctica** que sugiere buenas prácticas a usar durante el proceso.

Proceso Unificado Racional (RUP) – Fases

El **Rational Unified Process (RUP)** se organiza en cuatro fases discretas que gestionan el ciclo de vida del desarrollo de software:

1. **Concepción**: Se establece un caso empresarial para el sistema, identificando las entidades externas y evaluando el impacto del sistema en la empresa.
2. **Elaboración**: Se desarrolla una comprensión profunda del dominio del problema, se establece una arquitectura base y se identifican los riesgos clave.
3. **Construcción**: En esta fase se diseña, programa y prueba el sistema, desarrollando partes en paralelo e integrándolas progresivamente.
4. **Transición**: Se transfiere el sistema a la comunidad de usuarios, desplegándolo en el entorno operativo real.



Proceso Unificado Racional (RUP) – Buenas prácticas

El RUP promueve seis buenas prácticas clave para el desarrollo de software:

1. **Desarrollo iterativo**: Dividir el desarrollo en ciclos iterativos para gestionar mejor el riesgo.
2. **Gestión de requerimientos**: Documentar y rastrear los cambios en los requerimientos del sistema.
3. **Uso de arquitecturas basadas en componentes**: Estructurar el sistema en componentes modulares.
4. **Modelado visual del software**: Utilizar UML para representar gráficamente el sistema.
5. **Verificación continua de la calidad**: Asegurar que el software cumple con los estándares de calidad definidos.
6. **Gestión de cambios y configuración**: Controlar y gestionar los cambios al sistema de manera organizada.

El RUP no es un proceso adecuado para todos los tipos de desarrollo, por ejemplo, para desarrollo de software embebido. Sin embargo, sí representa un enfoque que potencialmente combina los tres modelos de proceso genéricos: Cascada, Incremental y Reutilización.

Universidad Intercontinental de la Empresa

Asignatura

Arquitectura del Software

Profesor

Yago Fontenla Seco

{*yago.fontenla1@uie.edu*}