TEMAS SELECTOS DE INGENIERÍA DE SOFTWARE

PROCESOS DE SOFTWARE

Introducción

El proceso de software es una serie de actividades relacionadas que conducen al desarrollo de un producto de software.

Las actividades fundamentales comunes a todos los procesos son:

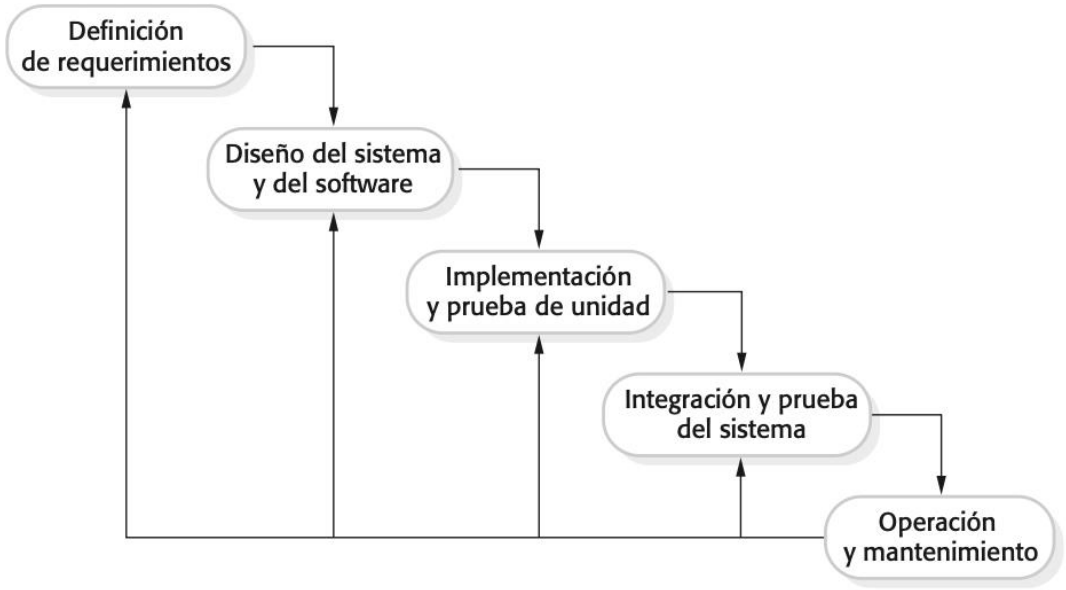
- Especificación del software: Definir funcionalidad y limitaciones.
- Diseño e implementación: Construir el software según especificaciones.
- Validación: Asegurar conformidad con requisitos del cliente.
- Evolución: Adaptar el software a cambios futuros.

¿Qué es un Modelo de Proceso de Software?

Resumen Modelos de Proceso

Modelo	Características Principales	Uso adecuado
Modelo en Cascada (Waterfall)	Desarrollo secuencial por fases claramente definidas	Requisitos estables, proyectos sencillos
Desarrollo Incremental	Versiones sucesivas con funcionalidades añadidas	Cambios frecuentes, sistemas empresariales
Ingeniería orientada a Reutilización	Integración de componentes previamente desarrollados	Sistemas con componentes disponibles
Prototipado	Construcción de versiones preliminares para clarificar requisitos	Cuando la especificación es incierta
Modelo en Espiral (Boehm)	Ciclos iterativos con enfoque en gestión de riesgos	Proyectos de alto riesgo o críticos

Modelo en Cascada

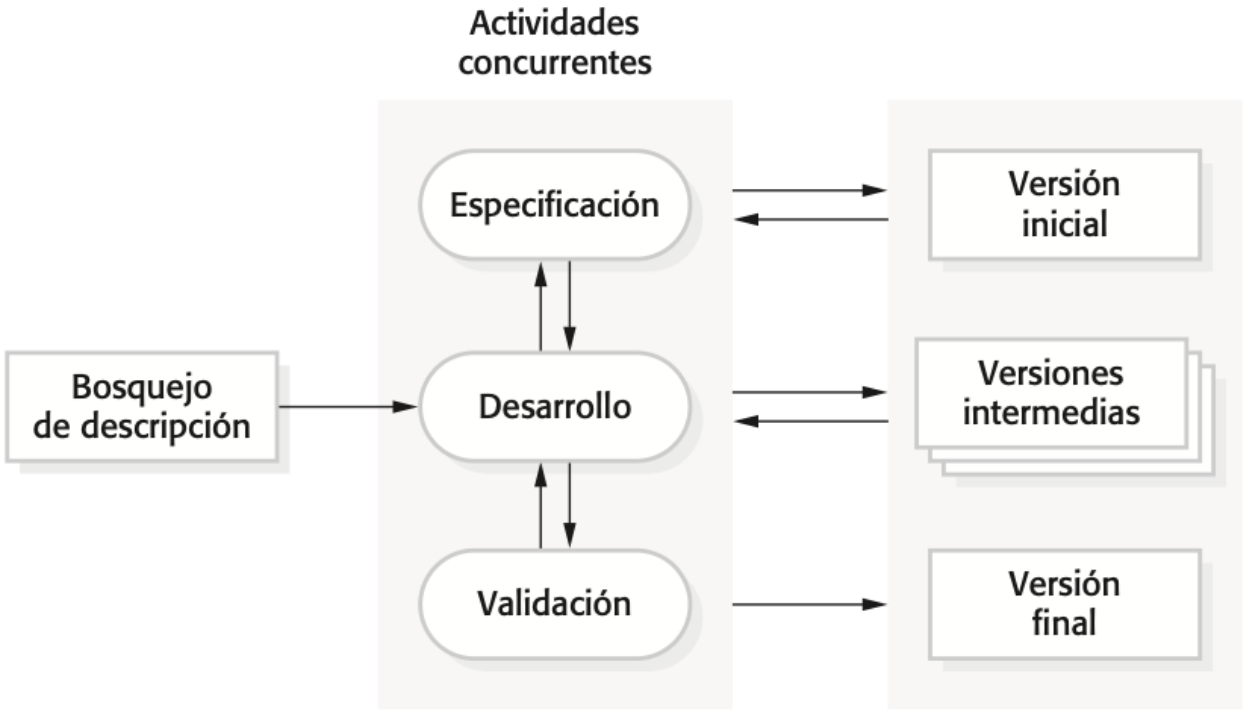


- El proceso clásico por etapas:

1. Requerimientos
2. Diseño
3. Implementación y prueba de unidades
4. Integración y prueba de sistema
5. Operación y mantenimiento

- Ventajas: fácil de entender, gestión visible con documentación formal.
- Desventajas: rígido, difícil adaptación a cambios, retroalimentación tardía.
- Uso: proyectos con requisitos muy estables y bien entendidos.

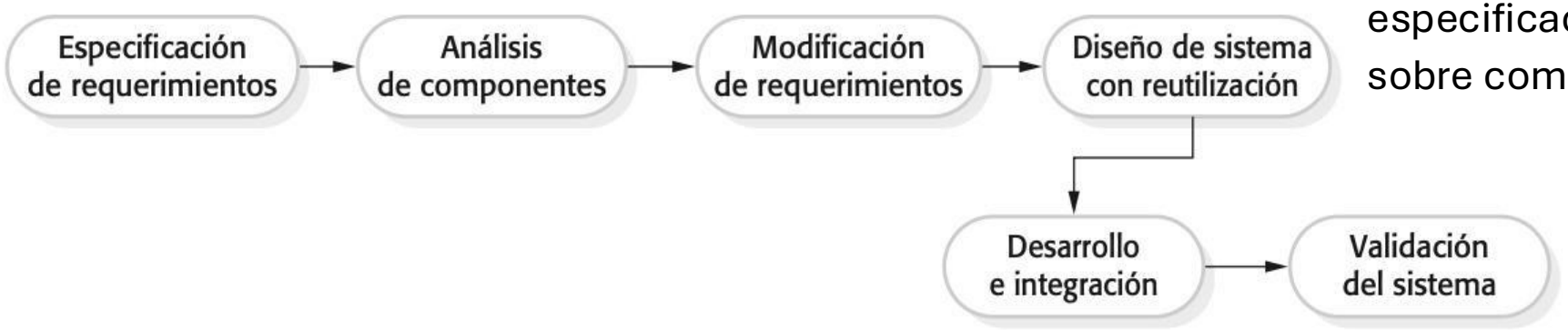
Desarrollo Incremental



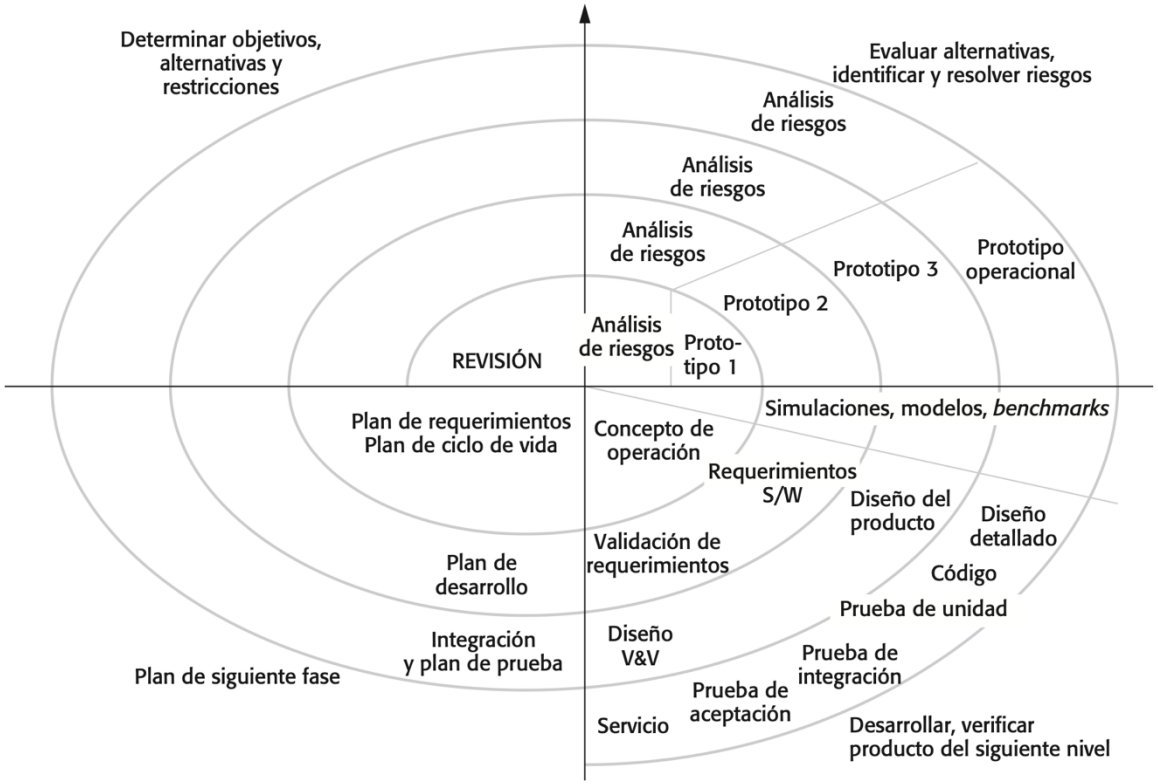
- El sistema se construye en versiones o incrementos funcionales.
- Actividades de especificación, desarrollo y validación entrelazadas.
- Permite retroalimentación temprana y adaptación a cambios.
- Ventajas: reduce costos de adaptación, entrega funcional temprana y mejora la satisfacción del cliente.
- Desventajas: riesgo de degradación de la estructura, coordinación compleja en grandes proyectos.

Ingeniería de Software Orientada a la Reutilización

- El desarrollo se basa en componentes reutilizables y sistemas comerciales (COTS).
- Etapas incluyen análisis de componentes, modificación de requerimientos, diseño con reutilización e integración.
- Ventajas: reducción de costos y riesgos, entregas más rápidas.
- Desventajas: compromiso en especificaciones, pérdida de control sobre componentes reutilizados.



Modelo en Espiral de Boehm



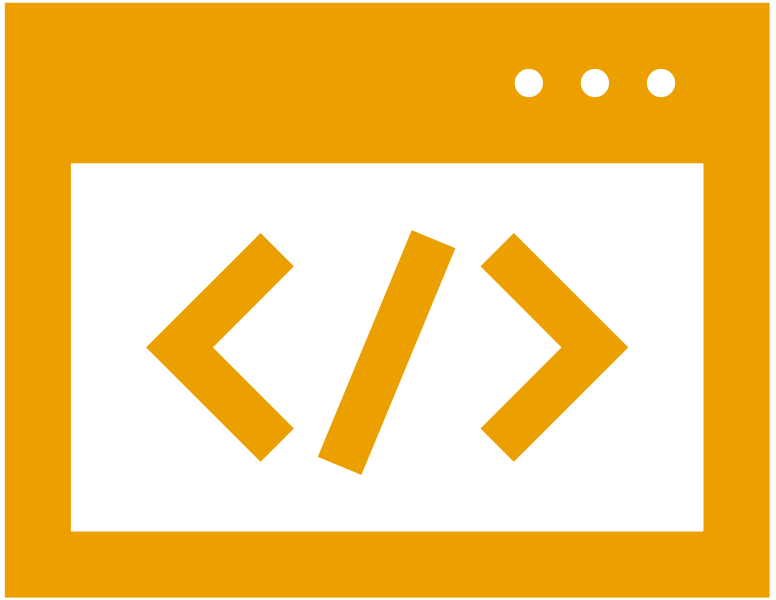
- Generador de modelo de proceso impulsado por el riesgo
- Cada loop en la espiral representa una fase del proceso
- Cuatro sectores principales:
  - Establecimiento de objetivos
  - Evaluación y reducción de riesgos
  - Desarrollo y validación
  - Planificación
- Especialmente útil para proyectos grandes y complejos

Actividades del proceso

Todas las metodologías de desarrollo incluyen cuatro actividades fundamentales:

1. Especificación de software
2. Diseño e implementación
3. Validación de software
4. Evolución del software

Diferentes modelos organizan estas actividades de formas distintas



Especificación de Software

Proceso de establecer qué servicios son requeridos

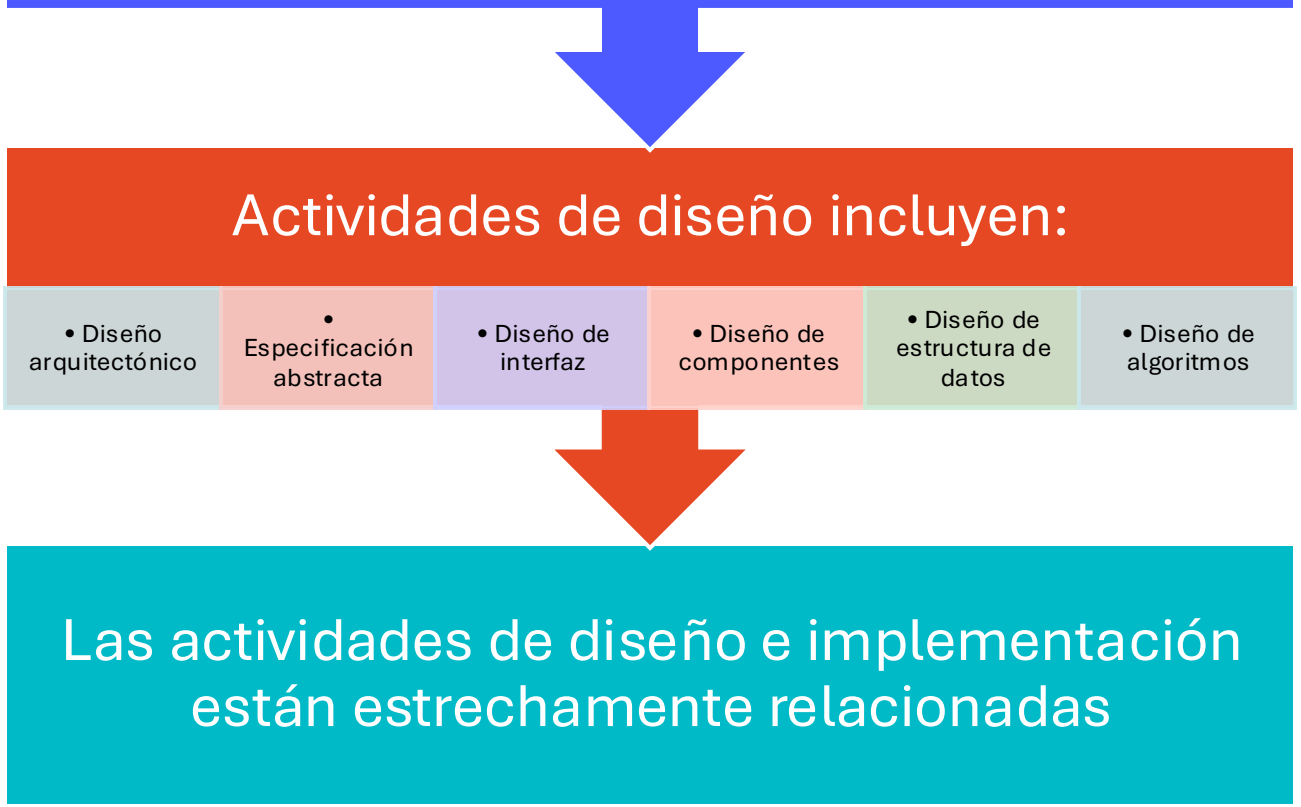
Define las restricciones de operación y desarrollo del sistema

Proceso de ingeniería de requisitos incluye:

- Estudio de factibilidad
- Elicitación y análisis de requisitos
- Especificación de requisitos
- Validación de requisitos

Diseño e implementación

Proceso de convertir la especificación del sistema en un sistema ejecutable



Validación de software

Verificación y validación (V&V) demuestra que el sistema cumple con su especificación

Involucra procesos de verificación, revisión y pruebas del sistema

Etapas de prueba:

- Pruebas de desarrollo/componente
- Pruebas de sistema
- Pruebas de aceptación

Es un proceso iterativo con retroalimentación entre etapas

Evolución del software

El software es inherentemente flexible y puede cambiar

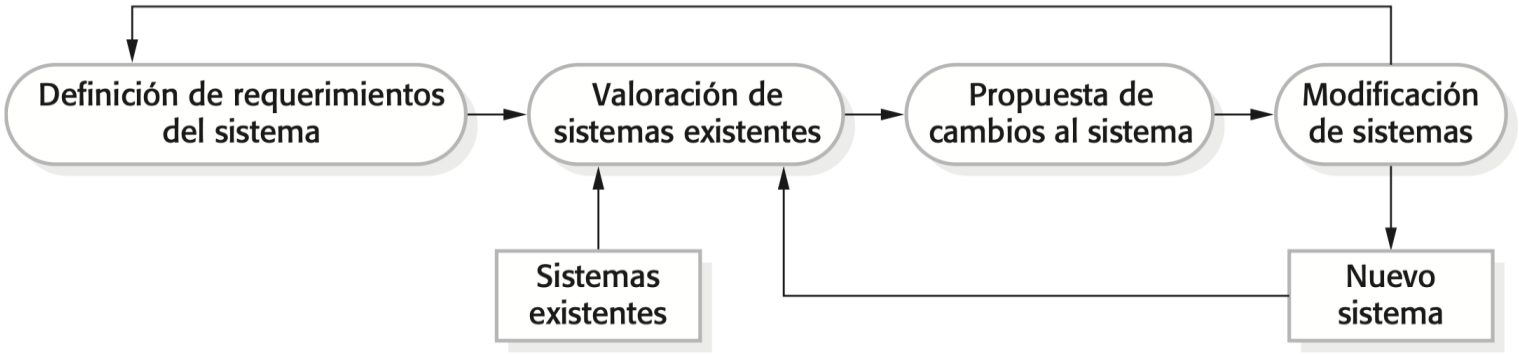
Los requisitos cambian debido a circunstancias de negocio cambiantes

El software debe evolucionar y cambiar para soportar el negocio

La distinción entre desarrollo y evolución (mantenimiento) es cada vez menos relevante

Pocos sistemas son completamente nuevos

Evolución del software



Los principios de los métodos ágiles

Principio	Descripción
Participación del cliente	Los clientes deben intervenir estrechamente durante el proceso de desarrollo. Su función consiste en ofrecer y priorizar nuevos requerimientos del sistema y evaluar las iteraciones del mismo.
Entrega incremental	El software se desarrolla en incrementos y el cliente especifica los requerimientos que se van a incluir en cada incremento.
Personas, no procesos	Tienen que reconocerse y aprovecharse las habilidades del equipo de desarrollo. Debe permitirse a los miembros del equipo desarrollar sus propias formas de trabajar sin procesos establecidos.
Adoptar el cambio	Esperar a que cambien los requerimientos del sistema y, de este modo, diseñar el sistema para adaptar dichos cambios.
Mantener simplicidad	Enfocarse en la simplicidad tanto en el software a desarrollar como en el proceso de desarrollo. Siempre que sea posible, trabajar de manera activa para eliminar la complejidad del sistema.

Limitaciones del Modelo Cascada en la Práctica

Casos reales de fracasos documentados:
FBI Sentinel System: $400 millones perdidos, tuvo que cambiar a metodología ágil
Microsoft Windows Vista: Proceso rígido resultó en problemas de usabilidad y rendimiento
Hershey's ERP Rollout: Enfoque 'big bang' causó pérdidas millonarias
Proyectos Telecom CRM: Abandonados por incapacidad de manejar requisitos cambiantes
CHAOS Report (1994): 31.1% de proyectos fallaron, 52.7% excedieron costos y tiempo
"Los proyectos Waterfall fallan cuando los requisitos cambian frecuentemente"

El Cambio de Paradigma - Década de los 90s

Factores que generaron el cambio:

Masificación del Internet (mediados de los 90s)

Proyectos con requisitos altamente cambiantes

Necesidad de tiempos de entrega muy cortos

Competencia feroz en mercados tecnológicos

Clientes demandaban mayor participación en el proceso

Surgimiento de 'metodologías livianas':

Scrum (1993) - Jeff Sutherland

Extreme Programming XP (1996)

Crystal Clear (1996)

Feature-Driven Development (1997)

Problema central: Se pasaba más tiempo pensando en el diseño que desarrollando software

El Manifiesto Ágil - 2001

17 desarrolladores se reunieron en Snowbird, Utah para definir una alternativa

Los 4 Valores Fundamentales:

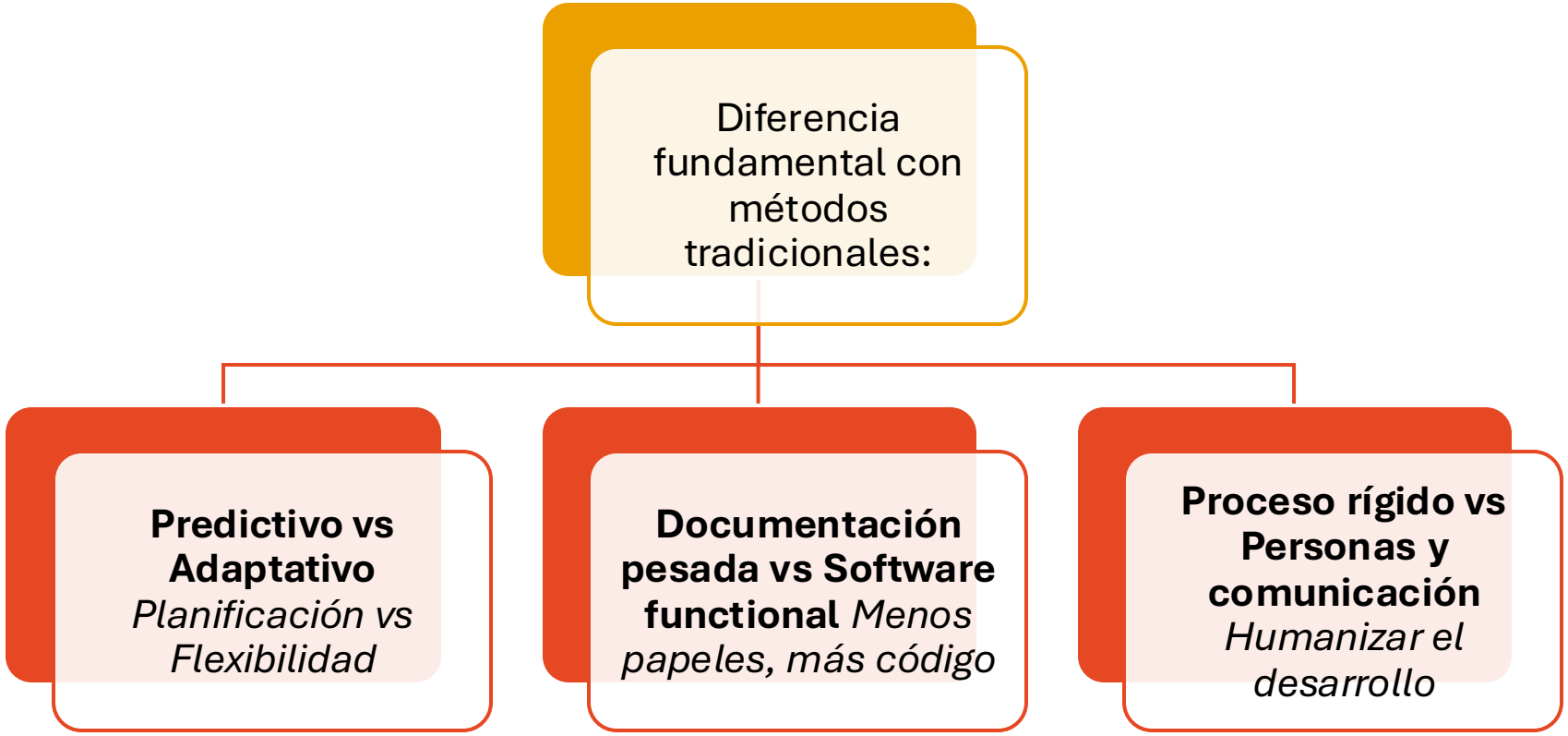
- Individuos e interacciones sobre procesos y herramientas
- Software funcionando sobre documentación extensiva
- Colaboración con el cliente sobre negociación contractual
- Respuesta ante el cambio sobre seguir un plan

Métodos Ágiles - Características Fundamentales

Características clave de los métodos ágiles:

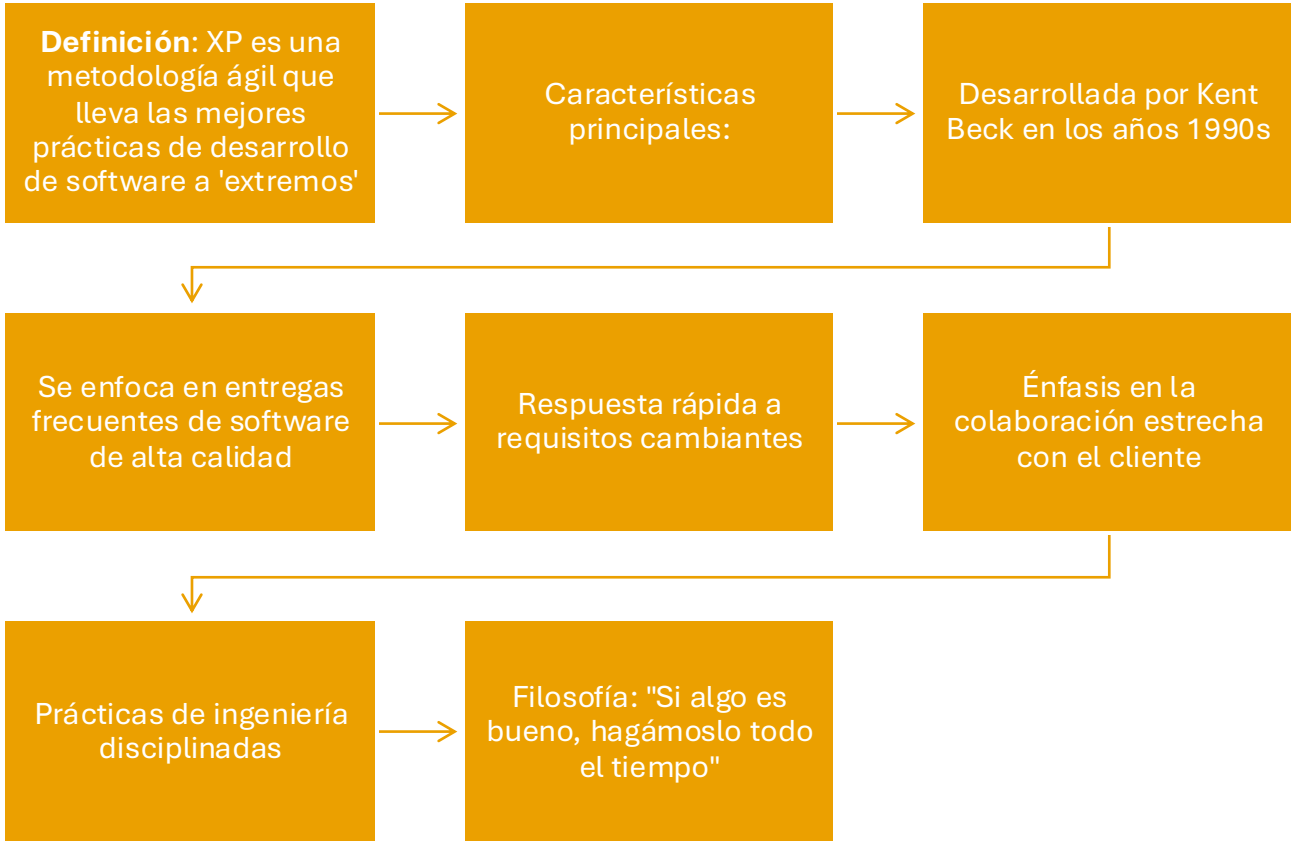
- Desarrollo iterativo e incremental - Construcción por etapas
- Entregas frecuentes de software funcional
- Participación activa del cliente durante todo el proyecto
- Equipos multifuncionales y autoorganizados
- Alta adaptabilidad a cambios de requisitos

Métodos Ágiles - Características Fundamentales



Extreme Programming (XP)

¿Qué es Extreme Programming (XP)?



Los 5 Valores Fundamentales de XP

💬 Communication (Comunicación) Comunicación constante entre todos los miembros del equipo, cliente y stakeholders

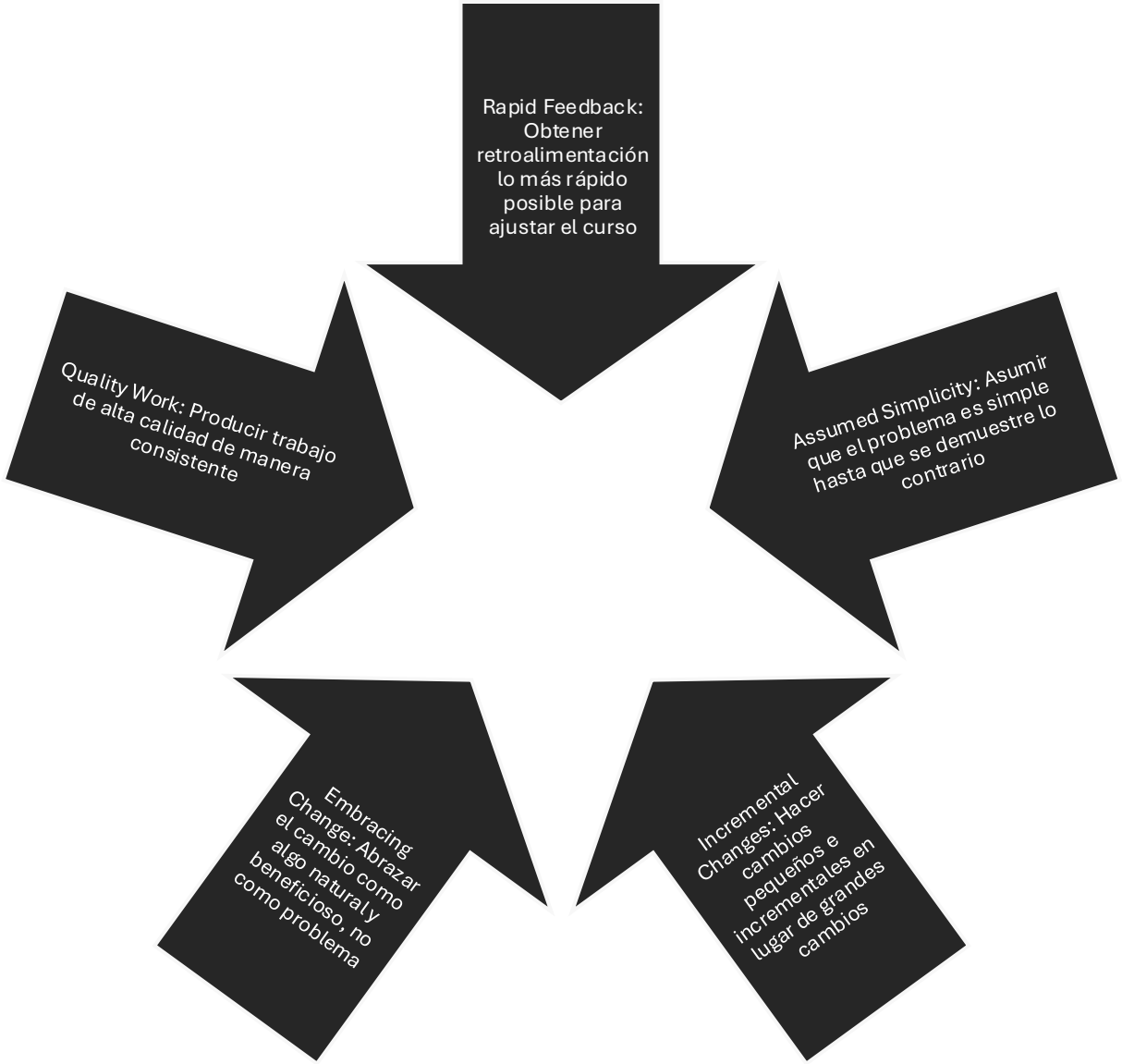
⚡ Simplicity (Simplicidad) Hacer lo más simple que funcione, evitar complejidad innecesaria

🔄 Feedback (Retroalimentación) Retroalimentación temprana y continua de código, diseño y usuarios

💪 Courage (Valentía) Coraje para hacer cambios necesarios, refactorizar código y decir la verdad

🤝 Respect (Respeto) Respeto mutuo entre todos los miembros del equipo y sus contribuciones

Principios de XP



Prácticas Clave de XP - Planificación

Planning Game (Juego de Planificación)

- Release Planning: Decidir qué funcionalidades van en cada release
- Iteration Planning: Planificar el trabajo de cada iteración (1-2 semanas)
- Tres fases: Exploración, Compromiso y Dirección

Otras Prácticas de Planificación

- On-site Customer: Cliente disponible tiempo completo
- Small Releases: Releases pequeños y frecuentes (cada 2-4 semanas)
- User Stories: Requisitos escritos en tarjetas desde perspectiva del usuario

Prácticas Clave de XP - Desarrollo

Test-Driven Development (TDD)

- Escribir pruebas antes que el código
- Código debe pasar todas las pruebas unitarias

Pair Programming: Dos programadores, un teclado - colaboración continua

Collective Code Ownership: Cualquiera puede modificar cualquier código del sistema

Continuous Integration: Integración continua del código varias veces al día

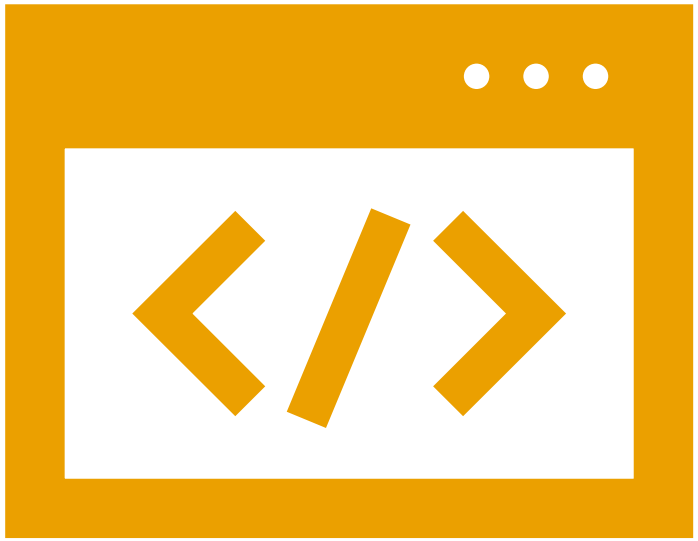
Refactoring: Mejora continua del diseño del código sin cambiar funcionalidad

Simple Design: Diseño simple que cumple únicamente los requisitos actuales

Prácticas Clave de XP - Gestión

- Sustainable Pace: Ritmo de trabajo sostenible (máximo 40 horas/semana)
- Coding Standards: Estándares de codificación consistentes para todo el equipo
- System Metaphor: Metáfora simple que describe cómo funciona el sistema
- Acceptance Tests: Pruebas definidas por el cliente para validar funcionalidades
- Open Workspace: Espacio de trabajo abierto que facilite la comunicación
- Just Rules: Reglas simples que el equipo acuerda seguir

Ventajas y limitaciones de XP



✅ **Ventajas**

- Alta calidad de código (TDD, refactoring, pair programming)
- Respuesta rápida a cambios de requisitos
- Alta satisfacción del cliente por entregas frecuentes
- Conocimiento compartido del equipo
- Entregas frecuentes de valor al negocio
- Reducción de riesgos por feedback temprano

⚠️ **Limitaciones**

- Requiere cliente dedicado tiempo completo
- Difícil de implementar con equipos distribuidos
- No funciona bien con equipos muy grandes (>12 personas)
- Resistencia cultural a pair programming
- Poca documentación formal puede ser problemática
- Requiere desarrolladores experimentados y disciplinados

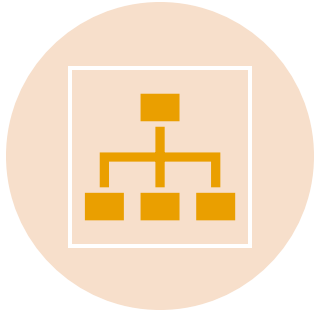
SCRUM

¿Qué es Scrum?

Scrum es un marco de trabajo ágil para desarrollo de productos complejos.

Basado en iteraciones cortas llamadas Sprints (¡inspección y adaptación!).

El equipo es autoorganizado, enfocado en entregar valor de negocio constante en cada Sprint.



Scrum se apoya en transparencia, inspección y adaptación.

Roles en Administración Ágil

👥 Product Owner

- Define y prioriza el Product Backlog
- Representa la voz del cliente/negocio
- Acepta o rechaza entregables completados
- Participa activamente en Planning Game

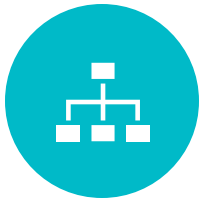
🎯 Scrum Master/Coach

- Facilita el proceso ágil y ceremonias
- Elimina impedimentos del equipo
- Protege al equipo de distracciones externas
- Coaching en prácticas ágiles

💻 Development Team

- Auto-organizado y multifuncional
- Responsable de entregar incrementos de software
- Estima esfuerzo de user stories
- Implementa prácticas técnicas (TDD, refactoring, etc.)

Eventos Principales de Scrum



Sprint: Ciclo de trabajo de 1-4 semanas.

Sprint Planning: Planificación del Sprint: ¿qué vamos a hacer y cómo?

Daily Scrum: Reunión diaria de 15 minutos, sincronización y ajuste.

Sprint Review: Revisión del Sprint, demostración del incremento y feedback.

Sprint Retrospective: Reflexión final para mejorar proceso y equipo en el próximo Sprint.

Artefactos Scrum

Product Backlog:

- Lista priorizada de requisitos del producto (historias, tareas, mejoras, errores).

Sprint Backlog:

- Conjunto de tareas seleccionadas para el Sprint actual (plan detallado).

Increment:

- El resultado funcional (software listo para producir valor).
- Debe cumplir “Definition of Done” acordada por el equipo.

Flujo del Proceso Scrum

1. El Product Owner prioriza el Product Backlog.
2. En el Sprint Planning, el equipo selecciona los elementos a trabajar.
3. Cada día, el equipo revisa su avance y ajusta el plan en el Daily Scrum.
4. Al finalizar el Sprint, se presenta el resultado (Sprint Review).
5. Se analiza y mejora el proceso en la Retrospectiva.
6. Se actualiza el Product Backlog y comienza un nuevo Sprint.

GRACIAS POR SU ATENCIÓN!