Proceso Software y Ciclo de Vida

Conceptos importantes

- Personas: los que trabajan
- Producto: lo que se obtiene
- Proyecto: la pauta a seguir para desarrollar un producto
- Proceso: la pauta a seguir para desarrollar un proyecto

	Personas	Producto	Proyecto	Proceso
TRAJE	El sastre	El traje	el sastre, el traje, el presupue el sastre, el traje, el presupuesto del traje, el traje en sí, los pasos a dar para hacer el traje	La secuencia de acciones para hacer un traje concreto La secuencia de acciones para hacer un traje concreto
CENA	Empleados de una empresa de catering Empleados de una empresa de catering	La cena que se sirve	El menú, el presupuesto, lo que hay que hacer para , el presupuesto, lo que hay que hacer para conseguir el men conseguir el menú	La secuencia de acciones de servir una cena La secuencia de acciones de servir una cena
MARCA DE AUTO	Empleados de la marca	Los autos	Desarrollo de un modelo nuevo	Las instrucciones de la empresa sobre c Las instrucciones de la empresa sobre cómo desarrollar un modelo nuevo desarrollar un modelo nuevo
USTEDES	Su grupo	la aplicación elegida	parte por parte práctica IS	entregas mensuales cómo ustedes deciden organizarse

Ingeniería de Software

Pressman (Un enfoque práctico 1997) caracteriza la Ingeniería de Software como “una tecnología multicapa”

El objetivo de la IS es lograr productos de software de calidad (tanto en su forma final como durante su elaboración), mediante un proceso apoyado por métodos y herramientas.

Cualquier disciplina de ingeniería (incluida la ingeniería del software) debe descansar sobre un esfuerzo de organización de **calidad**. La gestión total de la calidad y las filosofías similares fomentan una cultura continua de mejoras de procesos que conduce al desarrollo de enfoques cada vez más robustos para la IS.

Base de cualquier proceso de ingeniería.La IS se basa en calidad

El fundamento de la IS es la **capa proceso**. El proceso define un marco de trabajo para un conjunto de áreas clave, las cuales forman la base del control de gestión de proyectos de software y establecen el contexto en el cual: se aplican los métodos técnicos, se producen resultados de trabajo, se establecen hitos, se asegura la calidad y el cambio se gestiona adecuadamente.

Capa que une calidad y métodos. Desarrollo racional de la IS

Los **métodos** de la IS indican cómo construir técnicamente el software. Los métodos abarcan una gran gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento. Estos métodos dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.

Las **herramientas** de la IS proporcionan un soporte automático o semi-automático para el proceso y los métodos, a estas herramientas se les llama herramientas CASE (C*omputer-Aided Software Engineering//Ingeniería de Software Asistida por Computadora*).

El proceso de desarrollo del software

Un proceso de desarrollo de software tiene como propósito la producción eficaz y eficiente de un producto software que reúna los requisitos del cliente.

Un producto software en sí es complejo, es prácticamente inviable conseguir un 100% de confiabilidad de un programa por pequeño que sea. Existe una inmensa combinación de factores que impiden una verificación exhaustiva de las todas posibles situaciones de ejecución que se puedan presentar (entradas, valores de variables, datos almacenados, software del sistema, otras aplicaciones que intervienen, el hardware sobre el cual se ejecuta, etc.).

Un producto software es intangible y por lo general muy abstracto, esto dificulta la definición del producto y sus requisitos, sobre todo cuando no se tiene precedentes en productos software similares. Esto hace que los requisitos sean difíciles de consolidar tempranamente. Así, los cambios en los requisitos son inevitables, no sólo después de entregado en producto sino también durante el proceso de desarrollo.

El proceso de desarrollo de software no es único. No existe un proceso de software universal que sea efectivo para todos los contextos de proyectos de desarrollo. Debido a esta diversidad, es difícil automatizar todo un proceso de desarrollo de software.

A pesar de la variedad de propuestas de proceso de software, existe un conjunto de actividades fundamentales que se encuentran presentes en todos ellos.

Especificación de software: Se debe definir la funcionalidad y restricciones operacionales que debe cumplir el software.

Diseño e Implementación: Se diseña y construye el software de acuerdo a la especificación.

Validación: El software debe validarse, para asegurar que cumpla con lo que quiere el cliente.

Evolución: El software debe evolucionar, para adaptarse a las necesidades del cliente.

Además de estas actividades fundamentales, Pressman menciona un conjunto de “actividades protectoras”, que se aplican a lo largo de todo el proceso del software:

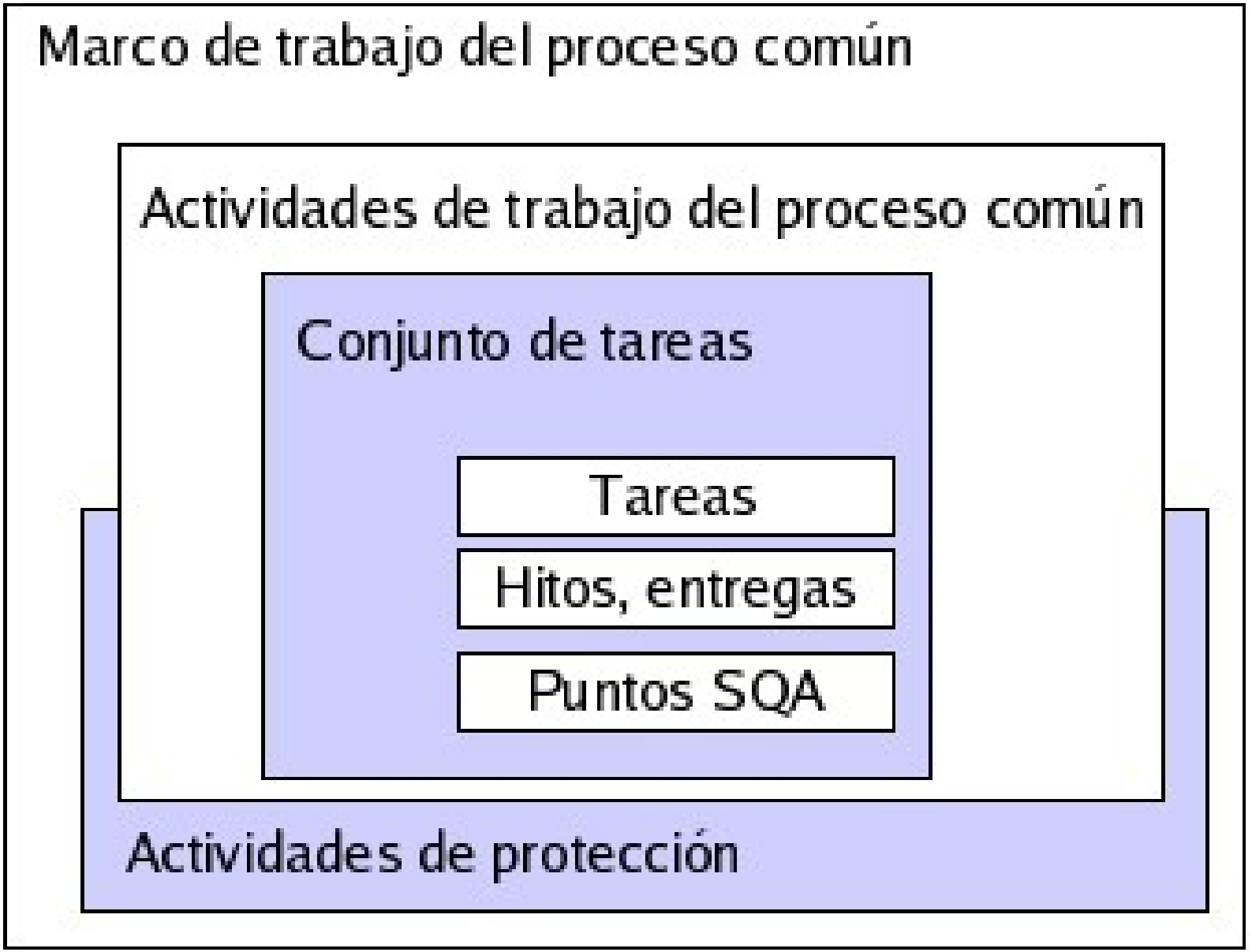
*Seguimiento y control de proyecto de software.
*Revisiones técnicas formales.
*Garantía de calidad del software.
*Gestión de configuración del software.
*Preparación y producción de documentos.
*Gestión de reutilización.
*Mediciones.
*Gestión de riesgos.

Pressman caracteriza un proceso de desarrollo de software con los siguientes elementos:

Un marco común del proceso, definiendo un pequeño número de actividades del marco de trabajo que son aplicables a todos los proyectos de software, con independencia del tamaño o complejidad.

Un conjunto de tareas, cada uno es una colección de tareas de IS, hitos de proyectos, entregas y productos de trabajo del software, y puntos de garantía de calidad, que permiten que las actividades del marco de trabajo se adapten a las características del proyecto de software y los requisitos del equipo del proyecto.

Las actividades de protección, tales como garantía de calidad del software, gestión de configuración del software y medición, abarcan el modelo del proceso. Las actividades de protección son independientes de cualquier actividad del marco de trabajo y aparecen durante todo el proceso.

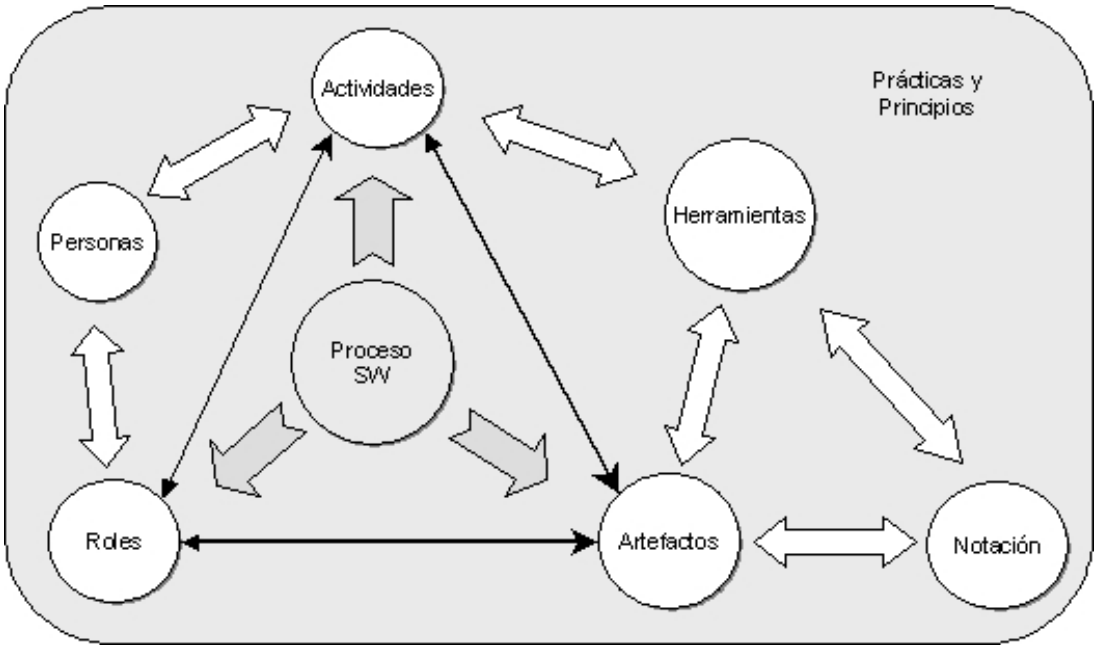

Marco de trabajo del proceso común
Actividades de trabajo del proceso común
Conjunto de tareas
Tareas
Hitos, entregas
Puntos SQA
Actividades de protección

Otra perspectiva utilizada para determinar los elementos del proceso de desarrollo de software es establecer las relaciones entre elementos que permitan responder **Quién** debe hacer **Qué**, **Cuándo** y **Cómo** debe hacerlo

Quién: Las Personas participantes en el proyecto de desarrollo desempeñando uno o más Roles específicos.

Qué: Un artefacto es producido por un rol en una de sus actividades. Los artefactos se especifican utilizando notaciones específicas. Las herramientas apoyan la elaboración de artefactos soportando ciertas notaciones.

Cómo y Cuándo: Las Actividades son una serie de pasos que lleva a cabo un rol durante el proceso de desarrollo. El avance del proyecto está controlado mediante hitos que establecen un determinado estado de terminación de ciertos Artefactos.



Modelos de proceso

Un modelo de proceso, o paradigma de IS, es una plantilla, patrón o marco que define el proceso a través del cual se crea software

Dicho de otra forma, los procesos son instancias de un modelo de proceso

En esta asignatura los términos proceso y modelo de proceso se utilizan indistintamente

- Una organización podría variar su modelo de proceso para cada proyecto, según:

  La naturaleza del proyecto

  La naturaleza de la aplicación

  Los métodos y herramientas a utilizar

  Los controles y entregas requeridas

Sommerville (Ingeniería de Software,2002) 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.”

Características del proceso

- Entendible
- Visibilidad: Grado en que las actividades del proceso proporcionan resultados
- Soportable por herramientas CASE
- Aceptabilidad: Grado en que los desarrolladores aceptan y usan el proceso
- Fiabilidad: Capacidad de evitar o detectar errores antes de que sean defectos
- Robustez: Continuidad del proceso a pesar de los problemas
- Mantenible: Capacidad de evolución para adaptarse
- Rapidez: Velocidad en que el proceso puede proporcionar un sistema a partir de una especificación

Modelos Genéricos de Desarrollo de Software

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.

*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 y 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 (Waterfall)

El primer modelo de desarrollo de software que se publicó en 1970. 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.

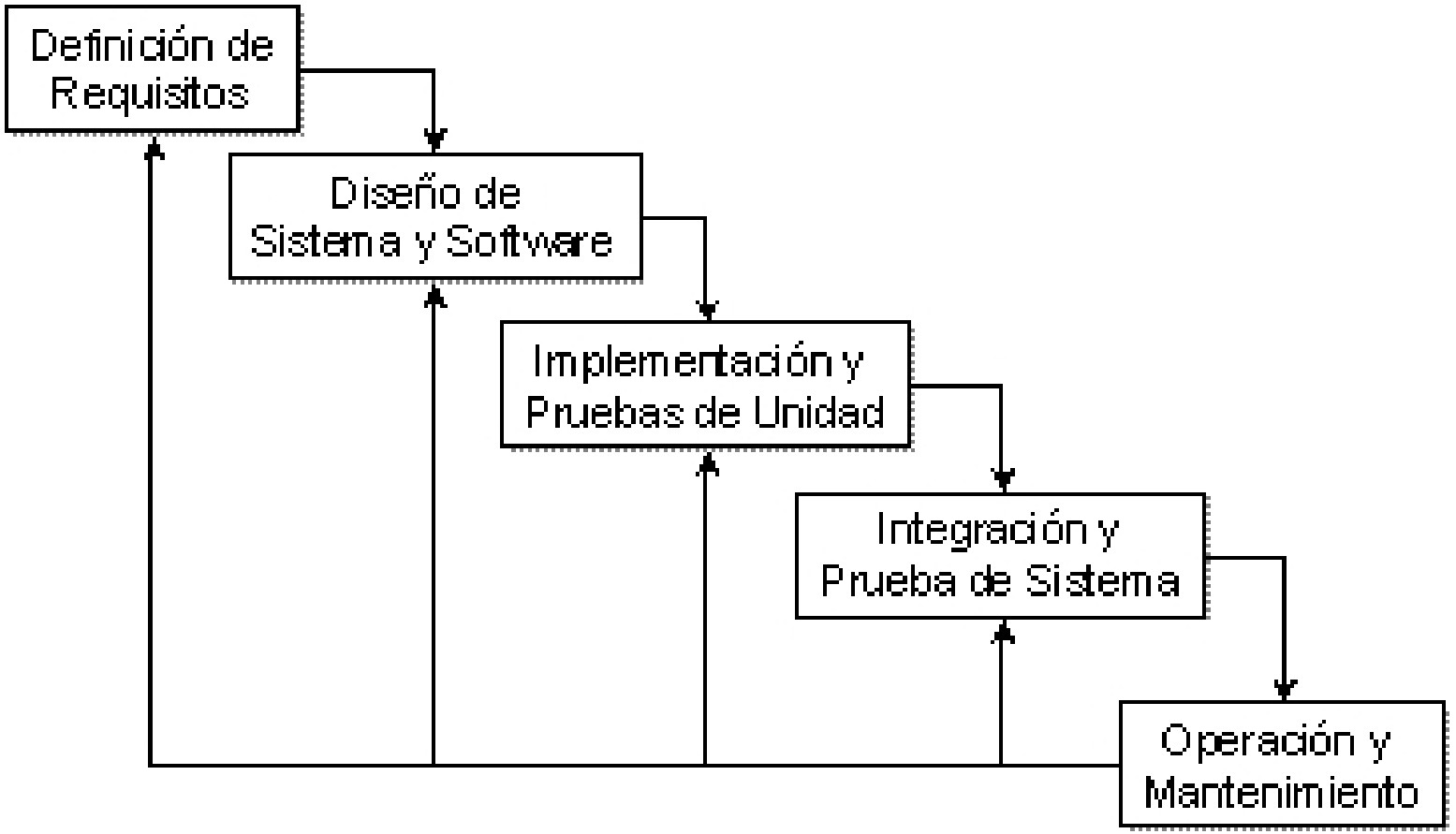
Implementación y pruebas unitarias: Construcción de los módulos y unidades de software. Se realizan pruebas de cada unidad.

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.

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.



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.

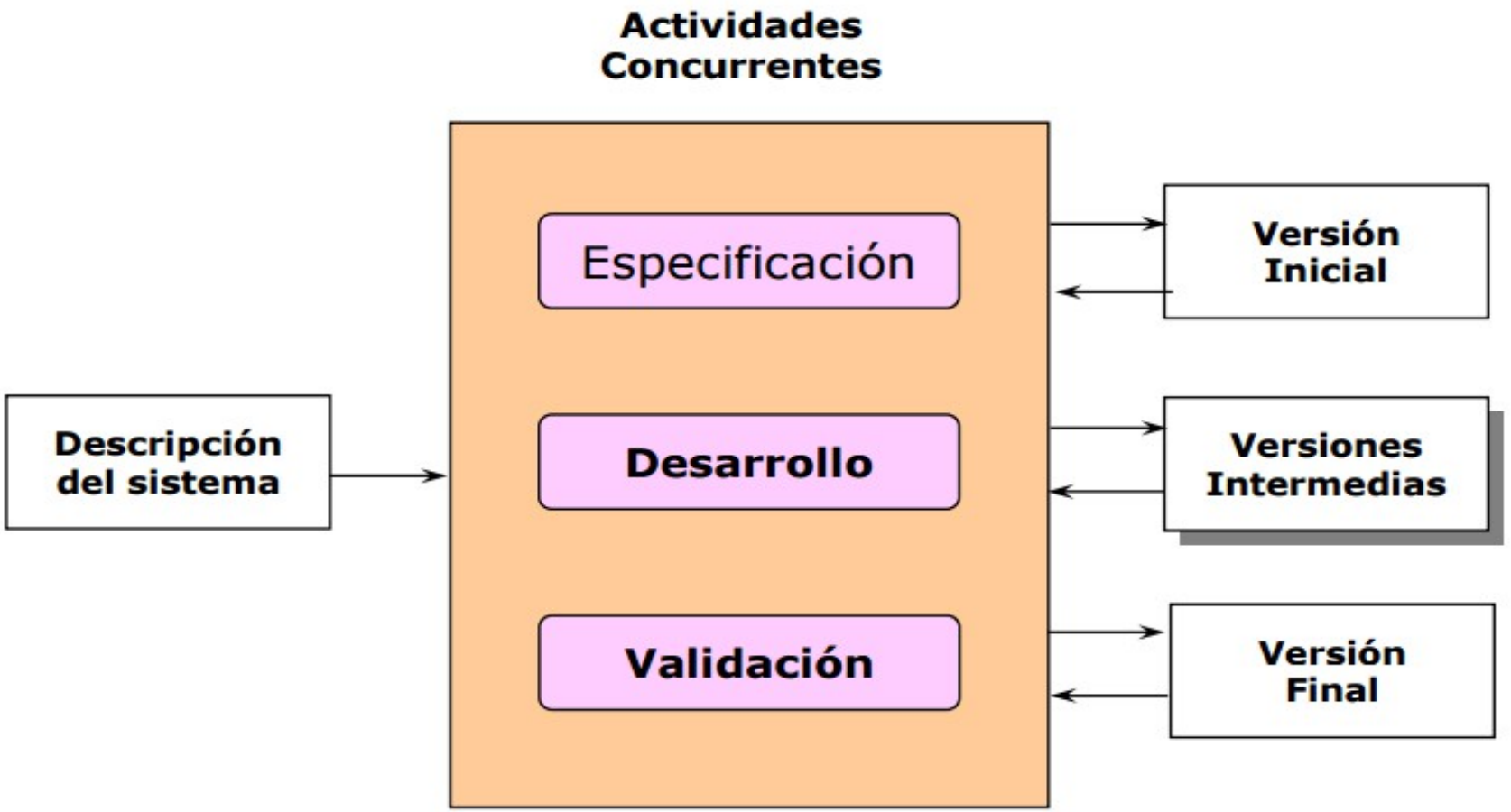
Este modelo sólo debe usarse si se entienden a plenitud los requisitos. Aún se utiliza como parte de proyectos grandes.

Desarrollo evolutivo

La idea detrás de este modelo es el desarrollo de una implantación del sistema inicial, exponerla a los comentarios del usuario, refinarla y generar N versiones hasta que se desarrolle el sistema adecuado.

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.


Actividades
Concurrentes
Descripción
del sistema
Especificación
Desarrollo
Validación
Versión
Inicial
Versiones
Intermedias
Versión
Final

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.

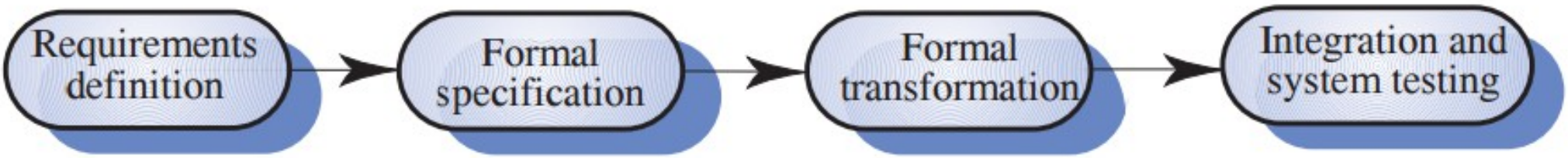
Ilustra un paradigma ideal de programación automática. Se distinguen dos fases globales: **especificación** (incluyendo validación) y **transformación**. Las características principales de este paradigma son: la especificación es formal y ejecutable (constituye el primer prototipo del sistema), la especificación 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 (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.


Requirements definition
Formal specification
Formal transformation
Integration and system testing

Desarrollo basado en reutilización

Fuertemente orientado a la reutilización. Este modelo consta de 4 fases:

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.

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

Dos enfoques híbridos, especialmente diseñados para el soporte de las iteraciones:

Desarrollo Incremental.

Desarrollo en Espiral.

Desarrollo incremental

Sugiere 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. 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.

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 1988. 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 y 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.

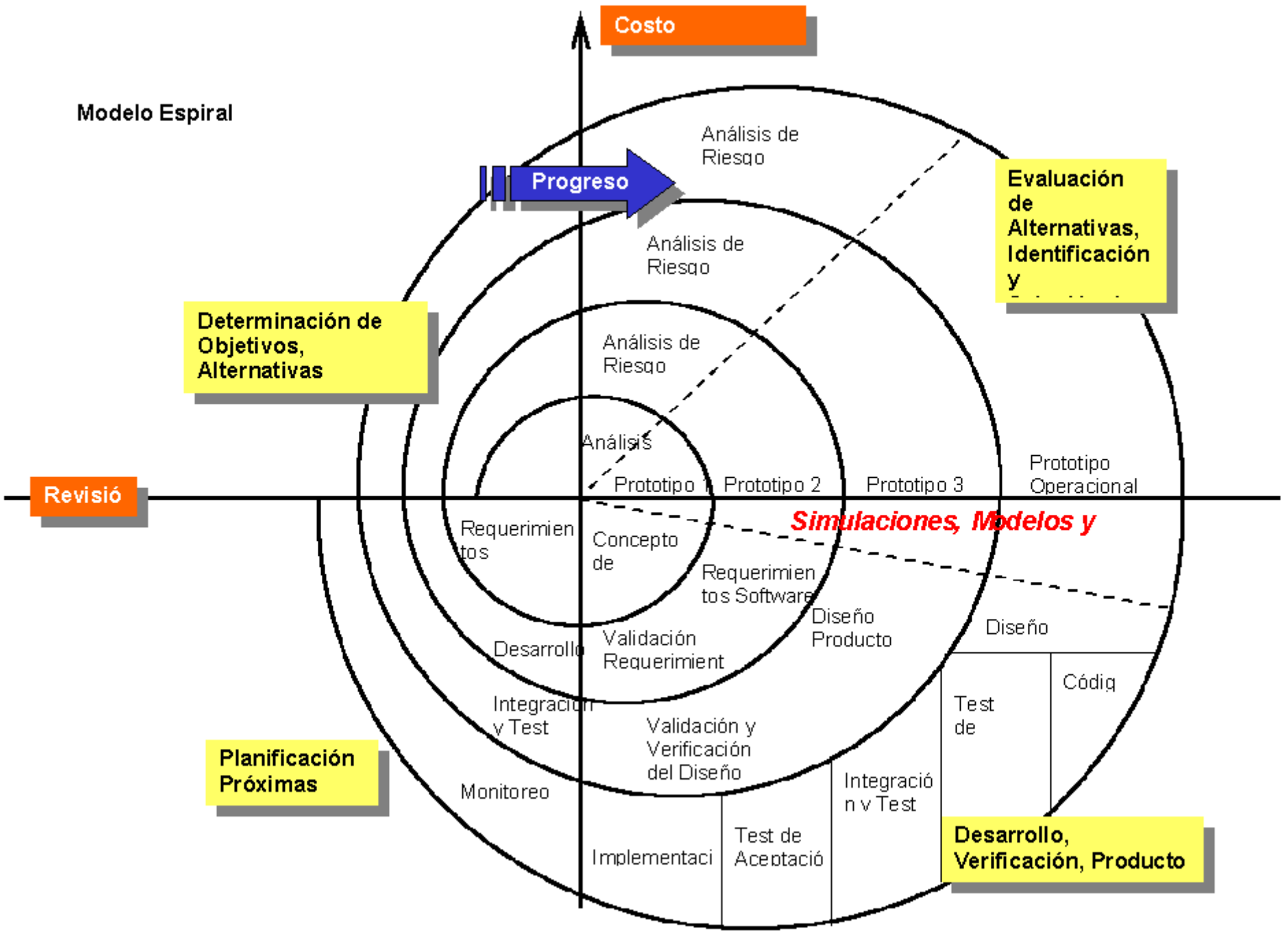
Evaluación y reducción de riesgos: Se realiza un análisis detallado de cada riesgo identificado. Pueden desarrollarse prototipos para disminuir el riesgo de requisitos dudosos. Se llevan a cabo los pasos para reducir los riesgos.

Desarrollo y validación: Elegir modelo de desarrollo Elegir modelo de desarrollo, algunos autores lo denominan metamodelo o modelo paramétrico.

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.


Modelo Espiral
Costo
Progreso
Análisis de Riesgo
Análisis de Riesgo
Análisis de Riesgo
Análisis
Evaluación de Alternativas, Identificación y
Determinación de Objetivos, Alternativas
Revisió
Prototipo 1
Prototipo 2
Prototipo 3
Prototipo Operacional
Simulaciones, Modelos y
Requerimien tos
Concepto de
Requerimien tos Software
Diseño Producto
Diseño
Desarrollo
Validación Requerimient
Integracion y Test
Validación y Verificación del Diseño
Test de
Códig
Integració n y Test
Monitoreo
Implementaci
Test de Aceptació
Planificación Próximas
Desarrollo, Verificación, Producto

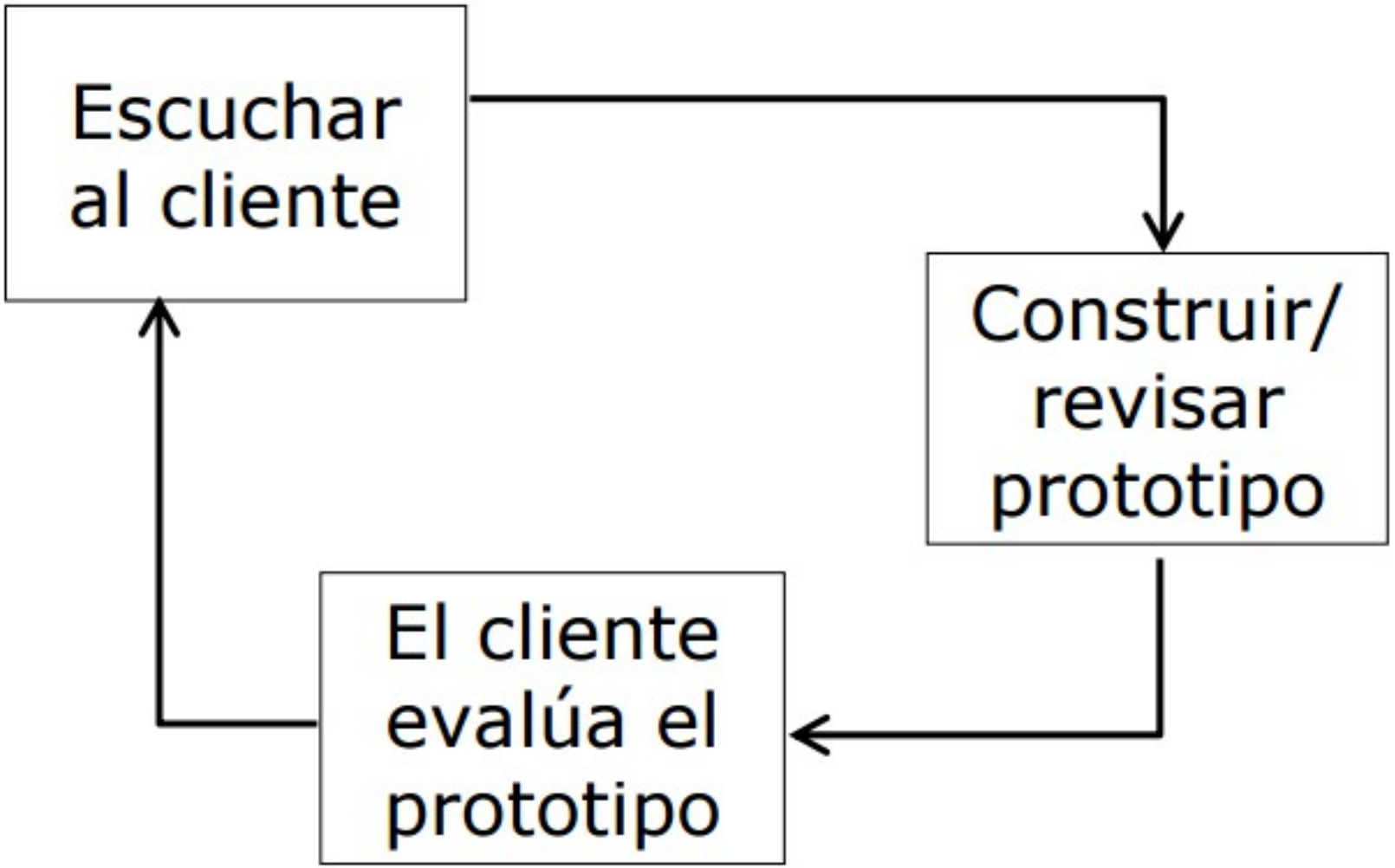
Prototipado

Se usa un prototipo para dar al usuario una idea concreta de lo que va a hacer el sistema

Se aplica cada vez más cuando la rapidez de desarrollo es esencial

Prototipado evolutivo: el prototipo inicial se refina progresivamente hasta convertirse en versión final

Prototipado desechable: de cada prototipo se extraen ideas buenas que se usan para hacer el siguiente, pero cada prototipo se tira entero


Escuchar al cliente
Construir/ revisar prototipo
El cliente evalúa el prototipo

Comienza con la recolección de requisitos
•Cliente y desarrolladores definen los objetivos del software.
•Además, identifican los requisitos conocidos y aquellos que ser más definidos.

Aparece un diseño rápido centrado en los aspectos visibles para el cliente
•El diseño rápido lleva a la construcción de un prototipo

El prototipo lo evalúa el cliente y lo utiliza para refinar los requisitos

El proceso se reitera... ¿desechando el prototipo?

Visión general de la IS

Con independencia del modelo de proceso hay tres fases genéricas:

- **Fase de definición**
- **Fase de desarrollo**
- **Fase de mantenimiento**

Cada una de estas fases se descompone en un conjunto de tareas

Fase de definición/especificación

Se identifican requisitos de sistema y software:

Información a procesar

Función y rendimiento deseados

Comportamiento del sistema

Interfaces establecidas

Restricciones de diseño

- Tareas principales:

  Planificación del proyecto software

  Ingeniería de sistemas o de información

  Análisis de requisitos

Fase de desarrollo

Se define:

Cómo diseñar las estructuras de datos

Cómo implementar las funciones

Cómo caracterizar las interfaces

Cómo traducir el diseño a programación

Cómo validar el producto (pruebas, verificación)

•Tareas principales:

Diseño del software

Generación del código

Pruebas del software

Fase de mantenimiento

Centrada en cambios que se pueda necesitar realizar sobre un producto

Se vuelven a aplicar las fases de definición y desarrollo, pero sobre software ya existente

Pueden producirse cuatro tipos de cambio:

Corrección:corregir los defectos

Adaptación: modificar por cambios externos.

Mejora: ampliar los requisitos funcionales originales a petición del cliente

Prevención: Cambio para facilitar el cambio

¿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 1 se expone un cuadro comparativo de acuerdo con algunos criterios básicos para la selección de un modelo de proceso [10], 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 del proyecto
Codificar y corregir	Bajo	Bajo	Bajo	Alto	Medio
Cascada	Bajo	Alto	Bajo	Bajo	Bajo
Evolutivo exploratorio	Medio o Alto	Medio o Alto	Medio	Medio o Alto	Medio o Alto
Evolutivo prototipado	Alto	Medio	Medio	Alto	Alto
Desarrollo formal de sistemas	Bajo	Alto	Bajo a Medio	Bajo	Bajo
Desarrollo orientado a reutilización	Medio	Bajo a Alto	Bajo a Medio	Alto	Alto
Incremental	Bajo	Alto	Medio	Bajo	Bajo
Espiral	Alto	Alto	Alto	Medio	Medio

Modelo de Proceso	**Visibilidad del Proceso**
Modelo de Cascada	Buena visibilidad, cada actividad produce un documento o resultado
Desarrollo Evolutivo	Visibilidad pobre, muy caro al producir documentos en cada iteración
Modelos Formales	Buena visibilidad, en cada fase deben producirse documentos
Desarrollo orientado a la reutilización	Visibilidad moderada. Importante contar con documentación de componentes reutilizables
Modelo de Espiral	Buena visibilidad, cada segmento y cada anillo del espiral debe producir un documento

Metodologías para desarrollo de software

Un proceso de software detallado y completo suele denominarse “Metodología”. Las metodologías se basan en una combinación de los modelos de proceso genéricos (cascada, evolutivo, incremental, etc.). Adicionalmente una metodología debería definir con precisión los artefactos, roles y actividades involucrados, junto con prácticas y técnicas recomendadas, guías de adaptación de la metodología al proyecto, guías para uso de herramientas de apoyo, etc. Habitualmente se utiliza el término “método” para referirse a técnicas, notaciones y guías asociadas, que son aplicables a una (o algunas) actividades del proceso de desarrollo, por ejemplo, suele hablarse de métodos de análisis y/o diseño.

Metodologías para desarrollo de software

La comparación y/o clasificación de metodologías no es una tarea sencilla debido a la diversidad de propuestas y diferencias en el grado de detalle, información disponible y alcance de cada una de ellas. A grandes rasgos, si tomamos como criterio las notaciones utilizadas para especificar artefactos producidos en actividades de análisis y diseño, podemos clasificar las metodologías en dos grupos: Metodologías Estructuradas y Metodologías Orientadas a Objetos. Por otra parte, considerando su filosofía de desarrollo, aquellas metodologías con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el apelativo de Metodologías Tradicionales (o peyorativamente denominada Metodologías Pesadas, o Peso Pesado). Otras metodologías, denominadas Metodologías Ágiles, están más orientadas a la generación de código con ciclos muy cortos de desarrollo, se dirigen a equipos de desarrollo pequeños, hacen especial hincapié en aspectos humanos asociados al trabajo en equipo e involucran activamente al cliente en el proceso. A continuación se revisan brevemente cada una de estas categorías de metodologías.

Metodologías estructuradas

Las metodologías no ágiles son aquellas que están guiadas por una fuerte planificación durante Los métodos estructurados comenzaron a desarrollarse a fines de los 70's con la Programación Estructurada, luego a mediados de los 70's aparecieron técnicas para el Diseño (por ejemplo: el diagrama de Estructura) primero y posteriormente para el Análisis (por ejemplo: Diagramas de Flujo de Datos). Estas metodologías son particularmente apropiadas en proyectos que utilizan para la implementación lenguajes de 3ra y 4ta generación.

Metodologías orientadas a objetos

Su historia va unida a la evolución de los lenguajes de programación orientada a objeto, los más representativos: a fines de los 60's SIMULA, a fines de los 70's Smalltalk-80, la primera versión de C++ por Bjarne Stroustrup en 1981 y actualmente Java1 o C# de Microsoft. A fines de los 80's comenzaron a consolidarse algunos métodos Orientadas a Objeto.

En 1995 Booch y Rumbaugh proponen el Método Unificado con la ambiciosa idea de conseguir una unificación de sus métodos y notaciones, que posteriormente se reorienta a un objetivo más modesto, para dar lugar al Unified Modeling Language (UML)2, la notación OO más popular en la actualidad.

Algunos métodos OO con notaciones predecesoras de UML son: OOAD (Booch), OOSE (Jacobson), Coad & Yourdon, Shaler & Mellor y OMT (Rumbaugh).

Algunas metodologías orientadas a objetos que utilizan la notación UML son: Rational Unified Process (RUP)3, OPEN4, MÉTRICA (que también soporta la notación estructurada).

Metodologías tradicionales (no ágiles)

Las metodologías no ágiles son aquellas que están guiadas por una fuerte planificación durante todo el proceso de desarrollo; llamadas también metodologías tradicionales o clásicas, donde se realiza una intensa etapa de análisis y diseño antes de la construcción del sistema.

Todas las propuestas metodológicas antes indicadas pueden considerarse como metodologías tradicionales. Aunque en el caso particular de RUP, por el especial énfasis que presenta en cuanto a su adaptación a las condiciones del proyecto (mediante su configuración previa a aplicarse), realizando una configuración adecuada, podría considerarse Ágil.

Metodologías ágiles

Un proceso es ágil cuando el desarrollo de software es **incremental** (entregas pequeñas de software, con ciclos rápidos), **cooperativo** (cliente y desarrolladores trabajan juntos constantemente con una cercana comunicación), **sencillo** (el método en sí mismo es fácil de aprender y modificar, bien documentado), y **adaptable** (permite realizar cambios de último momento) [11].

Entre las metodologías ágiles identificadas en [11]:

Extreme Programming [6].

Scrum ([12], [13]).

Familia de Metodologías Crystal [14].

Feature Driven Development [15].

Proceso Unificado Rational, una configuración ágil ([16]).

Dynamic Systems Development Method [17].

Adaptive Software Development [18].

Open Source Software Development [19]

Casos de éxito y fracaso

oftware, su construcción y resultados han sido históricamente cuestionados debido a los problemas asociados, entre ellos podemos destacar los siguientes [1]:

Los sistemas no responden a las expectativas de los usuarios.

Los programas “fallan” con cierta frecuencia.

Los costes del software son difíciles de prever y normalmente superan las estimaciones.

La modificación del software es una tarea difícil y costosa.

El software se suele presentar fuera del plazo establecido y con menos prestaciones de las consideradas inicialmente.

Normalmente, es difícil cambiar de entorno hardware usando el mismo software.

El aprovechamiento óptimo de los recursos (personas, tiempo, dinero, herramientas, etc.) no suele cumplirse.

- Inadecuada gestión de los requisitos.
- No existe medición del proceso ni registro de datos históricos.
- Estimaciones imprevistas de plazos y costos.
- Excesiva e irracional presión en los plazos.
- Escaso o deficiente control en el progreso del proceso de desarrollo.
- No se hace gestión de riesgos formalmente.
- No se realiza un proceso formal de pruebas.
- No se realizan revisiones técnicas formales e inspecciones de código.
- El primer reconocimiento público de la existencia de problemas en la producción de software tuvo lugar en la conferencia organizada en 1968 por la Comisión de Ciencias de la OTAN en Garmisch (Alemania), dicha situación problemática se denominó crisis del software. En esta conferencia, así como en la siguiente realizada en Roma en 1969, se estipuló el interés hacia los aspectos técnicos y administrativos en el desarrollo y mantenimiento de productos software. Se pretendía acordar las bases para una ingeniería de