Calidade e probas en robótica

2022/23

Introducción a la ingeniería de software

Marcos Boullón Magán

Depto. Electrónica e Computación

Motivación

Consideramos que el software es un aspecto esencial en esta sociedad:

- servicios públicos

  plataformas
- fabricación y distribución

  dispositivos industriales
- sistema financiero
- industria cultural y del ocio
- ...

Dependemos de diferentes tipos de software con necesidades, limitaciones y modos de operación diferentes. Es esencial la capacidad de construirlo *correctamente*.

Motivación: naturaleza del software

El software es una abstracción, intangible, y no limitado por leyes físicas, materiales o procesos de fabricación.

Características especiales:

- lógico, no físico
- inmaterial e invisible
- se deteriora, pero no se rompe
- complejo y con un comportamiento difícil de predecir
- desarrollado, no construido (no hay repuestos)
- desarrollado a medida
- se evalúa ya desarrollado

Motivación: naturaleza del software (2)

Ventajas:

- no hay límites artificiales para la construcción de software, sino que los límites vienen forzados exclusivamente por el entorno de ejecución

Desventajas:

- el proceso de desarrollo rápidamente se puede volver muy complejo y difícil de entender

Motivación: naturaleza del software (3)

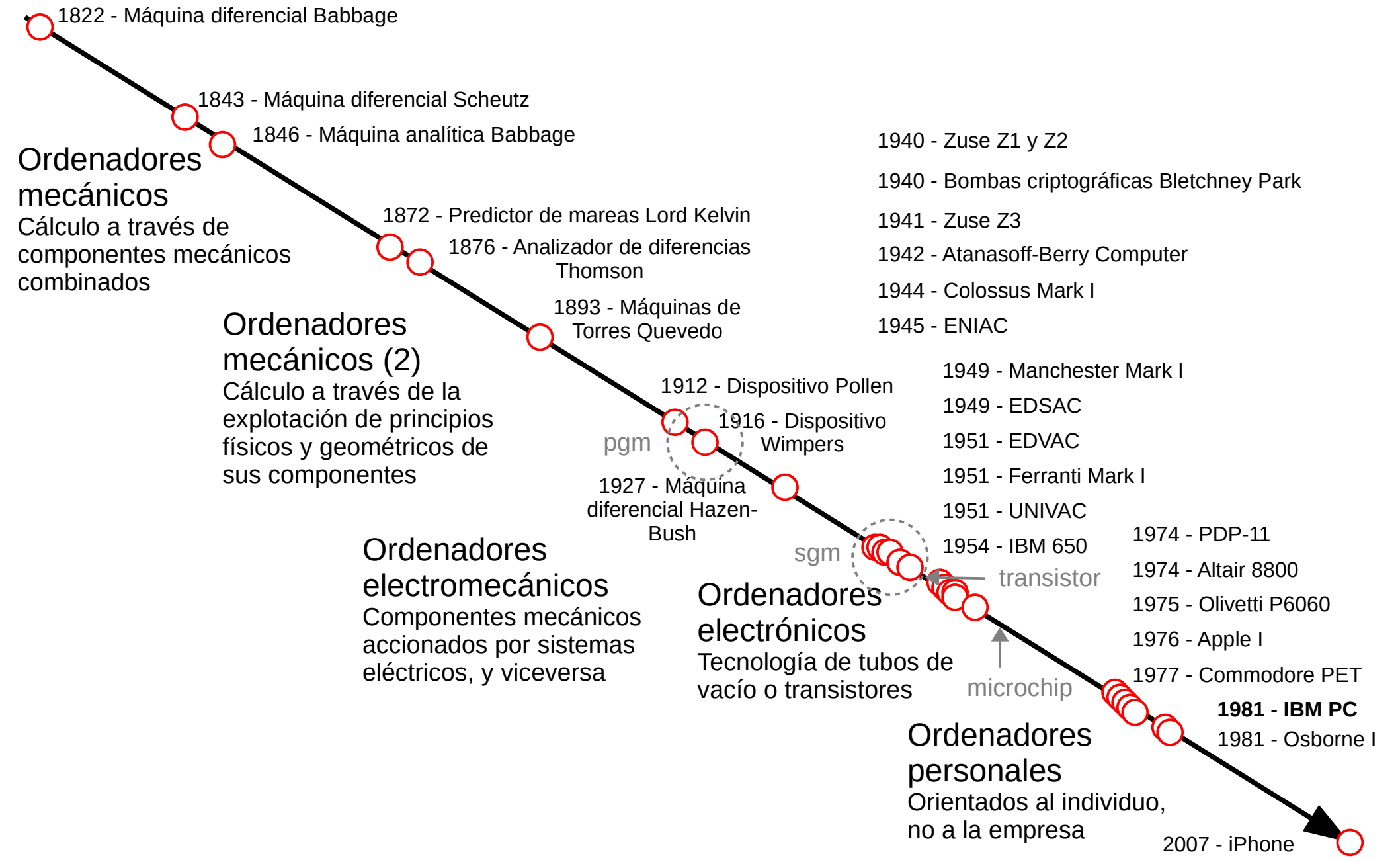
La *ingeniería de software* trae orden al caos:

Es el proceso para construir y mantener un sistema informático ajustándose a factores de calidad, coste, recursos, tiempo... Requiere de una planificación previa. Y proporciona ayuda en forma de métodos, técnicas, procedimientos, normas, recomendaciones...

Pero la naturaleza del software fuerza que las metodologías sean diferentes a otras ingenierías:

- el cliente, al principio, no sabe expresar qué quiere ni cómo
- el desarrollador no sabe si el producto agradará al cliente

Motivación: crisis del software



Motivación: crisis del software (2)

Desde los años 50 el desarrollo es artesanal. Este enfoque es útil al desarrollo de programas pequeños, pero no escala hacia los sistemas más grandes y complejos. Se obtiene una baja productividad y calidad.

En 1968 la OTAN preside una conferencia para resolver estos problemas (*crisis del software*). Se propone una nueva disciplina. Su objetivo es proporcionar metodologías, definir procedimientos, implementar técnicas y construir herramientas para reducir costes y mejorar la calidad.

A lo largo de los 70 y 80 se desarrollan estos métodos, herramientas, estándares y buenas prácticas que seguimos usando: programación estructurada, programación orientada a objetos, diseño descendente, ocultación de la información, SOLID, DRY, GRASP, YAGNI...

Ingeniería de software

Ingeniería de software. Es la aproximación sistemática al desarrollo, operación y mantenimiento del software (IEEE, 1983).

> *Sistemática*, que implica el uso de principios (métodos y procedimientos) de ingeniería robustos.

Es una ingeniería. Proporciona teorías, métodos y herramientas preparadas para el desarrollo de software, con un enfoque sistemático y organizado.

Desde ahora vamos a considerar que *software* incluye tanto código ejecutable, datos de configuración y toda la documentación del proyecto; *sistema informático*.

Ingeniería de software (2)

El objetivo es obtener resultados de calidad con las restricciones impuestas de **coste** y **tiempo**,

- con la **funcionalidad** requerida
- con un **rendimiento** adecuado

y además,

- **mantenibilidad;** debe evolucionar según cambian las necesidades del usuario/mercado
- **confiabilidad y seguridad;** abarca fiabilidad, seguridad y protección; no puede causar daño físico o económico en caso de falla o uso malicioso
- **eficiencia;** no desperdicia recursos del sistema (cpu, memoria, tiempo de respuesta...)
- **aceptabilidad;** es comprensible por el usuario, utilizable y compatible con estándares

Ámbito de aplicación de la ingeniería de software

Se aborda de manera diferente:

- desarrollo de software personal
- la mayoría del software se hace dentro de la empresa (equipos internos o subcontratados) para propósitos específicos del negocio...
- ... o como el propio producto de negocio: herramientas específicas, plataformas genéricas personalizables, controladores de dispositivos construidos por la compañía

El primer caso, no nos interesa (¿seguro?).

En el segundo caso, la empresa esa quien define y controla las especificaciones, que luego desarrolla en el producto.

En el tercer caso, el control lo tiene la empresa que desarrolla el producto y *con suerte* éste se adapta a las especificidades del cliente. Otro enfoque es construir productos/plataformas adaptables y configurables por el usuario.

Ámbito de aplicación (2)

La ingeniería de software se interesa sólo por desarrollos profesionales:

- gestión de recursos, definición de responsabilidades, coordinación del equipo
- aplicación de técnicas para la captura y validación de requerimientos, metodologías para el diseño e implementación de los componentes del sistema, técnicas de verificación y validación, configuraciones
- mantenimiento y mejora del producto durante su vida útil
- documentación del proceso completo
- ...

Estas características no son relevantes en productos personales, donde típicamente hay un único usuario que no necesita formación para usarlo (¿seguro? *github*).

Ámbito de aplicación (3)

La ingeniería de software se interesa por todos los aspectos de la construcción del software profesional:

- desarrollo de métodos y herramientas para facilitar la producción
- procesos técnicos de desarrollo
- gestión del proyecto

  estimación, planificación, seguimiento, dirección técnica, gestión de recursos, subproyectos, coordinación del equipo de trabajo...
- control del proyecto

  validar la ejecución del proyecto con métricas sobre la planificación, medidas de calidad, aseguramiento de la calidad, gestión de configuraciones...
- operación

  formación de usuarios, entrega del producto, puesta en marcha, soporte...

Proyecto, proceso, actividades del proceso

Proyecto. Esfuerzo temporal para crear un producto único.

Proceso. El conjunto coherente de actividades para cumplir un fin: la producción de nuestro producto.

En el caso de sistemas informáticos incluye código ejecutable, datos, configuraciones y documentación: plan de proyecto, documentos de diseño, manual de usuario...

Producto final. Es el resultado en forma de sistema informático de la ejecución del proyecto. También hay productos intermedios al final de cada fase del proyecto tanto en forma de código como de documentación.

La ingeniería de software permite conducir el proceso de desarrollo y mantenimiento de forma eficaz y obtener un producto *fiable*, de alta *calidad* y *bajo coste* para *satisfacción del cliente*.

Proyecto, proceso, actividades del proceso (2)

Diferentes tipos de sistema, diferentes necesidades:

- aplicaciones independientes
- aplicaciones interactivas basadas en transacciones o web

  heterogeneidad, seguridad, usabilidad, privacidad, integridad de datos...
- sistemas de control embebido

  verificación y validación exhaustiva, sin interfaz de usuario, respuesta rápida, bajo gasto energético...
- sistemas de proceso por lotes
- sistemas de entretenimiento
- sistemas de modelado y simulación
- sistemas de adquisición de datos y sensores

  alta fiabilidad, bajo mantenimiento, bajo gasto energético...

Proyecto, proceso, actividades del proceso (3)

Hay muchos procesos propuestos para desarrollar estos sistemas, pero todos incluyen las mismas **actividades fundamentales**:

- especificación: definir las funcionalidades y restricciones
- desarrollo: diferentes tipos de sistema requieren diferentes aproximaciones y tipos de metodologías para abordar el desarrollo

  problemas comunes: ejecución en entornos heterogéneos y fuera de nuestro control, integrar la evolución de las necesidades del usuario, asegurar seguridad y confianza en escenarios reales
- validación
- evolución: entrega, instalación y mantenimiento

Proyecto, proceso, actividades del proceso (4)

Estas actividades básicas contienen **subactividades**:

validación de requisitos, diseño de la arquitectura, especificación del modelo de datos, pruebas unitarias, diseño del interfaz de usuario...

y **actividades de soporte** al proceso:

gestión, documentación, administración de configuraciones...

Proyecto, proceso, actividades del proceso (5)

Estos procesos propuestos son una representación simplificada del proceso final, ya que una organización puede diseñar su propio proceso de desarrollo, adaptado a sus particularidades (empresa, sector, o sistema):

> técnicas de desarrollo, productos intermedios, roles, precondiciones/postcondiciones de validación de las fases...

La ingeniería de software está en evolución y actualiza continuamente sus recomendaciones y mejores prácticas. Siguiéndolas, la organización se beneficia de una estandarización en sus procesos (con una reducción en la diversidad de los mismos):

- mejora la comunicación
- es necesaria una menor capacitación
- el soporte al proceso es más económico
- una adaptación más rápida de la nueva tecnología

Proyecto, proceso, actividades del proceso (6)

Grandes categorías de procesos:

- **procesos dirigidos por un plan;** todas las actividades se planean por anticipado y el avance se mide contra dicho plan:

  actividades bien definidas, requisitos bien identificados, entregas planificadas...

- **procesos ágiles;** planificación incremental para permitir gestionar requisitos de usuario o de sistema cambiantes en el tiempo:

  continua interacción con el cliente, entregas parciales, ciclos iterativos cortos...

Cada enfoque es más adecuado para un cierto tipo de sistema software. Pero no son excluyentes y se pueden combinar (distintos subsistemas se pueden programar bajo distintos modelos).

Metodología, ciclo de vida, proceso

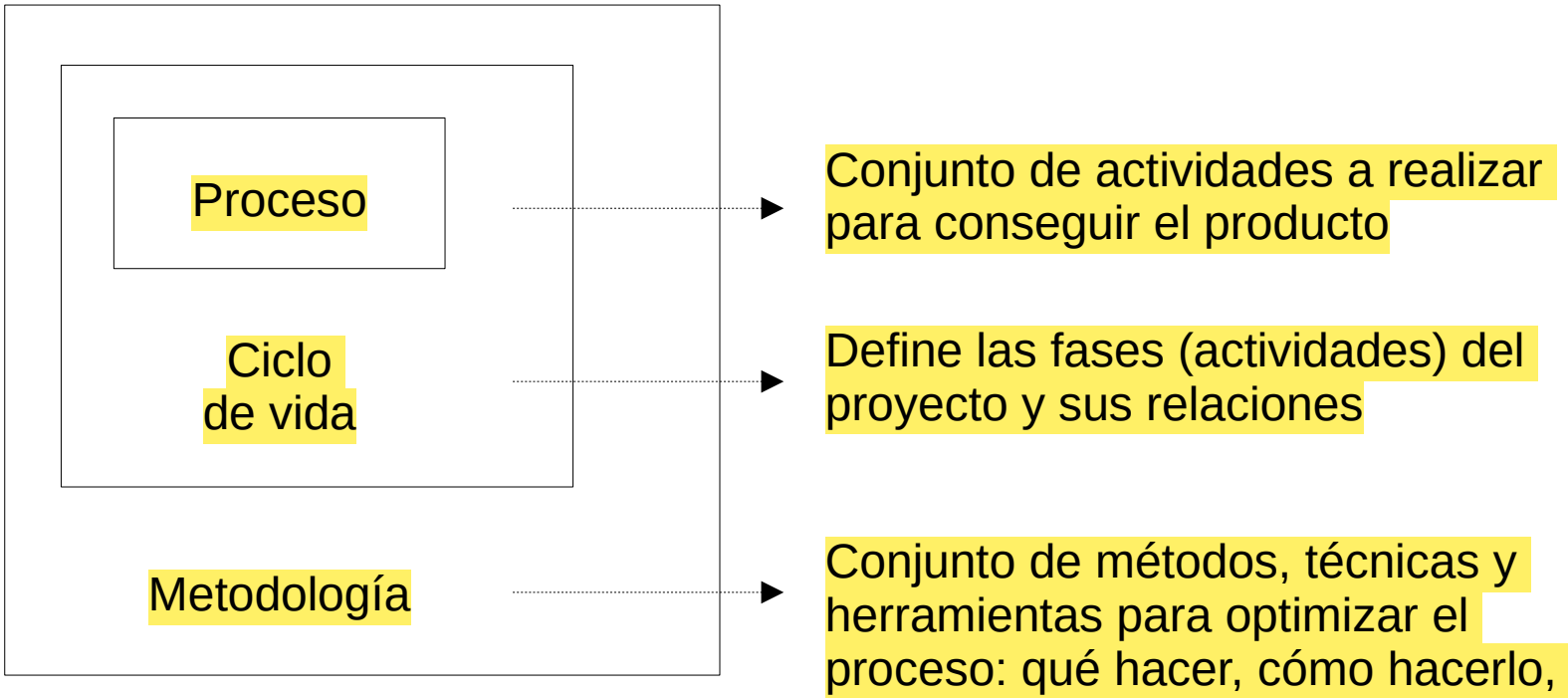
Metodología. Conjunto de métodos, técnicas y herramientas que se utilizan en un proyecto para formalizarlo y optimizarlo. Determina los pasos a seguir y cómo realizarlos.

Al aplicar metodologías a la ingeniería de software optimizamos el proceso y el producto final.

Ciclo de vida. Conjunto de fases por las que pasa el sistema que se desarrolla desde la idea inicial hasta que se retira. Determina el orden de las fases del proyecto, estradas y salidas de cada una, criterios de transición para cambiar entre ellas...

Proceso. El trabajo desarrollado en las fases.

Metodología, ciclo de vida, proceso (2)



Metodología, ciclo de vida, proceso (3)

Por tanto, una metodología

- define las fases
- establece el orden de las fases
- identifica:
  - tareas a realizar en cada fase
  - productos intermedios
    - documentación, interfaces, entradas y salidas
  - producto final
- proporciona ayudas a la realización de cada tarea

  procedimientos, herramientas, criterios de evaluación...

Una de las funciones principales de la **metodología** es definir el **ciclo de vida** más adecuado al proyecto (y/o a cada módulo) según características.

Metodología, ciclo de vida, proceso (4)

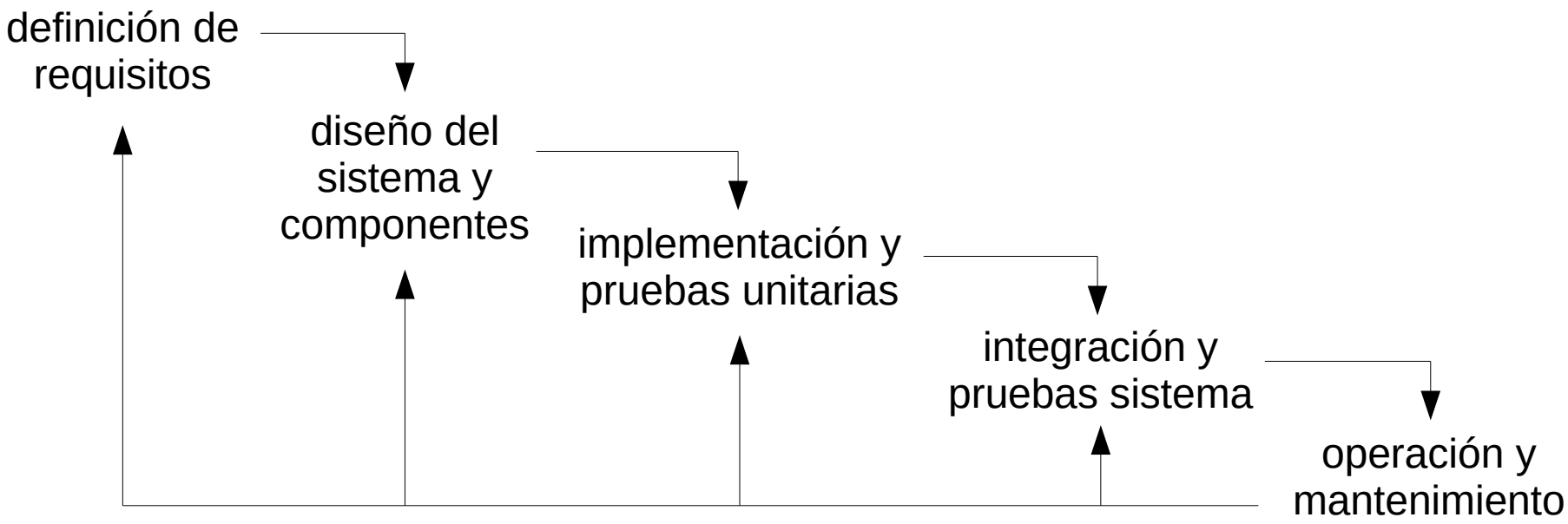
¡Vamos a ignorar la diferencia entre todos ellos (sabiendo que son distintos niveles de abstracción)! Y hablar únicamente de **proceso**.

Modelos de proceso típicos

Funcionan como abstracciones de un proceso general, pensados para extenderse y adaptarse; no son excluyentes, se pueden combinar (distintos subsistemas se pueden programar bajo distintos modelos):

- **Modelo en cascada.** Las cuatro actividades fundamentales (especificación, desarrollo, validación y evolución) son fases separadas e independientes en el proceso de desarrollo.
- **Modelo incremental.** Las actividades de especificación, desarrollo y validación están intrínsecamente vinculadas. El producto se construye como una sucesión de productos que van añadiendo funcionalidades nuevas sobre las versiones (*incrementos*) anteriores.
- **Modelo orientado a la reutilización.** Gracias a la existencia de un alto número de elementos reutilizables, poco desarrollo y mucho proceso de integración.
- Modelo en espiral, orientado al riesgo.
- Proceso unificado racional (RUP).

Modelo en cascada



Modelo en cascada (2)

Principales etapas:

- análisis y definición de requisitos; servicios, restricciones y metas se establecen con consultas al usuario; sirven como especificaciones de sistema (responde a *qué*)
- diseño del sistema y del software (responde a *cómo*):

  sistema; seleccionar la arquitectura global (hardware, software) para cumplir los requisitos

  software; identificar y describir las abstracciones fundamentales del software y sus relaciones
- implementación y pruebas unitarias; se concreta el diseño del software como un conjuntos de códigos (unidades), y se verifican sus especificaciones
- integración y pruebas del sistema; las unidades individuales se prueban como sistema completo, y se verifican sus especificaciones; se entrega al usuario

Modelo en cascada (3)

(...)

- operación y mantenimiento; el sistema se instala, se usa y va evolucionando

  corrección de errores, mejoras en la implementación, incremento de servicios (nuevos requisitos)...

Una etapa no debería poder comenzar hasta finalizar las anteriores. Pero en la práctica, solapan; incluso retroceden (hay un flujo de *realimentación* donde los productos de cada etapa se pueden ir refinando iterativamente).

¿Cuándo se usa cascada? Cuando lo requisitos se entienden correctamente y es improbable un cambio radical (debido a un requisito sorpresa).

Modelo en cascada (4)

Ventajas:

- es el modelo más simple
- cada etapa genera documentación; con ella los administradores pueden monitorizar el proceso contra el plan

Modelo de desarrollo de sistemas formales

Una variación del modelo en cascada.

La especificación del sistema viene dada por un modelo matemático (validable con código). Las modificaciones se aplican mediante transformaciones matemáticas que mantienen la consistencia. Los cambios son autovalidables:

Método B, *clean room*...

¿Cuándo se usa? Cuando se trata de un sistema crítico que debe asegurar la seguridad, fiabilidad y protección de datos.

Modelo de desarrollo incremental

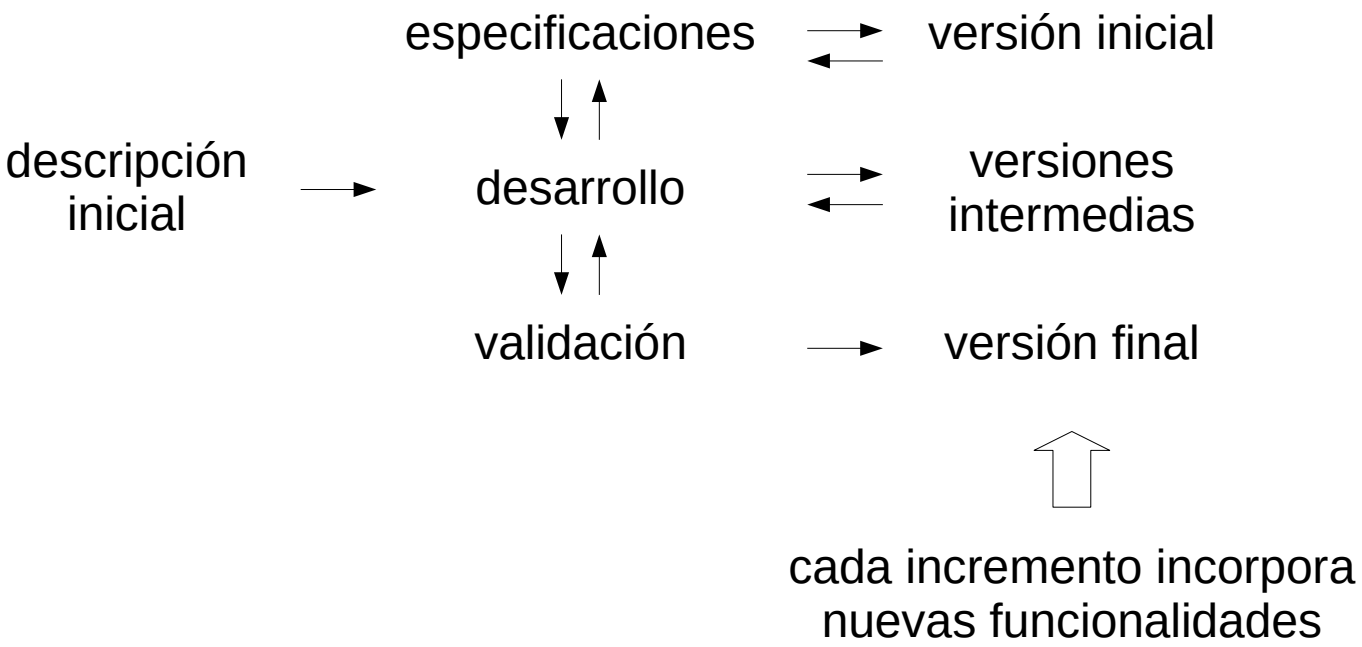
Las actividades de especificación, desarrollo y validación están interconectadas, con una rápida realimentación. El producto se desarrolla por incrementos, y dentro de cada uno el ciclo de vida es iterativo:

Hay una implementación inicial mínima. A partir de ese momento todas las fases se van recorriendo en orden para construir nuevas versiones (nuevos *incrementos*) que añaden funcionalidades y mejoran el producto. Las funcionalidades más urgentes se incluyen en las primeras versiones; las siguientes, se van añadiendo en orden de prioridad.

¿Cuándo se usa? Se usa tanto en procesos planeados como en procesos ágiles. En planeados, se identifican de antemano los incrementos a usar. En ágiles, los incrementos dependen de las prioridades del cliente.

Es el más usado en enfoques ágiles, ya que coincide con su filosofía: no se trabaja desde el principio con la solución completa, sino que se avanza en pasos pequeños.

Modelo de desarrollo incremental (2)



Modelo de desarrollo incremental (3)

Ventajas:

- se proporcionan las funcionalidades poco a poco; si hay problemas de validación, sólo hace falta corregir el incremento actual
- sencillez al incorporar cambios; se reduce el coste de adaptarse a requisitos cambiantes; el esfuerzo en análisis y documentación son menores porque son los del incremento actual
- las primeras entregas sirven como prototipos para validación; ¡prototipos que funcionan como el producto final!
- el usuario proporciona realimentación antes, a través de la ejecución del código (no sólo leyendo documentación)
- las funcionalidades críticas reciben más pruebas (porque están disponibles desde el principio)
- puesta en producción: las primeras entregas ofrecen ya las funcionalidades críticas

Modelo de desarrollo incremental (4)

Desventajas:

- el proceso no es visible, sólo los productos; el avance se mide con las entregas, y la documentación queda rápidamente desactualizada
- la estructura global se vuelve excesivamente compleja (frágil) con los incrementos
  - requiere etapas de *refactorización* para mantener la adaptabilidad (incrementos que no incorporan funcionalidades de usuario)
- es difícil identificar recursos comunes a varios subsistemas hasta que estén entregadas: *refactorización*
- mayores sistemas llevan a problemas más agudos (demasiados equipos) a no ser que las responsabilidades de los equipos puedan planificarse de antemano:
  - *propiedad del código*

Modelo de desarrollo incremental (5)

(...)

- en sistemas que deben reemplazar a otro, la funcionalidad se consigue en la última entrega
- las especificaciones son incrementales aunque se conozca ya el conjunto final completo de requisitos

Modelo de desarrollo incremental (6)

No debería usarse con:

- sistemas que dependan de hardware (si no es configurable)
- sistemas críticos con interacciones complejas
- escenarios donde no se controle por completo el entorno (redes remotas)

Aún así, un prototipo del sistema desarrollado incrementalmente para experimentar con los requisitos todavía es útil (es el enfoque más usado para desarrollo de el módulos de interfaz de usuario).

SCRUM

SCRUM es un tipo de proceso/metodología ágil (no se planifica por anticipado) que sigue un modelo incremental.

Características:

- permite cambios en los requisitos en cualquier fase del proceso
- se van entregando versiones evaluables de forma continua
- cliente y usuario trabajan juntos
- los equipos de desarrollo se autoorganizan; no hay jefes de proyecto

  reuniones diarias, reflexión sobre eficacia y calidad técnica…

Otras metodologías ágiles: XP (*extreme programming*), Kanban...

SCRUM (2)

El equipo SCRUM:

- roles principales
  - *project owner/product owner*; representa al cliente
  - *SCRUM master*; se asegura de que el equipo siga la metodología correctamente (*sprints* de 2-4 semanas)
  - desarrolladores; equipos pequeños (4-8 personas)
- roles secundarios
  - proveedores, vendedores, subcontratas

SCRUM (3)

Desarrollo:

1. Creación del *product backlog*. Listado de características y requisitos de alto nivel que definen el trabajo a realizar.

2. *Sprint planning*. Definir los elementos del *product backlog* que se van a implementar en la versión actual.

- el *product owner* decide cuáles
- el equipo de desarrolladores refina la selección según a lo que puedan comprometerse a completar (gestión de recursos)

3. Creación del *sprint backlog*. Listado de tareas a realizar en el *sprint*.

SCRUM (4)

4. *Sprint*. Desarrollo (diseño, implementación, validación) de las tareas.

Se realiza un *daily meeting* (10 min) donde cada miembro responde a:

- ¿ha finalizado el trabajo asignado?
- ¿problemas encontrados?
- ¿qué otro trabajo realizará a continuación?

Un tablero de tareas permite hacer un seguimiento visual del *sprint*: tareas por hacer, tareas en progreso, hechas.

5. Reuniones de evaluación y retrospectiva.

6. Repetir desde el paso 2 hasta la finalización del proyecto.

SCRUM (5)

El *product backlog* recoge los requisitos de alto nivel formulados como una historia de usuario.

Elementos:

- identificador de la historia
- enunciado: "*como <rol> quiero hacer <descripción de la funcionalidad>, con el objetivo de <razón|resultados>*"
- contiene valor para el cliente
- estimación del esfuerzo en personas/días
- identificador (número) del *sprint*
- prioridad

Modelo orientado a la reutilización

Siempre hay una reutilización informal de componentes, independientemente del modelo de desarrollo: librerías públicas de código, algoritmos estándar, productos similares...

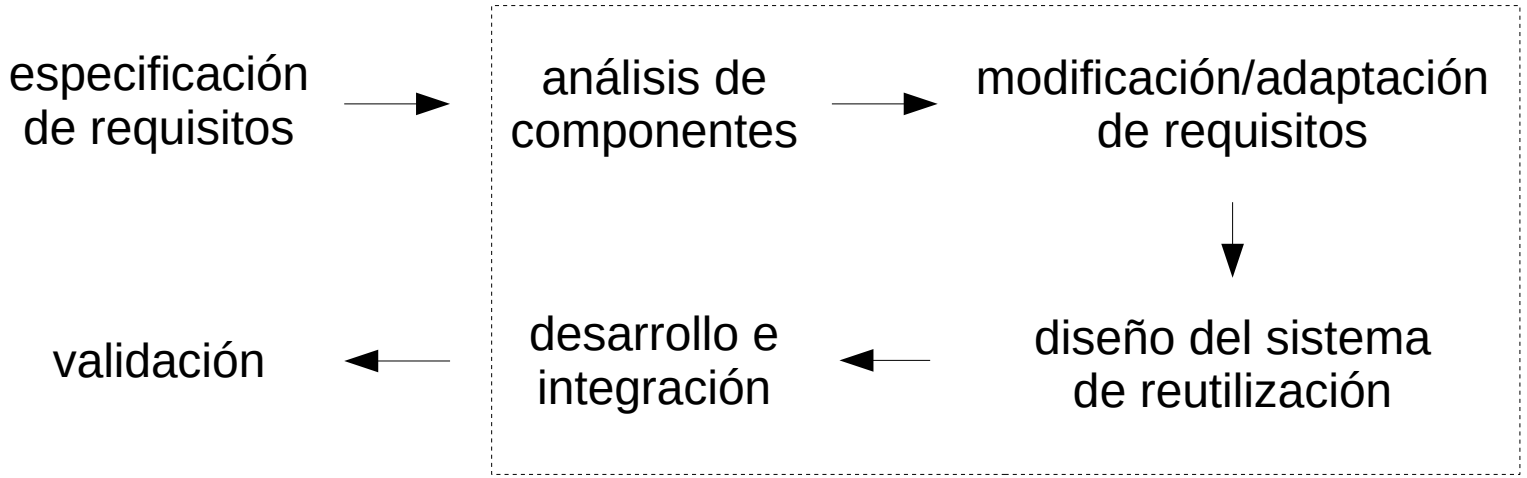
Este enfoque se basa en

- una gran base de componentes disponibles
- existencia de frameworks para la composición de estos elementos

Ejemplos:

- servicios web
- objetos .net
- scripts independientes colaborando (unix)

Modelo orientado a la reutilización (2)



Modelo orientado a la reutilización (3)

Etapas:

- búsqueda de componentes para implementar las especificaciones
  - obtenemos funcionalidad parcial
- los requisitos se actualizan con la funcionalidad encontrada
- se diseña/adapta el marco donde introducirlos
- se desarrollan componentes necesarios, y se integran todos para construir el sistema
  - la integración forma parte del desarrollo y no es una actividad independiente (no hay producto sin ella, ni siquiera un producto parcial)

Modelo orientado a la reutilización (4)

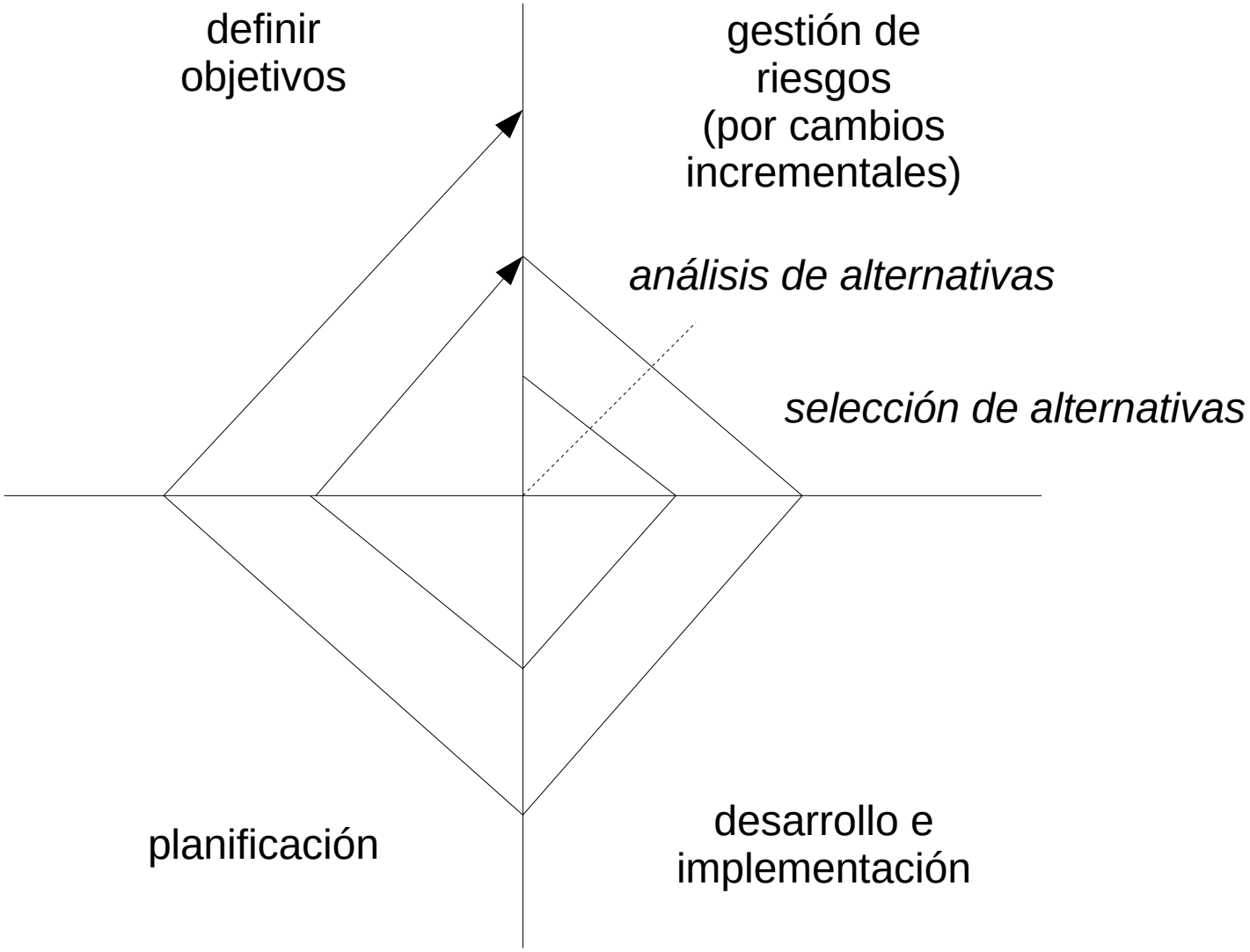
Ventajas:

- menos software a desarrollar, menores costes y riesgos
- entregas más rápidas

Desventajas:

- funcionalidad incompleta, compromisos en requisitos
- la evolución del sistema depende de componentes de terceros

Modelo orientado al riesgo



Modelo orientado al riesgo (2)

El desarrollo se estructura en ciclos que cubren una actividad: ciclo de análisis de factibilidad, ciclo de especificación de requisitos, ciclo de diseño del sistema...

Cada ciclo tiene las mismas fases:

- establecer objetivos; rendimiento y funcionalidades
- valoración y reducción del riesgo; buscar formas alternativas de alcanzar los objetivos; selección a través de
  - recopilación de información detallada
  - creación de prototipos
  - simulaciones
- desarrollo y validación por el cliente
- planificación del siguiente ciclo

Modelo orientado al riesgo (3)

Ventajas:

- se centra en eliminar alternativas poco atractivas con mecanismos para detectar y prevenir dificultades

Desventajas:

- entradas y salidas poco definidas entre fases

Gestión del proyecto

Realizada por el jefe o director del proyecto. Es una actividad continuada, a lo largo de todo el ciclo de vida del proyecto (desde la idea inicial hasta su retirada).

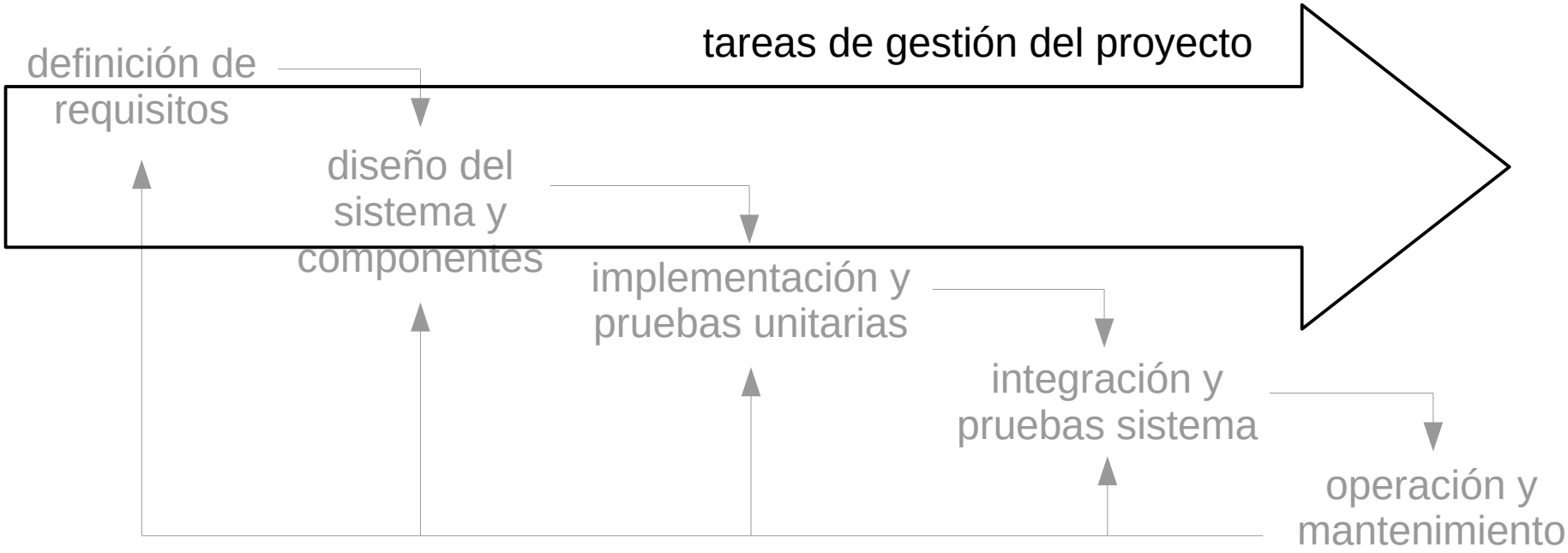
La **gestión eficaz** del proyecto involucra al personal, al producto, al proceso y al proyecto (hay actividades de gestión en cada uno de ellos).

Gestión no eficaz... motivo de el fracaso de un proyecto software:

- incorrecta estimación y planificación de recursos o duración
- pobre gestión del riesgo
- escasa gestión de la calidad
- errores en el seguimiento del desarrollo
  - desarrolladores con información insuficiente
- sin procedimientos de monitorización y control

Gestión del proyecto (2)

Gestión de proyectos. Es el proceso de planificar, organizar, proveer de personal, monitorizar, controlar y liderar el proyecto (IEEE).



Gestión del proyecto (3)

Tareas de gestión y dirección de proyectos:

Estimación. Predicción cuantitativa de aspectos de duración, esfuerzo y costes.

Planificación. Organización de recursos humanos y tareas.

Negociación. Acordar y acotar con el cliente los objetivos del proyecto.

Coordinación del equipo. Coordinación y motivación para ser eficaces y efectivos (productivos).

Seguimiento y control. Tanto del proceso como del producto, para asegurar que no se desvía de la planificación. Controlar la calidad. Controlar los límites en coste, esfuerzo y tiempo.

Gestión (delegable). Actividades administrativas, financieras, definición… para el buen término del proyecto.

Dirección técnica (delegable). Colaborar y supervisar actividades técnicas, resolviendo los problemas que surjan.

Gestión: coordinación del equipo

Tareas:

- diseñar la estructura y organización del equipo
- asignar roles y responsabilidades a los participantes
- definir los canales de comunicación
- cada persona debe saber:

  qué y para cuándo, quién le puede ayudar, quién es su jefe, cómo es evaluado, herramientas de las que dispone… *onboarding*
- cada persona recibe:
  - formación
  - información

    mediante reuniones periódicas para gestionar estado, problemas y soluciones
  - motivación

Gestión: estimación

Es una predicción sobre duración, esfuerzo y costes.

Unidades de medida:

- coste: moneda
- duración: unidades temporales
- esfuerzo (productividad): personas/día, p/semana, p/año
  - *p/día*, es el trabajo que de media puede realizar una persona a tiempo completo en un día
  - personas y tiempo no son intercambiables por las dependencias entre tareas

A partir del esfuerzo obtenemos el coste.

Hay que multiplicar por un coeficiente interno que indica el trabajo que debe realizar una persona en un tiempo fijo para que la empresa no tenga pérdidas: considera recursos humanos, viajes, hardware y software, otros...

Gestión: estimación (2)

Para la estimación del esfuerzo, se acostumbra a dividir el proyecto en subproblemas/subsistemas recursivamente hasta llegar a los más sencillos de estimar.

estimación total = sumatorio(estimación subsistemas)

Hay técnicas empíricas de estimación que utilizan fórmulas para predecir esfuerzo en base a atributos del proyecto: líneas de código, puntos de función…

- Modelo COCOMO II
- Modelo de puntos de función
- Modelo de puntos objeto
- Modelo de casos de uso

Gestión: planificación

Creación del **plan de proyecto** que distribuye entre las distintas tareas los recursos ya estimados: esfuerzo, tiempo y coste.

Previamente debemos tener dividido en proyecto en subsistemas, y éstos en tareas… ¿durante la estimación? Así podemos definir las dependencias y relaciones entre tareas y distribuir entre ellas los recursos.

A cada **tarea** se le asigna:

- una duración
- dependencias
- recursos humanos

Gestión: planificación (2)

Además de la planificación de recursos y tiempo por tareas, hay que planificar **hitos**, que son momentos del ciclo de vida donde se evalúa el proyecto y se compara con la planificación:

resultados (calidad), coste, tiempo, recursos consumidos...

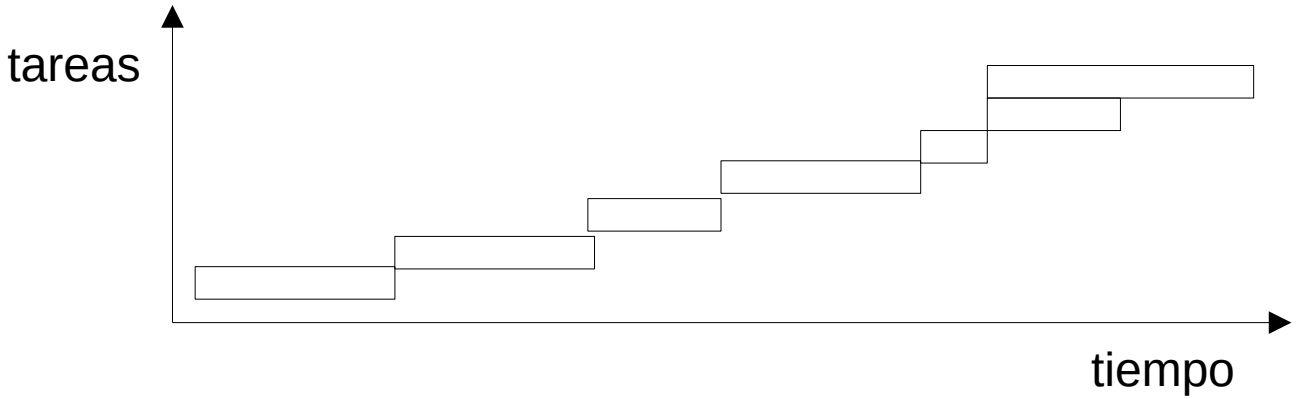
En caso de no estar satisfechos en el hito (no coincide con lo planificado), hay que:

- diagnosticar para buscar el desvío
- aplicar planes de contingencia para reconducir el proyecto
- actualizar el plan de proyecto

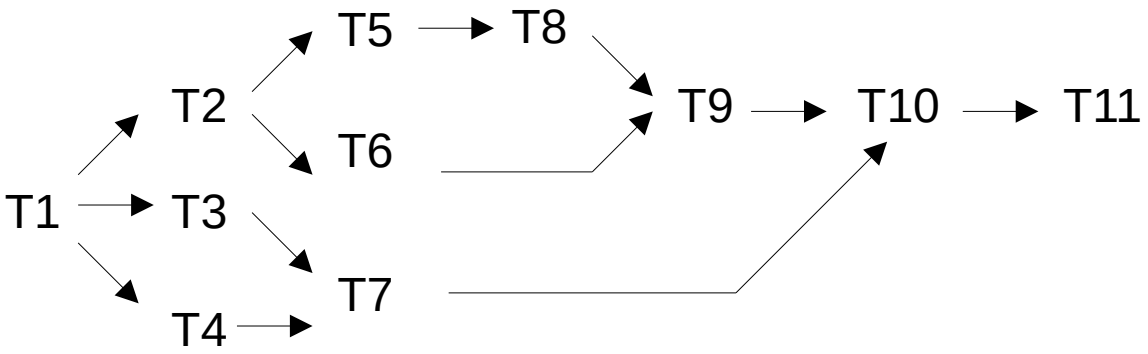
Gestión: planificación (3)

Ayudas a la planificación temporal de tareas:

Diagramas GANTT



Diagramas PERT (red de tareas): ¡caminos críticos!



Gestión: planificación: plan de proyecto

Alcance

- objetivos
- desglose de subsistemas
- funcionalidades principales
- otras características
- escenario de desarrollo
- restricciones técnicas y de gestión

Estimación de recursos

- recursos humanos
- recursos hardware y software
- ventanas de disponibilidad

Estimación de coste

Plan temporal

- red de tareas PERT
- diagrama GANTT

Equipo de desarrollo

- roles y responsabilidades

Requisitos de disponibilidad del cliente

- información/datos
- reuniones
- otros

Fases del modelo genérico

El proceso es una secuencia de actividades técnicas, colaborativas y organizativas con el objetivo de construir un sistema.

Las **actividades fundamentales**,

- especificación
- diseño e implementación
- validación
- evolución

pueden estár organizadas de distinta forma en los diferentes modelos,

secuenciales (cascada), entrecruzadas (incremental), delegadas (reutilización)...

y presentar variaciones según tipo de software o estructura organizativa

metodologías ágiles: pruebas como código, evolución a través de refactorización...

Actividad de especificación

Comprender y definir qué servicios se requieren del sistema, identificar las restricciones y metas. Es una de las etapas más críticas, que puede provocar errores en el diseño y la implementación.

Genera un **documento de requisitos** con dos niveles de detalles: nivel básico para usuarios y clientes (DRU, documento de requisitos de usuario), nivel detallado para desarrolladores (ERS, especificación de requisitos software).

¡Tema 3, Ingeniería de requisitos!

Actividad de especificación (2)

En esta fase de análisis de requisitos se **ANALIZA** el problema (con clientes, usuarios, proveedores e ingenieros; entrevistas, análisis de contexto. Simulaciones, prototipos) y se **DEFINE** el producto (una descripción detallada y completa del sistema final).

Principios:

- comprender el problema y su entorno
- funcionalidades y requisitos se determinan con una aproximación descendente
- la especificación de requisitos debe ser ampliable
- sin influencia de factores técnicos: OS, lenguaje, técnica
- para cada subsistema, identificar requisitos funcionales
- para el sistema global, identificar requisitos no funcionales

  documentación, roles, fiabilidad...

Actividad de especificación (3)

Subactividades:

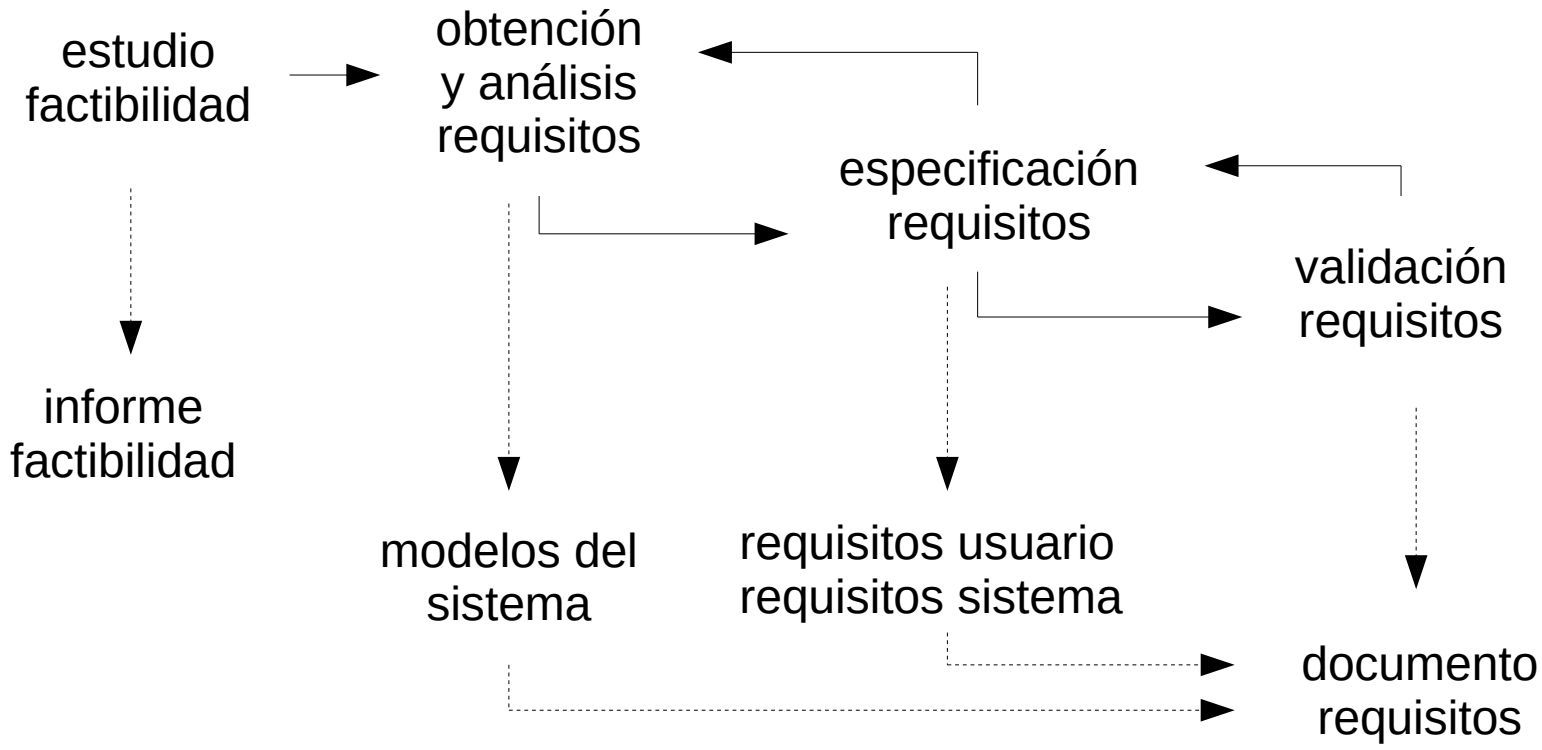
- estudio de factibilidad; rápido y barato; responde a ¿las necesidades del usuario se pueden cumplir? ¿se pueden desarrollar dentro de las restricciones? ¿tiene una buena relación coste/beneficio? ¿seguimos?
- obtención y análisis de requisitos; derivar los requisitos del sistema mediante observación de los sistemas actuales, discusión con los usuarios y proveedores, análisis de la lógica del negocio, usando modelos del sistema y prototipos…

  combinar, dividir, determinar viabilidad, priorizar… requisitos
- especificación de requisitos; transcribir la información anterior a un documento; ¿usar diagramas? dos tipos:
  - requisitos de usuario, descripción para usuario/cliente
  - requisitos de sistema: descripción para desarrolladores
- validación; comprobar que los requisitos sean realistas, coherentes y completos, sin ambigüedad y consistentes

Actividad de especificación (4)

No es necesario seguir estrictamente esa secuencia de subactividades.

Durante todo el proceso pueden aparecer nuevos requisitos. Por supuesto, en metodologías ágiles los requisitos se desarrollan de forma incremental.



Actividad de especificación (5)

Tipos de requisitos:

Funcionales. Acciones fundamentales (elementales) que tienen lugar durante la ejecución para el correcto funcionamiento del producto.

No funcionales. Representan características generales del sistema en lugar de acciones y funcionalidades. Hacen referencia a atributos y propiedades del sistema informático. Pueden estar enfocados al usuario o al ingeniero.

IEEE 830/1993 establece hasta 13 tipos de requisitos no funcionales:

de interfaz y usabilidad; de rendimiento; de seguridad; de disponibilidad; operacionales; de documentación; de fiabilidad; de recursos; de soporte; legales...

Actividad de especificación: ERS

Especificación de requisitos software:

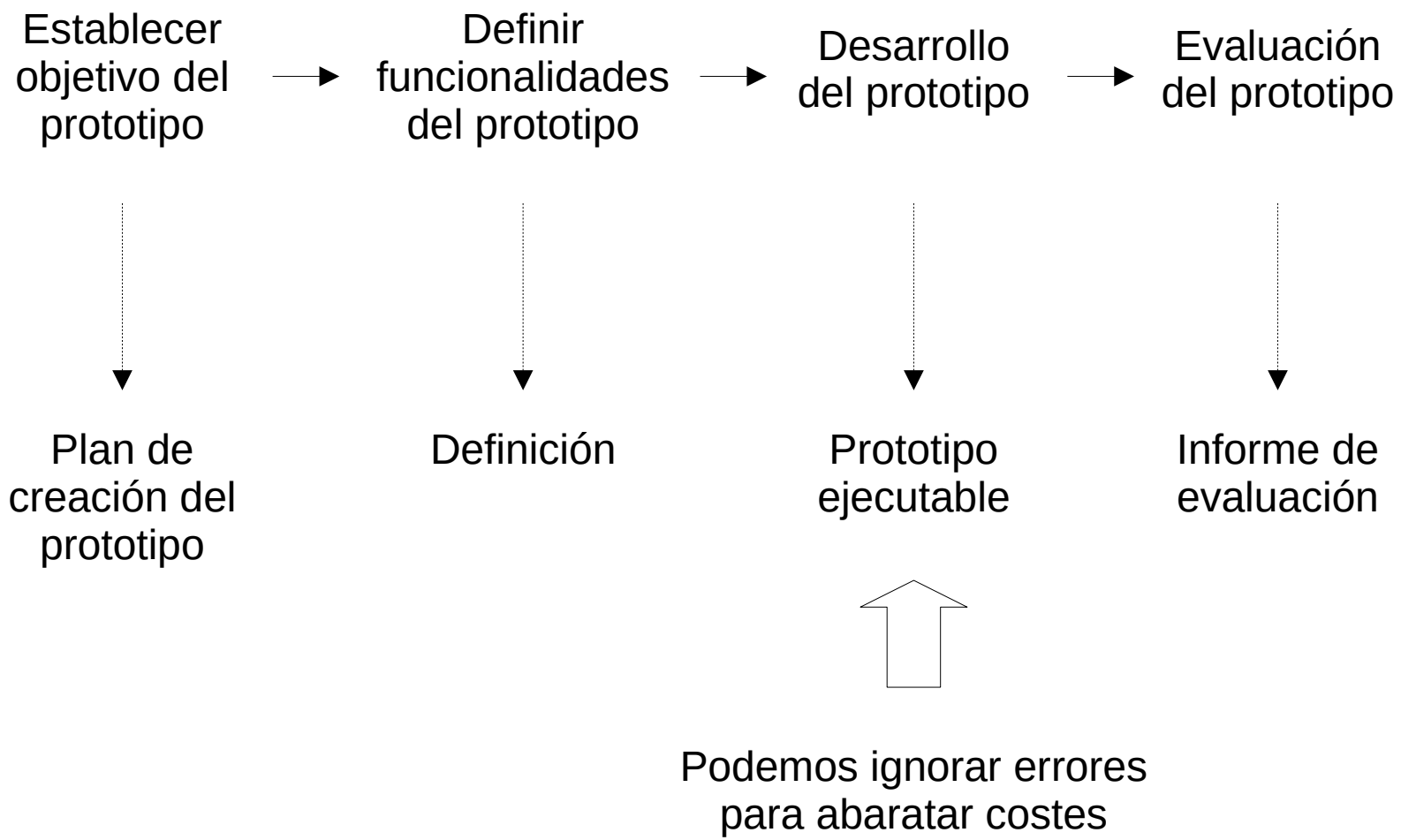
- introducción
- documentación general
- descripción de la información
- descripción funcional
- requisitos específicos
- descripción del comportamiento
  - diagramas
- criterios de validación
- conclusiones

Diseño de prototipos

Etapas donde utilizarlos:

- en la especificación de requisitos; selección y validación de requisitos de sistema
  - cuando hay dudas sobre requisitos software; se construyen subsistemas con funcionalidad completa; se van refinando iterativamente e implementando nuevas funcionalidades… a veces hasta convertirse en el producto
- en el diseño de sistema; buscar soluciones específicas: diseño interfaz de usuario... ¡no va a cumplir todos los objetivos! sólo está interesado por los requisitos funcionales

Diseño de prototipos (2)



Actividad de diseño e implementación

Es el proceso de convertir las especificaciones software (qué) en un sistema ejecutable (cómo).

Distinguimos:

- proceso de diseño, típicamente iterativo
- proceso de programación, que puede corregir las especificaciones si se sigue un enfoque incremental

Diseño e implementación: diseño

No hay una única solución de diseño válida (elegimos la más simple).

El DISEÑO describe:

- estructura software
  - arquitectura del sistema
  - componentes y módulos
- modelos y estructura de datos
- interfaces entre componentes
- algoritmos

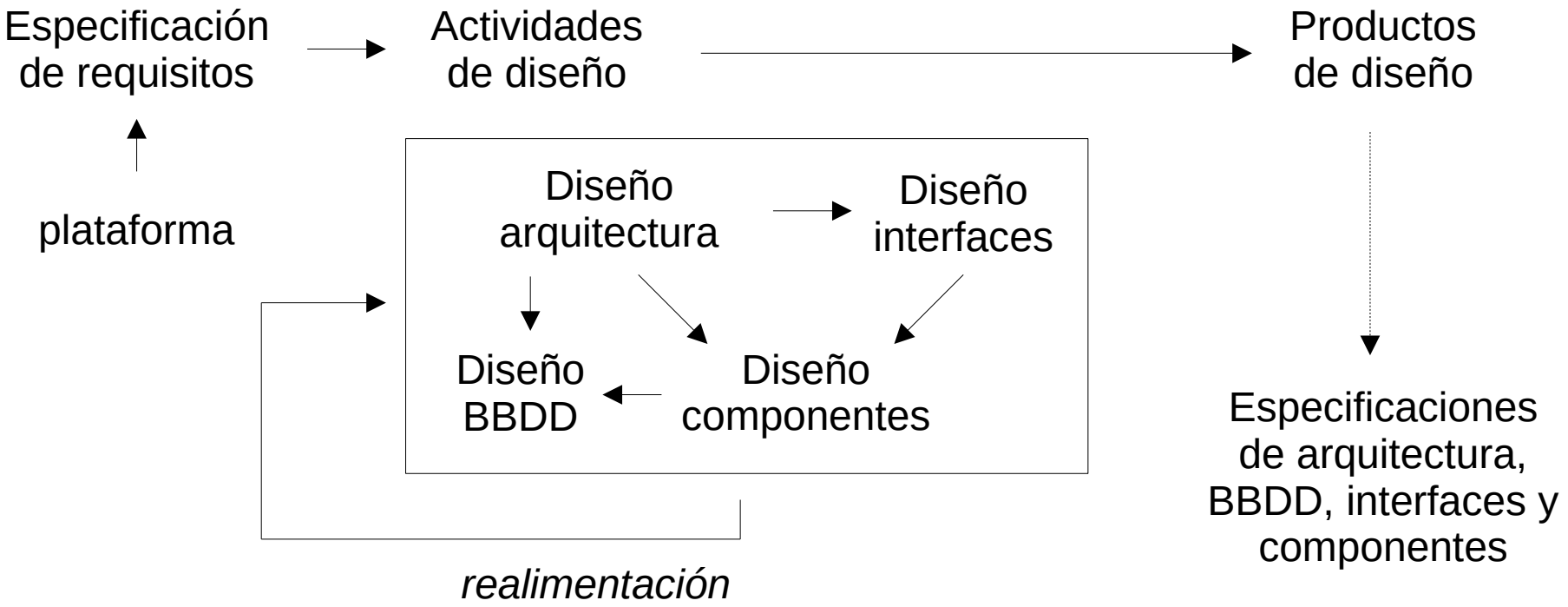
Diseño e implementación: diseño (2)

Dos niveles de diseño:

- alto nivel (diseño de arquitectura); define los componentes, interfaces y estructuras de datos
- bajo nivel, detallado; define la lógica y estructura de los componentes,sus estructuras de datos y los procedimientos de acceso

En la etapa de diseño también se define la **estrategia de pruebas** que se utilizará en el proyecto.

Diseño e implementación: diseño (3)



Diseño e implementación: diseño (4)

Principios de diseño:

- refinamiento, estrategia descendente
- modularidad, dividir el sistema en unidades pequeñas con entidad propia
- abstracción, identificar conceptos generales
- ocultación de la información, cada módulo debería ser privado

Un buen diseño se forma con una descomposición en módulos independientes, donde cada uno

- tiene bajo acoplamiento (dependencia en la interconexión de módulos)

  simplifica el entendimiento, limita la propagación de errores
- tiene alta cohesión (relación entre elementos del módulo)

  menor esfuerzo de programación y pruebas, mayor calidad

Diseño e implementación: diseño (5)

Subactividades:

- diseño de la arquitectura; selección de una arquitectura global, principales módulos y relaciones entre ellos
- diseño de interfaz; conexión entre componentes (se define el interfaz independientemente de la implementación)
- diseño de componentes; funcionamiento de cada componente; si ya existe, lista de cambios necesarios para reutilizar el componente
    - diseño de datos; puede tratarse como un componente más
- diseño de las bases de datos; cómo representar las estructuras de datos en el almacenamiento y sus relaciones

Las actividades concretas dependen del sistema y productos (¿sensores? ¿temporización? ¿conexión redes? ¿base de datos?...)

Diseño e implementación: diseño (6)

A veces un modelo de diseño puede generar automáticamente las implementaciones; por ejemplo, con herramientas UML.

Productos:

- en enfoques dirigidos por modelo, las salidas son diagramas
- en enfoques ágiles, se representan directamente en el código del programa
- en...

Diseño e implementación: documento de diseño

Introducción

propósito del documento, entorno hardware y software, principales funciones del software, bbdd externas, restricciones y limitaciones, referencias

Descripción del diseño

diagramas, descripción de datos, descripción de interfaces, descripción de comunicaciones

Descripción de los módulos

descripción, descripción de interfaz, módulos relacionados, organización de los datos

Descripción de archivos externos y datos globales

descripción, métodos de acceso

Especificaciones de programas

Referencias cruzadas con los requisitos

Plan de pruebas

estrategia de integración, estrategia de pruebas, consideraciones especiales

Conclusiones

Diseño e implementación: métodos estructurados

Los métodos estructurados de diseño utilizan modelos gráficos para representar el diseño y generan automáticamente las implementaciones.

Ejemplo:

- Métodos dirigidos por modelo (MDD); un tipo de métodos estructurados que genera implementaciones con diferentes niveles de abstracción
- UML; una evolución del anterior

Diseño e implementación: implementación

Traducir las especificaciones de diseño a código, usando un lenguaje de programación. Productos:

- programas
- manual técnico
- manual de usuario
- guía de instalación

No hay un proceso general para programar (excepto en la generación automática a partir del diseño), sólo recomendaciones y buenas prácticas.

¡Tema 2, Metodología TDD para el desarrollo de componentes!

Diseño e implementación: implementación (2)

El desarrollo puede estar mezclado con las pruebas. En ese caso, durante el proceso se realizan:

- **pruebas de código,** para establecer la existencia de defectos: prueba de componentes, pruebas de sistema... pruebas de aceptación
- **depuraciones,** para localizar y corregir los defectos; el proceso implica establecer una hipótesis sobre el comportamiento observado y llevar a cabo pruebas para demostrarlo
  - de nuevo, no hay un proceso general recomendado para la depuración (excepto en los desarrollos formales, que usan las transformaciones matemáticas para introducir nuevas funcionalidades así que no sería necesaria)

Actividad de validación

Incluye actividades de verificación y validación para comprobar que el sistema cumple las especificaciones (requisitos funcionales) y expectativas del cliente (requisitos no funcionales).

Verificación. Es el proceso de evaluación de un sistema o de uno de sus componentes para determinar si los productos (de la fase) satisfacen las condiciones impuestas al principio: ¿estamos construyendo correctamente el producto?

Validación. Es el proceso de evaluación de un sistema o de uno de sus componentes durante o al final del desarrollo para determinar si satisface los requisitos específicos: ¿estamos construyendo el producto correcto?

¡Tema 5, Pruebas de software!

Actividad de validación (2)

Se utilizan:

- pruebas; ejecución con datos de prueba simulados o reales

  se llevan a cabo durante la implementación y justo después

  aumentan la fiabilidad, pero no garantizan la ausencia de defectos

  un sistema o alguno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y se registran, y se realiza la evaluación de algún aspecto

- procesos externos de comprobación:
  - inspecciones
  - revisiones en cada etapa

Actividad de validación (3)

Principios básicos para pruebas:

- el resultado esperado es parte integrante de la prueba: entradas, salidas, salidas esperadas
- los casos de prueba deben ser escritos para condiciones de entrada válidas y no válidas
- el código también deberá ser probado por otro ingeniero distinto al autor y alguien con perfil similar al cliente
- se documentan los casos de prueba
- si el resultado es negativo, se deben indicar los pasos usados para convertirlo en positivo

Pruebas

El usuario desarrollador va probando ya durante la actividad de implementación (es una buena práctica).

Prueba de componentes. Cada componente (funciones, clases, agrupaciones de ellas) se pone a prueba individualmente.

Rápidas, no exhaustivas. Diseñadas para detectar errores en la menor cantidad de tiempo y con los menores recursos.

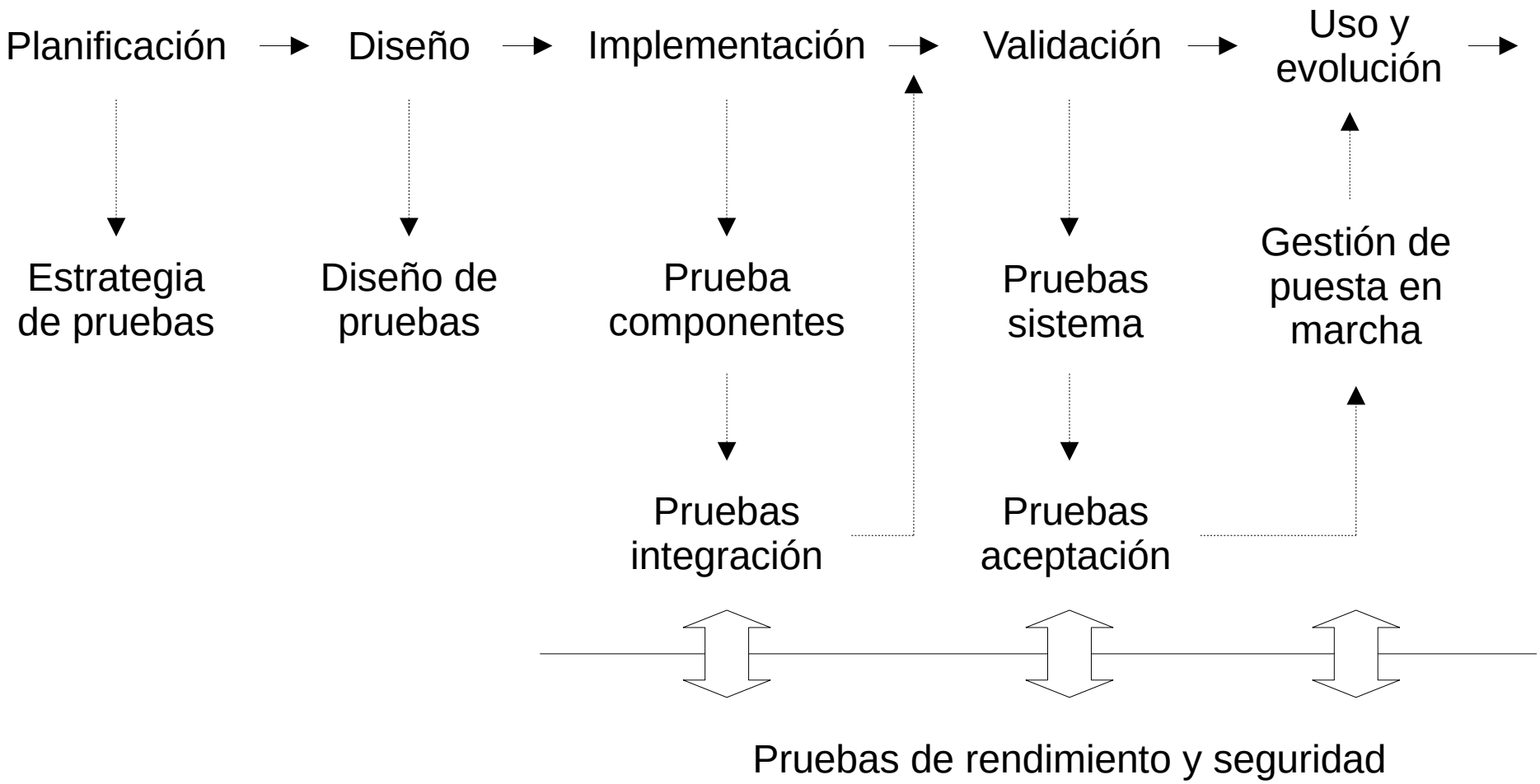
Pruebas de sistema. Integración de componentes. Detecta problemas de interacciones no anticipadas. Demuestra requisitos funcionales y no funcionales.

En sistemas grandes, la integración y validación se realiza en múltiples etapas.

Pruebas de aceptación. Uso de datos reales del cliente. Detecta errores u omisiones en requisitos. Identifica problemas con requisitos no funcionales.

Pruebas (2)

Pruebas en ciclo de vida genérico



Pruebas (3)

Para enfoques incrementales, cada uno de los incrementos debe probarse (se generan sistemas completos para producción). Cada incremento tiene especificaciones propias.

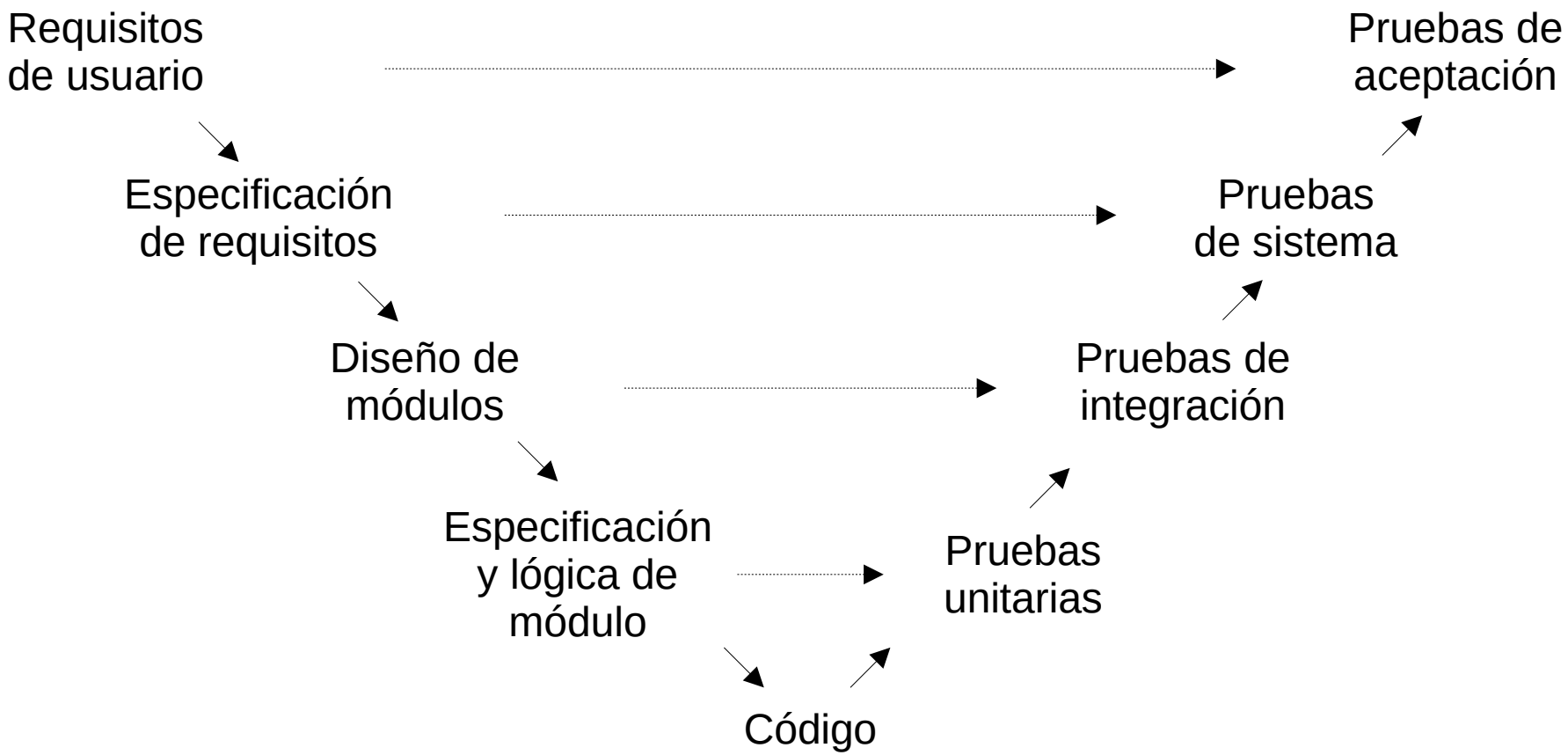
- En el enfoque ágil las pruebas se definen durante los requisitos (sirven como documento de requisitos).

Para enfoques planificados, se construyen conjuntos de planes de prueba en función de los requisitos y resultados del diseño.

- Modelo en V (estrategia de aplicación de pruebas)
- Documentación de diseño de pruebas (IEEE Std 829)

Pruebas (4)

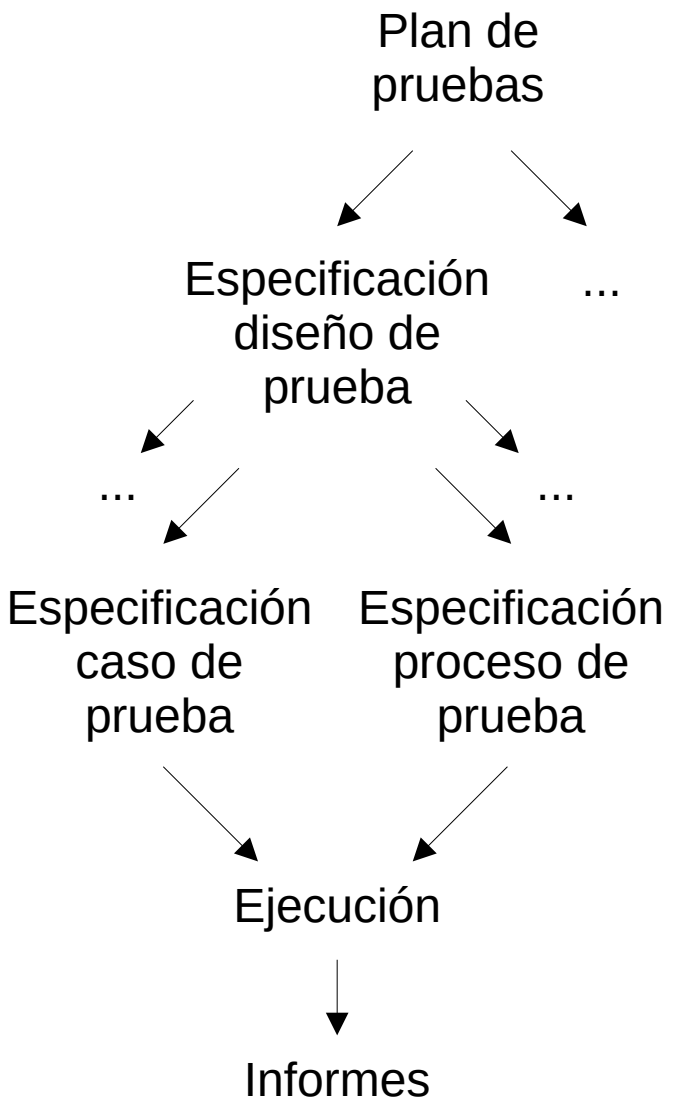
Estrategia de aplicación de pruebas



Pruebas (5)

Documentación del plan de pruebas

IEEE Std 829



Actividad de evolución

La fase de mantenimiento agrupa las actividades realizadas sobre el software cuando ya está en operación:

- corregir errores
- mejorar rendimiento
- añadir funcionalidades
- adaptarlo a un cambio de entorno

Uno de los objetivos de la codificación es la mantenibilidad (facilidad para corregir, adaptar y mejorar el software); es una métrica de calidad. Nos ocupamos de ella en todo el ciclo de vida.

¡Distinguimos entre *ciclo de vida del desarrollo del proyecto* y *ciclo de vida de mantenimiento del proyecto*!

Actividad de evolución (2)

Si es necesario rediseñar el sistema para cumplir los nuevos requisitos, queremos mantener la mayor cantidad posible de los componentes.

Sistema tolerante al cambio. Se diseñan para permitir cambios incrementales, aunque luego no se desarrollen.

- la refactorización es el mecanismo básico para apoyar la tolerancia al cambio

Sistemas no modificables. Se anticipan todas las necesidades y no se tocan. Sistemas temporales (prototipos).

Actividad de evolución (3)

Tipos de mantenimiento:

- correctivo; eliminar errores
- preventivo; prevenir errores
- perfectivo; mejorar, añadir requisitos o ampliar
- estructural; reingeniería que modifica la arquitectura interna
- adaptativo; acomodar software a cambios en el entorno o normativa

Actividad de evolución (4)

Estrategias:

- mantenimiento estructurado; almacenar solicitudes para realizarlas cada cierto tiempo

  se evalúan, se analizan y se aceptan, para realizarlas al principio del siguiente período de mantenimiento

  permite una mejor estimación de recursos

- no estructurado; ejecución bajo demanda

- híbrido; combinación de ambas

  se analiza su severidad; si es alta, se ejecuta inmediatamente (correctivas graves); si es media o baja, se reserva para el siguiente período

Actividad de evolución: documentos

Contrato de mantenimiento con el cliente (condiciones bajo las que se realiza el proyecto).

Plan de mantenimiento (establece qué, quién, cuándo y cómo), que se va actualizando con el desarrollo.

Formulario de solicitud de mantenimiento (para que el cliente pueda hacer solicitudes).

Informe de cambios (recoge información de cada cambio: objetivos, fecha, esfuerzo, responsable…)

Informe de **mantenimiento correctivo**: identificador, fecha, nombre de la persona que informa, error encontrado, condiciones de entorno, versión, consecuencias, soporte (datos de entrada)…

Informe de **mantenimiento adaptativo y perfectivo**: identificador, fecha de petición, petición, motivo de petición, quién realiza la petición...

Actividad de evolución: configuraciones

La gestión de configuraciones es la tarea de identificar, controlar y organizar el sistema software (código, documentación y datos) durante todo el ciclo de vida del proyecto: desarrollo y mantenimiento.

Ofrece coherencia entre versiones.

Permite:

- seguridad ante pérdidas de personal
- reutilización del software

Actividad de evolución: configuraciones (2)

Elemento de configuración. Cada uno de los componentes básicos del sistema que cumplen:

- evolucionan durante el ciclo de vida
- nos interesa controlar su evolución
- puede asignárseles un identificador único

Por ejemplo los documentos de análisis, documento de diseño de la base de datos, código…

Línea base. La configuración de referencia en el ciclo de vida a partir de la cual hacemos revisiones formales. Se encarga de controlar los cambios en el software, pero sin impedir llevar a cabo los cambios justificados.

Base de datos de proyecto. Almacena los elementos de configuración de una línea base. Ante un cambio, el elemento se extrae, se modifica, se reintroduce y se avisa.

Actividad de evolución: configuraciones (3)

Plan de proyecto

Fase de análisis

- documento de requisitos de usuario, especificación de requisitos software

Fase de diseño

- diseño de arquitectura, diseño detallado de módulos y datos externos

Fase de codificación

- código, pruebas, manual técnico

Entrega

- código, manual de usuario, guía de instalación, documento de resultados relevantes

Informe de esfuerzo final (frente a la estimación inicial)

Líneas base, control de versiones, control de cambios