


Fundamentos de Ingeniería del Software



Capítulo 7. Modelos del ciclo de vida del software



“Caminar sobre las aguas y desarrollar programas a partir de las especificaciones es fácil, si ambas están congeladas”

Edward V. Berard

Cap. 7. Modelos del ciclo de vida del software. Estructura



1. Concepto de modelo del ciclo de vida
2. Modelos del ciclo de vida
 - Modelo en cascada
 - Modelo en cascada con prototipado desechable
 - Paradigma de programación por transformaciones
 - Ciclos de vida evolutivos
 - Modelo incremental
 - Modelo en espiral
 - Modelo de ensamblaje de componentes
 - Técnicas de 4ª generación
 - Ciclos de vida orientados a objetos
3. Modelado del proceso software

Cap. 7. Modelos del ciclo de vida del software. Bibliografía



- (Pressman 2006) Cap. 3
- (Pressman 2002) Cap. 2. Aptdos. 2.3-2.11
- (Piattini et al. 04) Cap. 3. Aptdo. 3.6
- (Piattini et al. 96) Cap. 3. Aptdos. 3.3-3.6
- (Larman 03) Larman, C. *"UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado"*, Segunda Edición, Prentice-Hall, 2003. Aptdo. 37.5
- (Balzer et al. 83) Balzer, R., T.E. Cheatham, and C.C. Green, *Software Technology in the 1990's: Using a New Paradigm*. Computer, 1983. **16**(11): pp. 39-45

1. Concepto de modelo del ciclo de vida



Recordemos que (Capítulo 6):

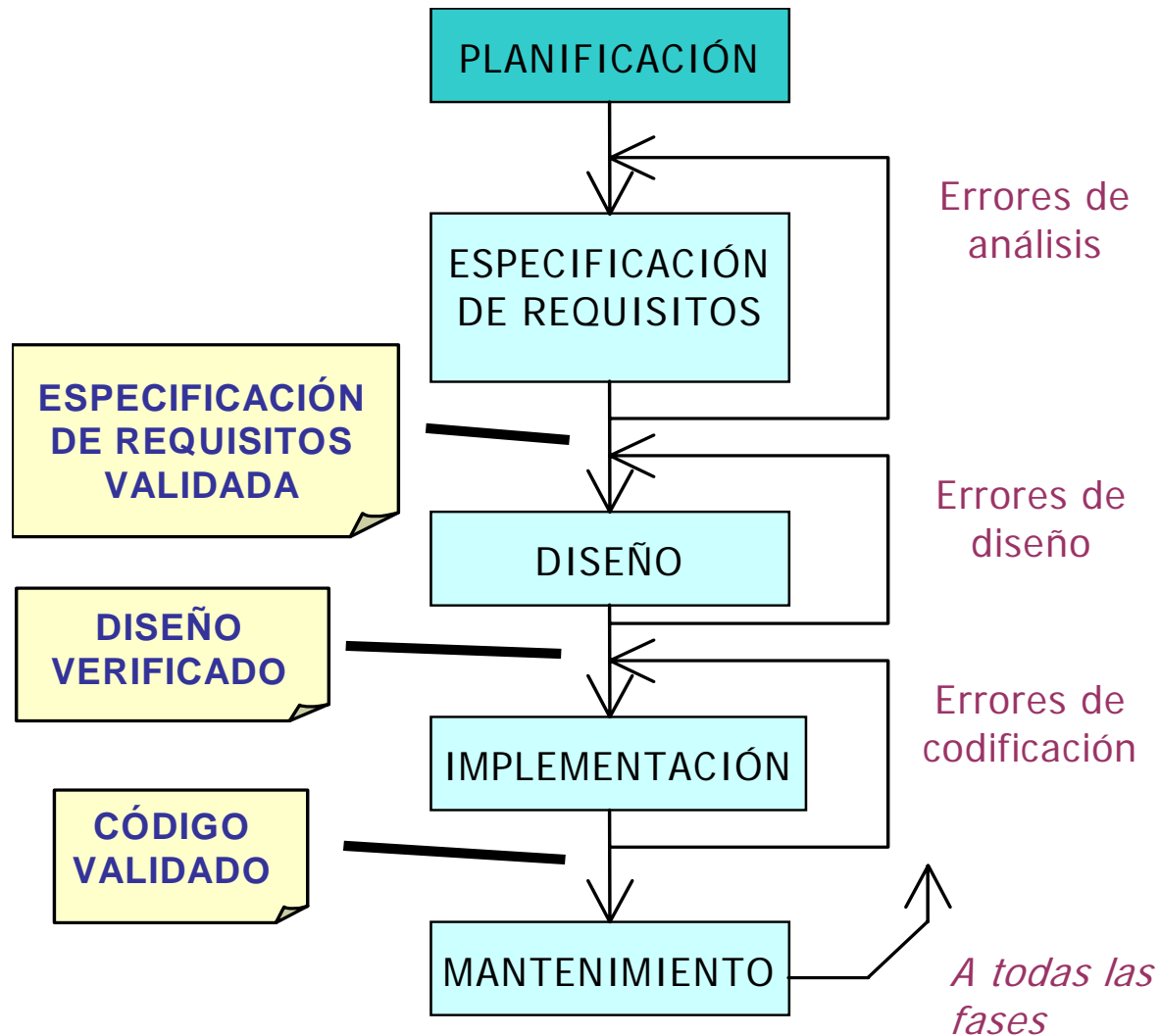
- Es una descripción de un proceso de software que se presenta desde una perspectiva particular.
- Es una abstracción de un proceso real.
- Existe una gran variedad de modelos diferentes “genéricos” o paradigmas de desarrollo de software.

2. Modelos del ciclo de vida. Modelo en cascada (waterfall model)



- Primer modelo empleado (Royce 1970)
- También denominado
 - “ciclo de vida clásico” o “paradigma clásico”
 - “orientado a fases”
 - “lineal secuencial”
- Ejecución secuencial de una serie de fases
- Cada fase genera documentación para la siguiente
- Varias propuestas

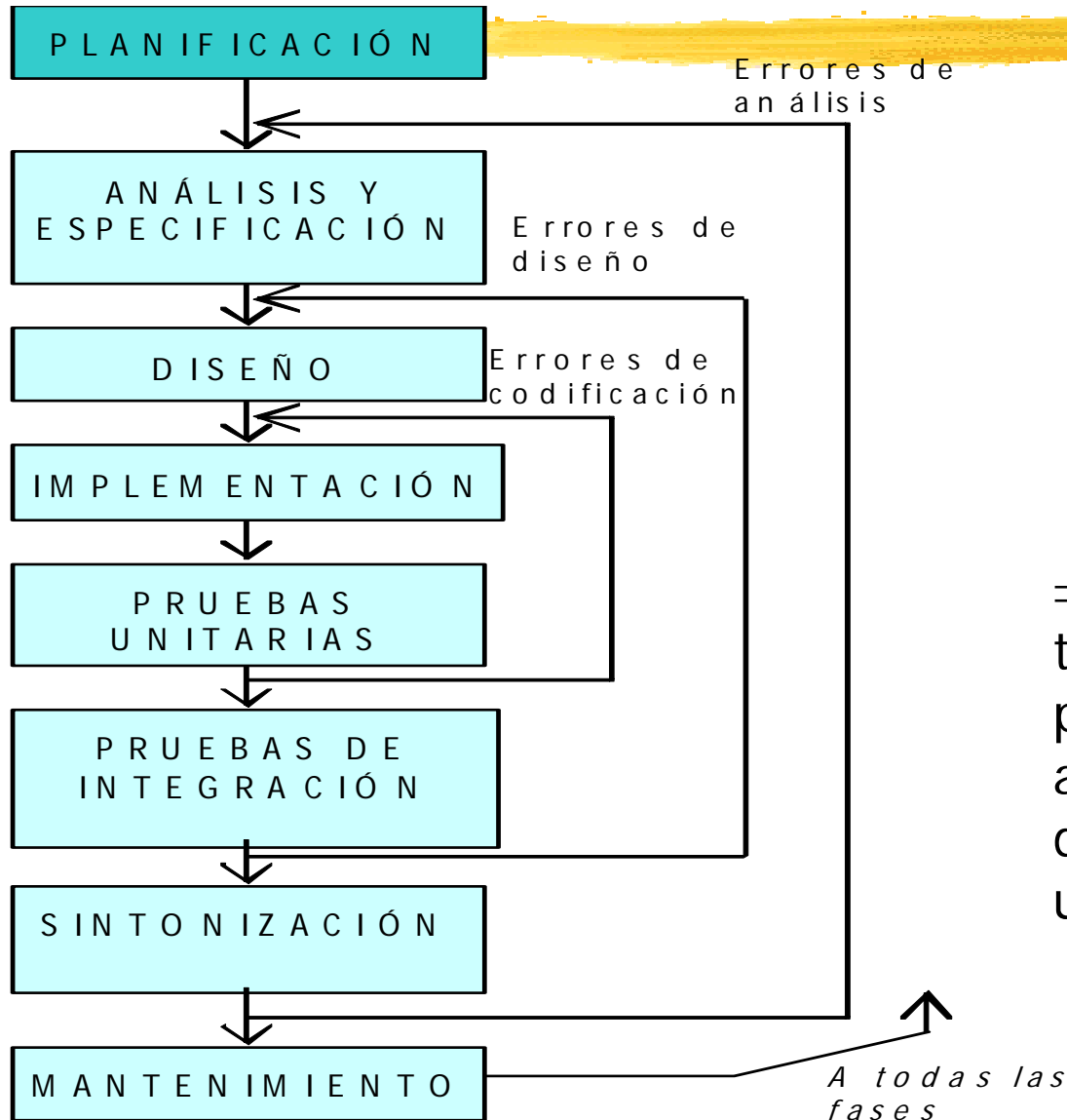
Modelo en cascada ideal



⇒ ¿es realista?

Modelo en cascada.

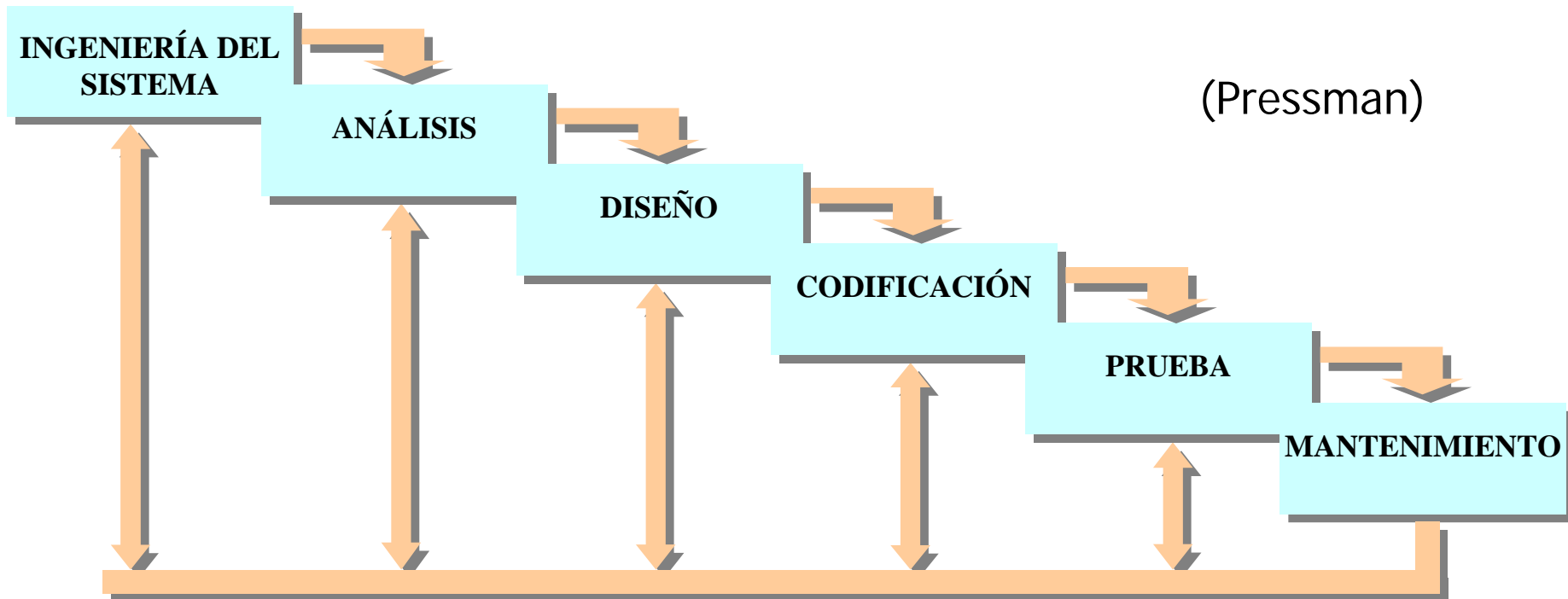
Definición alternativa



⇒ nótese el tiempo que transcurre desde que se produce un error de análisis hasta que se detecta por parte de los usuarios

Modelo en cascada.

Definición alternativa (II)



■ Es similar al enfoque de ingeniería según el cual se construyen los edificios o los puentes (Larman 2003)

⇒ sin embargo, en el caso del software no funciona bien

Modelo en cascada. Fases



- Ingeniería (análisis) del sistema
 - El software suele formar parte de un sistema mayor:
 - Identificar los requisitos de todos los elementos del sistema
 - Diseñar la arquitectura del sistema
 - Asignar un subconjunto de dichos requisitos al software
 - *¿Qué debe hacer el sistema?*

Modelo en cascada. Fases

(II)



■ Análisis de los requisitos del software

- El proceso de recopilación de los requisitos se centra especialmente en el software
- Hay que especificar:
 - funciones que el software debe realizar
 - la información que el software va a gestionar
 - condicionantes existentes: rendimiento, utilización de recursos, etc.
- Los requisitos del software se documentan y se revisan con el cliente
- Se genera la ERS (Especificación de Requisitos del Software)

Modelo en cascada. Fases (III)



■ Diseño

- Definir la estructura del software que satisfaga los requisitos con la calidad necesaria:
 - Estructura de los datos
 - Arquitectura del software
 - Representaciones de interfaz
 - Determinar los algoritmos
- *¿Cómo se ha de construir el sistema?*
- Diseño preliminar (arquitectónico)
- Diseño detallado

Modelo en cascada. Fases (IV)



Codificación

- A veces se puede realizar de forma automática a partir de un diseño detallado, al menos la generación de esqueletos de código y del esquema de la BD

Prueba

- Pruebas unitarias, de integración, del software, del sistema, de aceptación
- *¿Se ha construido el sistema que se deseaba?*

Modelo en cascada. Fases

(V)



Mantenimiento

- Si se usa, es seguro que el software evolucionará, sufrirá cambios después de que se entregue al cliente
 - *errores, nuevas funciones, aumentos del rendimiento, etc.*
- Volver a ejecutar sobre la aplicación cada una de las fases anteriores

Modelo en cascada. A favor.



- “Es mejor que nada”
(tener en cuenta que fue el primer modelo empleado)
- Proporciona un marco para aplicar métodos, técnicas y herramientas
- Es “sencillo” controlar qué productos se deben generar durante el desarrollo del proyecto
- Para un proyecto corto de p.ej. dos meses, se puede usar un ciclo de vida en cascada (Larman 2003)
 - ⇒ Las dificultades aparecen cuando la escala de tiempo se alarga

Modelo en cascada. Críticas



■ Cuando el proyecto se alarga...

(Larman 2003, apdo. 37.5)

- La complejidad pasa a ser alta.
- En general, las cuestiones de alto riesgo no se abordan lo suficientemente pronto, no existe un intento activo de identificar y mitigar en primer lugar las cuestiones de más riesgo.
- Las decisiones especulativas se incrementan y complican, ya que los requisitos se congelan y no existe retroalimentación a partir de implementaciones y pruebas reales.
- Es irreal pretender congelar los requisitos del sistema. Da una sensación al equipo de desarrollo de estar trabajando sobre un sistema "ficticio".

Modelo en cascada. Críticas

(II)



- En general, establecer todos los requisitos al principio del proceso es un “mito inalcanzable”
 - ⇒ los requisitos no se pueden “congelar”, ¡la única constante es el cambio!
 - “Lo sabré cuando lo vea”: las personas involucradas cambian de idea o no pueden imaginarse lo que quieren hasta que no ven un sistema concreto
 - El mercado cambia
- Es poco realista, los proyectos reales raramente pueden seguir el flujo secuencial que se propone

Modelo en cascada. Críticas (III)



- *“Si, como creo, las estructuras conceptuales que construimos hoy son demasiado complicadas para que se especifiquen con precisión por adelantado, y demasiado complejas para que se construyan sin errores, entonces debemos utilizar un enfoque radicalmente diferente (desarrollo iterativo, incremental)”*

“No Silver Bullet” (Brooks 87)

⇒ actualmente, la atención en ingeniería del software está en modelos de proceso iterativos e incrementales, que ayudan a mitigar los problemas del paradigma clásico

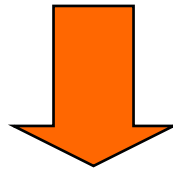
Modelo en cascada. Críticas

(IV)

- Se tarda mucho tiempo en pasar por todo el ciclo

⇒ “el usuario debe tener paciencia”

⇒ hasta que no termina una fase no empieza la siguiente



retrasos innecesarios

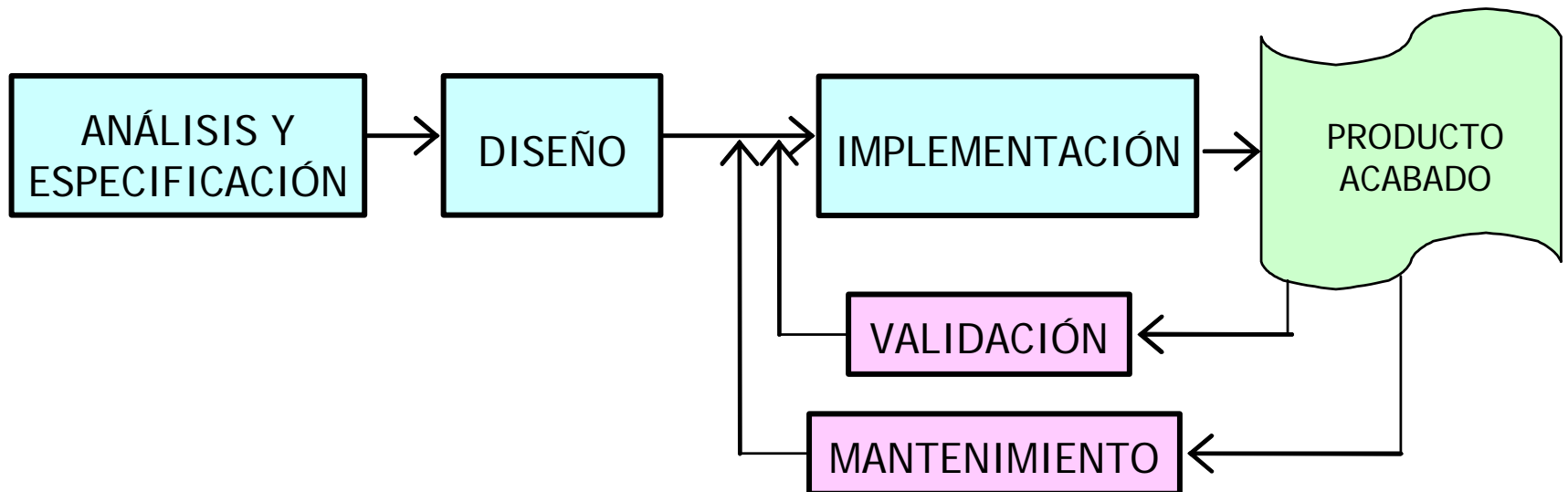
Modelo en cascada. Críticas

(V)

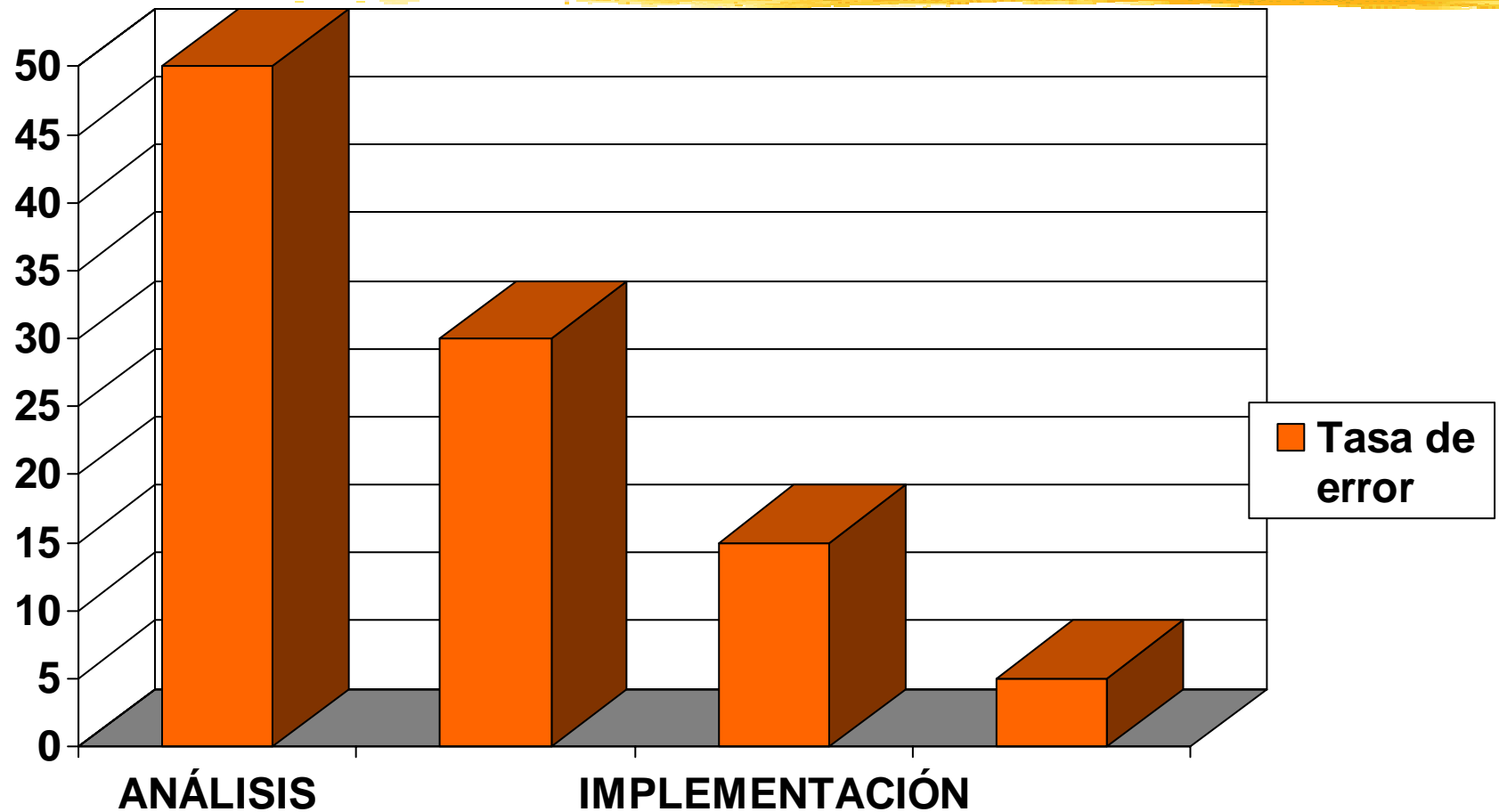


- Los errores de análisis y diseño son difíciles de eliminar, y se propagan a las etapas siguientes con un efecto conocido como "bola de nieve"
- En la práctica, el modelo tiende a deformarse, y el peso de la validación y el mantenimiento recae, en su mayor parte, sobre el código fuente
- El software va deteriorándose y resulta cada vez más difícil de mantener

Modelo en cascada deformado

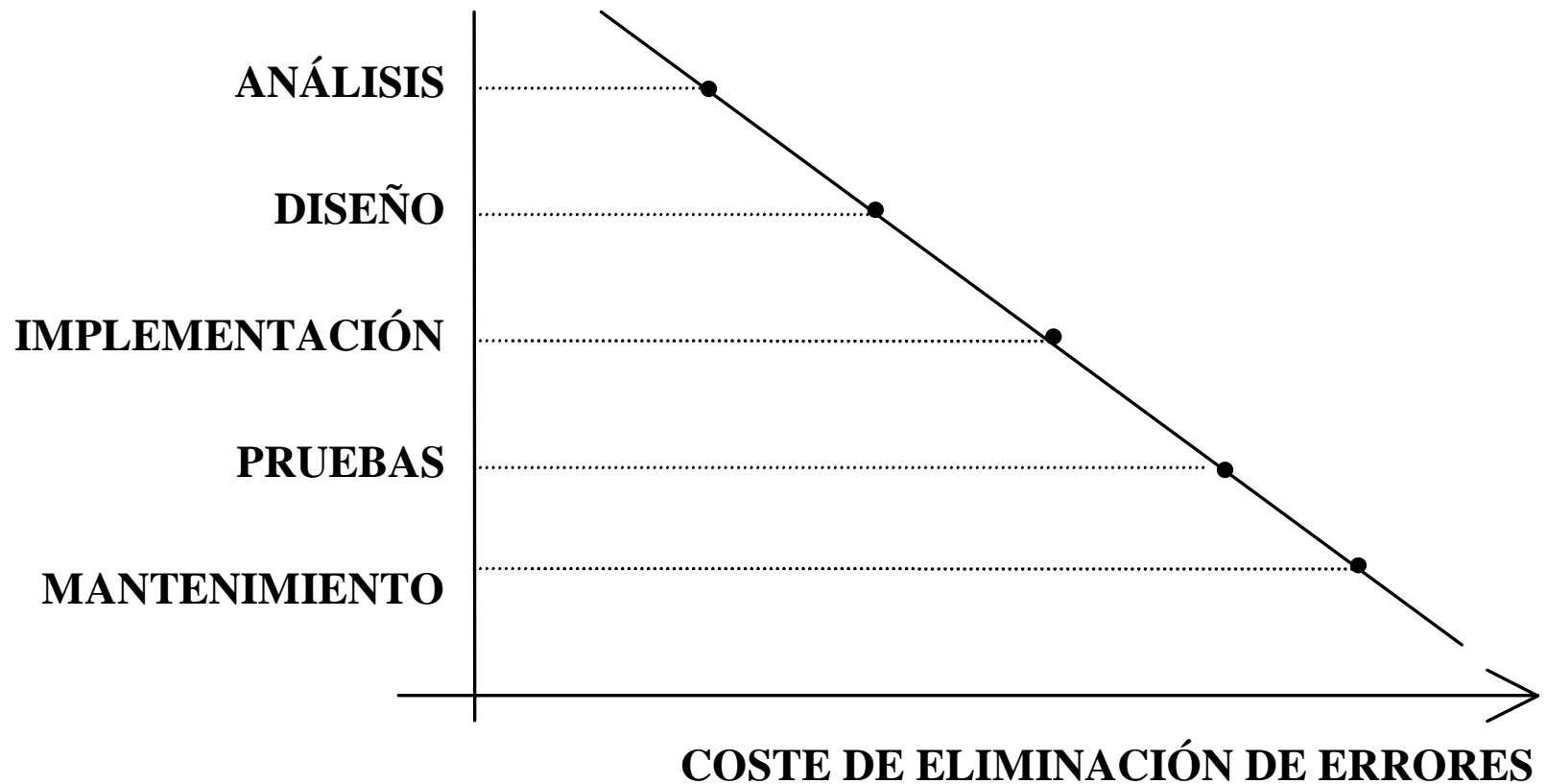


Tasa de errores

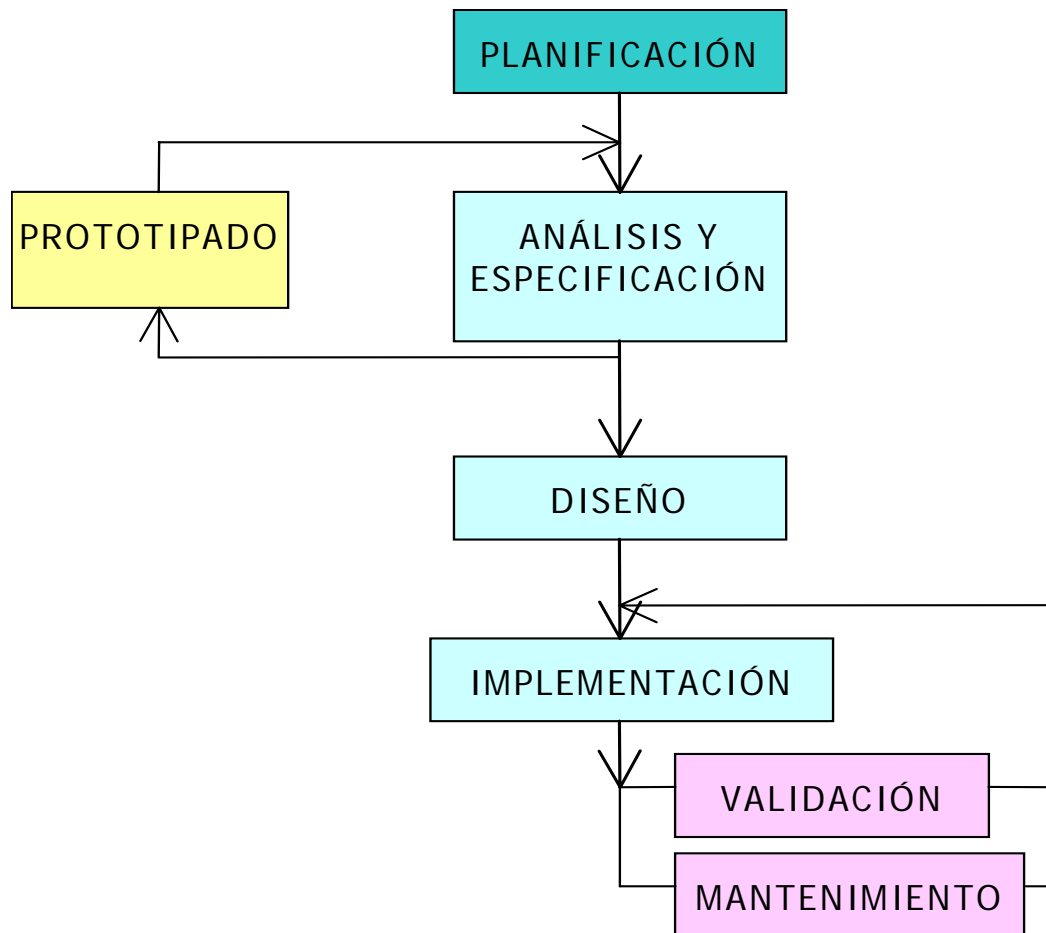


⇒ muchos errores ocurren al principio del ciclo de vida del software: requisitos no descubiertos, mal entendidos, incompletos, mal negociados, etc.

Efecto "bola de nieve"



Modelo en cascada con prototipado desechable



Modelo en cascada con prototipado desechable (II)

- Durante el análisis de requisitos, se construye un prototipo rápido, que ayudará a refinar y validar la especificación de requisitos.
- Después el prototipo se desecha y el desarrollo prosigue en cascada, con mayor seguridad de que los requisitos se han especificado correctamente.

Modelo en cascada con prototipado. Críticas

- El cliente ve en funcionamiento una versión preliminar, sin asumir que no es robusta ni completa \Rightarrow a veces puede pretender “parchear” el prototipo.
- Es frecuente arrastrar malas decisiones (de diseño -SO, algoritmos, etc.-, de planificación...) que sólo eran apropiadas para la obtención rápida del prototipo.
- El tiempo invertido en la construcción del prototipo puede hacer que el producto pierda oportunidad.
- La inversión en un producto desechable puede no ser rentable.
- Ayuda a mitigar el efecto “bola de nieve”, pero no el mantenimiento sobre el código.

Paradigma de programación por transformaciones (Balzer et al. 83)

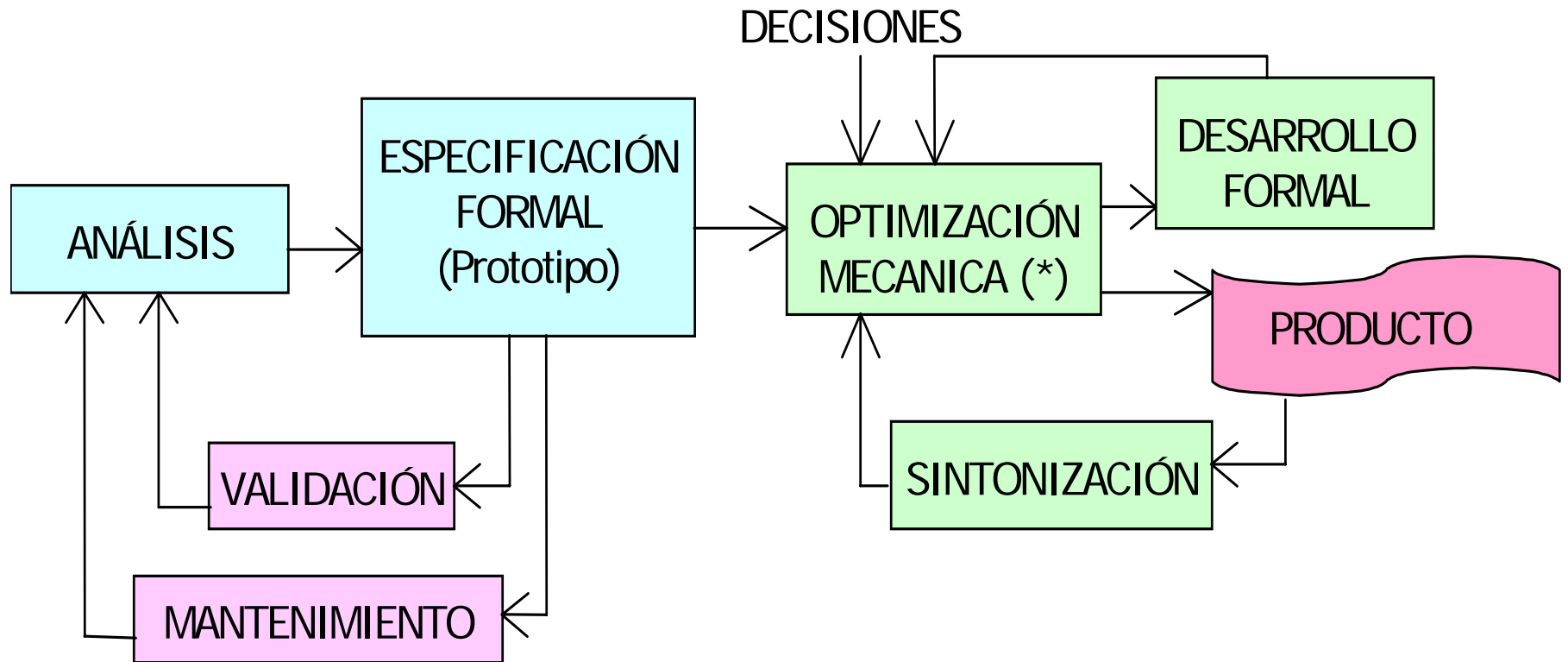
- Objetivo de Balzer: introducir automatización en el proceso de desarrollo del software, y cambiar radicalmente dicho proceso
 - El paradigma clásico está limitado por la debilidad del modelo
 - dispersión de información de optimización por todo el sistema
 - no se documentan suficientemente los procesos, decisiones y razones
 - “se necesita un paradigma nuevo para obtener una ganancia de productividad de órdenes de magnitud”
⇒ Programación automática o por transformaciones

Programación automática



- Idea base: “programación por transformaciones”:
 - Construcción de una primera versión que expresa formalmente el comportamiento deseado.
 - Transformación en una versión más eficiente, preservando la funcionalidad.
- No se trata de dos productos (especificación y prototipo), sino de la conversión automática de una especificación en un prototipo, y luego en el producto definitivo.
- Se puede considerar un antecedente de MDE (*Model Driven Engineering*), como MDA (*Model Driven Architecture*).

Programación automática (II)



(*) (en teoría) rápida, fiable y barata

Programación automática (III)



- Se utilizan lenguajes de especificación formal.
- El prototipo es la propia especificación (o se deriva automáticamente de ella).
- Los requisitos se refinan “animando la especificación”.
- La validación y el mantenimiento recaen sobre la especificación.
- El mantenimiento consiste en *revisar y reemplazar* la especificación, y *rederivar* el prototipo.
- El producto final se obtiene a través de un proceso mecánico de transformación.
- No existe el problema de “bola de nieve”.

Programación automática.

Críticas.



- Compromiso entre las ventajas obtenidas y el nivel al que debe elevarse la especificación
⇒ dificultad en la especificación \approx
dificultad de “programación del sistema”
- La tecnología necesaria para cubrir todo el ciclo de vida no está a punto
 - línea de investigación MDE, MDA
- Puede ser conveniente usar un paradigma combinado:
 - especificación formal → permite V&V formales
 - desarrollo manual, en vez de mecánico

Programación automática y paradigma clásico con prototipado



■ Clásico con prototipado

- Especificación informal (o "semiformal")
- Prototipo "manual"
- El prototipo se desecha
- Se valida comportamiento final frente a código
- Implementación "manual"
- El mantenimiento recae sobre el código (parches o remiendos)

■ Programación automática

- Especificación formal
- La especificación es el prototipo
- El prototipo evoluciona hacia el sistema final
- Se valida especificación contra los requisitos
- Implementación automática (o fuertemente asistida)
- El mantenimiento recae sobre la especificación (reemplazos)

Ventajas del prototipado (ambos paradigmas) sobre el modelo clásico

- El prototipo ayuda a determinar los requisitos, demostrar la viabilidad de una aplicación e investigar sobre los aspectos que producen más incertidumbre.
- El prototipo es un documento vivo para especificar el buen funcionamiento del sistema.
- El prototipo es un contrato con el cliente para el desarrollo del producto (\subseteq ERS).
- Aumenta la productividad del grupo y la calidad del producto.

Ventajas adicionales de la programación automática

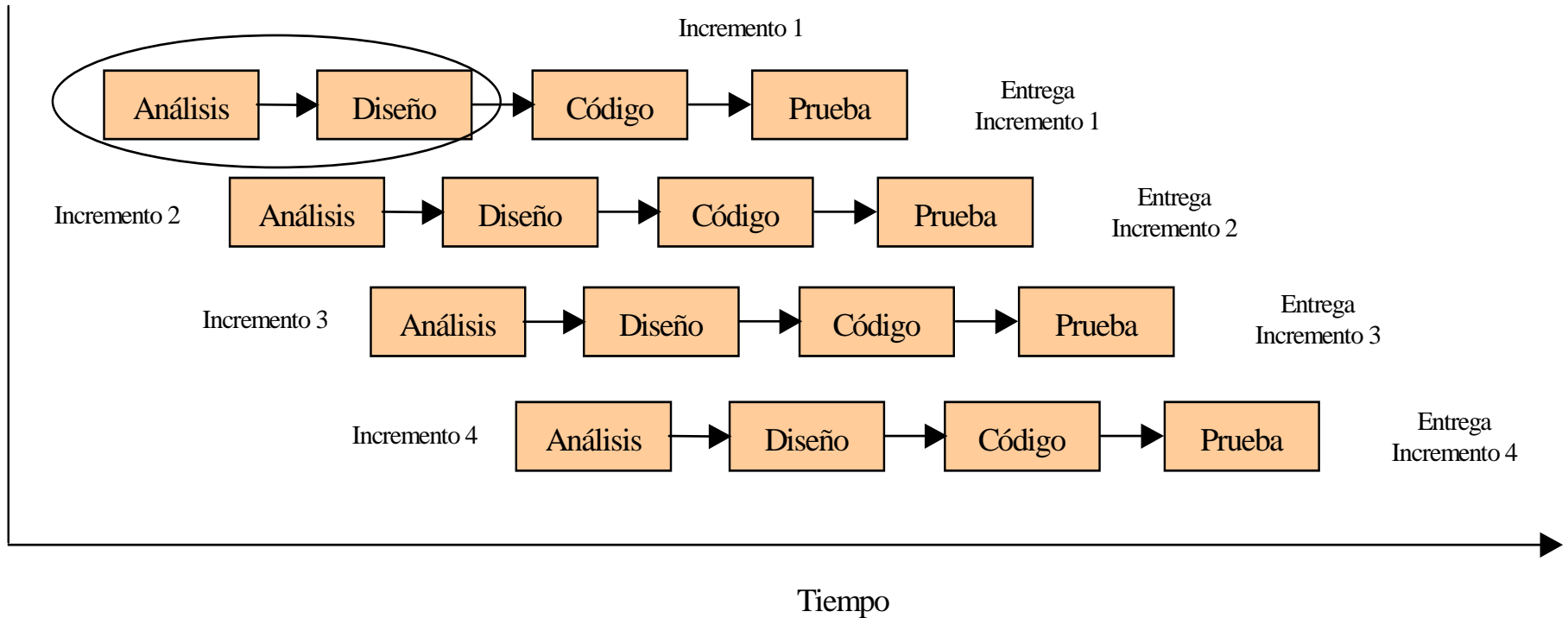


- La especificación es la única interfaz entre usuarios y técnicos, y podría ser creada y mantenida por los propios usuarios, con un lenguaje de especificación apropiado.
- Mayor implicación del cliente en el proceso de desarrollo.
- Ahorro de personal.
- “Reduce el tamaño del sistema” y los costes de mantenimiento.
- El mantenimiento mejora la calidad del software, en vez de degradarla.
- “No más paquetes estándar”.

Ciclos de vida evolutivos.

Modelo incremental

(Pressman 2006) Apto. 3.3.1

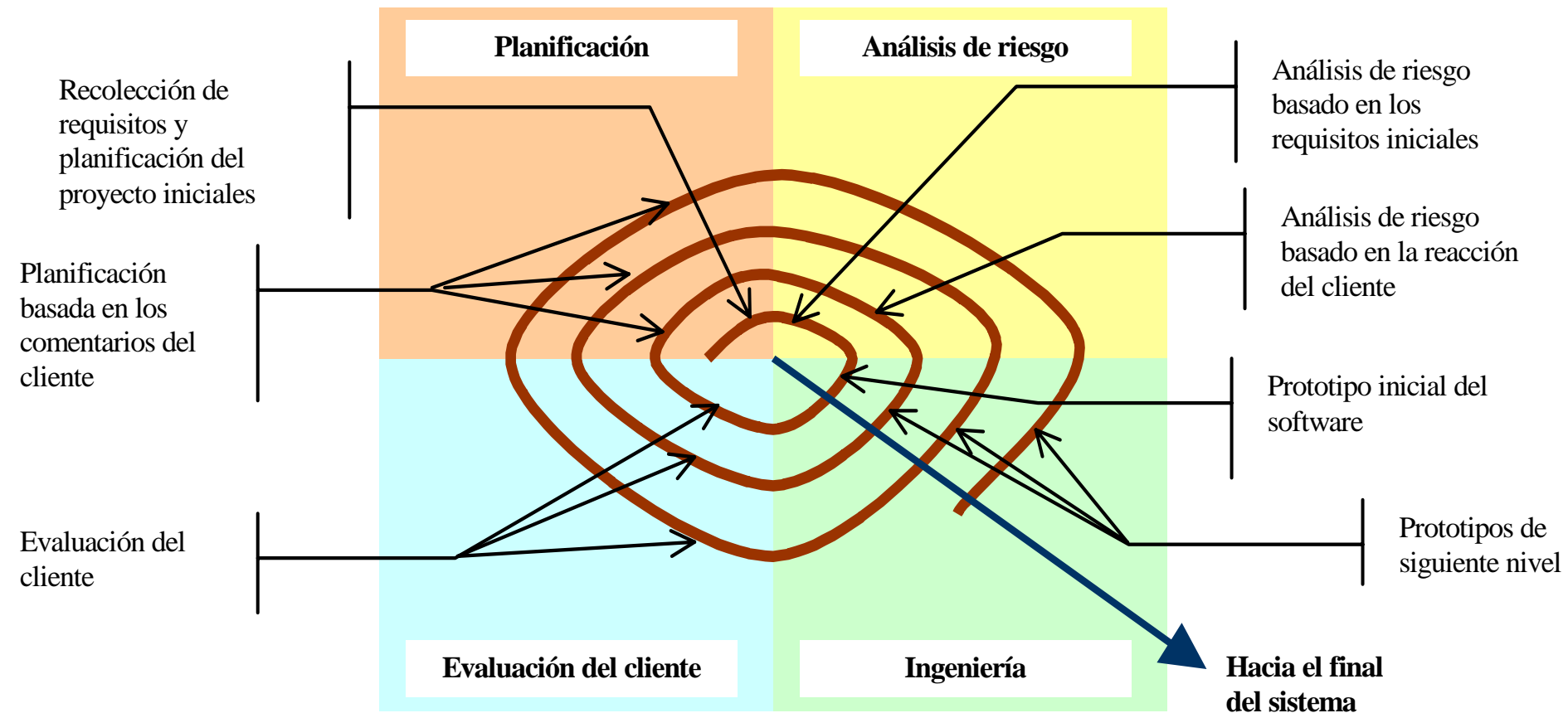


- Se maneja mejor que el paradigma clásico cuando hay fuertes presiones en los plazos de entrega
- Cada secuencia produce un "incremento" del sw.
- Con cada incremento, se entrega un producto totalmente operacional

Ciclos de vida evolutivos.

Modelo en espiral (Boehm 88)

(Pressman 2006) Apto. 3.4.2



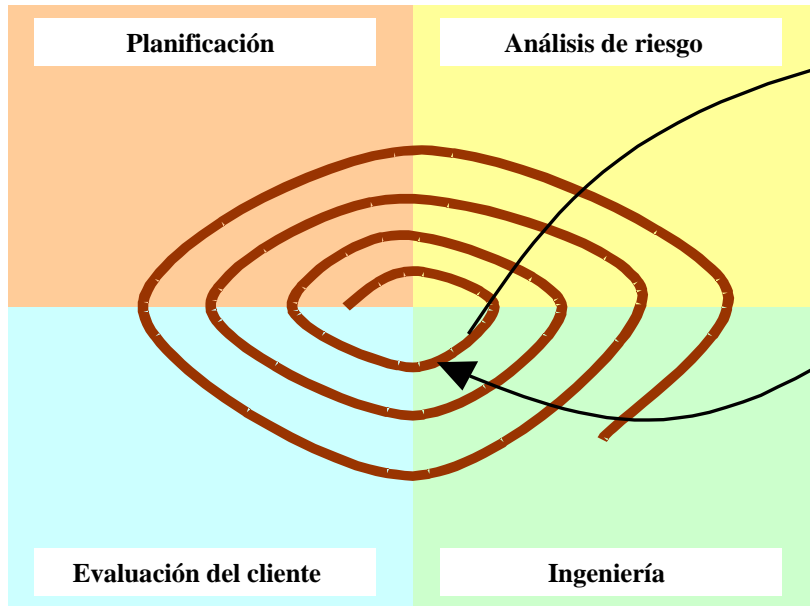
■ Más realista que el ciclo de vida clásico

Modelo en espiral (II)

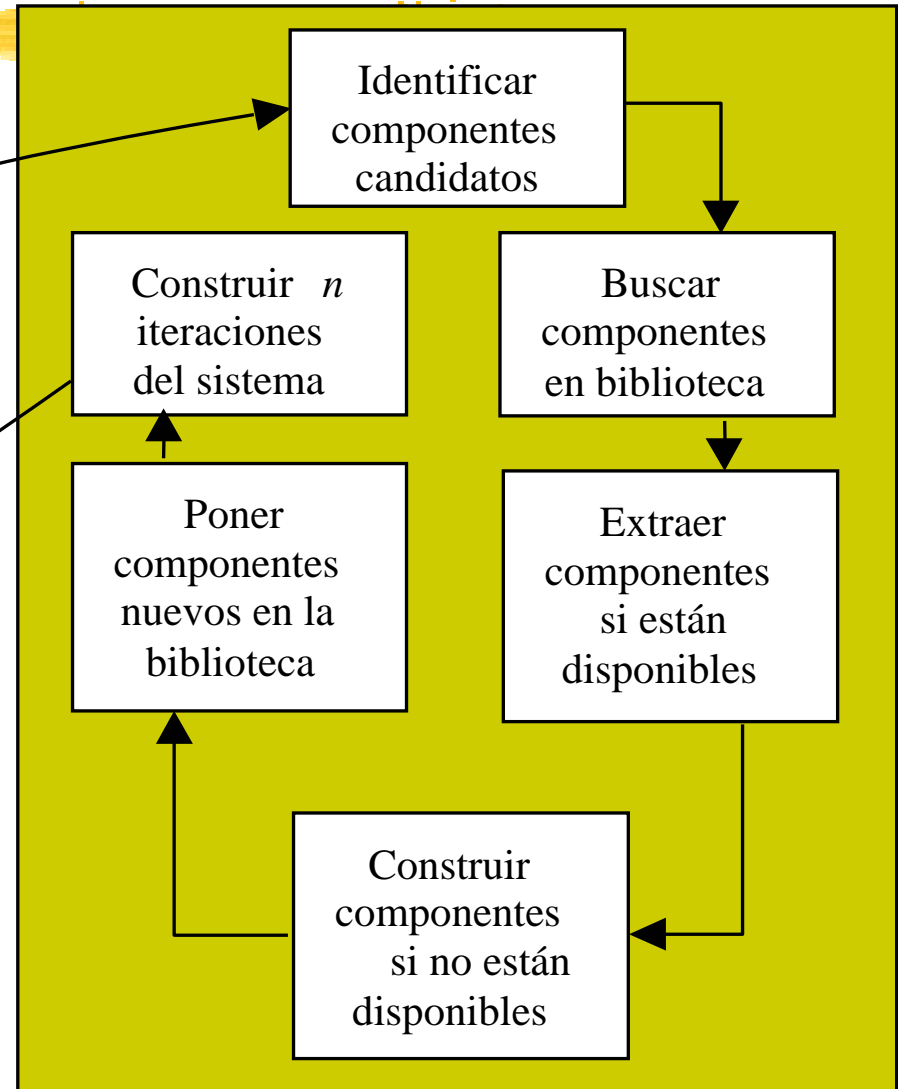


- Se diseña para aprovechar las ventajas del paradigma clásico y las del prototipado evolutivo
- Los productos de las diferentes fases de desarrollo se van reajustando sucesivamente durante la vida del sistema, retomándose de forma sucesiva, como si de una espiral se tratase
- En la planificación se determinan los objetivos, las alternativas y las restricciones.
- En el análisis de riesgo se evalúan las alternativas y se resuelven los riesgos.
- Especialmente útil en desarrollos con requisitos inciertos, o en los que hay áreas importantes de riesgo.

Modelo de ensamblaje de componentes *(Pressman 2002) Aptdo. 2.8*




- Ligado a la OO
- Promueve reutilización del sw.
⇒ t. desarrollo, costes↓↓



Técnicas de 4^a generación


(T4G) (Pressman 2002) Aptdo. 2.10.



Dos pasos:

- 1) Especificación de algunas características del software de alto nivel.
- 2) La herramienta genera automáticamente el código en L4G (4GL):
 - lenguajes no procedimentales de consulta
 - generación de informes
 - manejo de datos
 - interacción y definición de pantallas
 - ...

Técnicas de 4^a generación (II)



*En aplicaciones
pequeñas...*

Análisis de requisitos → implementación

*En aplicaciones
grandes...*

Exige el mismo tiempo de análisis,
diseño y prueba

Ventajas:

reducción t. desarrollo
mayor productividad

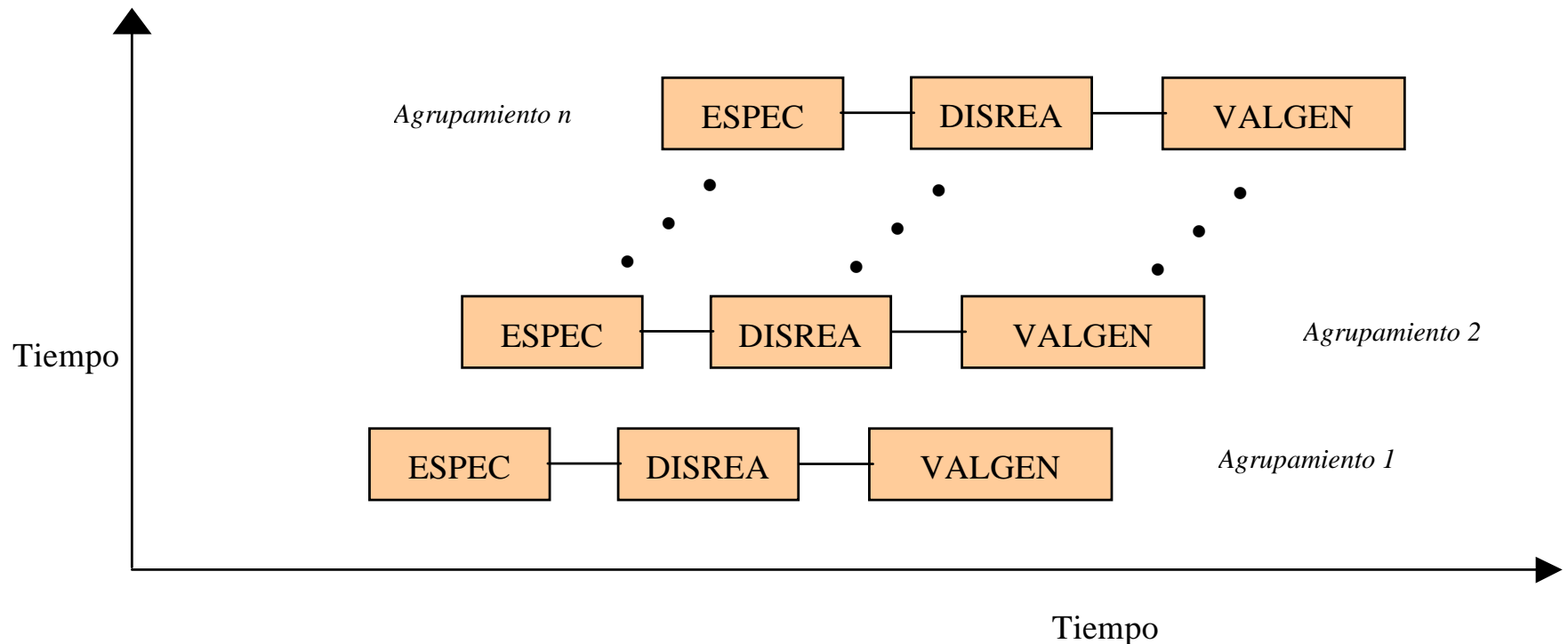
Inconvenientes:

no son más fáciles de usar
código ineficiente
mantenimiento difícil

Ciclos de vida orientados a objetos.

Modelo cluster (agrupamiento) (Meyer 90)

(Piattini et al. 96) pp. 54-55

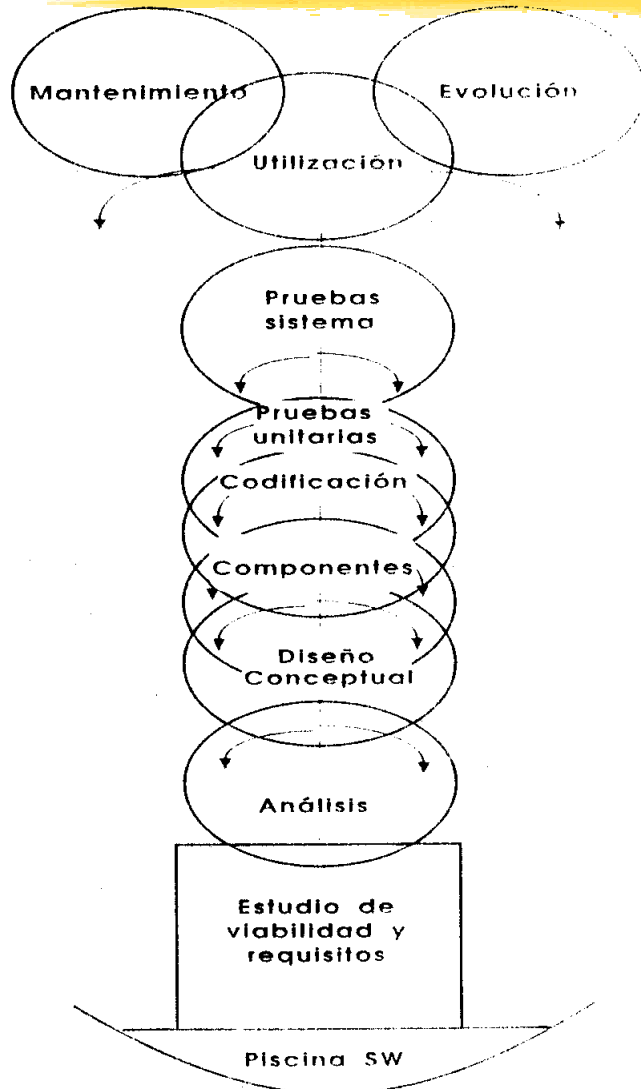


- **Cluster:** conjunto de clases relacionadas con objetivo común
- Cada subciclo de vida: Especificación, Diseño y Realización, Validación y Generalización

Ciclos de vida orientados a objetos.

Modelo fuente (Henderson-Sellers Edwards 90)

(Piattini et al. 96) pp. 55-56



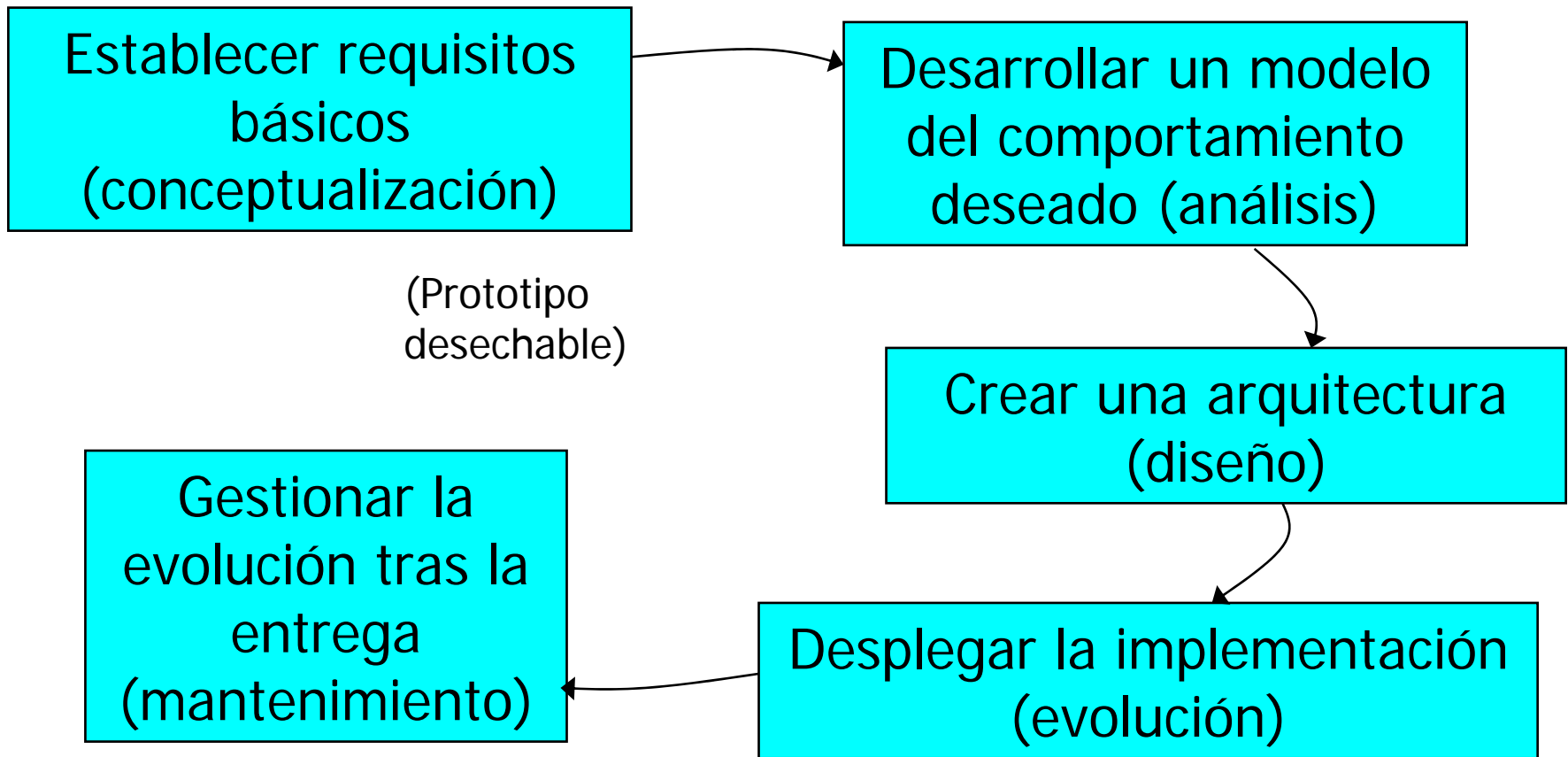
- Alto grado de solapamiento/iteración entre fases
- Cada clase/agrupamiento tiene un ciclo de vida propio
- La "piscina sw" (repositorio de clases) refleja reutilización
 - ⇒ el ciclo de desarrollo "brota" de la piscina sw.

Ciclos de vida orientados a objetos.

Booch 94

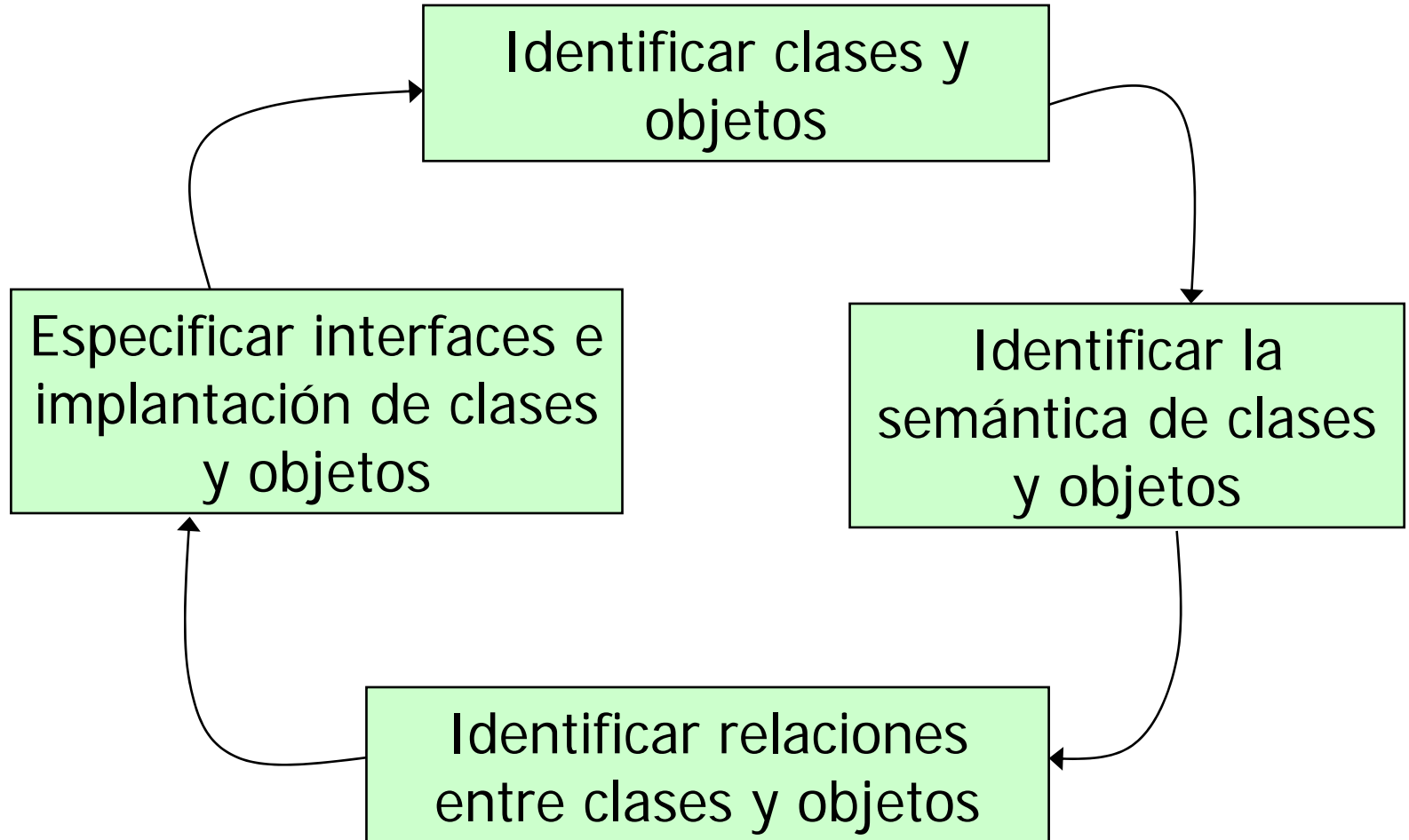
- El *macroproceso* es el marco que controla el *microproceso*: planificación, fechas de entrega, productos a entregar, evaluar el riesgo.
- El microproceso es una guía para las acciones que se desarrollan cuando se está desarrollando la arquitectura del sistema. Todas las actividades son intencionadamente borrosas.
- El macroproceso es de interés a la dirección técnica del equipo de desarrollo, y el microproceso al programador.
- La combinación de macroproceso y microproceso es parecida al ciclo de vida en espiral: el desarrollo evolutivo es muy útil, pero puede ser difícil gestionar el proyecto. Se combina así el ciclo de vida clásico y el OO.

Ciclos de vida orientados a objetos. Booch 94 (Macroproceso)




Ciclo de vida OO

Booch 94 (Microproceso)



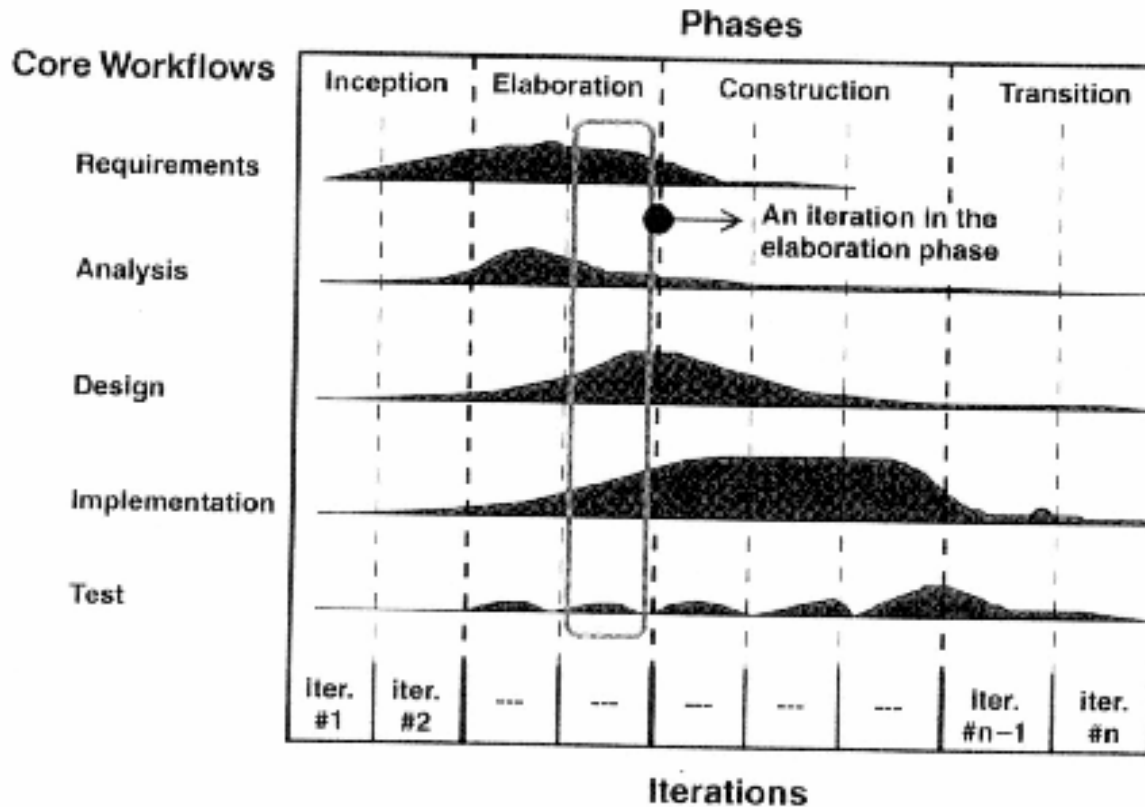
Ciclos de vida orientados a objetos.

Proceso Unificado (Jacobson, Booch y Rumbaugh 99)



- Soporte al estándar del OMG UML
- Entre otros, integra los métodos
 - OMT
 - Booch
 - OOSE/Objectory
- Características principales:
 - Dirigido por casos de uso
 - Centrado en la arquitectura
 - Iterativo e incremental
- Realmente es un framework de proceso, más que un proceso concreto

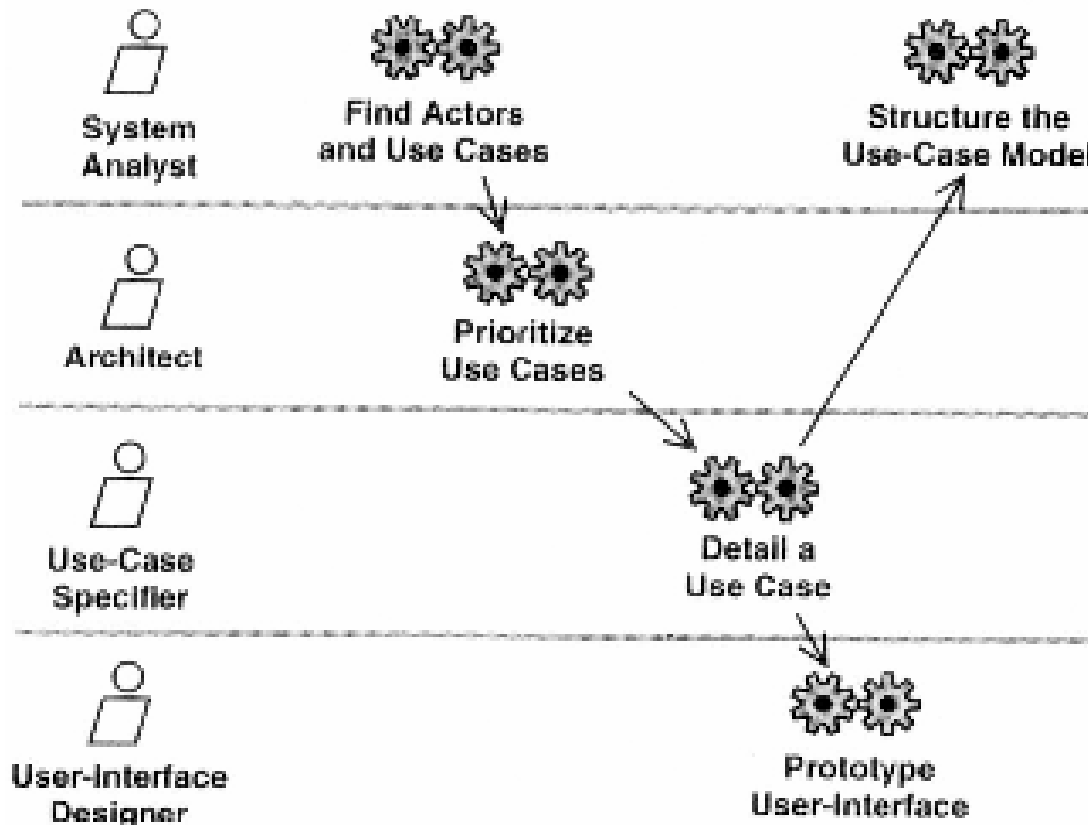
Proceso Unificado. Fases e iteraciones. Disciplinas (workflows)



(Jacobson, Booch y Rumbaugh 99)

Proceso Unificado. Ejemplo de flujo de trabajo (workflow)

Captura de requisitos como casos de uso



(Jacobson, Booch y Rumbaugh 99)

3. Modelado del proceso software (Pressman 2006) Aptdo. 2.7



- Objetivo de las herramientas de tecnología de procesos:
 - Construir un modelo automatizado de la estructura del proceso
 - Redes de Petri, hipergrafos, workflows, etc.
- Beneficios: determinar flujo de trabajo típico, estructuras alternativas de menor tiempo o coste, organizar tareas, controlar el proceso y los productos que se generan, gestionar la calidad técnica, coordinar el uso de otras herramientas CASE, etc.