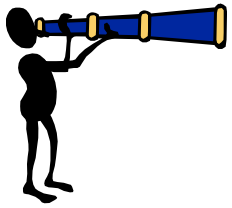
Ingeniería del Software



Tema 1: Introducción a la Ingeniería del Software

Dr. Francisco José García Peñalvo
(fgarcia@usal.es)
Miguel Ángel Conde González
(mconde@usal.es)
Sergio Bravo Martín
(ser@usal.es)

3º I.T.I.S.
Fecha de última modificación: 24-9-2008

Universidad de Salamanca – Departamento de Informática y Automática

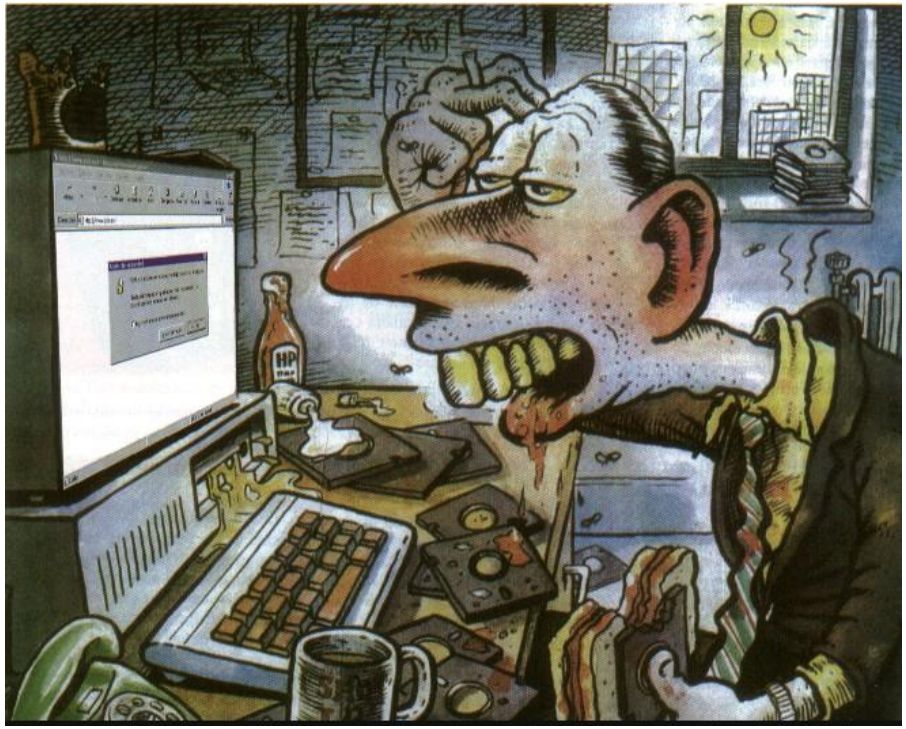
Resumen

Resumen	Se presentan los conceptos clásicos relacionados con el software y la Ingeniería del Software. El objetivo de este tema es tomar conciencia de la importancia de abordar la construcción del software desde una perspectiva de ingeniería. Se exponen los elementos constituyentes de un paradigma de desarrollo del software. Se ofrece una visión general del concepto de proceso, así como de los diferentes modelos de proceso software. Se introduce el concepto de metodología de desarrollo como contraposición al desarrollo anárquico y artesanal de aplicaciones, tan relacionado con la tan nombrada crisis del software. Se presenta, como ejemplo de proceso, el Proceso Unificado. Y se termina el tema hablando de herramientas CASE
Descriptores	Software; Aplicaciones del software; Crisis del software; Proceso software; Modelos de proceso; Ciclo de vida; Metodología; Método; Proceso Unificado; Herramienta CASE
Bibliografía	[Larman, 2003] Capítulo 2 [Piattini et al., 2004] Capítulos 3 y 4 [Pfleeger, 2002] Capítulos 1 y 2 [Pressman, 2006] Capítulos 1, 2, 3 y 4 [Sommerville, 2005] Capítulos 1, 2, 3, 4 y 17

Esquema

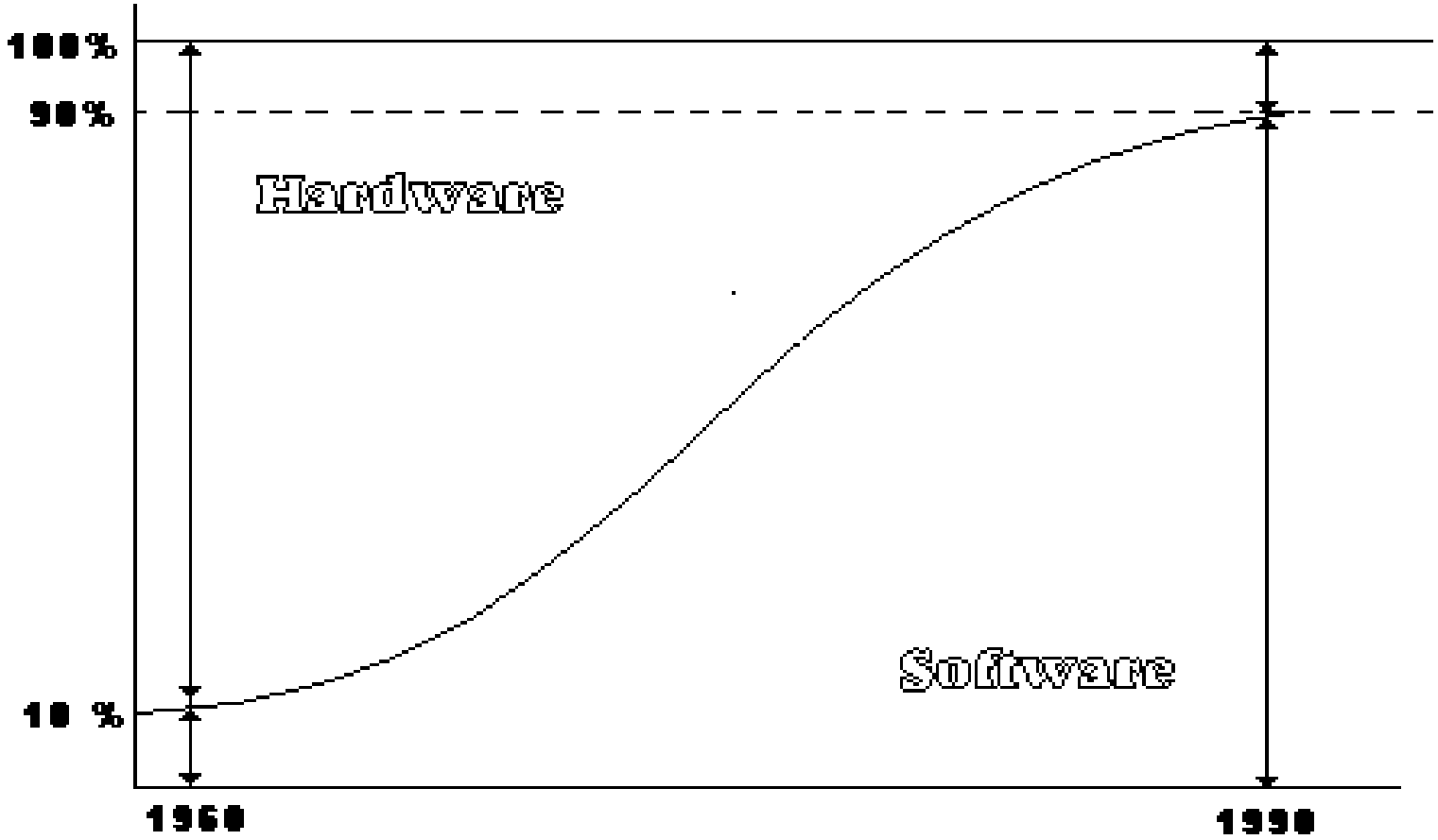
- Software
- Conceptos básicos de la Ingeniería del Software
- Proceso software
- Modelos de proceso software
- Metodologías
- Proceso Unificado
- CASE
- Aportaciones principales del tema
- Ejercicios
- Lecturas complementarias
- Referencias

1. Software



Economía del software (i)

- Evolución de los Costes del Software



Economía del software (ii)

- El software es un producto de consumo con un gran peso en la economía
  - El desarrollo del software de las empresas USA
    - 2 trillones de dólares en desarrollo
    - 30.000 millones de dólares anuales en mantenimiento
  - Gasto en proyectos software en el año 1995 en USA
    - 175.000 proyectos / 250.000 millones de dólares
    - 59.000 millones de dólares de desviación de los costes estimados
    - 81.000 millones de dólares en proyectos software cancelados
  - Contribución del software a la economía USA en 1996 [Minasi, 2000]
    - Gran superávit en las exportaciones
    - Se exportó software por un valor de 24.000 millones de dólares, se importó software por valor de 4.000 millones de dólares, se obtuvo una balanza positiva de 20.000 millones de dólares
    - Comparativa
      - Agricultura: Exportaciones 26.000 millones; Importaciones 14.000 millones; Balance: 12.000 millones
      - Industria Aeroespacial: Exportaciones 11.000 millones; Importaciones 3.000 millones; Balance: 8.000 millones
      - Química: Exportaciones 26.000 millones; Importaciones 19.000 millones; Balance: 7.000 millones
      - Vehículos: Exportaciones 21.000 millones; Importaciones 43.000 millones; Balance: -22.000 millones
      - Bienes manufacturados: Exportaciones 200.000 millones; Importaciones 265.000 millones; Balance: -65.000 millones
  - Industria del software en USA en el año 2000
    - Ventas: 180 billones de dólares
    - Trabajadores: 697.000 ingenieros de software – 585.000 programadores
  - El gobierno USA estima que las empresas han gastado cerca de 3,3 trillones de dólares en tecnologías de la información en la última década

Realidades del software (i)

- El 55% de los sistemas cuestan más de lo esperado, el 68% superan la fecha de entrega y el 88% tuvieron que ser sustancialmente rediseñados

  Informe de IBM (1994)

- La media era 100 dólares por línea de código, se esperaba pagar 500 dólares por línea, y se terminó pagando entre 700 y 900 dólares por línea, 6.000 millones de dólares de trabajo fueron descartados

  Advanced Automation System (FAAm 1982-1994)

- Cada 6 nuevos sistemas puestos en funcionamiento, 2 son cancelados, la probabilidad de cancelación está alrededor del 50% para sistemas grandes, la media de proyectos que sobrepasa el calendario es del 50%, 3 de cada 4 sistemas son considerados como fallos de operación

  Bureau of Labor Statistics (1997)

Realidades del software (ii)

- El 74% de todos los proyectos de tecnologías de la información fallan porque se pasan de presupuesto, porque no cumplen el plazo de entrega… y el 28% de los proyectos fallan completamente

  The Standish Group (1998)

- Cada año se gastan 75 billones de dólares en proyectos de tecnologías de la información fallidos en USA

  The Standish Group (1998)

- El 52,7% de los proyectos relacionados con las tecnologías de la información cuestan el 189% de su coste inicial estimado

  The Standish Group, as reported by Solutions Integrator (Junio de 1999)

- En grandes compañías (donde la media de coste de un proyecto de desarrollo es de 2.322.000 dólares), sólo el 9% de los proyectos estuvieron en la fecha prevista y dentro del presupuesto

  The Standish Group, as reported by Solutions Integrator (Junio de 1999)

Realidades del software (iii)

- El 31,1% de los proyectos se cancelan antes de completarse

  The Standish Group, as reported by Solutions Integrator (Junio de 1999)
- Cerca de la mitad de los proyectos de desarrollo cuestan un 70% más de lo que inicialmente fue presupuestado. Los gestores citan a la falta de información de los usuarios como principal razón para el fallo de un proyecto

  The Standish Group, as reported by InternetWeek (Septiembre de 1999)

Historias preocupantes y catastróficas (i)

"The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in. We're computer professionals. We cause accidents"

Nathaniel Borenstein, inventor of MIME, in: *Programming as if People Mattered: Friendly Programs, Software Engineering and Other Noble Delusions*, Princeton University Press, Princeton, NJ, 1991

- Un caso ¿divertido?
  - El computador de un hospital comete un error fatal: "De acuerdo con esto, yo estoy muerto"
    - <… El problema sucedió durante una actualización rutinaria de los ficheros del ordenador del [hospital] Saint Mary's en octubre, Jennifer Cammenga, la portavoz del Saint Mary's, declaró al Grand Rapid Press…>. <… "Un dígito fue omitido en el código, indicando que los pacientes habían fallecido, en lugar de indicar que habían sido dados de alta"…>

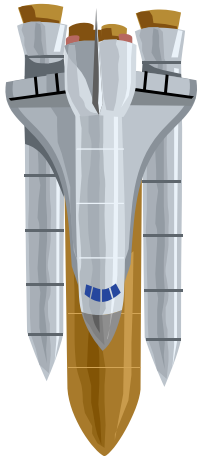
(Noticia de prensa del 8 de enero de 2003)

Historias preocupantes y catastróficas (ii)

- Otros casos no tan divertidos (i)
  - Varias muertes de pacientes de cáncer acaecidas entre 1985-1987 se debieron a una sobredosis de radiación debida a un problema en las tareas concurrentes en el software de la máquina de radioterapia Therac-25 (http://sunnyday.mit.edu/therac-25.html), [Leveson y Turner, 1993], [Leveson, 1995]
  - En mayo de 2001 la Agencia Internacional para la Energía Atómica declaró una emergencia radiológica en Panamá. 28 pacientes sufrieron una sobre exposición, 8 murieron, y ¾ partes de los supervivientes pueden sufrir serias complicaciones que en algunos casos pueden llegar a ser mortales. Se concluyó que uno de los factores que provocaron el accidente se debió a un error en el software que controlaba ciertas entradas de datos (http://www.fda.gov/cdrh/ocd/panamaradexp.html)
  - Servicio de Ambulancias de Londres (1992). Fallo en las pruebas de instalación del sistema y su compatibilidad con los ya existentes provocaron pérdida de llamadas y salidas múltiples debidas a llamadas duplicadas (http://www.cs.ucl.ac.uk/staff/A.Finkelstein/las.html), [Finkelstein y Dowell, 1996]

Historias preocupantes y catastróficas (iii)

- Otros casos no tan divertidos (ii)



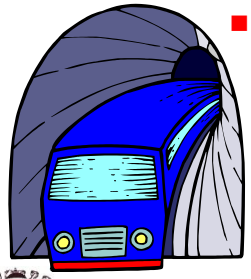
  - El fallo en el lanzamiento del satélite Ariane 5 en 1996 fue causada por una rutina de excepción defectuosa en el código Ada, que se invocaba como resultado de una conversión errónea de un número en coma flotante de 64 bits a un entero de 16 bits (http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html)
  - Dos oficiales de policía en una región escocesa utilizaban una pistola de radar para identificar a motociclistas que infringían los límites de velocidad. Repentinamente la pistola de radar quedó bloqueada apuntando al cielo e indicando una velocidad de 300 millas por hora. Segundos más tarde, un caza Harrier, volando a baja altura, pasó por allí. El buscador de blancos del avión había detectado el radar y lo había tomado por un enemigo. Por fortuna, el Harrier volaba desarmado, ya que el comportamiento normal hubiera sido el disparo de un misil de contraataque automático (http://catless.ncl.ac.uk/Risks/17.67.html#subj1.1), [Pfleeger, 2002]

Historias preocupantes y catastróficas (iv)

- Otros casos no tan divertidos (iii)

- El 15 de enero de 1990 la red de comunicaciones de larga distancia de AT&T estuvo fuera de servicio durante nueve horas a consecuencia de un fallo de software. Millones de llamadas quedaron bloqueadas. Algunos negocios que dependían en gran medida de los servicios telefónicos, como agencias de viajes, quedaron prácticamente colapsados, con la consiguiente pérdida económica

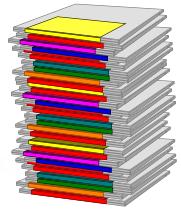
- En febrero de 2003 un fallo en la red de Vodafone deja sin servicio a sus 8,6 millones de usuarios. La avería se produjo en el transcurso de las tareas de mantenimiento del software de la red llevadas a cabo por la operadora, impidiendo las comunicaciones con sus líneas en toda España desde las siete de la mañana



- Un fallo humano, no negligente, motivado por la complejidad del sistema ergonómico de la pantalla del ordenador del centro de control de operaciones del Metro causó, el día 31 de octubre de 2004, el choque de dos trenes del Metro de Barcelona resultando heridas 50 personas

Historias preocupantes y catastróficas (v)

- Otros casos no tan divertidos (iv)
  - La Mars Polar Lander se estrelló en su aterrizaje en Marte en diciembre de 1999 por un fallo de software. Los motores de descenso se apagaron prematuramente porque un fallo en los sensores indicaban que había tomado tierra cuando estaba a unos 40 metros [Clark, 2000]
  - El robot Spirit en Marte tuvo que ser reseteado y actualizado desde la tierra por fallos en su memoria *flash* (enero de 2004) http://www.msnbc.msn.com/id/3855168/



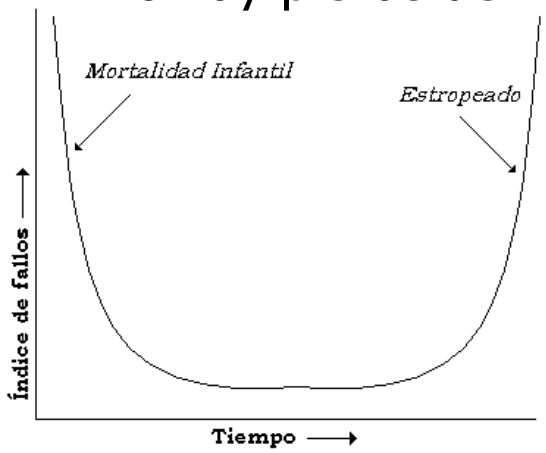
Definición

- Programas, procedimientos, reglas y la posible documentación asociada y datos que pertenezcan a la explotación de un sistema de ordenador [AECC, 1986]
- Una colección organizada de programas de ordenador, procedimientos, documentación asociada y datos referidos a un ordenador que realiza una función específica o un conjunto de funciones [IEEE, 1999a][1], [NIST, 1994], [DOD, 1995]
- Conjunto de programas, procedimientos y documentación asociada a la operación de un sistema informático [Piattini et al., 2004]

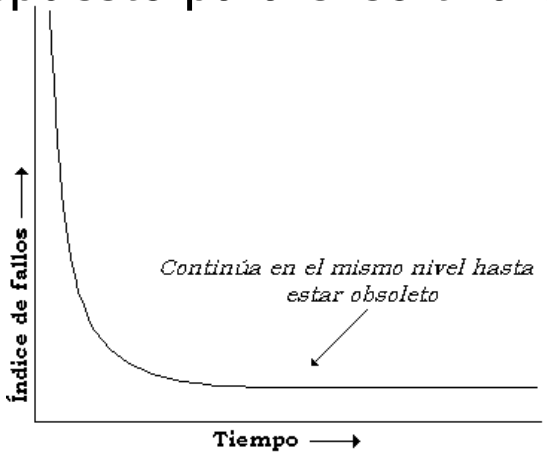
[1]IEEE Std 610.12-1990 *Standard Glossary of Software Engineering Terminology*

Características del producto software

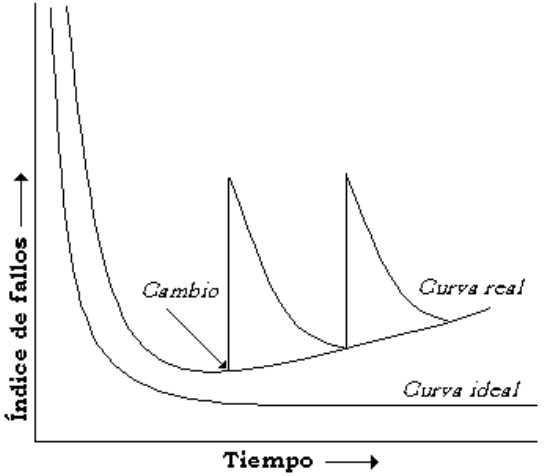
- El software se desarrolla, no se fabrica en el sentido clásico
  - Los costes del software se encuentran en la ingeniería
- El software no se estropea, se deteriora
  - Cambios en las fases de mantenimiento
  - No hay piezas de repuesto para el software



Curva de fallos del hardware



Curva de fallos del software

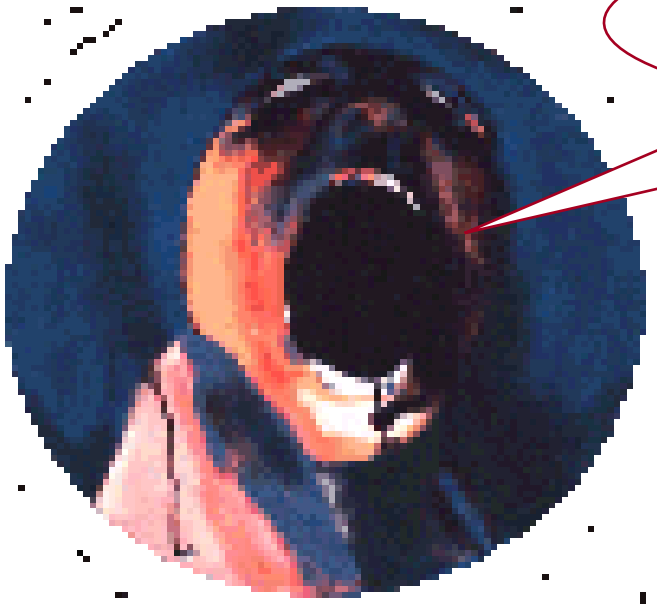


Curva real de fallos del software

- A pesar de las últimas tendencias, el software se sigue construyendo a medida

[Pressman, 2006]

La crisis del software (i)



¡¡¡¡ La crisis del software !!!!

- **Problemas del software**
  - Calidad cuestionable
    - Mal funcionamiento
    - Insatisfacción de los clientes
  - Cómo desarrollar software
    - Imprecisión en la planificación y la estimación
    - Baja productividad
  - Cómo mantener el volumen creciente de software existente
  - Cómo afrontar la incesante demanda de software
  - Barrera del mantenimiento

La crisis del software (ii)

- Dificultad inherente
- Gran complejidad
  - Número de estados posibles es muy elevado
  - Conexiones entre entidades
  - Complejidad arbitraria que surge de instituciones humanas
- Sujeto a continuos cambios
- Especificación de requisitos
- Comunicación del equipo

> "La construcción de software siempre será una tarea difícil. No hay bala de plata"
>
> Frederick P. Brooks, Jr. (1987)

Algunas causas a los problemas del software

- Responsables no cualificados
- Falta de comunicación entre las partes
- Desconocimiento de las nuevas tendencias
- Resistencia al cambio
- Falta de reconocimiento de la figura del informático
- Una amplia mitología y falta de "cultura informática" de la sociedad
  - Mitos de gestión
    - Resistencia al cambio en la gestión de proyectos
    - Concepto de la horda mongoliana
    - …
  - Mitos del cliente
    - Ideas genéricas al principio, detalles al final
    - Requisitos en continua evolución
    - …
  - Mitos del desarrollador
    - El trabajo acaba cuando se ha escrito el programa y funciona
    - Sólo se entrega un programa funcionando
    - Lo que uno crea sólo debe entenderlo él
    - …

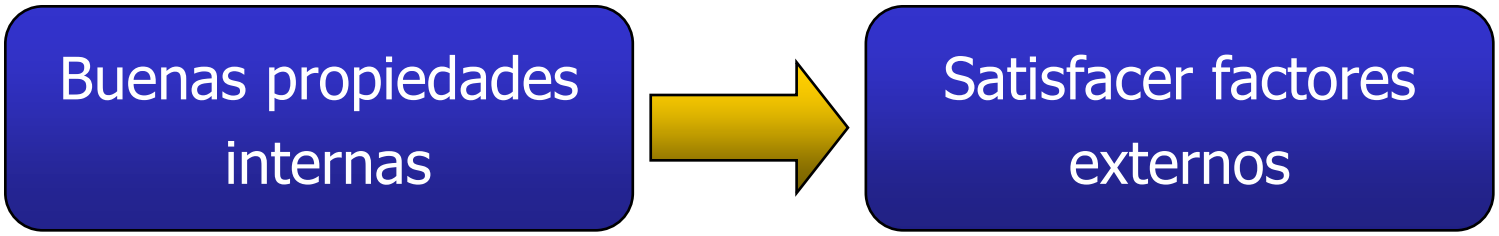
Calidad del software (i)

- Los ingenieros del software deben encontrar los métodos para asegurar que sus productos sean de utilidad y tengan una calidad aceptable
  - Así, la Ingeniería del Software siempre debe incluir una estrategia para producir **software de calidad**
- La calidad se percibe desde diferentes puntos de vista [Garvin, 1984]
  - Vista trascendental o ideal
  - Vista del usuario — **Vista externa**
  - Vista de la construcción o de proceso — **Vista externa**
  - Vista del producto — **Vista interna**
  - Vista basada en el valor

Calidad del software (ii)

- Factores externos
  - Pueden ser detectados por los usuarios
  - Es de suma importancia
- Factores internos
  - Sólo los perciben los ingenieros del software
  - Es el medio de conseguir la calidad externa

OBJETIVO



Atributos de un buen producto software (i)

- Factores externos
  - Facilidad de mantenimiento
    - Ha de poder evolucionar para adaptarse a las necesidades de cambio de los clientes
  - Confiabilidad
    - No debe causar daños físicos o económicos en el caso de fallo del sistema
    - Fiabilidad, seguridad y protección
  - Eficacia
    - Hacer efectivo el propósito del software
  - Usabilidad
    - Fácil de utilizar
    - Debe tener una interfaz de usuario apropiada y una documentación adecuada
  - Reusabilidad
    - Capacidad de que un software pueda utilizarse en un contexto diferente al de su creación
  - Portabilidad
    - Facilidad de transferir productos software a diferentes plataformas
  - ...

Atributos de un buen producto software (ii)

- Factores internos
  - Facilidad de traza
  - Modularidad
  - Tolerancia a fallos
  - Eficiencia de ejecución
  - Eficiencia de almacenamiento
  - Autodescripción
  - Legibilidad
  - Facilidad de expansión
  - Independencia del sistema
  - Independencia del hardware
  - Estandarización de datos
  - Estandarización de comunicaciones
  - ...

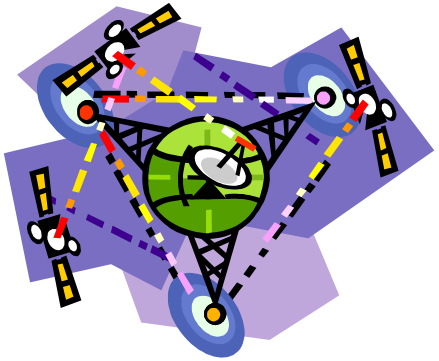
Tipos de productos software (i)

- Un producto software es un sistema software que se acompaña de la documentación necesaria para su instalación y uso

- Tipos de mercados
  - Productos genéricos
    - Sistemas autónomos producidos por una organización para su venta en el mercado abierto a cualquier cliente que pueda adquirirlo
    - El desarrollador controla la especificación
  - Productos personalizados
    - Sistemas encargados por un cliente particular
      - Desarrollos a medida
    - Las especificaciones las determina el cliente

Tipos de productos software (ii)

- Áreas de aplicación [Pressman, 2006] (i)
  - Software de sistemas
    - Software para dar servicio a otros programas: compiladores, editores...
      - Fuerte interacción con el hardware
      - Operación concurrente
      - Recursos compartidos
      - Gestión de procesos complicada
      - Estructuras de datos complejas



  - Software de tiempo real
    - Coordina/analiza/controla sucesos en el mundo real en el momento en el que suceden: control de vuelo, plantas químicas, telefonía...
      - Tiempo de respuesta crítico: magnitud de milisegundos
      - Interaccionan directamente con dispositivos físicos y sensores
      - Requisitos de rendimiento críticos
      - Programación de bajo nivel
      - Concurrencia

Tipos de productos software (iii)

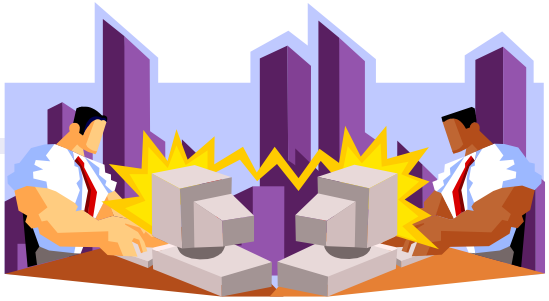
- Áreas de aplicación [Pressman, 2006] (ii)
  - Software de ingeniería y científico

    - Algoritmos de tratamiento numérico: simulación, estadística, CAD...
    - Diseño de algoritmos y estructuras de datos
    - Cálculo intensivo
    - Paralelización
  - Software empotrado
    - Reside en memoria de sólo lectura y se utiliza para controlar productos y sistemas de los mercados industriales y de consumo
    - Características similares al de tiempo real

  - Software de Inteligencia Artificial

    - Algoritmos no numéricos para resolver problemas complejos: sistemas expertos, reconocimiento de patrones, demostradores de teoremas...

Tipos de productos software (iv)



- Áreas de aplicación [Pressman, 2006] (iii)
  - Software de gestión
    - Proceso de información comercial: nóminas, clientes, inventarios...
      - Gran volumen de datos
      - Complejidad de la información
      - Integración
    - Sistemas transaccionales (TPS)
      - Soportan las operaciones diarias de un negocio: pedidos, compras...
      - Los requisitos, los datos y el procesamiento se conoce y está bien estructurado
    - Análisis de datos
      - Aplicaciones de consulta (*query*)
      - El usuario especifica qué desea no cómo obtenerlo
        - Lenguajes declarativos
      - *Datawarehouse*
        - Almacenamiento de versiones históricas de entradas a la base de datos, registros de transacciones y datos históricos
    - Soporte a la toma de decisiones (DSS – *Decision Support System*)
      - Herramienta de usuario final
      - Resolución de problemas no estructurados
      - Análisis "*what-if*", estadístico, tendencias...

Tipos de productos software (v)

- Áreas de aplicación [Pressman, 2006] (iv)
  - Software de computadoras personales
    - Herramientas de escritorio, software para ocio…
  - Aplicaciones Web
    - Software accedido a través de un navegador Web
    - Los sistemas Web tienen una naturaleza y unos requisitos que difieren del software tradicional
    - Los sistemas Web
      - Están orientados a documentos que contienen páginas Web estáticas o dinámicas
      - Se centran en el *look & feel* y enfatizan la creatividad visual y la presentación en la interfaz
      - Son conducidos por el contenido, incluyendo el desarrollo del contenido
      - Necesitan ofrecer servicios a usuarios con diversidad de características y capacidades
      - Ejemplifican los vínculos entre el arte y la ciencia que generalmente aparecen en el desarrollo del software
      - Requieren acortar el tiempo de desarrollo, dificultando aplicar el mismo nivel de formalidad en la planificación y prueba que se aplica en el software tradicional
      - Presentan un formato de distribución y explotación diferente al software tradicional
      - Los desarrolladores de los sistemas Web
        - Difieren en gran medida en su formación, características, conocimiento y comprensión del sistema
        - Diferencias en su percepción de la Web y de la calidad del sistema Web

Tipos de productos software (vi)

- Categorías de las aplicaciones Web

Categoría	Ejemplos
De información	Periódicos en línea, catálogos de productos, libros electrónicos en línea…
Interactivas	Formularios de registro, presentación de información personalizada, juegos en línea…
Transaccionales	Compra electrónica, banca electrónica…
Workflow	Sistemas de planificación en línea, gestión de inventario, monitorización de estado…
Entornos de trabajo cooperativo	Sistemas de autor distribuido, herramientas de diseño colaborativas…
Comunidades en línea, marketplaces	Grupos de *chat*, sitios que recomiendan productos o servicios, *marketplaces* en línea, subastas en línea…
Portales Web	Centros comerciales, intermediarios en línea…

[Ginige y Murugesan, 2001]

Tipos de productos software (vii)

- Sistemas de software intensivo [Moreno-Navarro, 2005]
  - Combinan tecnologías emergentes de sistemas empotrados (automoción, aviónica, ropas…) pero a la vez están inmersos en sistemas globales de cómputo (Internet, *grids*, sistemas orientados a servicios…)
  - Estos sistemas son sistemas programables que
    - Son dinámicos y evolucionan
    - Su comportamiento es adaptativo y anticipatorio
    - Procesan conocimiento y no sólo datos

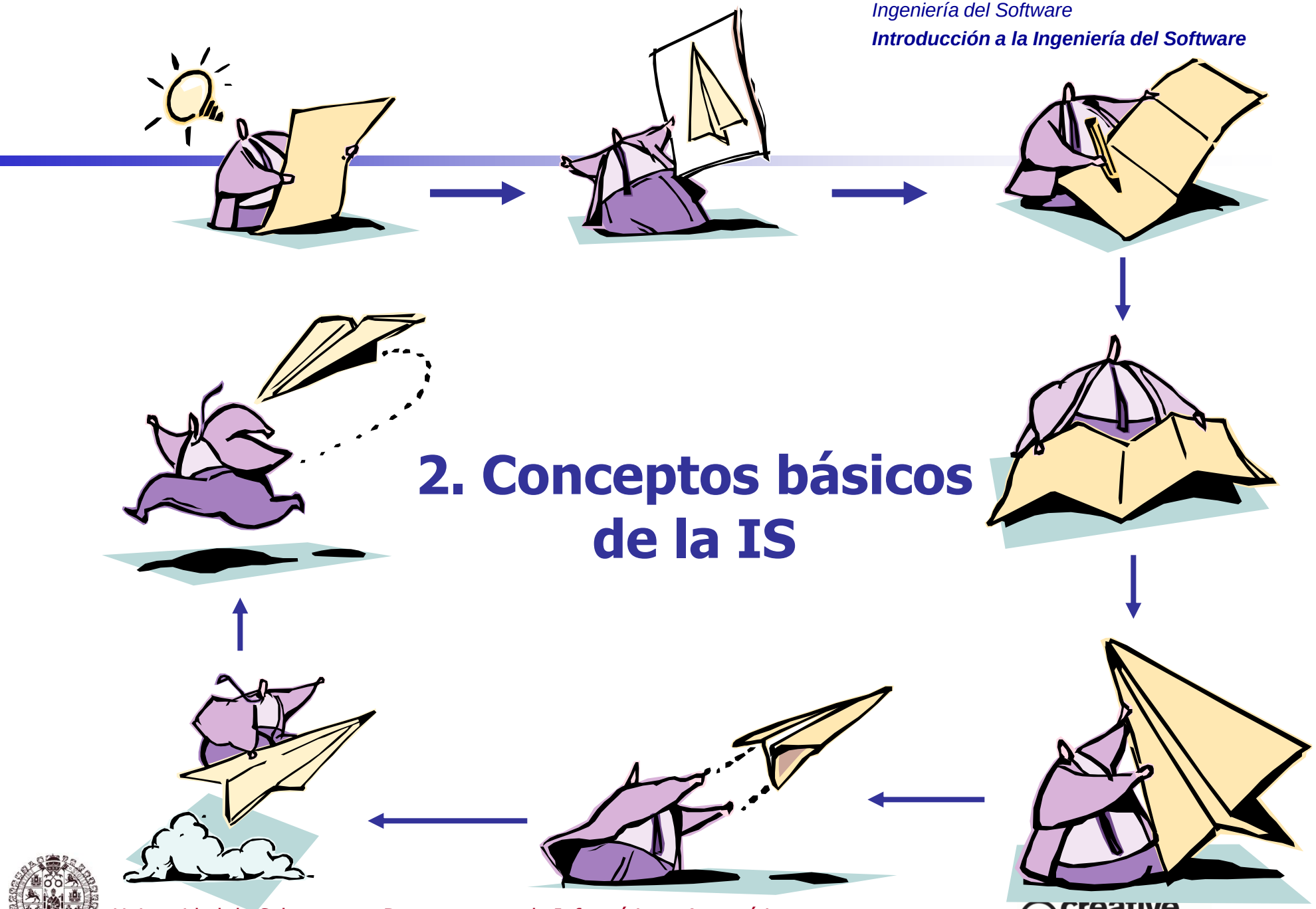
"*Computing is becoming a utility and software service […] applications will no longer be a big chunk of software that runs on a computer but a combination of web services; and the platform for which developers write their programs will no longer be the operative system, but application servers*" (The Economist, June 2003)

Producto software. Conclusiones

- Los sistemas software son productos complejos
  - Gran funcionalidad
  - Objetivos diferentes y en ocasiones conflictivos
  - En su concepción, desarrollo y mantenimiento interviene un gran número de personas con diferentes perfiles
  - Elevado tamaño
    - Windows 98 – 18 millones de líneas de código
    - Windows 2000 (2001) – 35 millones de líneas de código
    - Windows XP (2002) – 40 millones de líneas de código
    - Linux (Debian) – 55 millones de líneas de código – 14.000 personas/año – 1900 millones de dólares
    - Rotor (2002) – 3,6 millones de líneas de código
  - Sujeto a cambios continuos
    - Requisitos, tecnología...

Microsoft .net

Conclusión final: *La producción de software ha de estar regida por los principios de la **INGENIERÍA***



2. Conceptos básicos de la IS

Introducción

- Objetivos de la Ingeniería del Software
  - Desarrollo de software de Calidad
  - Aumento de la productividad
  - Disminución del tiempo
  - Desarrollo de software económico
- Diferentes puntos de vista sobre el mismo tema
  - Diseño, construcción y mantenimiento de grandes sistemas software
  - Construcción multipersona de software multiversión
  - Conjunto de técnicas que se enfrentan al software como un producto de ingeniería que requiere: planificación, análisis, diseño, implementación, pruebas y mantenimiento
  - Aplicación disciplinada de los principios y métodos de la ingeniería, la ciencia y las matemáticas para la producción económica del software de calidad
  - Conjunto de teorías, métodos y herramientas para el desarrollo profesional del software

Definiciones (i)

- Ingeniería del software es el establecimiento y uso de principios sólidos de ingeniería, orientados a obtener software económico que sea fiable y trabaje de manera eficiente en máquinas reales [Bauer, 1972]
- Tratamiento sistemático de todas las fases del ciclo de vida del software. Se refiere a la aplicación de metodologías para el desarrollo del sistema software [AECC, 1986]
- La construcción de software multiversión por un equipo de varias personas [Parnas y Weiss, 1987]
- La aplicación disciplinada de principios, métodos y herramientas de ingeniería, ciencia y matemáticas para la producción económica de software de calidad [Humphrey, 1989]
- Disciplina tecnológica y de gestión concerniente a la invención, producción sistemática y mantenimiento de productos software de alta calidad, desarrollados a tiempo y al mínimo coste [Frakes et al., 1991]

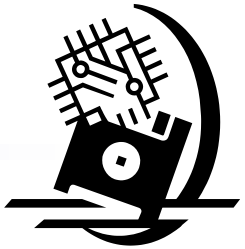
Definiciones (ii)

- Aplicación de herramientas, métodos y disciplinas para producir y mantener una solución automatizada de un problema real [Blum, 1992]
- La aplicación de principios científicos para la transformación ordenada de un problema en una solución software funcional, así como en el consiguiente mantenimiento del software hasta el final de su vida útil [Davis, 1993]
- Es la aplicación de herramientas, métodos y disciplinas de forma eficiente en cuanto al coste, para producir y mantener una solución a un problema de procesamiento real automatizado parcial o totalmente por el software [Horan, 1995]
- La aplicación de métodos y conocimiento científico para crear soluciones prácticas y rentables para el diseño, construcción, operación y mantenimiento del software y los productos asociados, al servicio de las personas [Shaw y Garlan, 1996]
- (1) La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, la operación y el mantenimiento del software; es decir, la aplicación de la Ingeniería al Software. (2) El estudio de las aproximaciones en (1) IEEE Std 610.12-1990 [IEEE, 1999a]

Método de Ingeniería

- Formulación del problema
- Análisis del problema
- Búsqueda de soluciones
- Elección de la solución más adecuada
- Especificación de la solución

Método de ingeniería en Ingeniería del Software



- Recolección y análisis de requisitos
  - Actividad: Formulación del problema con el cliente
  - Resultado: **Modelo del dominio del problema**

  ⇨ Formulación y análisis del problema
- Diseño del sistema
  - Actividad: Análisis del problema
  - Actividad: Descomposición en partes
  - Actividad: Selección de estrategias para diseñar el sistema
  - Actividad: Selección del diseño detallado para cada una de las partes
  - Resultado: **Modelo del dominio de la solución**

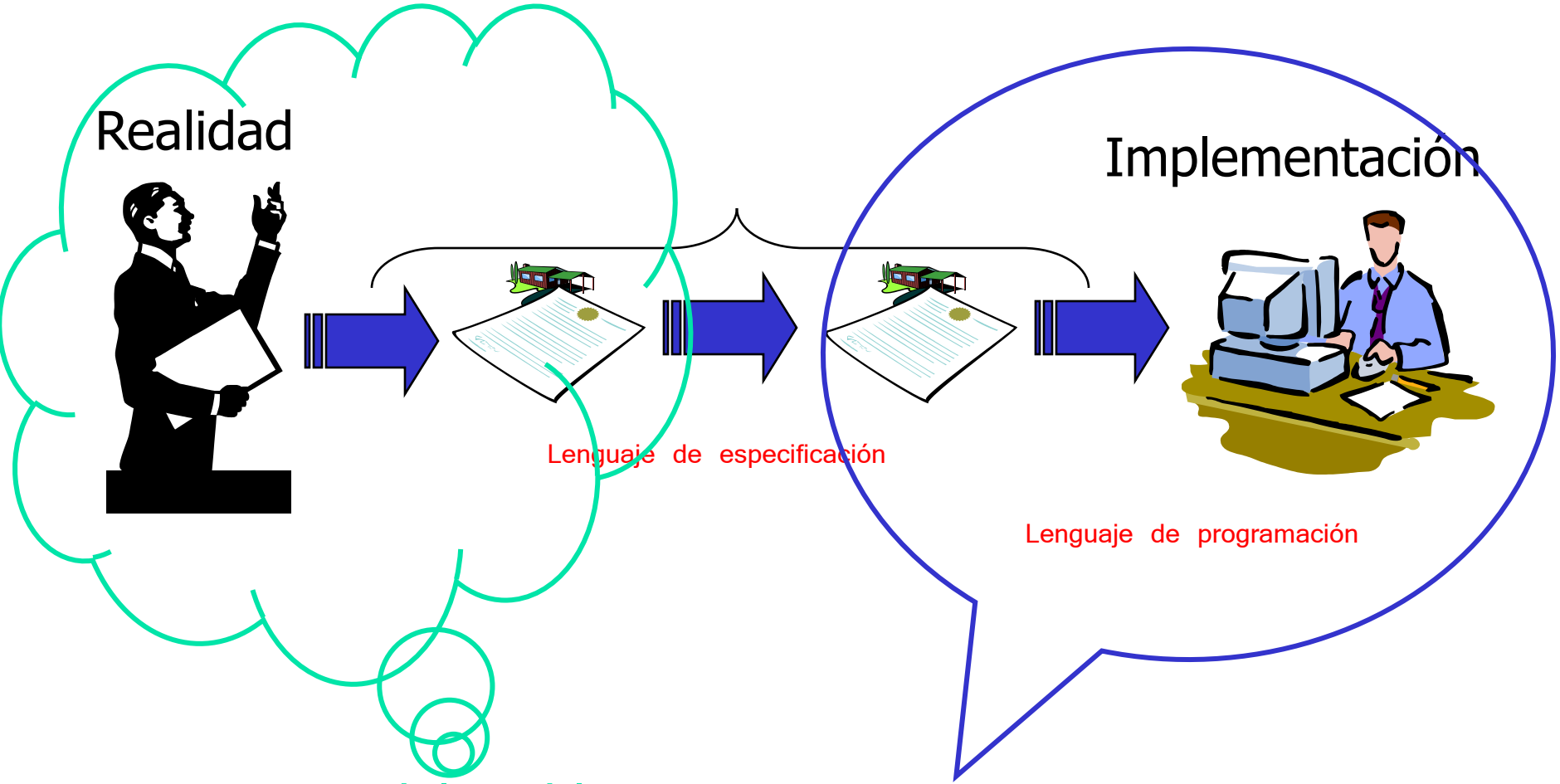
  ⇨ Búsqueda de soluciones; elección de la solución más adecuada
- Implementación
  - Actividad: Trasladar el modelo del dominio de la solución en representaciones ejecutables

  ⇨ Especificación de la solución

Modelo del problema vs. modelo de la solución (i)

- **Modelo del Dominio del Problema**
  - Descripción de aquellos aspectos del sistema del mundo real que son relevantes para el problema en consideración
  - Comprensión del entorno en el que ha de funcionar el sistema
- **Modelo del Dominio de la Solución**
  - Comprensión de los sistemas que se han de construir
  - Evaluar diferentes soluciones alternativas
  - Participación de un equipo de desarrollo en la construcción del sistema

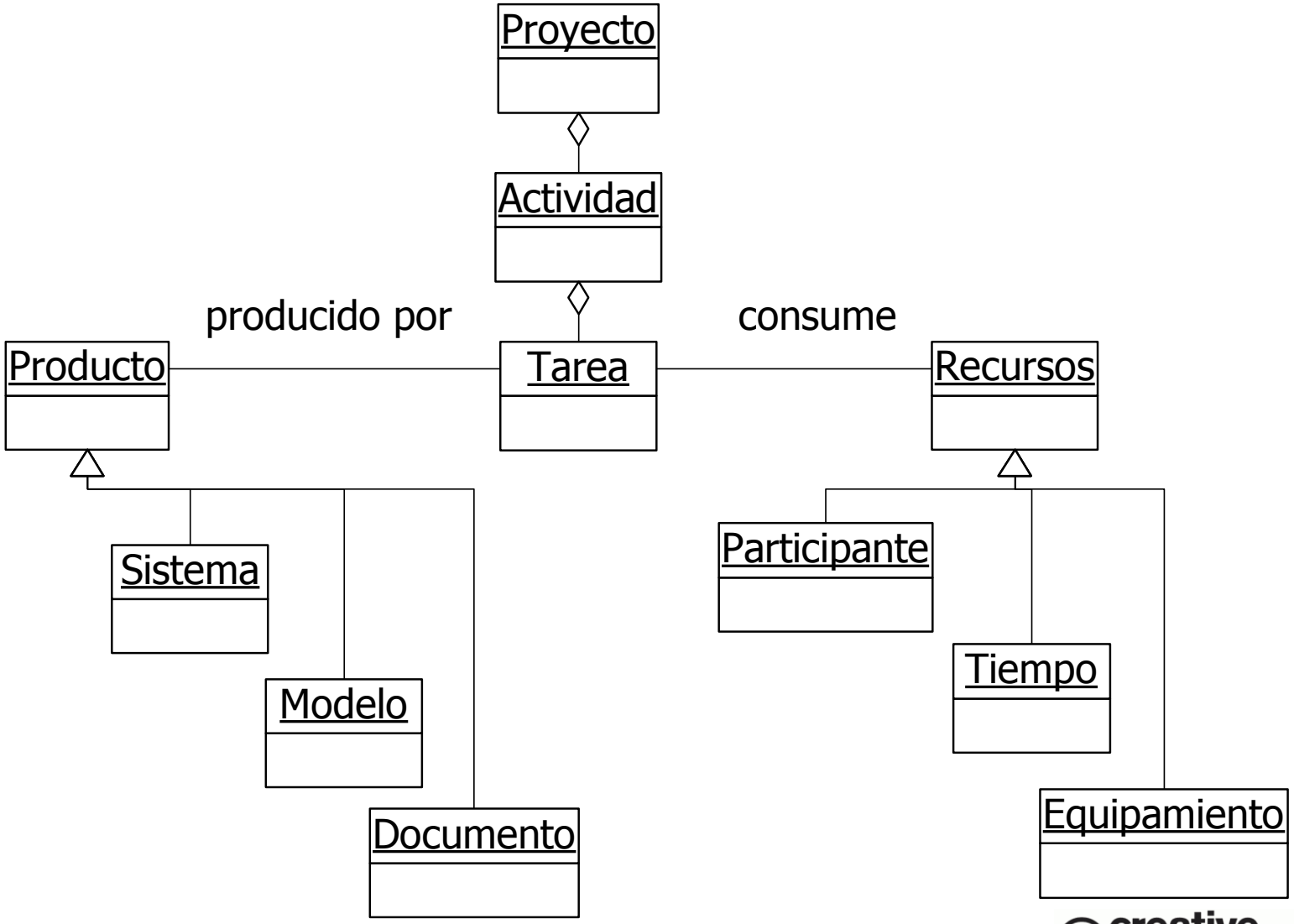
Modelo del problema vs. modelo de la solución (ii)



Marco conceptual de la Ingeniería del Software

- Un sistema software es siempre una parte de un sistema mayor que lo engloba como componente
- La Ingeniería del Software será solamente una parte del diseño del sistema
  - Los requisitos del software han de ajustarse a los requisitos del resto de los elementos que constituyen este sistema
  - El ingeniero del software ha de estar implicado en el desarrollo de los requisitos del sistema completo, comprendiendo el dominio de actividad en su totalidad
- La **Teoría General de Sistemas** es el antecedente conceptual en el que se apoya la teoría sobre los Sistemas de Información a los que la Ingeniería del Software intenta aportar soluciones

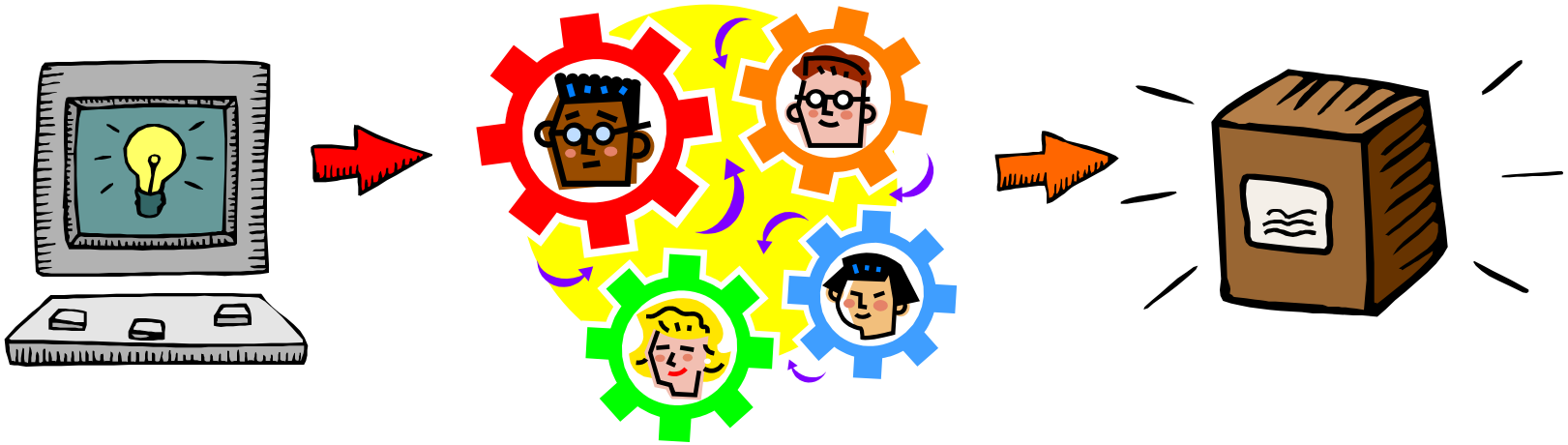
Conceptos (i)



Conceptos (ii)

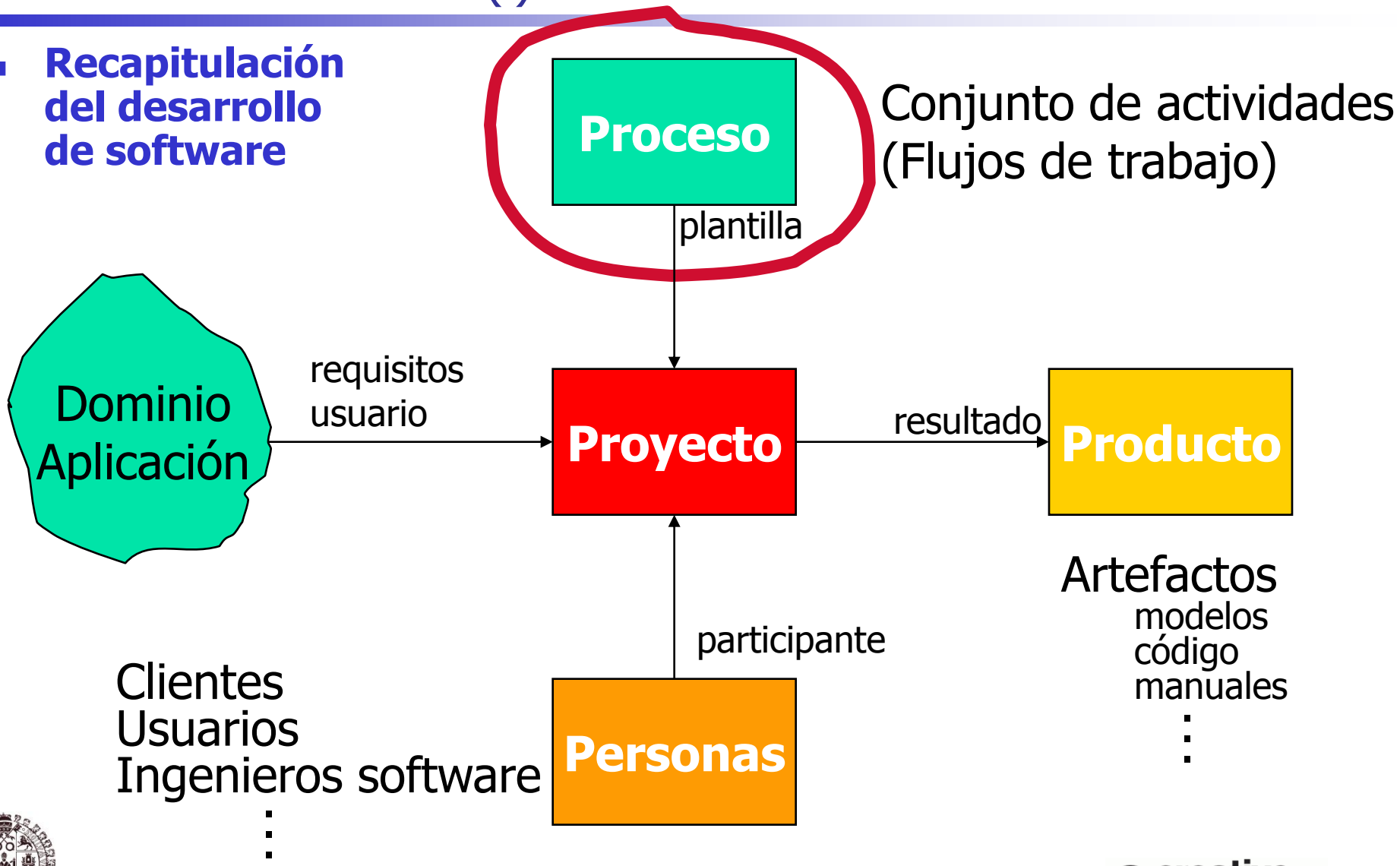
- Proceso
  - Define el marco de trabajo y permite un desarrollo racional y oportuno de la Ingeniería del Software
- Método
  - Indica cómo construir técnicamente el software. Se incluyen técnicas de modelado y otras técnicas descriptivas
- Herramientas
  - Proporcionan el soporte automático o semiautomático para el proceso y para los métodos
- Notación
  - Conjunto de reglas gráficas o textuales para la representación de un modelo
- Metodología
  - Colección de métodos para resolver un tipo de problemas
  - Descompone el proceso de desarrollo en actividades y proporciona los métodos adecuados para llevar a cabo dichas actividades

3. Proceso software



Introducción (i)

- **Recapitulación del desarrollo de software**



Introducción (ii)

- La Ingeniería del Software es una tecnología multicapa

"*El fundamento de la Ingeniería del Software es la capa proceso. El proceso de la Ingeniería del Software es la unión que mantiene juntas las capas de tecnología y que permite un desarrollo racional y oportuno de la Ingeniería del Software*"

[Pressman, 2006]

"*Un proceso bien definido es necesario para desarrollar sistemas software de manera repetible y predecible*"

"*Permite un negocio sostenible y que puede mejorar en cada nuevo proyecto, incrementando la eficiencia y productividad de la organización*"

G. Booch

Definición de proceso software

- Conjunto de actividades necesarias para transformar las ideas iniciales del usuario, que desea automatizar un determinado trabajo, en software
- Conjunto de actividades y resultados asociados necesarios para producir un producto software. Estas actividades son: especificación del software, desarrollo del software, validación del software y evolución del software [Sommerville, 2005]
- Conjunto ordenado de actividades; una serie de pasos que involucran tareas, restricciones y recursos que producen una determinada salida esperada [Pfleeger, 2002]
- Marco de trabajo de las tareas que se requieren para construir software de alta calidad [Pressman, 2006]

Características de un proceso (i)

- Cualquier proceso tiene las siguientes características [Pfleeger, 2002]
  - El proceso establece todas las actividades principales
  - El proceso utiliza recursos, está sujeto a una serie de restricciones y genera productos intermedios y finales
  - El proceso puede estar compuesto de subprocesos que se encadenan de alguna manera. Puede definirse como una jerarquía de procesos organizada de modo que cada subproceso tenga su propio modelo de proceso
  - Cada actividad del proceso tiene criterios de entrada y de salida, de modo que se conoce cuándo comienza y cuándo termina una actividad
  - Las actividades se organizan en secuencia de modo que resulta claro cuando una actividad se realiza en orden relativo a otras actividades
  - Todo proceso tiene un conjunto de principios orientadores que explican las metas de cada actividad
  - Las restricciones o controles pueden aplicarse a una actividad, recurso o producto

Características de un proceso (ii)

- Otras características que van a definir un proceso son
  - Comprensión
    - Está definido y es comprensible
  - Visibilidad
    - Se visualizan los progresos externamente
  - Soporte
    - Está soportado por herramientas CASE
  - Aceptación
    - Es aceptable para todos los actores implicados
  - Confianza
    - Los errores del proceso se detectan antes de que se produzcan errores en el producto
  - Robustez
    - Se puede continuar a pesar de problemas inesperados
  - Capacidad de mantenimiento
    - Puede ajustarse a las necesidades de cambio de la organización
  - Rapidez
    - Con qué "velocidad" se producen los sistemas
  - Adaptación
    - Capacidad que tiene un usuario del mismo de adaptarlo a sus necesidades

Importancia del proceso en el desarrollo del software (i)

- Un proceso software debe especificar
  - La **secuencia de actividades** a realizar por el equipo de desarrollo
    - Flujo de actividades
  - Los **productos** que deben crearse
    - Resultados del trabajo (modelos, documentos, datos informes...)
    - Qué y cuándo
  - La **asignación de tareas** a cada miembro del equipo y al equipo como un todo
  - Los **criterios** para controlar el proceso
    - Se establece el control de gestión de los proyectos software
    - Establecimiento de hitos
  - Las posibles **heurísticas**

Importancia del proceso en el desarrollo del software (ii)

- Facilita la gestión del proyecto
- Establece una división del trabajo
- Facilita la comunicación de los miembros del equipo
- Permite la reasignación y la reutilización de personal especializado
  - Transferencia entre proyectos
- Mejora la productividad y el desarrollo
  - El desarrollo es reproducible
- Establece el contexto en el que se aplican los métodos técnicos
- Gestiona el cambio adecuadamente
- Asegura la calidad

4. Modelos de proceso software

Ciclo de vida del software (i)

- Cuando un proceso implica la construcción de algún producto, suele referirse al proceso como un **ciclo de vida**
  - El proceso de desarrollo de software suele denominarse **ciclo de vida del software**
- La evolución del software representa el ciclo de actividades involucradas en el desarrollo, uso y mantenimiento de sistemas software [Scacchi, 1987]
- Los **proyectos software** se desarrollan en una serie de **fases**
  - Van desde la **concepción del software** y su desarrollo inicial hasta su **puesta en funcionamiento** y **posterior retirada** por otra nueva generación de software
  - Estas fases pueden ser
    - Temporales
      - Forman una secuencia en el tiempo
    - Lógicas
      - Cuando representan pasos o etapas que no constituyen una secuencia temporal

Ciclo de vida del software (ii)

- Se puede definir ciclo de vida del software como (i)
  - Las distintas fases por las que pasa el software desde que nace una necesidad de mecanizar un proceso hasta que deja de utilizarse el software que sirvió para ese objetivo, pasando por las fases de desarrollo y explotación [Frakes et al., 1991]
  - El período de tiempo que comienza cuando se concibe un producto software y finaliza cuando el producto pierde su utilidad. El ciclo de vida del software incluye las siguientes fases: fase de requisitos, fase de diseño, fase de realización, fase de pruebas, fase de instalación y aceptación, fase de operación y mantenimiento y, algunas veces, fase de retirada [AECC, 1986]
  - Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de requisitos hasta la finalización de su uso [ISO/IEC, 1995]
  - Una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software IEEE Std 1074-1997 *Standard for Developing Software Life Cycle Processes* [IEEE, 1999b]

Ciclo de vida del software (iii)

- Se puede definir ciclo de vida del software como (ii)
  - El ciclo de vida del software consiste de las siguientes fases: análisis de requisitos, diseño, implementación, prueba y mantenimiento. El proceso de desarrollo tiende a una iteración de estas fases más que a un proceso lineal [CERN, 1996]
  - El ciclo de vida del software usa el modelo de que un elemento software tiene vida. Un elemento software tiene una fase de concepción (una idea en una mente de un usuario potencial), después de una fase de gestación (la fase de desarrollo del software) hacia una fase de madurez (la revisión y corrección de errores, o fase de mantenimiento), y finalmente la fase de retirada [Leaney, 2004]
- Se puede definir ciclo de desarrollo del software como
  - El período de tiempo que comienza con la decisión de desarrollar un producto software y finaliza cuando se ha entregado éste. Este ciclo incluye, en general, una fase de requisitos, una fase de diseño, una fase de implantación, una fase de pruebas, y a veces, una fase de instalación y aceptación [AECC, 1986]

¿Qué es un modelo de proceso software?

- Un modelo de proceso software es una representación abstracta de un proceso software [Sommerville, 2005]
- Hay varios modelos de procesos definidos en la bibliografía de Ingeniería del Software
- Cada modelo de proceso representa un proceso desde una perspectiva particular, por lo que sólo ofrece una información parcial sobre dicho proceso
- Los modelos de proceso genéricos, también llamados paradigmas de proceso
  - Presentan un proceso desde una perspectiva arquitectónica, es decir, ofrecen un marco de definición para el proceso, pero no detallan las actividades específicas
  - No son descripciones definitivas de los procesos software, más bien son abstracciones útiles que se utilizan para explicar diferentes aproximaciones al desarrollo del software

Razones para modelar un proceso

- Cuando se pone por escrito una descripción de un proceso, se da forma a una comprensión común de las actividades, recursos y restricciones relacionados con el desarrollo del software
- Ayuda al equipo de desarrollo a encontrar las inconsistencias, las redundancias y las omisiones en el proceso y en las partes que lo constituyen
- El modelo debe reflejar las metas del desarrollo. A medida que se construye el modelo el equipo de desarrollo evalúa las actividades candidatas por su adecuación para alcanzar dichas metas
- Ayuda al equipo de desarrollo a comprender dónde debe adaptarse el proceso
- Los modelos de proceso de desarrollo de software incluyen los requisitos del sistema como entrada y un producto entregado como salida

[Pfleeger, 2002]

Modelo general de proceso en Ingeniería

- **Especificación**
  - Formulación de los requisitos y restricciones del sistema
- **Diseño**
  - Elaboración de un documento con el modelo del sistema
- **Fabricación**
  - Construcción del sistema
- **Prueba**
  - Comprobación de que el sistema cumple las especificaciones requeridas
- **Instalación**
  - Entrega del sistema al cliente y garantía de que es operativo
- **Mantenimiento**
  - Reparación de los fallos que aparecen en el sistema

Modelo general de proceso en Ingeniería de Software

- En el proceso de construcción de sistemas informáticos se pueden distinguir tres fases genéricas
  - La definición
  - El desarrollo
  - El mantenimiento

Fase de definición

- Se identifican los requisitos claves del sistema y del software
- Se desarrolla
  - Un **Análisis de Sistemas**
    - Se define el papel de cada elemento en el sistema automatizado de información, incluyendo el que jugará el software
  - Un **Análisis de Requisitos**
    - Se especifican todos los requisitos de usuario que el sistema tiene que satisfacer
    - Esta fase está orientada al **QUÉ**
      - Qué información ha de ser procesada, qué función y rendimiento se desea, qué interfaces han de establecerse, qué ligaduras de diseño existen y qué criterios de validación se necesitan para definir un sistema correcto
- Existe un paso complementario: la planificación del proyecto software
  - Se asignan los recursos
  - Se estiman los costes
  - Se planifican las tareas y el trabajo

Fase de desarrollo

- Fase orientada al **CÓMO**
- El primer paso de esta fase corresponde al **Diseño del Software**
  - Se trasladan los requisitos del software a un conjunto de representaciones que describen la estructura de datos, arquitectura del software y procedimientos algorítmicos que permiten la construcción física de dicho software
- Los otros dos pasos de la fase de desarrollo corresponden a la **Codificación** y a la **Prueba del Software**

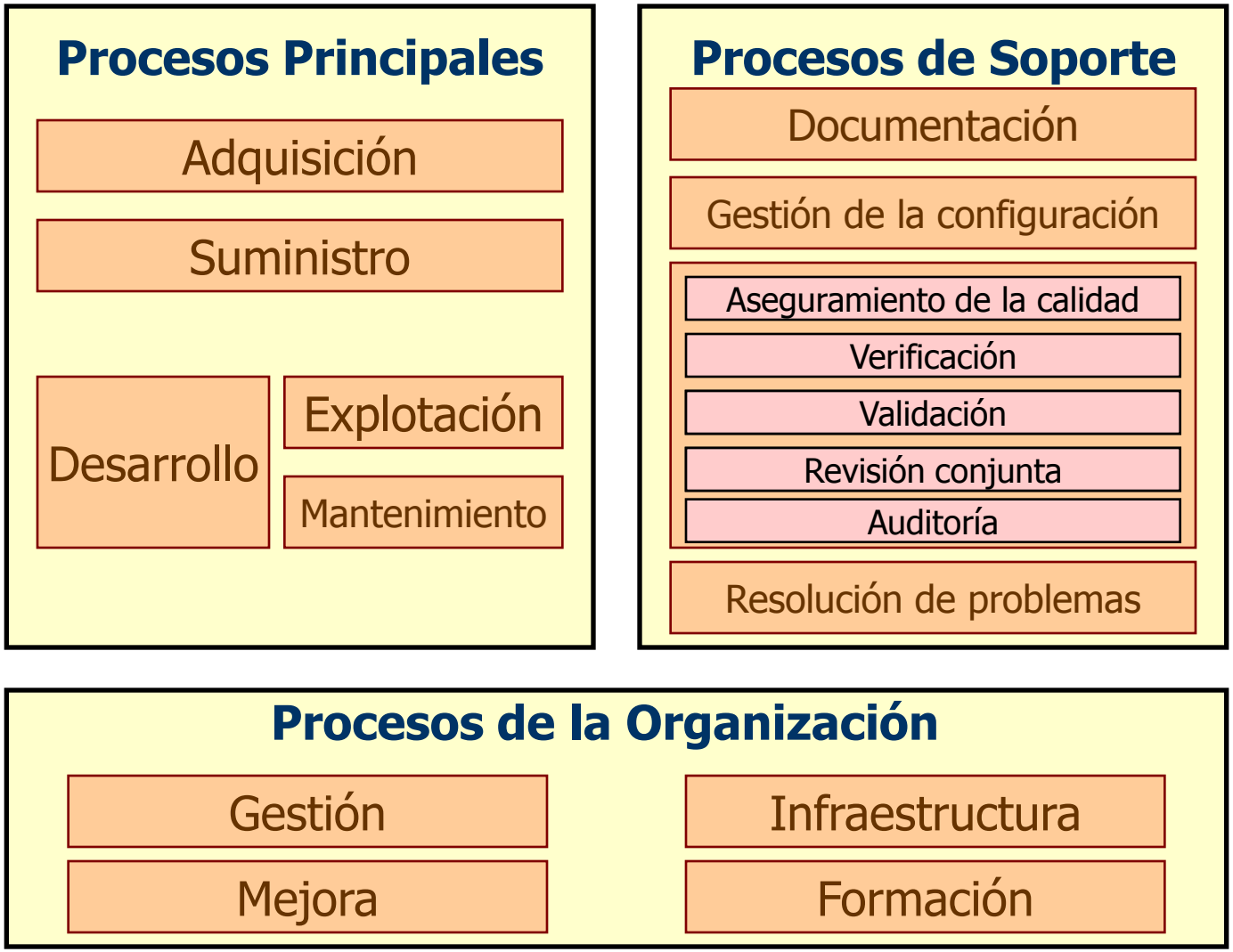
Fase de mantenimiento

- Mantenimiento correctivo
  - Corrección de errores
- Mantenimiento adaptativo
  - Adaptaciones requeridas por la evolución del entorno del software
- Mantenimiento perfectivo
  - Las modificaciones debidas a los cambios de requisitos del usuario para mejorar el sistema
- Mantenimiento preventivo
  - Mejora de las características internas del producto para hacer más mantenible

Norma ISO 12207-1 (i)

- En la norma ISO 12207-1 [ISO/IEC, 1995], las actividades que se pueden realizar en el ciclo de vida del software se agrupan en **cinco procesos principales**, **ocho procesos de soporte** y **cuatro procesos generales**, así como un proceso que permite adaptar el ciclo de vida a cada caso concreto
- Esta norma no fomenta o especifica ningún modelo concreto de ciclo de vida, gestión del software o método de ingeniería, ni prescribe cómo realizar ninguna de las actividades

Norma ISO 12207-1 (ii)



Norma ISO 12207-1 (iii)

- **Procesos principales**: Son los que resultan útiles a las personas que inician o realizan el desarrollo, la explotación o el mantenimiento del software durante su ciclo de vida
  - **Proceso de adquisición**: Actividades y tareas que se realizan para comprar un producto software
  - **Proceso de suministro**: Actividades y tareas que el suministrador realiza
  - **Proceso de desarrollo**: Contiene las actividades de análisis de requisitos, diseño, codificación, integración, pruebas e instalación y aceptación
  - **Proceso de explotación**: Incluye la explotación del software y el soporte operativo a los usuarios
  - **Proceso de mantenimiento**: Aparece cuando el software necesita modificaciones, ya sea en el código o en la documentación asociada, debido a un error, una deficiencia, un problema o la necesidad de mejora o adaptación

Norma ISO 12207-1 (iv)

- **Procesos de soporte**: Sirven de apoyo al resto y se aplican en cualquier punto del ciclo de vida
  - **Proceso de documentación**: Registra la información producida por un proceso o actividad en el ciclo de vida
  - **Proceso de gestión de la configuración**: Aplica ciertos procedimientos y técnicos durante todo el ciclo de vida del sistema
  - **Proceso de aseguramiento de la calidad**: Aporta la confianza de que los procesos y los productos software del ciclo de vida cumplen los requisitos especificados y se ajustan a los planes establecidos
  - **Proceso de verificación**: Determina si los requisitos de un sistema o del software están completos y son correctos
  - **Proceso de validación**: Sirve para determinar si el sistema o software final cumple con los requisitos previstos para su uso
  - **Proceso de revisión conjunta**: Sirve para evaluar el estado del software y sus productos en una actividad del ciclo de vida o una fase de un proyecto
  - **Proceso de auditoría**: Permite determinar, en los hitos predeterminados, si se han cumplido los requisitos, los planes y el contrato
  - **Proceso de resolución de problemas**: Permite analizar y eliminar los problemas descubiertos durante el desarrollo, explotación, el mantenimiento u otro proceso

Norma ISO 12207-1 (v)

- **Procesos generales**: Los emplea una organización para llevar a cabo funciones tales como la gestión, la formación del personal o la mejora del proceso
  - **Proceso de gestión**: Actividades y tareas genéricas que puede emplear cualquier organización que tenga que gestionar sus procesos, incluyendo planificación, seguimiento y control, y revisión y evaluación
  - **Proceso de infraestructura**: Establece la infraestructura necesaria para cualquier otro proceso
  - **Proceso de mejora**: Sirve para establecer, valorar, medir, controlar y mejorar los procesos del ciclo de vida del software
  - **Proceso de formación**: Sirve para proporcionar y mantener al personal formado

Norma ISO 12207-1 (vi)

- **Proceso de adaptación**: Sirve para realizar la adaptación básica de la norma ISO-12207 con respecto a los proyectos software. Es necesario comprender los procesos, las organizaciones y sus relaciones bajo diferentes puntos de vista
  - Bajo el punto de vista del contrato, el comprador y el proveedor negocian y firman un contrato, empleando los procesos de adquisición y suministro
  - Bajo el punto de vista de gestión, el comprador, el proveedor, el desarrollador, el operador y el personal de mantenimiento gestionan sus respectivos procesos para el proyecto software
  - Bajo el punto de vista de explotación, el operador proporciona el servicio de explotación del software a los usuarios
  - Bajo el punto de vista de ingeniería, el desarrollador o el personal de mantenimiento llevan a cabo sus respectivas tareas de ingeniería para producir o modificar los productos software
  - Bajo el punto de vista de soporte, los grupos (tales como el de la gestión de la configuración, el aseguramiento de la calidad...) proporcionan servicios de apoyo a otros grupos en el cumplimiento de tareas únicas y específicas

Tipos de modelos de procesos

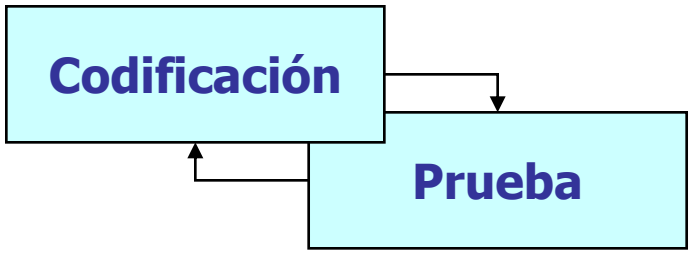
- **Modelos lineales o secuenciales**
- **Modelos basados en prototipos**
- **Modelos basados en métodos formales**
- **Modelos evolutivos (iterativos e incrementales)**
- **Modelos basados en reutilización**

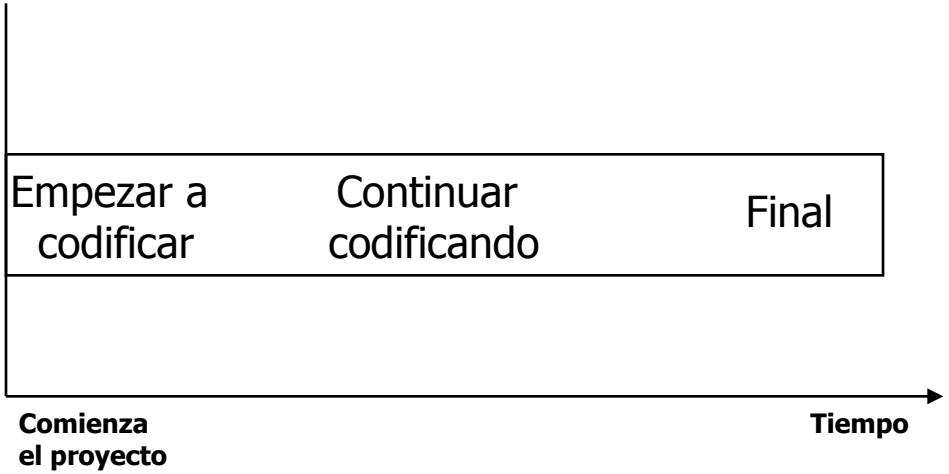
Modelos lineales o secuenciales

- Han sido ampliamente utilizados
  - Ofrecen grandes facilidades a los gestores para controlar el progreso de los proyectos
  - Proponen un enfoque sistemático, secuencial, para el desarrollo del software
  - Comienza en un nivel de sistemas y progresa con el análisis, diseño, codificación, pruebas y mantenimiento
  - Fases separadas en la especificación y el desarrollo
- La filosofía de estos modelos de proceso no es realista
  - No se ajusta al proceso de desarrollo software
  - Raramente sigue un flujo secuencial sino que exige diversas iteraciones
  - No ofrece un soporte adecuado a las técnicas de desarrollo basadas en objetos y componentes

Modelo primitivo (i)

- Se le conoce también con el nombre de **Modelo Prueba y Error** o **Modelo Codifica y Mejora**
- Proceso de desarrollo aplicado en las primeras experiencias de programación
- Supone una iteración de fases codificación-depuración sin ninguna planificación ni diseños previos





Modelo primitivo (ii)

- Inconvenientes
  - Código pobremente estructurado tras varias iteraciones
    - Código espagueti
  - Caro de desarrollar por las numerosas recodificaciones
  - Posible rechazo del usuario al no existir análisis de requisitos
  - Caro de depurar por la falta de planificación
  - Caro de mantener por la falta de estructura y documentación

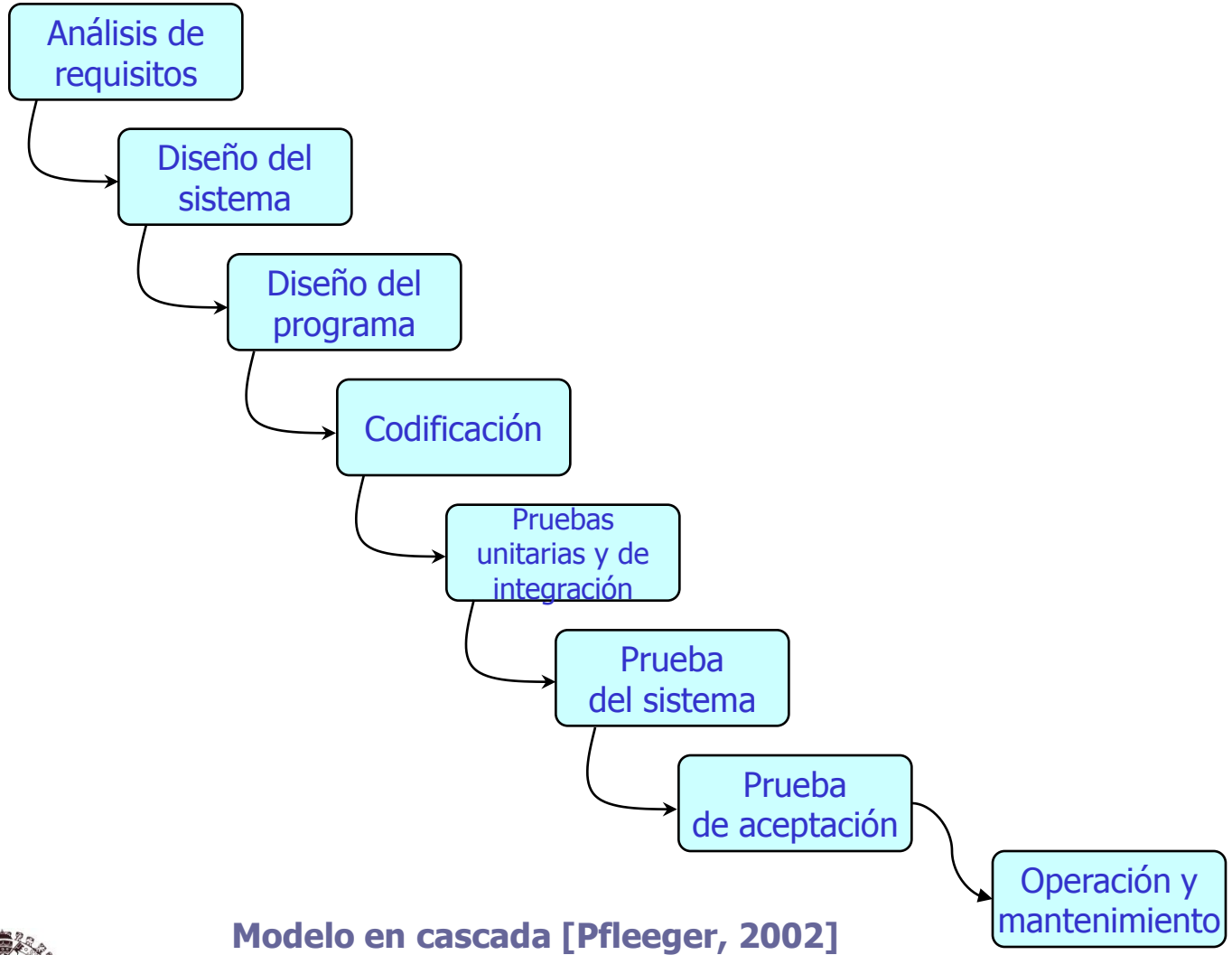
Modelo clásico o en cascada (i)

- Enunciado por W. Royce (1970)
- Uno de los primeros modelos propuestos, derivado de otros procesos de ingeniería
- Recibe el nombre de "en cascada" porque las etapas se representan cayendo en cascada desde una etapa a la siguiente
- Ha sido utilizado para prescribir las actividades de desarrollo de software en varios contextos
  - Se encuentra definido en la norma Estándar 2167-A del DoD de EEUU

Modelo clásico o en cascada (ii)

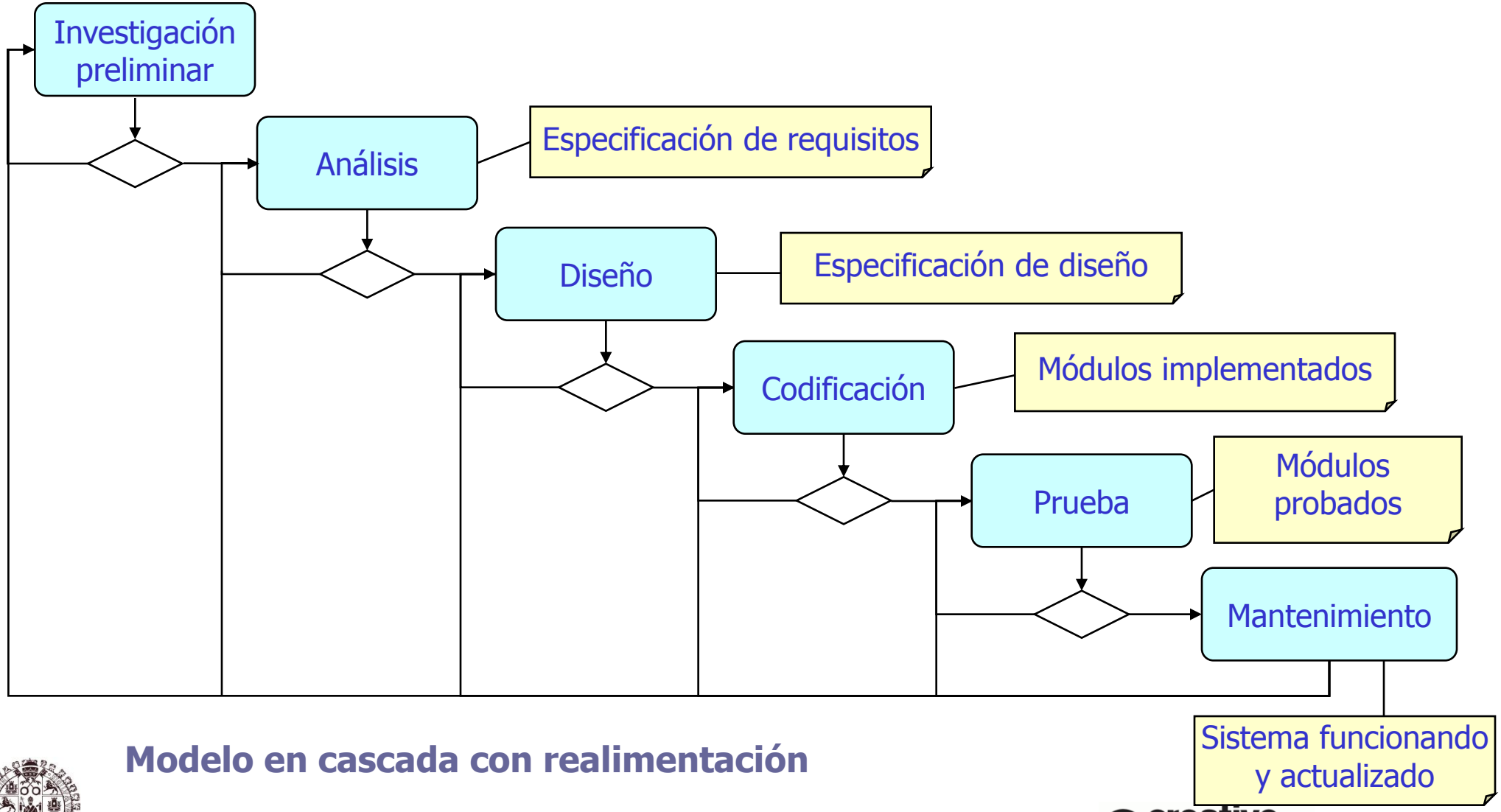
- Fases secuenciales
  - Comienzo de una fase con el término de la anterior
  - Paso de fase al conseguir los objetivos
  - Obtención de documentos como criterio de finalización de fase
  - El final de una fase puede suponer un punto de revisión
- Apoyo a los gestores
- Distintas configuraciones
  - Muchos modelos más complejos son variaciones del modelo en cascada que incorporan lazos de realimentación y fases adicionales

Modelo clásico o en cascada (iii)



Modelo en cascada [Pfleeger, 2002]

Modelo clásico o en cascada (iv)



Modelo en cascada con realimentación

Modelo clásico o en cascada (v)

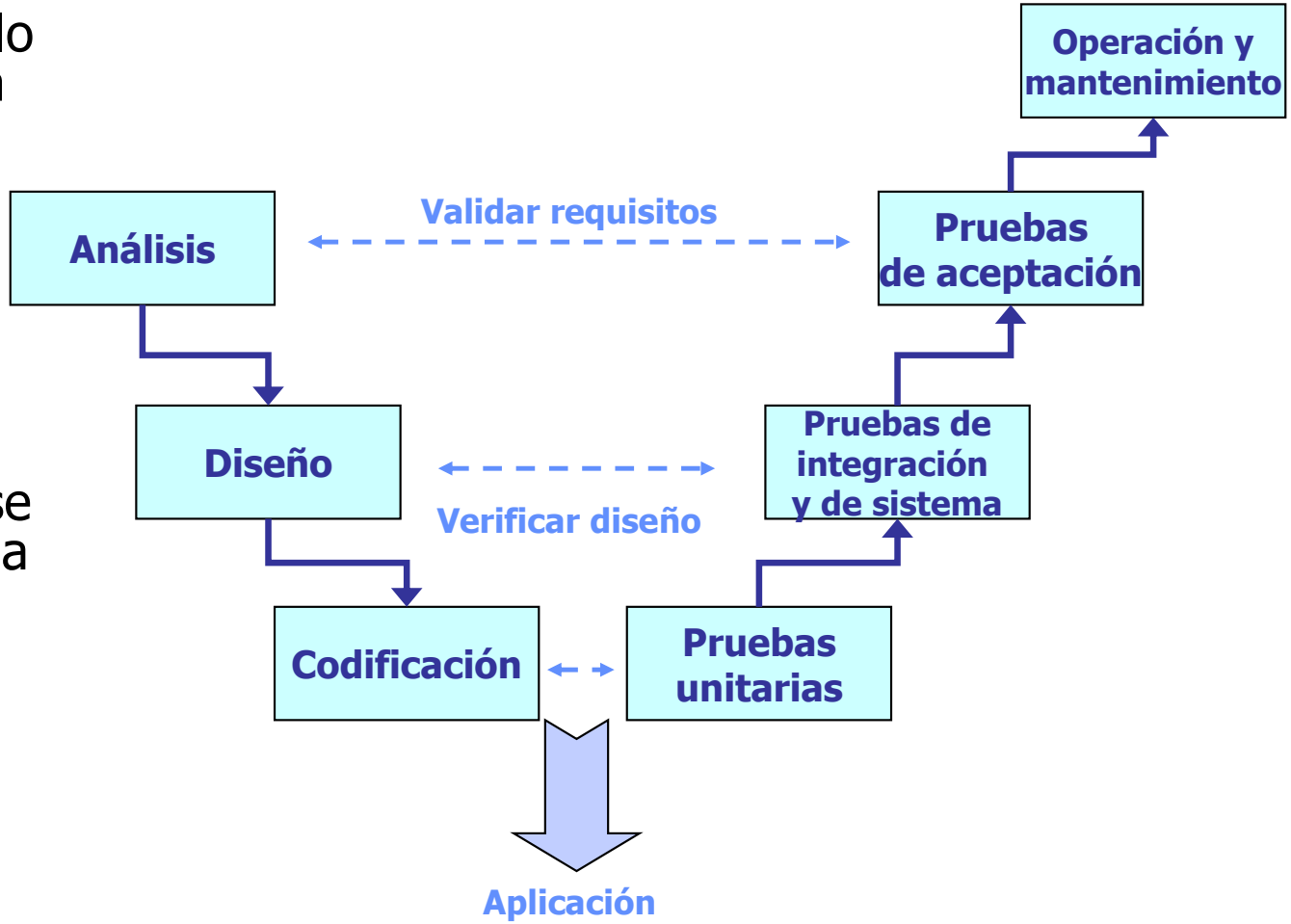
- Inconvenientes
  - Su progresión secuencial o lineal no refleja la manera en que realmente se desarrolla el software [Pfleeger, 2002; Pressman, 2006]
  - Es un modelo que adolece de rigidez [Pressman, 2006]
    - Exige al usuario que exponga explícitamente todos los requisitos al principio, presentando problemas para gestionar la incertidumbre natural propia del comienzo de la mayoría de los proyectos
  - Se tarda mucho tiempo en pasar por todo el ciclo [Piattini et al., 2004]
  - Es un modelo monolítico [Pressman, 2006]
    - Hasta llegar a las etapas finales del desarrollo no habrá una versión operativa del programa, lo que influye negativamente en el descubrimiento a tiempo de errores o incongruencias en los requisitos
  - Impone una estructura de gestión de proyecto al desarrollo del sistema [McCracken y Jackson, 1981]
  - No trata al software como un proceso de resolución de problemas [Curtis et al., 1987]

Modelo clásico o en cascada (vi)

- Consideraciones finales
  - Tiene un lugar destacado en la Ingeniería del Software
  - Proporciona una plantilla para adecuar los métodos
  - Es muy utilizado
  - Tiene problemas pero es mejor que desarrollar sin guías

Modelo V

- Es una variación del modelo en cascada que demuestra cómo se relacionan las actividades de prueba con las de análisis y desarrollo [GMD, 1992]
- Presenta una implantación ascendente
- Demuestra que el desarrollo de las pruebas se efectúa de manera síncrona con el desarrollo del programa
- Mientras que el modelo clásico centra su atención en los documentos y artefactos producidos, el modelo en V lo hace en la actividad y la exactitud



Modelo V

Modelos basados en prototipos (i)

- Un prototipo es un modelo experimental de un sistema o de un componente de un sistema que tiene los suficientes elementos que permiten su uso
  - Es una solución parcial que describe la interacción entre el hombre y la máquina, mostrando parte de su funcionalidad no optimizada

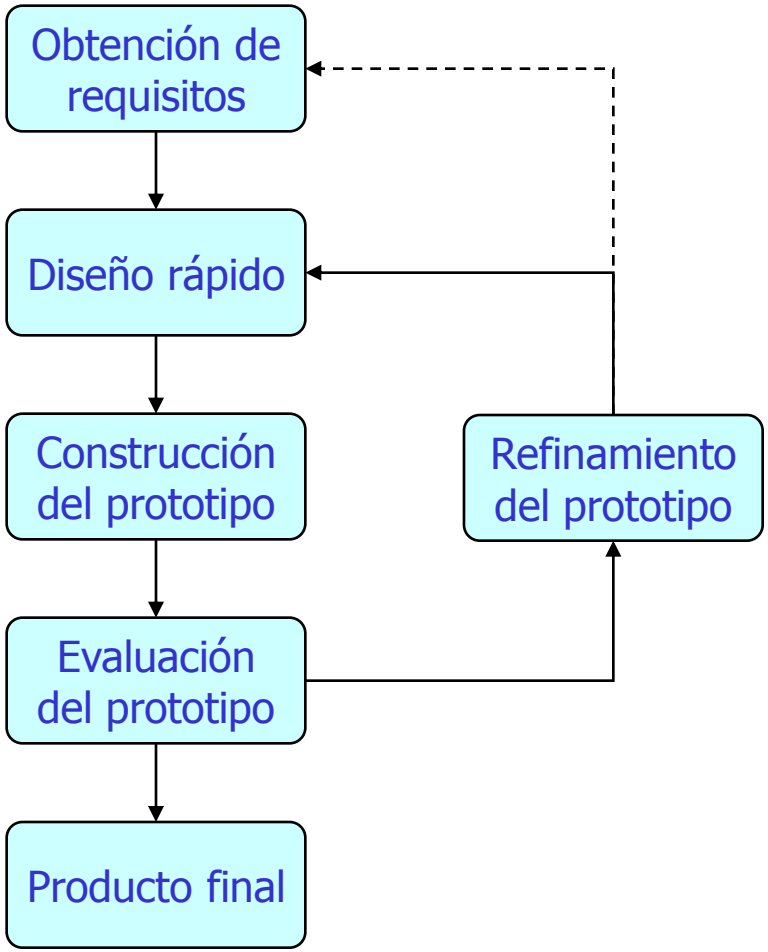
- Los modelos de proceso basados en prototipos presentan un componente iterativo que facilita involucrar a los clientes/usuarios en el desarrollo del software para ayudar a éstos a comprender los requisitos

Modelos basados en prototipos (ii)

- Estos modelos pueden presentar diferentes enfoques en la utilización de los prototipos en el ciclo de vida
  - **Enfoque desechable**
    - El propósito del prototipo es validar algún aspecto del sistema, sirviendo, en este caso, como herramienta auxiliar a la especificación de requisitos y el diseño [Davis, 1982]
    - Este enfoque suele derivar en un modelo lineal una vez que el prototipo ha cumplido su misión
  - **Enfoque evolutivo**
    - El prototipo se utiliza como alternativa de ciclo de vida [Basili y Turner, 1975]
    - Es la base de los modelos de proceso evolutivos
  - **Enfoque mixto**
    - Conocido con el nombre de prototipado operativo [Davis, 1992]
    - Combina ambos tipos de prototipos

Modelo de prototipos (i)

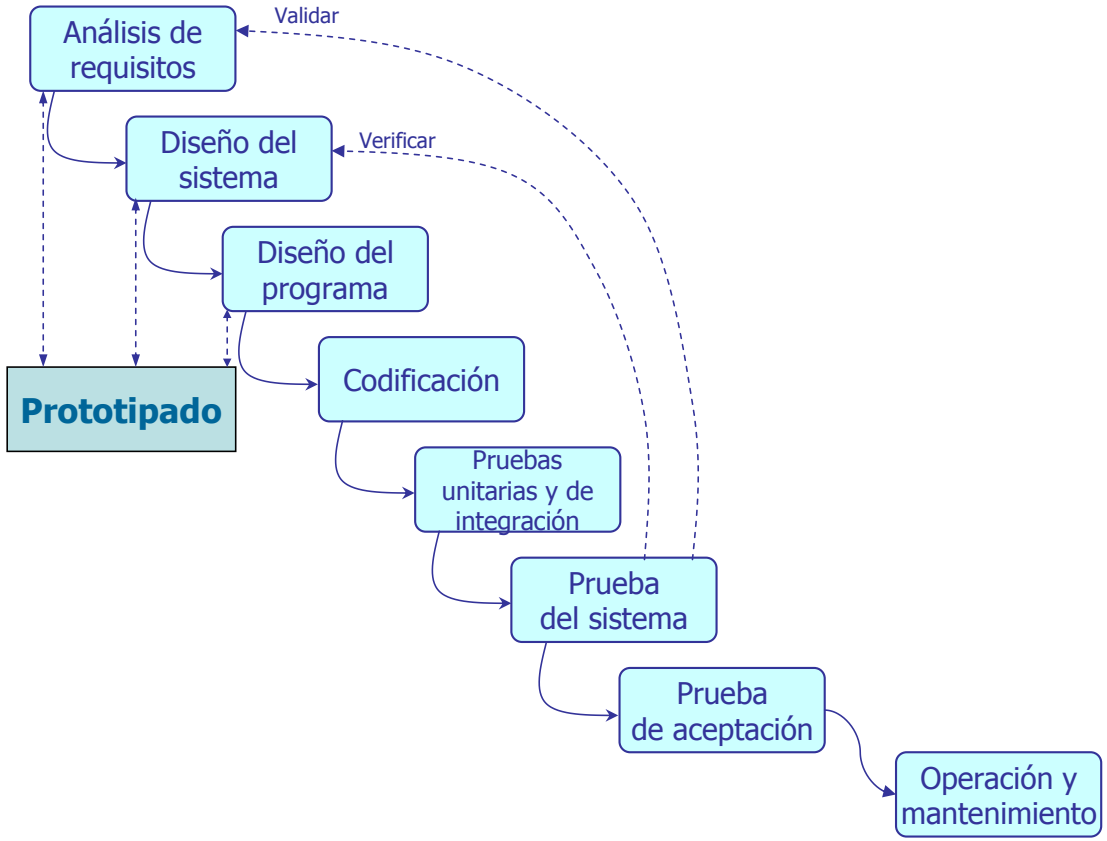
- Basado en prototipos desechables
  - Construcción rápida y a bajo coste
- Idea del software en líneas generales desde el punto de vista del usuario
- Idealmente sirve para identificar los requisitos del software
  - Introduce cierta flexibilidad en la introducción de requisitos
- Proceso iterativo
  - La iteración ocurre cuando el prototipo se pone a punto para satisfacer las necesidades del cliente, permitiendo a la vez que el desarrollador comprenda mejor lo que necesita hacer



Modelo de prototipos

Modelo de prototipos (ii)

- Ventajas
  - Permite solventar objeciones del usuario
  - Sirve para formalizar la aceptación previa
  - Introduce flexibilidad en la captura de requisitos
    - Ayuda a combatir la rigidez de los modelos lineales
  - Es útil cuando el área de aplicación no está definida, cuando el riesgo de rechazo el alto, o como forma de evaluar el impacto de una aplicación
  - El prototipado es un subproceso que puede incluirse como parte de otros modelos de proceso
    - Por ejemplo puede combinarse con un ciclo en cascada para intentar solventar ciertas carencias de éste



Modelo en cascada con prototipado [Pfleeger, 2002]

Modelo de prototipos (iii)

- Inconvenientes
  - El sistema se puede llegar a deteriorar tendiendo hacia el modelo primitivo
  - Se suele refinar el prototipo hacia el sistema final en lugar de desecharlo y empezar desde el principio
  - El cliente puede encontrar atractivo el prototipo y quedarse con el prototipo como sistema final
  - Relajación de los desarrolladores
  - No disminuye el tiempo entre la definición de los requisitos y la entrega del producto
  - Al usuario le desagrada que se deseche código

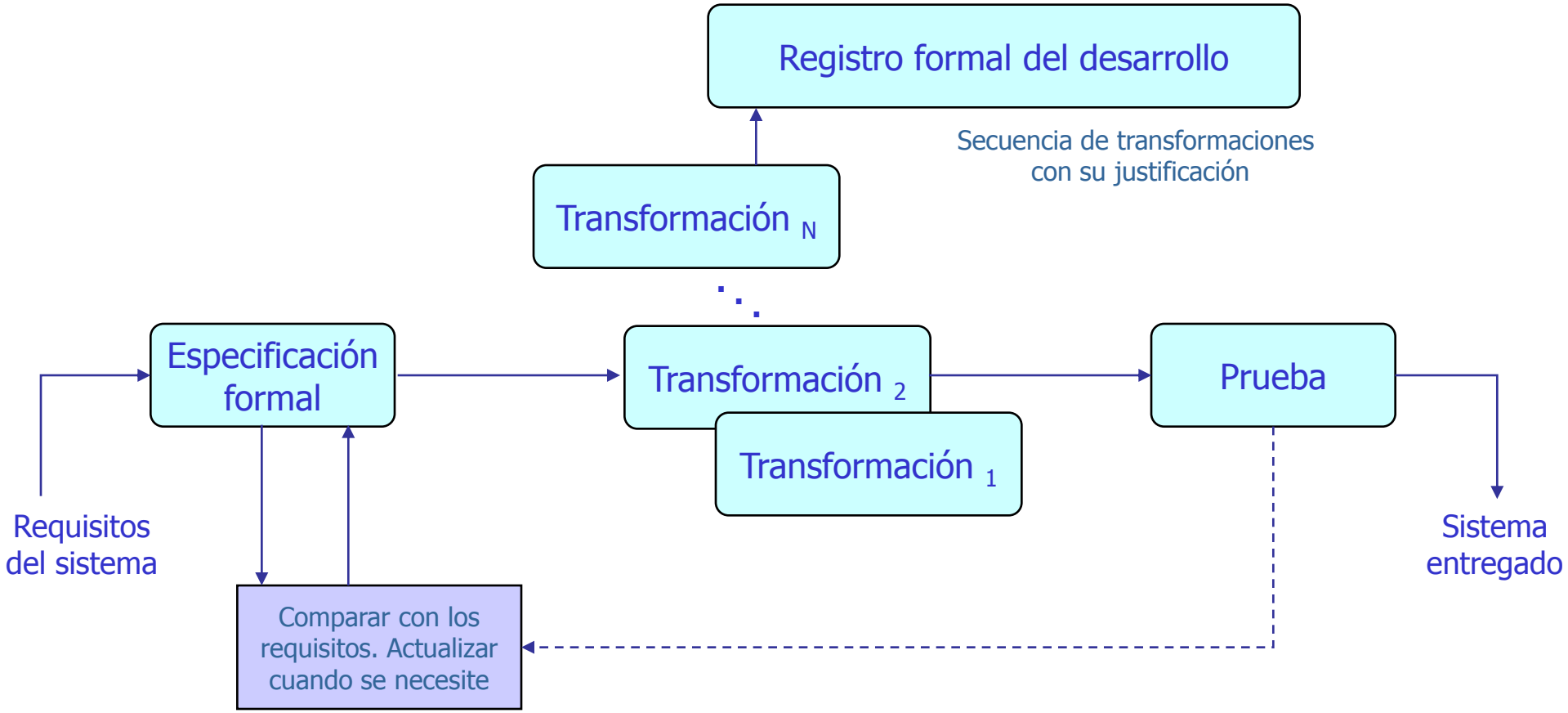
Modelos basados en métodos formales

- Permiten especificar, desarrollar y verificar un sistema aplicando una notación matemática
- Algunas organizaciones de desarrollo de software aplican una variación de este enfoque, llamado **ingeniería del software de sala limpia** [Mills et al., 1987; Dyer, 1992]
- Ventajas
  - La ambigüedad, lo incompleto y la inconsistencia se descubren y se corrigen fácilmente
  - Los métodos formales de diseño sirven como base para la verificación de programas
  - Sirven de base para la generación automática de código
- Barreras para su implantación
  - El desarrollo es bastante caro y lleva mucho tiempo
  - El estudio de los métodos es costoso
  - Es difícil establecer modelos formales como mecanismo de comunicación con los clientes

Modelo de transformación (i)

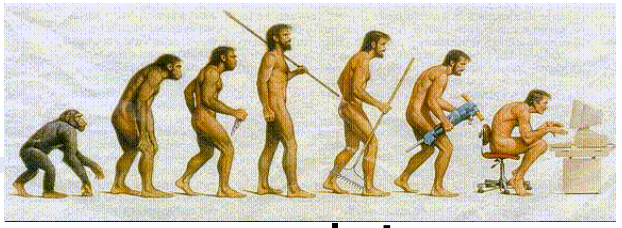
- Usando un soporte automatizado se aplican una serie de transformaciones para convertir una especificación formal en un sistema ejecutable [Balzer, 1981]
- Intenta reducir la probabilidad de la existencia de errores eliminándolos en las diferentes iteraciones de que consta el proceso
  - En las iteraciones no se modifica el código sino la especificación formal, obteniéndose un código que siempre se corresponde con la especificación
- Inconvenientes
  - Poca aceptación de los métodos formales
  - No se dispone de transformaciones automáticas más que en pequeños dominios y entornos
  - Ofrecen muy poca flexibilidad al usuario ante situaciones no planificadas a priori

Modelo de transformación (ii)



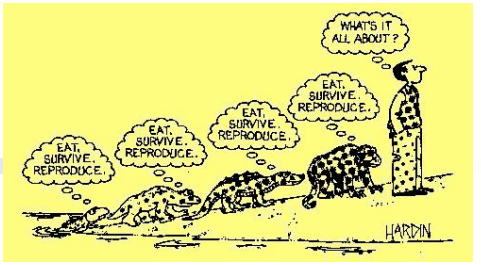
Modelo de transformación, basado en [Pfleeger, 2002]

Modelos evolutivos (i)



- El software, al igual que todos los sistemas complejos, evolucionan con el tiempo [Gilb, 1988]
- Se caracterizan porque permiten desarrollar versiones cada vez más completas del software, teniendo en cuenta la naturaleza evolutiva del software
- Presentan la filosofía de poner un producto en explotación cuanto antes
  - Están muy ligados a la idea de prototipado evolutivo
- Existen muchos modelos de proceso evolutivos
- Los modelos evolutivos son iterativos [Pressman, 2006]
  - Se caracterizan por la forma en que permiten a los ingenieros de software desarrollar versiones cada vez más completas del producto software, que puede ir entregándose al cliente en forma de incrementos

Modelos evolutivos (ii)

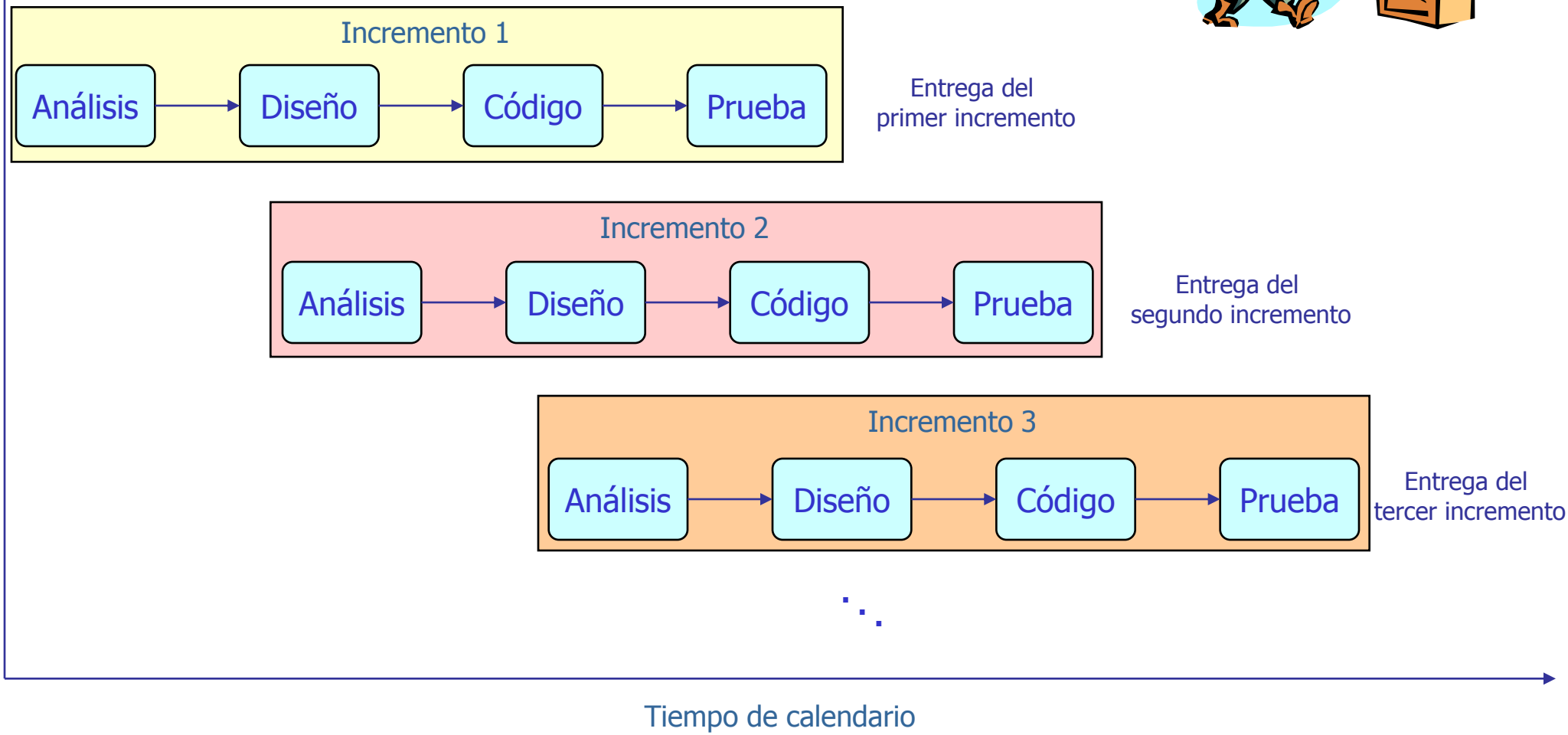


- Los desarrollos orientados a objetos se ajustan a un modelo de proceso iterativo e incremental
- Se puede argumentar lo mismo para los desarrollos basados en componentes
- Esto es así porque
  - Las tareas de cada fase se llevan a cabo de una forma iterativa
  - A la vez que existe un ciclo de desarrollo **análisis-diseño-implementación-análisis** que permite hacer evolucionar al sistema
  - En el desarrollo incremental el sistema se divide en un conjunto de particiones
    - Cada una se desarrolla de forma completa hasta que se finaliza el sistema
- Esta idea de iteratividad máxima propia de la orientación a objetos ha sido equiparada por autores como James Rumbaugh (1992) o L. B. S. Raccoon (1995) a las fractales o la teoría del caos

Modelo incremental (i)

- En este modelo, el sistema, tal y como está especificado en la especificación de requisitos del software, se divide en subsistemas de acuerdo a su funcionalidad
- Las versiones se definen comenzando con un subsistema funcional pequeño y agregando funcionalidad con cada nueva versión
  - Cada nueva parte entregada se denomina incremento
- Combina elementos del modelo en cascada con la filosofía interactiva de construcción de prototipos
- Aplica secuencias lineales de forma escalonada mientras progresa el calendario del proyecto
  - Cada secuencia lineal supone un incremento [McDermid y Rook, 1993]

Modelo incremental (ii)

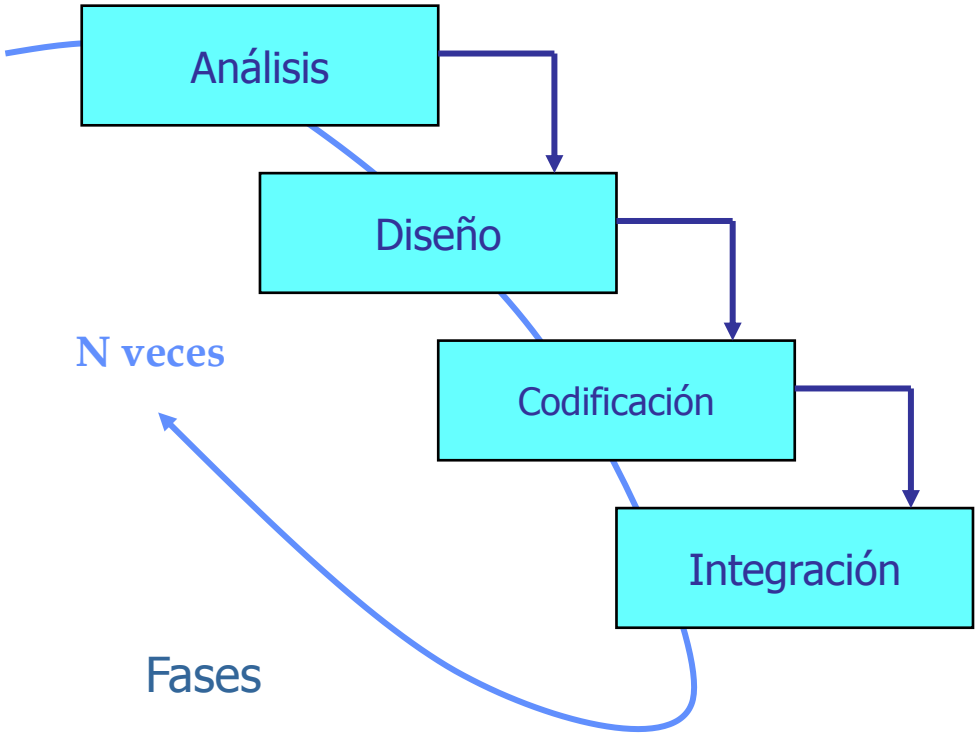


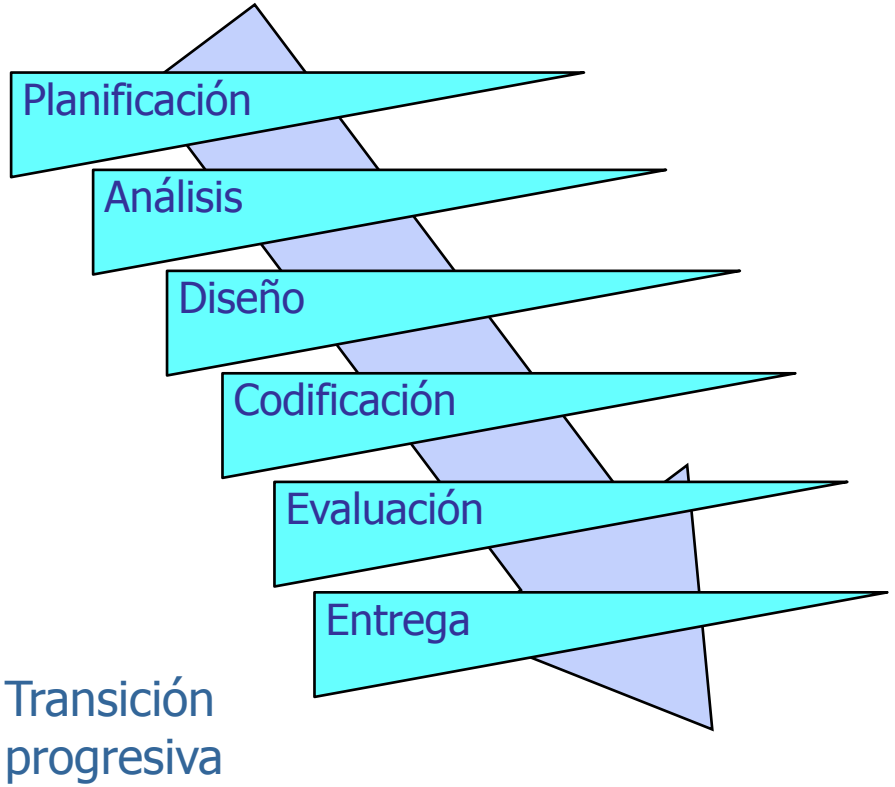
Modelo incremental [Pressman, 2006]

Modelo iterativo (i)

- Entrega un sistema completo desde el principio, para posteriormente cambiar la funcionalidad de cada subsistema con cada versión
- Características del ciclo iterativo [Muller, 1997]
  - Se basa en la evolución de prototipos ejecutables, mensurables y evaluables
  - Se van incorporando cambios en cada iteración
  - Exige más atención e implicación de todos los actores del proyecto
- Minicascada
  - Cada iteración reproduce el ciclo de vida en cascada, pero a una escala menor
  - Los objetivos de cada iteración se establecen en función de la evaluación de las iteraciones precedentes
  - Las fases tradicionales se cubre gradualmente en las diversas iteraciones
  - Las actividades internas se solapan porque dentro de una iteración no necesitan terminarse de golpe, siendo la transición entre dos actividades progresiva

Modelo iterativo (ii)



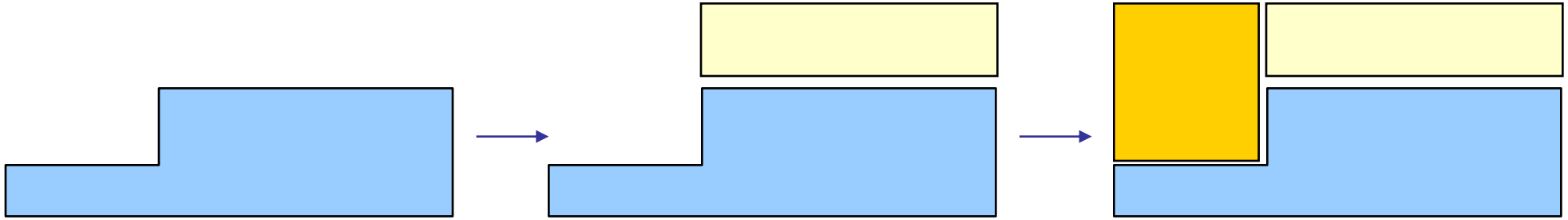


Modelo iterativo (iii)

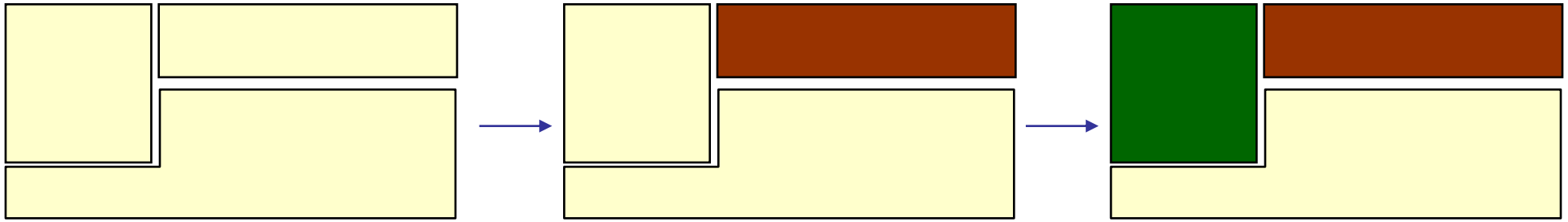
- Evaluación de las iteraciones
  - Deben definirse criterios de evaluación de las iteraciones
  - Una iteración se marca por etapas intermedias que permitan medir los progresos. Debe haber al menos dos etapas
    - Revisión inicial: fija los objetivos y criterios de la iteración
    - Revisión de evaluación: valida los resultados
- Mitos sobre el ciclo de vida iterativo [Muller, 1997]
  - El ciclo de vida iterativo favorece los apaños
  - El ciclo de vida iterativo engendra problemas
  - El ciclo de vida iterativo e incremental exige recomenzar *n* veces hasta que el resultado sea el adecuado
  - El ciclo de vida iterativo es una excusa para no planificar y gestionar un proyecto
  - El ciclo de vida iterativo sólo concierne a los desarrolladores
  - El ciclo de vida iterativo favorece siempre añadir nuevas necesidades, sin fin

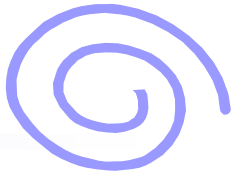
Incremental vs. iterativo

Desarrollo incremental: sistema parcial, funcionalidad completa



Desarrollo iterativo: sistema completo; funcionalidad parcial



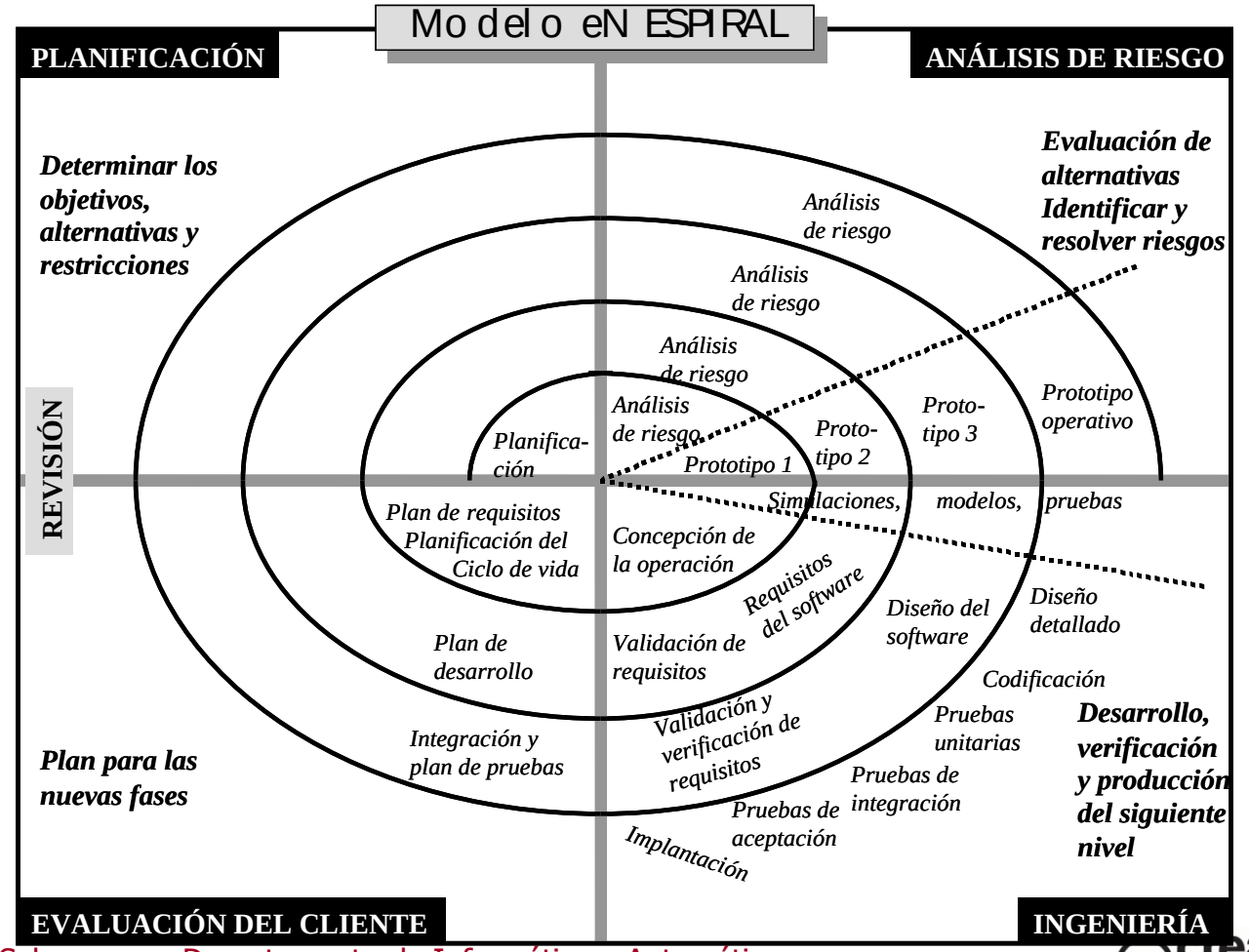


Modelo en espiral (i)

- Los grandes sistemas se componen por regla general de subsistemas
- No tiene porque ser aplicable el mismo modelo de proceso a todos los subsistemas
  - Prototipado para las especificaciones de alto riesgo
  - Modelo lineal para los desarrollos claros
- El modelo en espiral propuesto por Boehm (1986) es un modelo híbrido que incluye estos aspectos
  - Combina la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal
  - Se introduce el análisis de riesgos
    - Riesgo es toda circunstancias potencialmente adversa que puede afectar a un proceso de desarrollo o a la calidad de los productos [Ghezzi et al., 1991]
  - No se fijan fases
    - Las fases las decide el gestor del proyecto
  - El software se desarrolla en una serie de versiones incrementales
  - El modelo se divide en sectores de tareas
    - Pueden existir entre tres y seis regiones de tareas

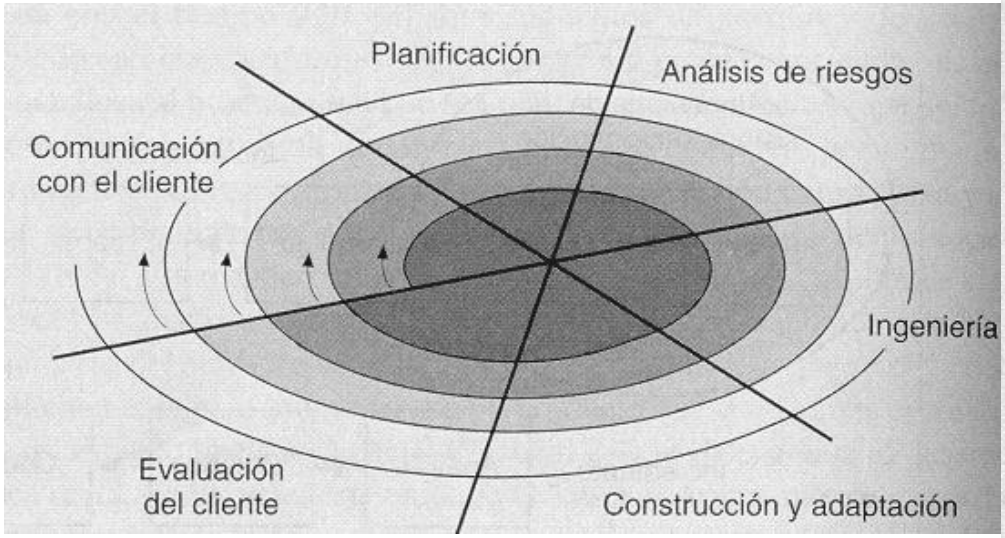
Modelo en espiral (ii)

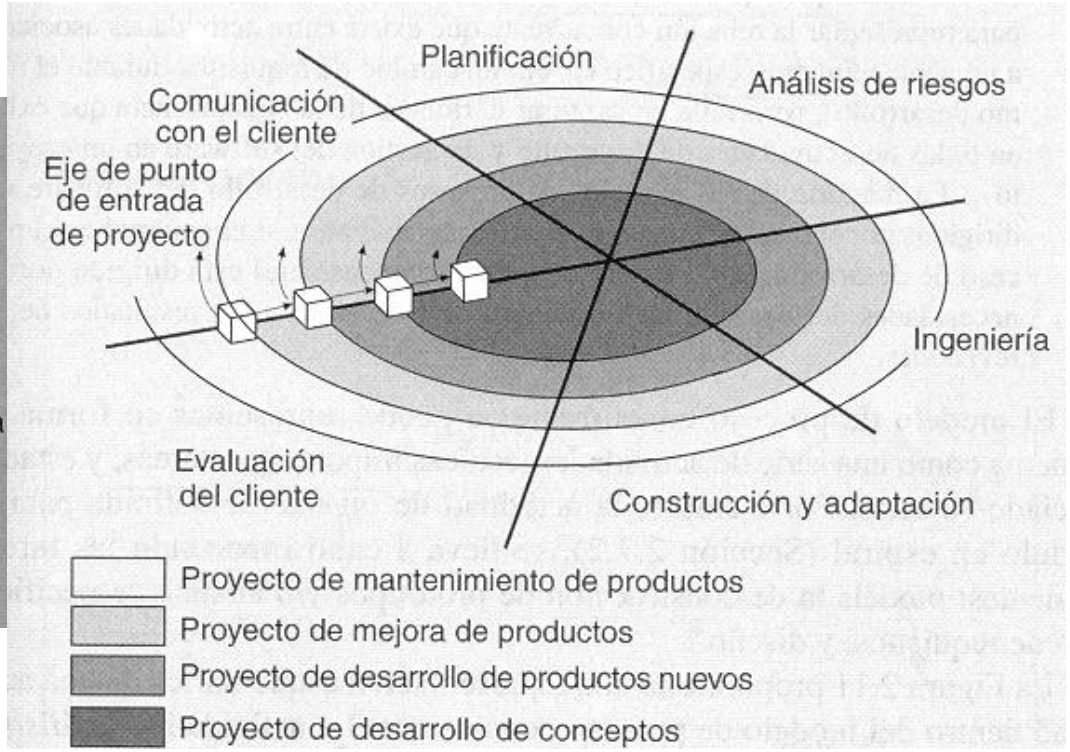
- En el modelo original de **Boehm** (1986) aparecen cuatro regiones de tareas
  - Planificación, Análisis de riesgos, Ingeniería, Evaluación del cliente



Modelo en espiral (iii)

- En [Pressman, 2006] se presenta una variante con seis regiones de tareas
  - Comunicación con el cliente, Planificación, Análisis de riesgos, Ingeniería, Construcción y adaptación, Evaluación del cliente





Ciclo de vida en espiral [Pressman, 2006]

Modelo en espiral (iv)

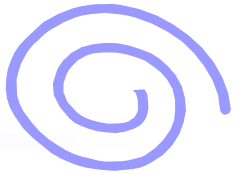
- Progresión de la espiral
  - Se progresa a partir del centro de la espiral en el sentido de las agujas del reloj
    - Desarrollo de una especificación de productos
    - Desarrollo de un prototipo
    - Desarrollo de versiones más sofisticadas del software
  - Cada paso de la región de planificación produce ajustes en el plan del proyecto
    - El coste y la planificación se ajustan según la reacción ante la evaluación del cliente
    - El gestor del proyecto ajusta el número planificado de iteraciones requeridas para completar el software
  - Presenta un enfoque realista del desarrollo de sistemas y de software a gran escala
    - Según progresa el proceso, el desarrollador y el cliente comprenden y reaccionan mejor ante riesgos en cada uno de los niveles evolutivos

Modelo en espiral (v)

- Ventajas
  - Refleja de forma más realista la idiosincrasia del desarrollo de software
  - Toma lo mejor y evita lo peor de los demás modelos, según la situación en cada momento
  - Las opciones de reutilización se tienen en cuenta desde el primer momento
  - Proporciona una preparación para la evolución, crecimiento y cambio
  - Proporciona un mecanismo para incorporar objetivos de calidad en el desarrollo
  - Se centra en la eliminación de errores y opciones no atractivas desde el principio
  - Determina el nivel de esfuerzo de cada fase en cada proyecto
  - Se sigue el mismo procedimiento para el desarrollo que para el mantenimiento, con lo que se evitan los problemas de las "mejoras rutinarias" de alto riesgo
  - Permite una gran flexibilidad
  - Se adapta bien al diseño y programación orientado a objetos

Modelo en espiral (vi)

- Inconvenientes
  - No ha sido desarrollado en el mundo de la contratación comercial sino en el de desarrollo interno
    - Puede resultar difícil convencer a grandes clientes de que el enfoque evolutivo es controlable
  - Necesita experiencia en la evaluación de riesgos, expertos, que no siempre están disponibles
  - Necesita una elaboración adicional de los pasos del modelo



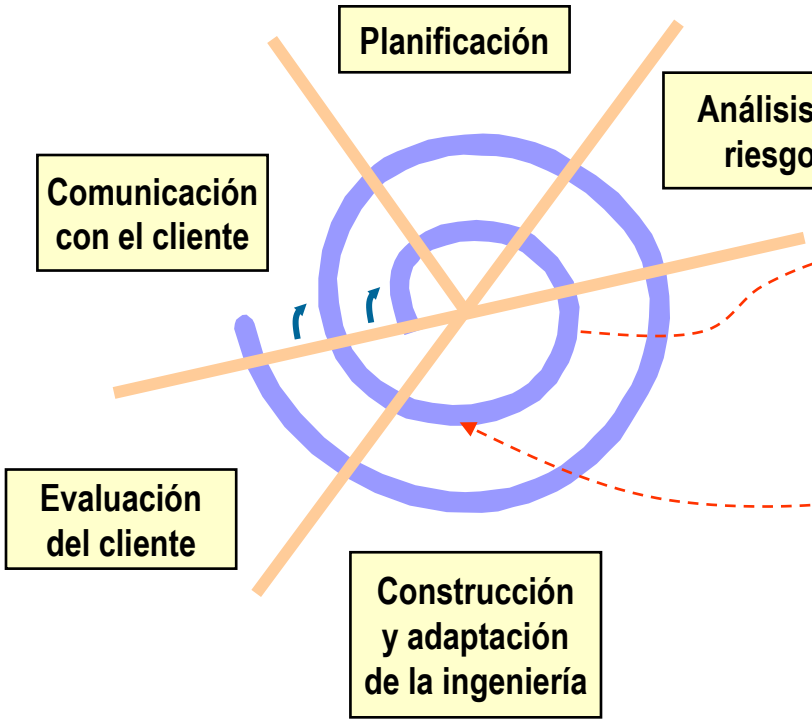
Modelo en espiral (vii)

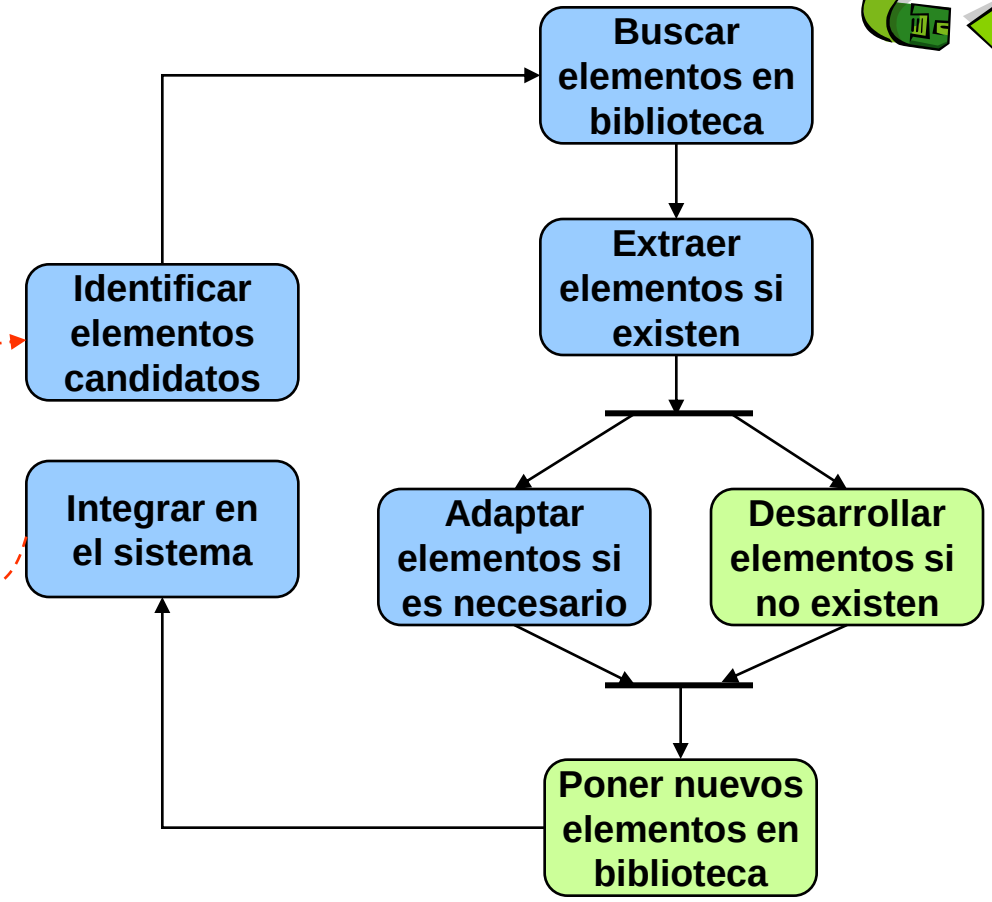
- Diferencias con otros modelos [Wolff, 1989]
  - Existe un reconocimiento explícito de las diferentes alternativas para alcanzar los objetivos del proyecto
  - El modelo se centra en identificar los riesgos de cada alternativa, así como las formas de solventarlos
  - La división de los proyectos en ciclos, cada uno con un acuerdo al final, implica que existe un acuerdo para los cambios a realizar o para la finalización del mismo, en función de lo aprendido a lo largo del proyecto
  - Es un método que se adapta a cualquier tipo de actividad, alguna de las cuales no existen en otros paradigmas, como puede ser la consulta a asesores externos

Modelos basados en reutilización (i)

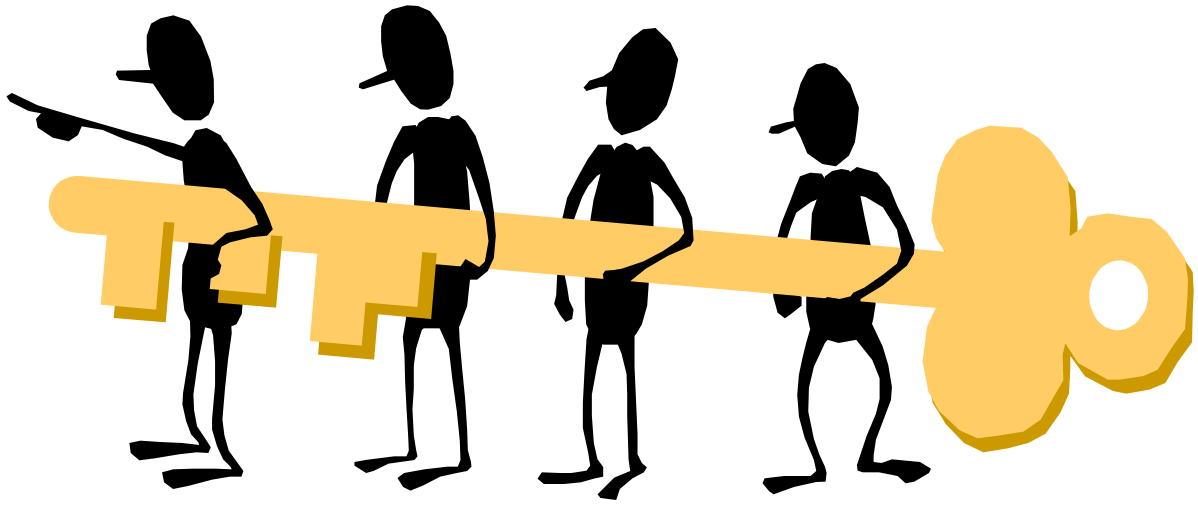
- Esta aproximación se basa en la existencia de un número significativo de elementos reutilizables
- El proceso de desarrollo del sistema se centra en la integración de estos elementos en un sistema, en lugar de desarrollarlo desde cero
- Incorpora muchas características del modelo en espiral
  - Es evolutivo por naturaleza [Nierstrasz, 1992] y existe un enfoque iterativo para la creación de software
- El proceso software tiende a estructurarse en dos subprocesos distintos y separados [Karlsson, 1995]
  - El desarrollo para reutilización
    - Construcción de elementos reutilizables dentro de un dominio concreto
  - El desarrollo con reutilización
    - Construcción de aplicaciones utilizando elementos reutilizables

Modelos basados en reutilización (ii)





5. Metodologías



Introducción (i)

- Resulta necesario establecer un enfoque sistemático y disciplinado para llevar a cabo un desarrollo software
- El uso de una **metodología** permite el dominio del proceso descrito
- Una metodología es *el conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal* [RAE, 2001]
- Una metodología software es un enfoque, una manera de interpretar la realidad o la disciplina en cuestión, que en este caso particular correspondería a la Ingeniería del Software
- Se elaboran a partir del marco definido por uno o varios ciclos de vida
- No existe un consenso entre los diversos autores sobre el concepto de metodología

Introducción (ii)

- Desde una perspectiva de Ingeniería de Software, una metodología
  - Describe cómo se organiza un proyecto
  - Establece el orden en el que la mayoría de las actividades tienen que realizarse y los enlaces entre ellas
  - Indica cómo tienen que realizarse algunas tareas proporcionando las herramientas concretas e intelectuales
- Con una metodología se intentan cubrir las siguientes necesidades [Piattini et al., 2004]
  - Mejores aplicaciones
  - Mejor proceso de desarrollo
  - Establecer un proceso estándar en una organización

Definiciones (i)

- Conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas de información. Según esto, una metodología es un conjunto de componentes que especifican: *Cómo dividir un proyecto en etapas; Qué tareas se llevarán a cabo en cada etapa; Qué salidas se producen y cuándo deben producirse; Qué restricciones se aplican; Qué herramientas van a ser utilizadas; Cómo se gestiona y se controla el proyecto* [Maddison, 1983]
- Una metodología es una aproximación organizada y sistemática para el ciclo de vida del sistema o sus partes. Especifica las tareas individuales y sus secuencias [Palvia y Nosek, 1993]
- Conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo software [Piattini et al., 2004]

Definiciones (ii)

- Se puede definir **metodología de Ingeniería del Software** como

> *Un proceso para producir software de forma organizada, empleando una colección de técnicas y convenciones de notación predefinidas* [Rumbaugh et al., 1991]

- Confusión entre los términos metodología, método y ciclo de vida por abuso del lenguaje técnico
  - Una metodología puede seguir uno o varios modelos de ciclo de vida, esto es, el ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto, pero no cómo. Esto sí lo debe indicar la metodología
  - Una metodología es un concepto más amplio que el de método. Así, se puede considerar a la metodología como un conjunto de métodos

Objetivos de las metodologías

- Establecer los requisitos de un sistema software de una forma acertada
- Proporcionar un método sistemático de desarrollo de forma que se pueda controlar su proceso
- Construir un sistema software dentro de un tiempo apropiado y unos costes aceptables
- Construir un sistema que esté bien documentado y que sea fácil de mantener
- Ayudar a identificar, lo antes posible, cualquier cambio que sea necesario realizar dentro del proceso de desarrollo
- Proporcionar un sistema que satisfaga a todas las personas afectadas por el mismo

[Piattini et al., 2004]

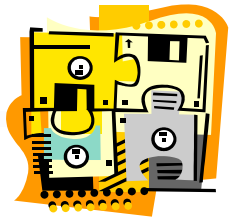
Características deseables en una metodología

- Una metodología debe cubrir [Henderson-Sellers y Firesmith, 1999]
  - Un proceso de ciclo de vida completo, que comprenda aspectos tanto del negocio como técnicos
  - Un conjunto completo de conceptos y modelos que sean internamente consistentes
  - Una colección de reglas y guías
  - Una descripción completa de artefactos a desarrollar
  - Una notación con la que trabajar, idealmente soportada por diversas herramientas CASE y diseñada para una usabilidad óptima
  - Un conjunto de técnicas probadas
  - Un conjunto de métricas, junto con asesoramiento sobre calidad, estándares y estrategias de prueba
  - Identificación de los roles organizacionales
  - Guías para la gestión de proyectos y aseguramiento de la calidad
  - Asesoramiento para la gestión de bibliotecas y reutilización

Clasificación de la metodologías

- Estructuradas
  - Orientadas a procesos
  - Orientadas a datos
- Orientadas a estados y transiciones
- Orientadas al diseño del conocimiento
- Orientadas a objetos
- Orientadas al desarrollo de sistemas hipermediales
- Basadas en métodos formales

Metodologías estructuradas



- Proponen la creación de modelos del sistema que representan los procesos, los flujos y la estructura de los datos de una manera descendente
- Se pasa de una visión general del problema, nivel de abstracción alto, a un nivel de abstracción sencillo
- Esta visión se puede enfocar
  - Hacia un punto de vista funcional del sistema
    - Metodologías orientadas a procesos
  - Hacia la estructura de datos
    - Metodologías orientadas a datos

Metodologías orientadas a procesos

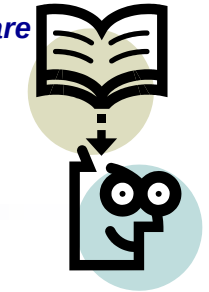
- La Ingeniería del Software se fundamenta en el modelo básico **entrada/proceso/salida** de un sistema
- Estas metodologías se enfocan fundamentalmente en la parte de **proceso**
- Utilizan un enfoque de descomposición descendente para evaluar los procesos del espacio del problema y los flujos de datos con los que están conectados
- Este tipo de metodologías se desarrolló a lo largo de los años 70
- Representantes de este grupo son las metodologías de análisis y diseño estructurado como
  - **Merise** [Tardieu et al., 1986]
  - **YSM** (*Yourdon Systems Method*) [Yourdon Inc., 1993]
  - **SSADM** (*Structured Systems Analysis and Design Method*) [Ashworth y Goodland, 1990]
  - **METRICA v.2.1** [MAP, 1995]
  - **METRICA v3.0** (Parcialmente) [MAP, 2001]

Metodologías orientadas a datos

- Estas metodologías se centran más la parte de **entrada/salida**
- Las actividades de análisis comienzan evaluando en primer lugar los datos y sus interrelaciones para determinar la arquitectura de datos subyacente
- Cuando esta arquitectura está definida, se definen las salidas a producir y los procesos y entradas necesarios para obtenerlas
- Representantes
  - **JSP** (*Jackson Structured Programming*) [Jackson, 1975]
  - **JSD** (*Jackson Structured Design*) [Jackson, 1983]
  - **LCP** (*Logical Construction Program*) [Warnier, 1974]
  - **DESD** (Desarrollo de Sistemas Estructurados de Datos), también conocido como metodología **Warnier-Orr** [Orr, 1977]

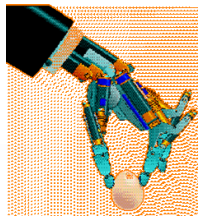
Orientadas a estados y transiciones

- Están dirigidas a la especificación de
  - Sistemas en tiempo real
  - Sistemas que tienen que reaccionar continuamente a estímulos internos y externos (eventos o sucesos)
- Representantes
  - Extensiones de las metodologías de análisis y diseño estructurado de **Ward y Mellor** (1985) y de **Hatley y Pirbhai** (1987)



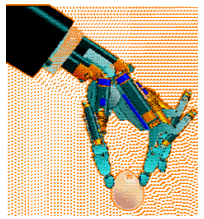
Orientadas al diseño del conocimiento

- Aproximación que se encuentra aún en una fase temprana de desarrollo
- Utiliza técnicas y conceptos de Inteligencia Artificial para especificar y generar sistemas de información
- Representantes
  - **KADS** (*Knowledge Acquisition and Development Systems*) [Wielinga et al., 1991]
  - **IDEAL** [Gómez et al., 1998]



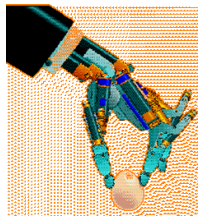
Orientadas a objetos (i)

- Se fundamentan en la integración de los dos aspectos de los sistemas de información: **datos** y **procesos**
- En este paradigma un sistema se concibe como un conjunto de objetos que se comunican entre sí mediante mensajes
- El objeto encapsula datos y operaciones
  - Este enfoque permite un modelado más natural del mundo real y facilita enormemente la reutilización del software
- Las metodologías orientadas a objetos acortan la distancia existente entre el espacio de conceptos (lo que los expertos o usuarios conocen) y el espacio de diseño e implementación



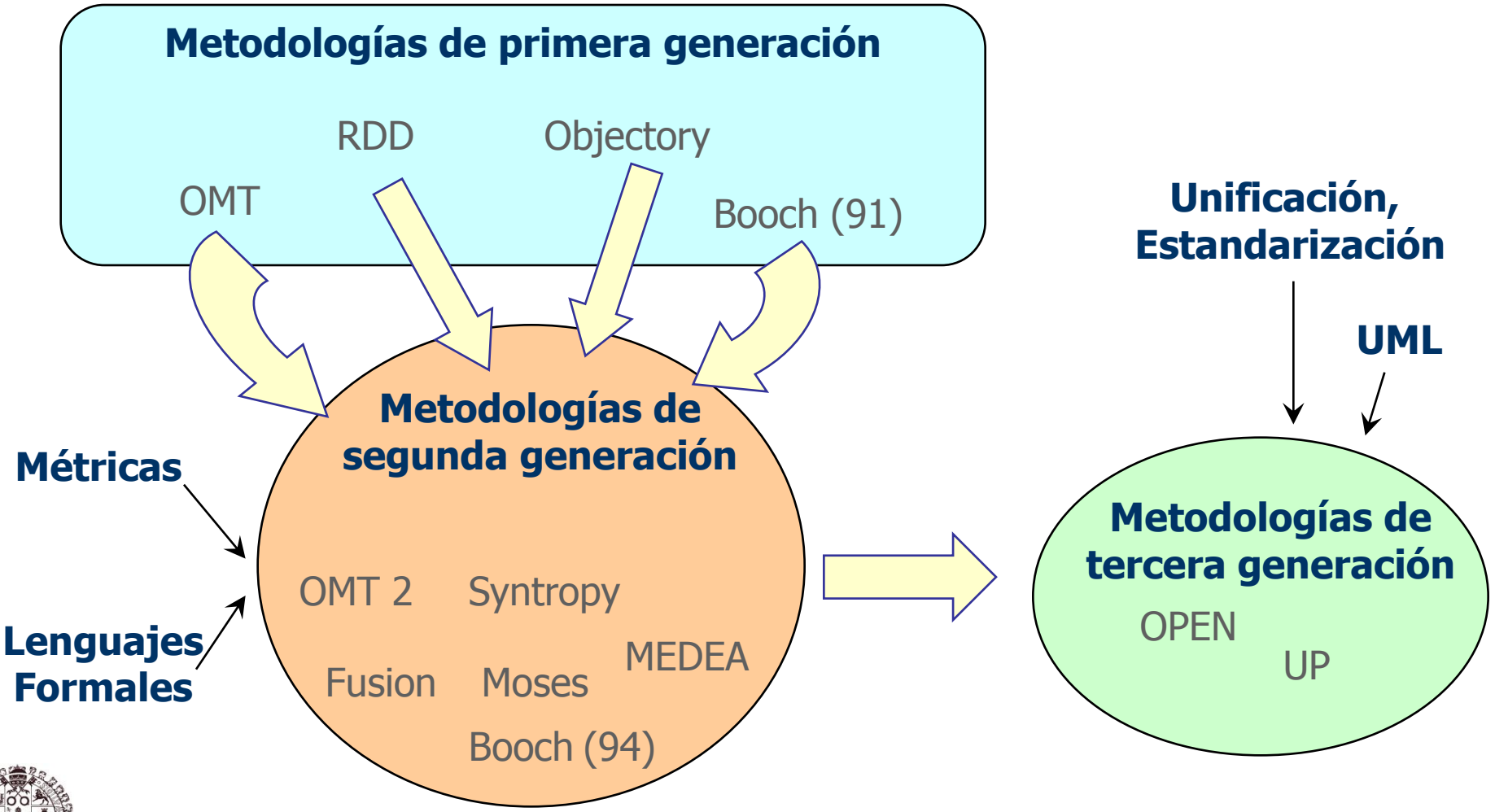
Orientadas a objetos (ii)

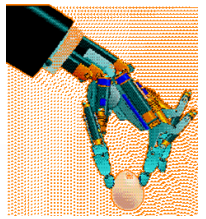
- Gran cantidad de representantes
  - Metodologías dirigidas por los datos
    - **OMT** (*Object Modeling Technique*) [Rumbaugh et al., 1991]
    - **Fusion** [Coleman et al., 1994]
  - Metodologías dirigidas por las responsabilidades
    - **RDD** (*Responsibility Driven Design*) [Wirfs-Brock et al., 1990]
    - **OBA** (*Object Behavior Analysis*) [Rubin y Goldberg, 1992]
  - Metodologías dirigidas por los casos de uso
    - **Objectory** [Jacobson et al., 1992]
    - **Proceso Unificado** [Jacobson et al., 1999]
  - Metodologías dirigidas por estados
    - **Metodología de Shlaer y Mellor** [Shlaer y Mellor, 1992]



Orientadas a objetos (iii)

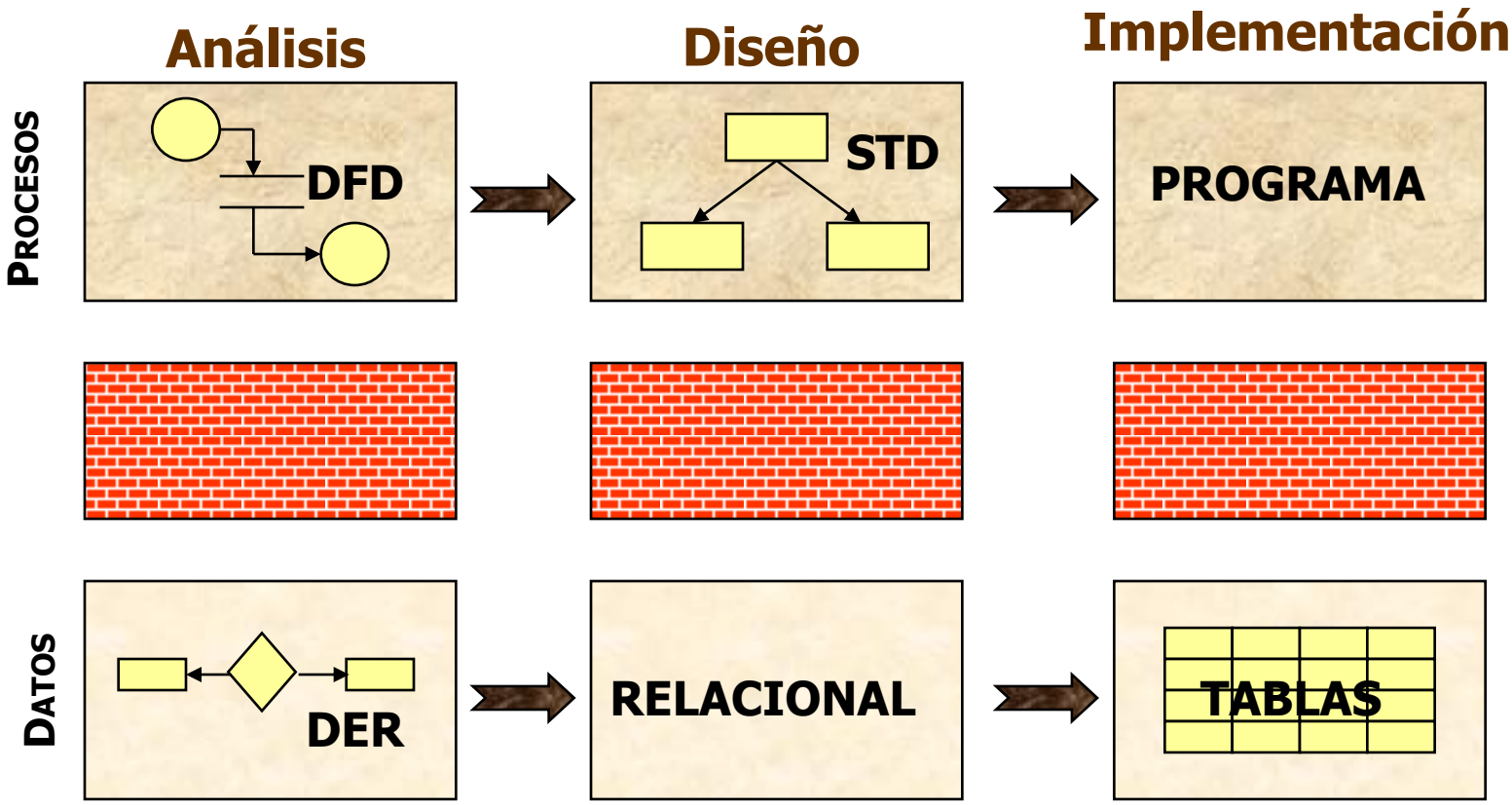
- Evolución de las metodologías OO

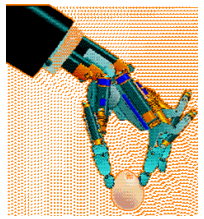




Orientadas a objetos (iv)

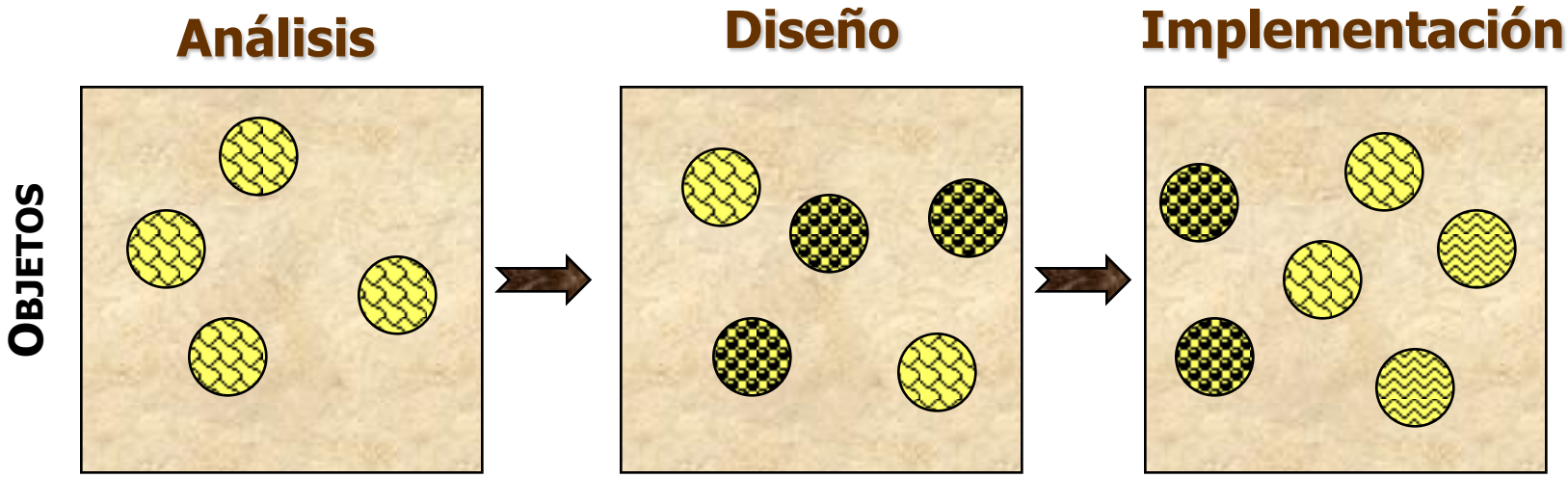
- Metodologías estructuradas vs. Metodologías OO (i)

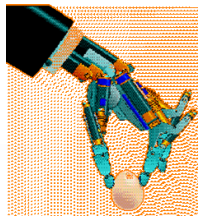




Orientadas a objetos (v)

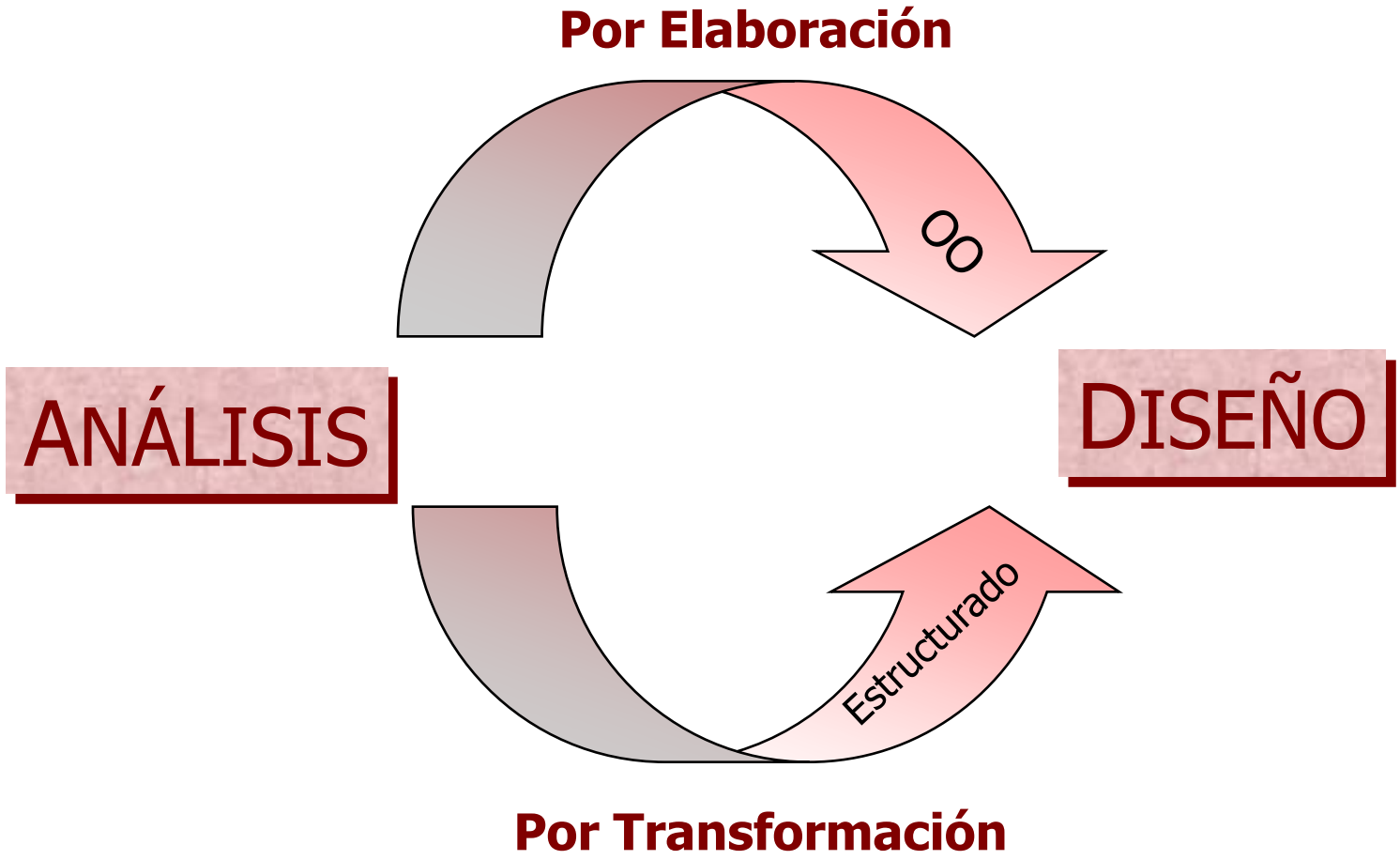
- Metodologías estructuradas vs. Metodologías OO (ii)





Orientadas a objetos (vi)

- Metodologías estructuradas vs. Metodologías OO (iii)



Orientadas al desarrollo de sistemas hipermediales (i)

- Pretenden sistematizar la creación de aplicaciones Web dentro de un proceso de creación de software bien definido
- Muchas de estas aproximaciones adolecen de tratar de forma separada los aspectos hipermediales de los meramente funcionales
  - Esto dificulta el afrontar el problema del desarrollo de aplicaciones Web dentro de un contexto uniforme
- No obstante, salvando estas soluciones parciales, se tiene ampliamente asumido que los sitios Web tradicionales están evolucionando de meros almacenes de información hipermedia a aplicaciones hipermedia distribuidas, comúnmente denominadas **aplicaciones Web** [Baresi et al., 2000]

Orientadas al desarrollo de sistemas hipermediales (ii)

- Se han definido diversas propuestas metodológicas para la construcción de aplicaciones Web
  - Proponen diferentes pasos y actividades
  - Algunas se centran sólo en el diseño o en la representación visual, mientras que otras cubren todo el proceso de desarrollo de una aplicación Web
  - Todas prescriben diferentes técnicas y notaciones
  - Algunas están soportados por herramientas
- Representantes
  - **HDM** (*Hypermedia Design Model*) [Garzotto et al., 1993]
  - **HFPM** (*Hypermedia Flexible Process Modeling*) [Olsina, 1998]
  - **OOHDM** (*Object-Oriented Hypermedia Design Method*) [Rossi, 1996]
  - **OOH-Method** [Gómez et al., 2000]
  - **OOWS** (*Object-Oriented Web-Solutions*) [Pastor et al., 2001a]
  - **WSDN** (*Web Site Design Method*) [De Troyer y Leune, 1997]

Basadas en métodos formales

- Implican una revolución en los procedimientos de desarrollo, ya que a diferencia de todas las anteriores
- Se basan en teorías matemáticas que permiten una verdadera aproximación científica y rigurosa al desarrollo de sistemas de información y software asociado
- Representantes
  - **OO-Method** [Pastor et al., 2001b]

Metodologías ágiles (i)

- Los sistemas web caracterizados, entre otras cosas, por su presión temporal priorizan el *cuándo* sobre el *qué* como sucede en las aplicaciones tradicionales

- Esta reducción del ciclo de desarrollo del software, sin perder calidad ni capacidad de evolución y de mantenimiento, es uno de los retos a los que se enfrenta la Ingeniería Web
  - A los procesos software que intentan dar solución a estas circunstancias se les conoce con el nombre de procesos ágiles o procesos ligeros

Métodos ágiles (ii)

- Las aproximaciones ágiles emplean procesos técnicos y de gestión que continuamente se adaptan y se ajustan a [Turk et al., 2002]
  - Cambios derivados de las experiencias ganadas durante el desarrollo
  - Cambios en los requisitos
  - Cambios en el entorno de desarrollo
- La agilidad en el desarrollo del software no significa únicamente poner en el mercado o en explotación los productos software más rápidamente
  - Esto choca frontalmente con los modelos de procesos tradicionales que son monolíticos y lentos, centrados en una única iteración o ciclo de larga duración
- Diversas aproximaciones
  - XP (*eXtreme Programming*) [Beck, 2000], que ha originado toda una corriente de desarrollo *extremo*

eXtreme Programming (i)

- Nuevo y controvertido enfoque de desarrollo de software basado en el modelo incremental
- Está indicado para
  - Equipos de tamaño mediano o pequeño
  - Requisitos imprecisos y cambiantes
- Características
  - El juego de la planificación
  - Versiones pequeñas
  - Metáfora
  - Diseño sencillo
  - Hacer pruebas
  - Refactorización
  - Programación en parejas
  - Propiedad colectiva
  - Integración continua
  - Cliente in-situ
  - Estándares de codificación

eXtreme Programming (ii)

- Según Beck (2000) la Programación Extrema descansa sobre cuatro valores
  - Comunicación
  - Sencillez
  - Realimentación
  - Valentía
- La Programación Extrema se diferencia de otros métodos en [Beck, 2000]
  - Su inmediata, concreta y continua realimentación de los ciclos cortos
  - Su enfoque de planificación incremental, que rápidamente plantea un plan global que se espera que evolucione a lo largo de la vida del proyecto
  - Su capacidad para programar de forma flexible la implementación de las funcionalidades, respondiendo a las necesidades cambiantes de los negocios
  - Su confianza en las pruebas automatizadas, escritas por los programadores y los clientes para controlar el progreso del desarrollo, para permitir la evolución del sistema y captar los defectos lo antes posible
  - Su confianza en la comunicación oral, las pruebas y el código fuente para comunicar la estructura e intención del sistema
  - Su confianza en el proceso de diseño evolutivo que perdura mientras perdura el sistema
  - Su confianza en la colaboración estrecha entre programadores con habilidades normales
  - Su confianza en las prácticas que funcionan tanto con los instintos a corto plazo de los programadores como con los intereses a largo plazo del proyecto

eXtreme Programming vs. Proceso Unificado (i)

- Detractores del Proceso Unificado
  - Argumentan, en favor de las aproximaciones ágiles en general y de la Programación Extrema en particular, que el Proceso Unificado es un proceso pesado que obliga a hacer gran cantidad de actividades inútiles y antinaturales
- Seguidores del Proceso Unificado
  - El Proceso Unificado es un marco de trabajo abierto y flexible con un modelo de proceso riguroso, que se puede adaptar al proceso de desarrollo que se necesite, pero sin caer en el caos al que invita la Programación Extrema per se [Kruchten, 2001]

eXtreme Programming vs. Proceso Unificado (ii)

- El Proceso Unificado puede adaptarse para incluir algunas de las prácticas de la Programación Extrema, pero no son propuestas idénticas en ningún caso
- El Proceso Unificado permite construir procesos para dar soporte a proyectos que están fuera del alcance la Programación Extrema, tanto por su escala como por su tipo
- El Proceso Unificado es un proceso pesado por la completa descripción de una familia de procesos, que pueden ser tanto ligeros como pesados, mientras que la Programación Extrema es totalmente ligera porque pone su énfasis en la implementación, implicando que las cosas hay que descubrirlas, inventarlas o definirlas ad hoc

[Smith, 2001]

6. Proceso Unificado

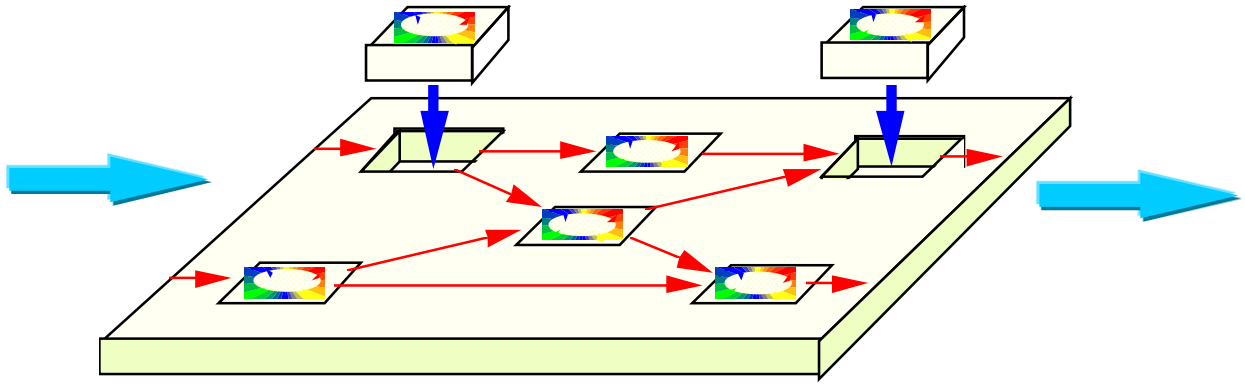
Introducción (i)

El Proceso Unificado es más que un simple proceso [Jacobson et al., 1999], es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyectos

- Características generales
  - Está basado en componentes
  - Utiliza UML [Booch et al., 1999; OMG, 2003]
- Características principales [Jacobson et al., 1999]
  - Es un proceso conducido por casos de uso
  - Está centrado en la arquitectura
  - Es iterativo e incremental

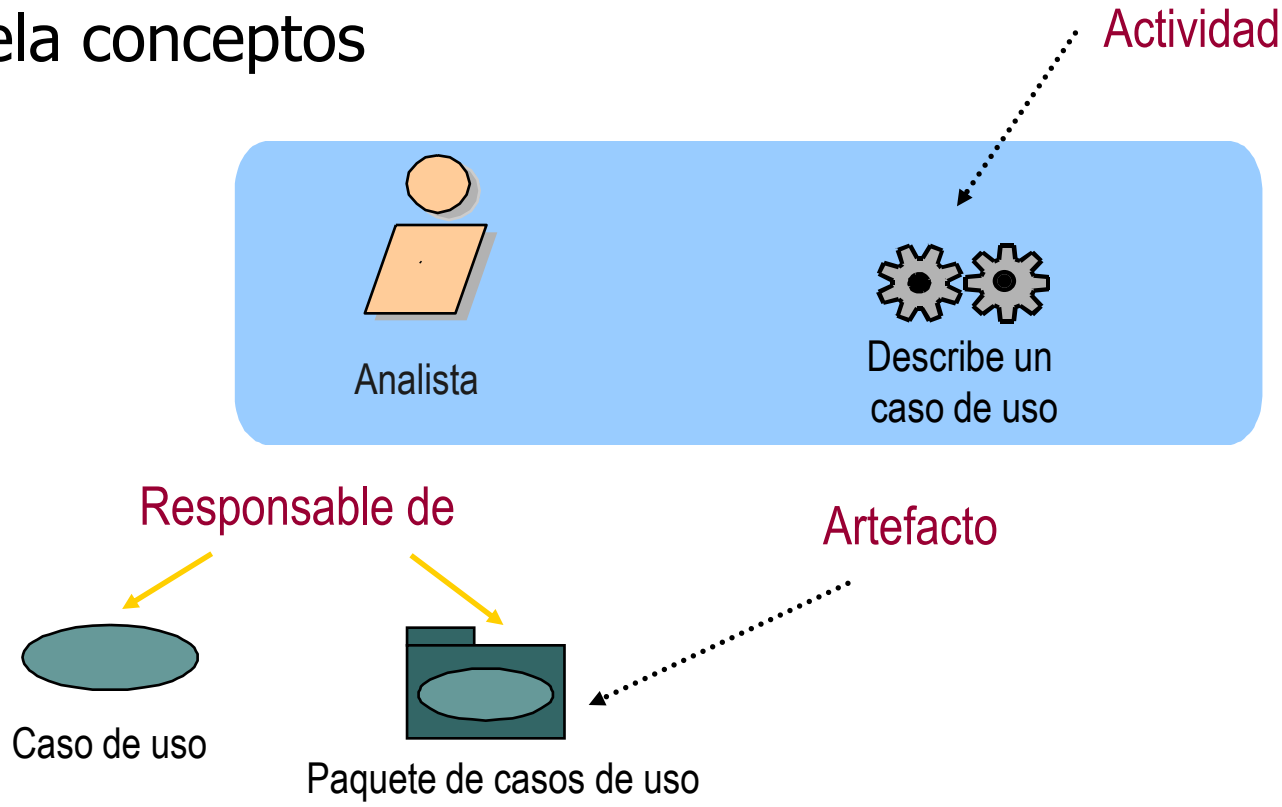
Introducción (ii)

- **Un marco de trabajo genérico**
  - No existe un proceso universal
  - Puede extenderse y especializarse para una gran variedad de sistemas de software
    - Flexibilidad
    - Está basado en componentes
  - Permite gran variedad de estrategias de ciclo de vida
    - Se pueden definir diferentes conjuntos de productos
    - Se pueden definir actividades y encargados de las mismas



Introducción (iii)

- Selecciona qué artefactos producir
- Define actividades y *stakeholders*
- Modela conceptos



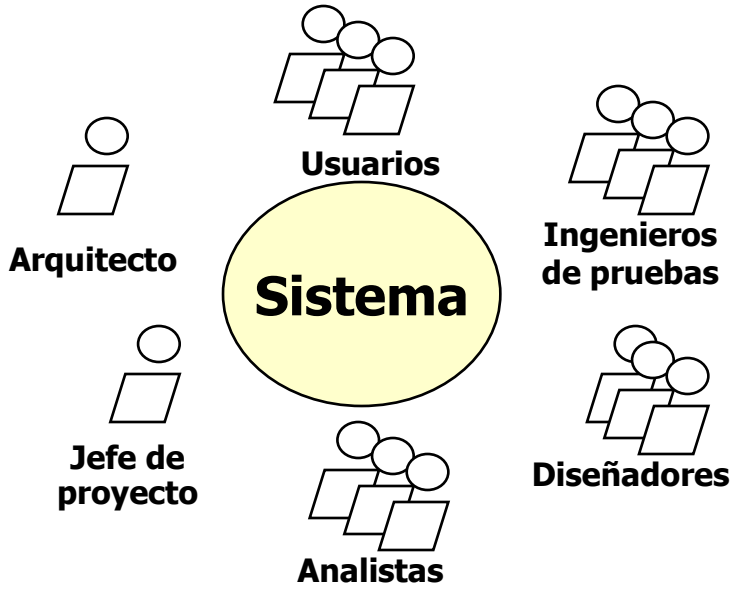
El producto (i)

- El producto que se obtiene es un **sistema de software**
- El sistema lo componen todos los "artefactos" necesarios para representarlo de forma comprensible
- **Artefacto**
  - Término general para cualquier tipo de información creada, producida, cambiada o utilizada por los *stakeholders* en el desarrollo del sistema. Puede ser
    - De ingeniería
    - De gestión
- El artefacto más importante del Proceso Unificado es el **modelo**
- Un sistema posee una colección de modelos y las relaciones entre ellos

El producto (ii)

Un modelo es una abstracción semánticamente cerrada del sistema

- Los modelos recogen diferentes perspectivas del sistema (perspectivas de todos los *stakeholders*)

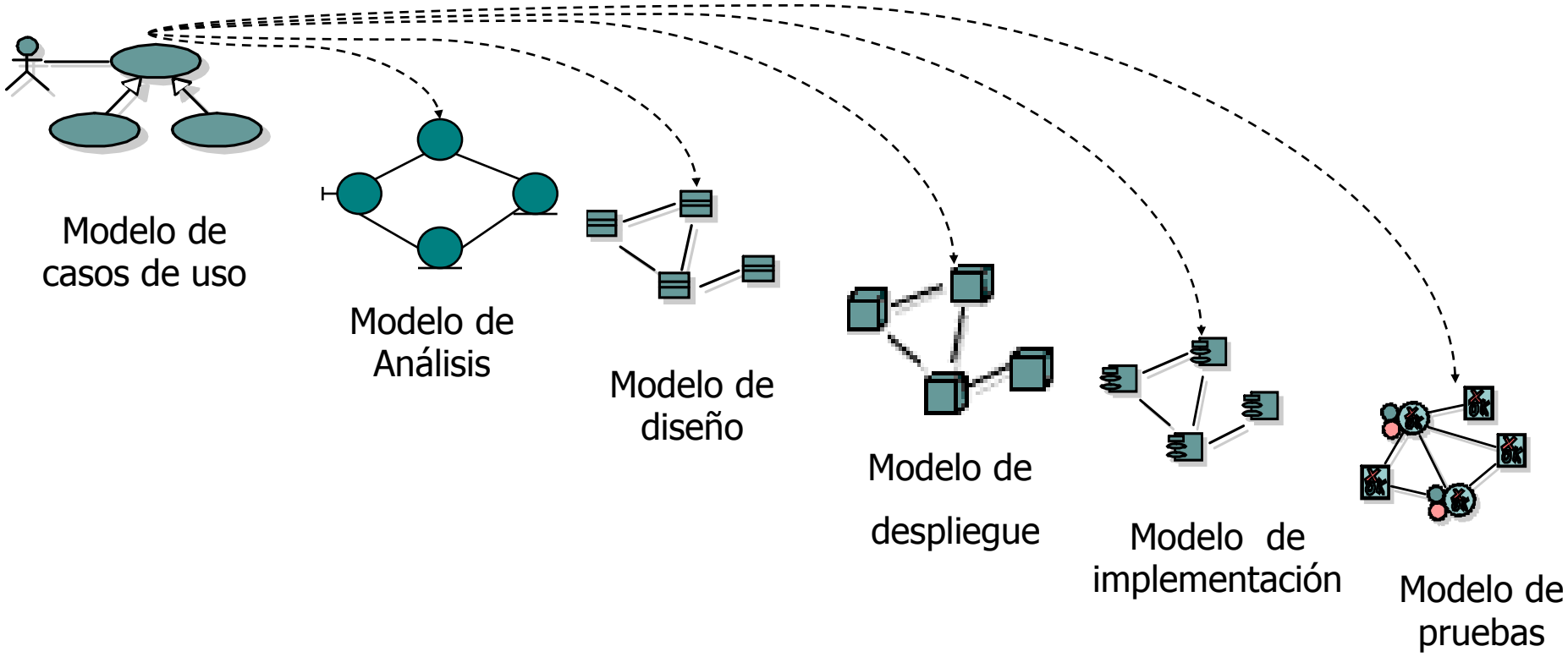


El producto (iii)

- **Modelos**
  - **Modelo de casos de uso**
    - Diagramas de casos de uso, secuencia, colaboración y actividad
  - **Modelos de análisis y diseño**
    - Diagramas de clases, objetos, secuencia, colaboración y actividad
  - **Modelo de despliegue**
    - Diagramas despliegue, secuencia y colaboración
  - **Modelo de implementación**
    - Diagramas de componentes, secuencia y colaboración
  - **Modelo de pruebas**
    - Todos los diagramas

El producto (iv)

- Existen dependencias entre el modelo de casos de uso y los demás modelos



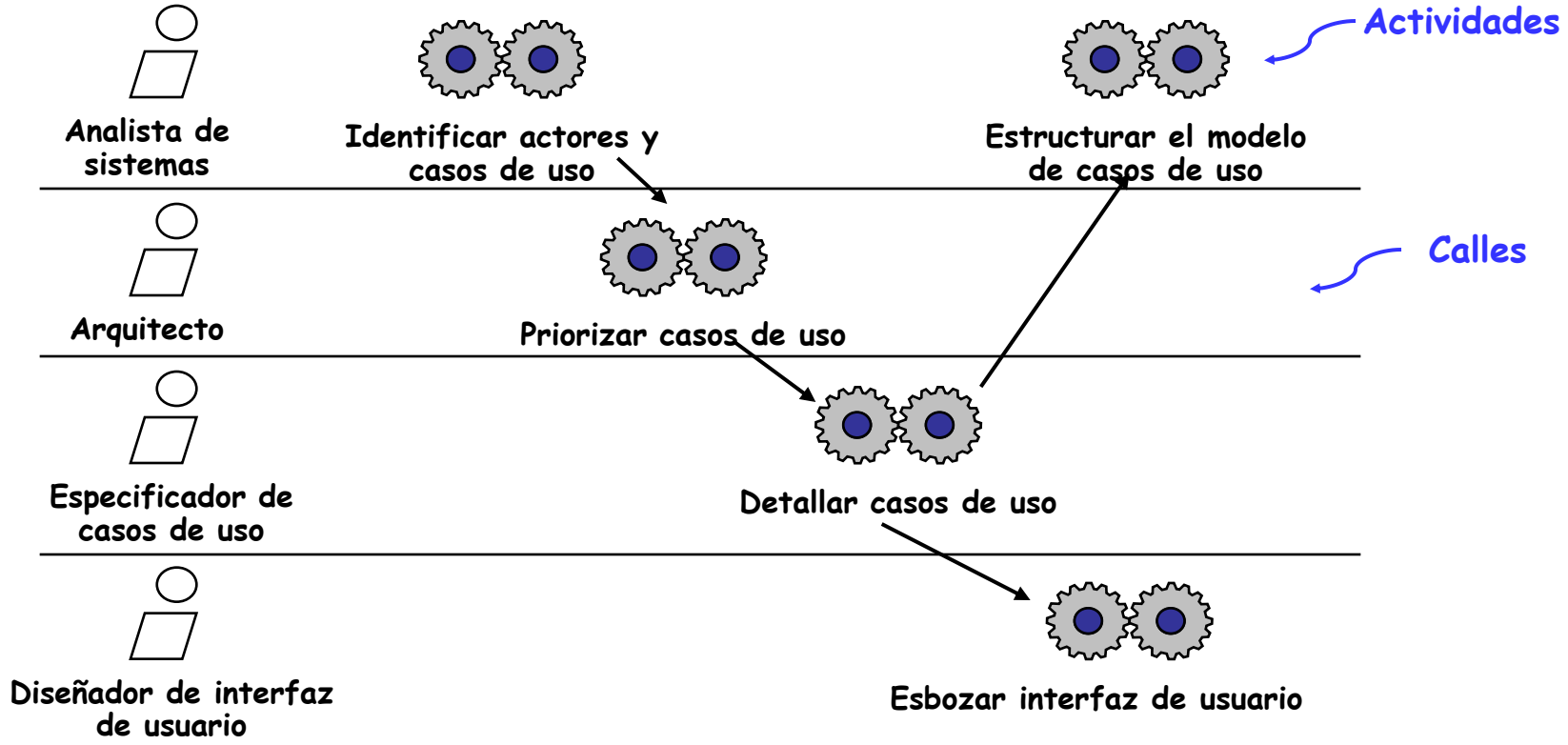
El proceso (i)

El **proceso de desarrollo de software** es una definición de un conjunto completo de actividades necesarias para convertir los requisitos de usuario en un conjunto consistente de artefactos que conforman un producto software, y para convertir los cambios sobre esos requisitos en un nuevo conjunto consistente de artefactos

- El proceso hace referencia a un contexto que sirve como plantilla que pueda reutilizarse para crear instancias de ella (proyectos)
- Las actividades relacionadas conforman **disciplinas** o **flujos de trabajo**
  - Su identificación parte de la identificación de los *stakeholders* y de los artefactos para cada tipo de *stakeholder*
  - Describen como fluye el proceso a través de los *stakeholders*

El proceso (ii)

- Representación de las disciplinas mediante flujos de trabajo



Disciplina del modelado de casos de uso

Características principales del proceso

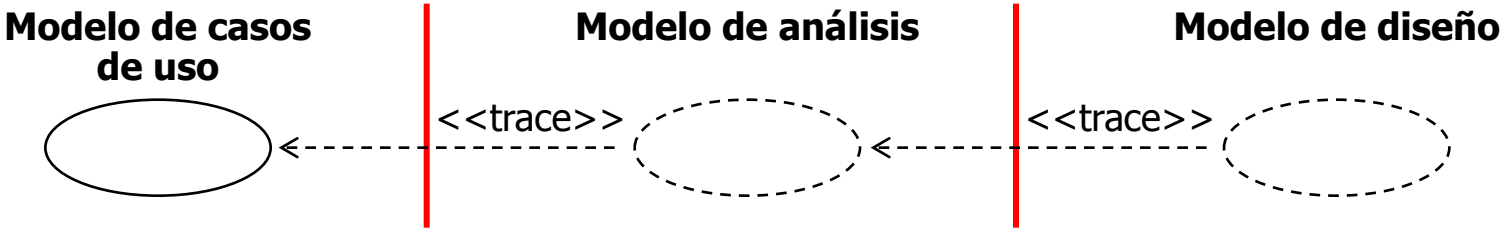
- **Conducido por casos de uso**
  - Los casos de usos guían el desarrollo del sistema
  - Como los casos de uso contienen las descripciones de las funciones, afectan a todas las fases y vistas
- **Centrado en la arquitectura**
  - La arquitectura se representa mediante vistas del modelo
  - Se puede tomar como arquitectura de referencia el denominado modelo de arquitectura de 4+1 vistas propuesto por Philippe Kruchten (1995)
- **Iterativo e Incremental**
  - En cada iteración se identifican y especifican los casos de uso relevantes, se crea un diseño basado en la arquitectura seleccionada, se implementa el diseño mediante componentes y se verifica que los componentes satisfacen los casos de uso
  - Si una iteración cumple con sus objetivos se pasa a la siguiente
  - En cada iteración se va desarrollando el sistema de forma incremental

Proceso dirigido por casos de uso (i)

- Dirigen las actividades de desarrollo
  - Creación y validación de la arquitectura del sistema
  - Definición de casos de prueba y procedimientos
  - Planificación de iteraciones
  - Creación de documentación de usuario
  - Despliegue del sistema
- Sincronizan el contenido de los diferentes modelos

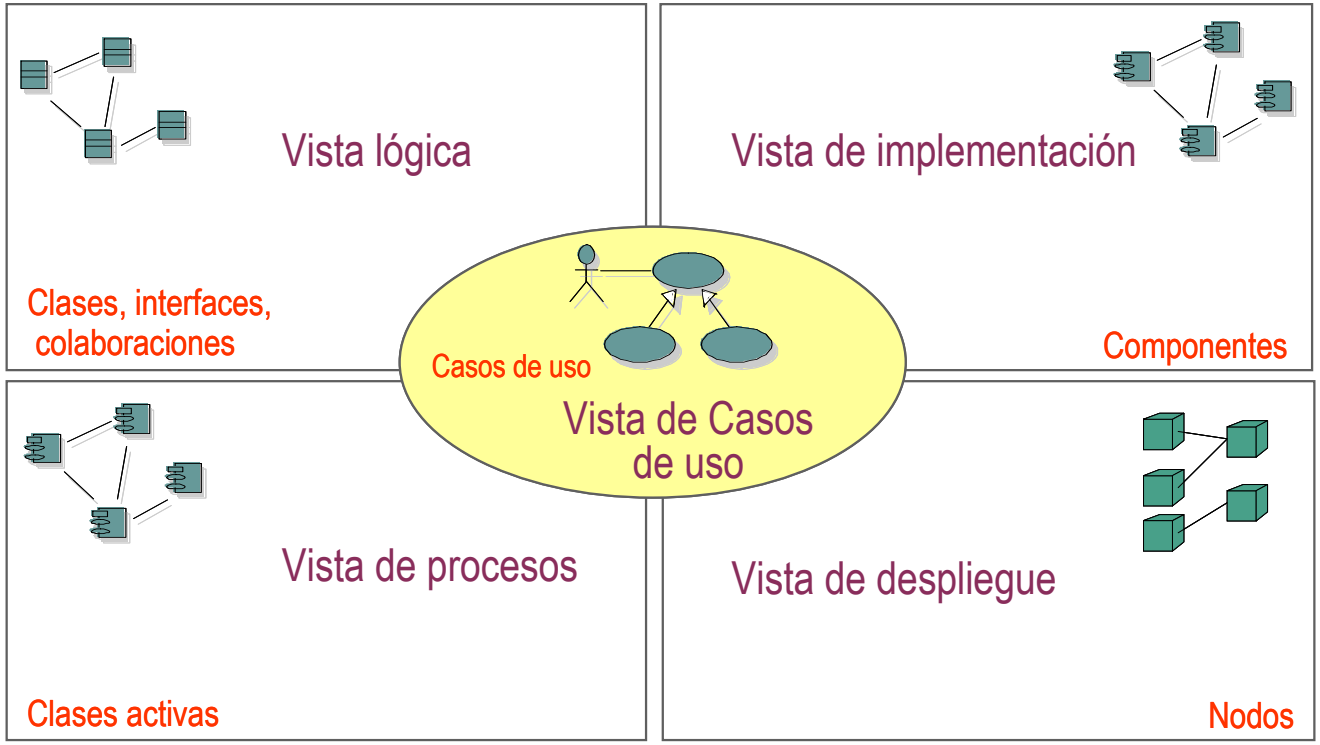
Proceso dirigido por casos de uso (ii)

- Inicialmente los casos de uso se utilizan para la captura de requisitos funcionales
- Durante el análisis y el diseño se transforma el modelo de casos de uso mediante un modelo de análisis en una estructura de clasificadores y **realizaciones de casos de uso**
- En cada iteración, los casos de uso sirven de guía a través del conjunto completo de disciplinas



Proceso centrado en la arquitectura (i)

- Se puede tomar como arquitectura de referencia el denominado modelo de arquitectura de 4+1 vistas, propuesto por Philippe Kruchten (1995)
  - Cada vista es una parte de un modelo



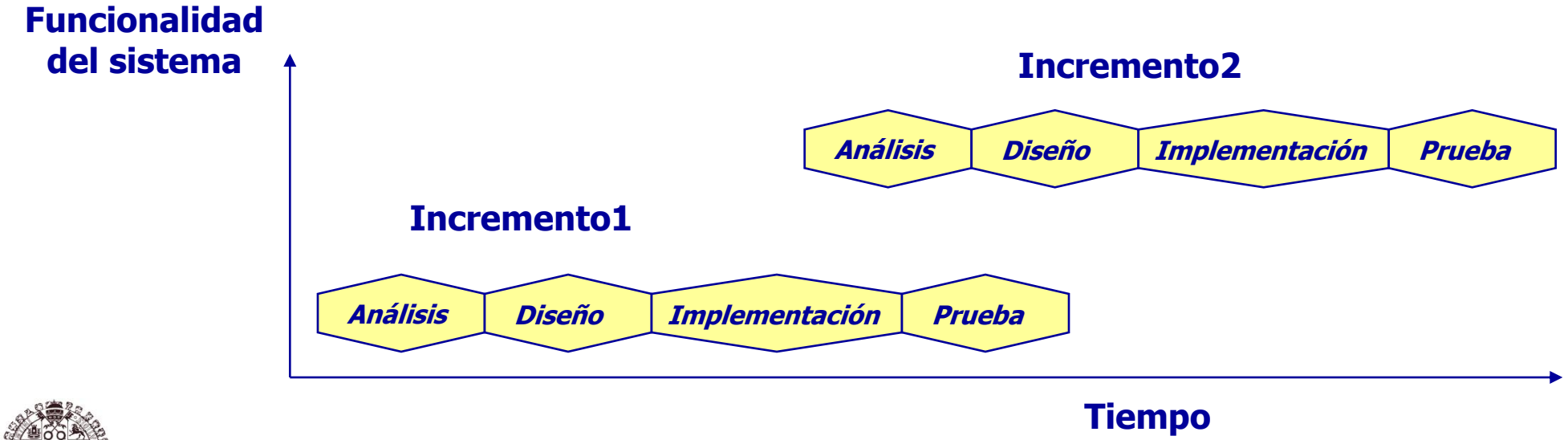
Proceso centrado en la arquitectura (ii)

- Diseño de la arquitectura
  - Seleccionar escenarios: aspectos críticos y riesgos
  - Identificar las clases principales y sus responsabilidades
  - Distribuir el comportamiento en clases
  - Estructurar en subsistemas, capas y definir interfaces
  - Definir distribución y concurrencia
  - Implementar prototipos de arquitectura
  - Derivar casos de prueba a partir de los casos de uso
  - Evaluar la arquitectura

    Iterar
- La arquitectura se desarrolla mediante iteraciones (**en capas**)
  - Comienza con una línea base de arquitectura (primera versión de los modelos)
  - La línea base evoluciona hasta convertirse en un sistema estable

Proceso iterativo e incremental (i)

- La característica fundamental del Proceso Unificado es ser un **proceso iterativo**
  - Se basa en la ampliación y el refinamiento del sistema
  - Una serie de desarrollos cortos (mini proyectos de 2 a 6 semanas, cada iteración reproduce el ciclo de vida a menor escala)
  - No sólo se mejora sino que el sistema también crece: proceso iterativo e incremental



Proceso iterativo e incremental (ii)

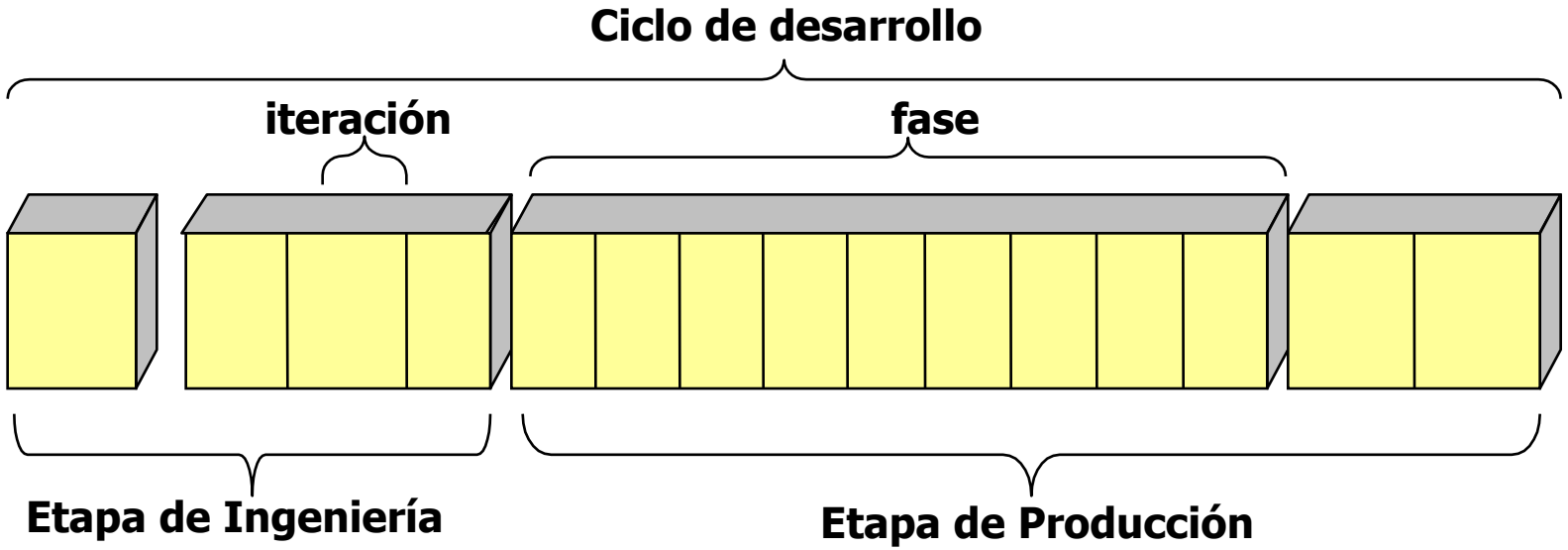
- El resultado de cada iteración es un sistema ejecutable (aunque sea incompleto y no esté listo para su instalación)
- Un sistema instalable requiere varias iteraciones
- Evolución de prototipos ejecutables
- Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes
- Concepto de "*time-boxing*"
  - Cada iteración debe tener una duración fija (el máximo, 6 meses)
    - En lugar de retrasar el final de una iteración se recomienda eliminar algunos de los requisitos (se dejan para la siguiente iteración)
- La realimentación del usuario es fundamental en este proceso
- El progreso es visible

Elementos del Proceso Unificado

- Fases
  - Es preciso diferenciar temporalmente las fases del ciclo de vida
  - La división temporal necesita puntos de control
- Puntos de control o hitos
  - Separan las etapas, las fases, las iteraciones
- Disciplinas o Flujos de trabajo
  - Organizan las actividades fundamentales de gestión y desarrollo
  - Se pueden solapar en el tiempo
  - El resultado de las actividades de los flujos de trabajo son los artefactos
- Artefactos
  - Cualquier tipo de información producida por los desarrolladores de un sistema (diagramas UML, código, ejecutables, casos de prueba...)
  - Se construyen de forma incremental

Planificación temporal del proyecto

- El Proceso Unificado propone una serie de ciclos de desarrollo
  - Hay que separar claramente la etapa de Ingeniería de la etapa de Producción
  - Cada una de las dos grandes etapas se dividen en fases
  - Las fases se dividen en iteraciones



La vida del Proceso Unificado (i)

- El Proceso Unificado se repite a lo largo de una serie de **ciclos de desarrollo** que constituyen la vida de un sistema
- Cada **ciclo de desarrollo** concluye con una **versión entregable** del producto
- Cada ciclo consta de cuatro **fases**
  - **Inicio**
    - Se define el alcance del proyecto y se desarrollan los casos de negocio
  - **Elaboración**
    - Se planifica el proyecto, se especifican en detalle la mayoría de los casos de uso y se diseña la arquitectura del sistema
  - **Construcción**
    - Se construye el producto
  - **Transición**
    - El producto se convierte en versión beta
    - Se corrigen problemas y se incorporan mejoras sugeridas en la revisión

La vida del Proceso Unificado (ii)

- Etapa de Ingeniería
  - Equipos pequeños, actividades poco predecibles (análisis, viabilidad, planificación)
  - Comprende las fases
    - Inicio
    - Elaboración
- Etapa de Producción
  - Equipos grandes, actividades predecibles, menos riesgos (programación, pruebas)
  - Comprende las fases
    - Construcción
    - Transición

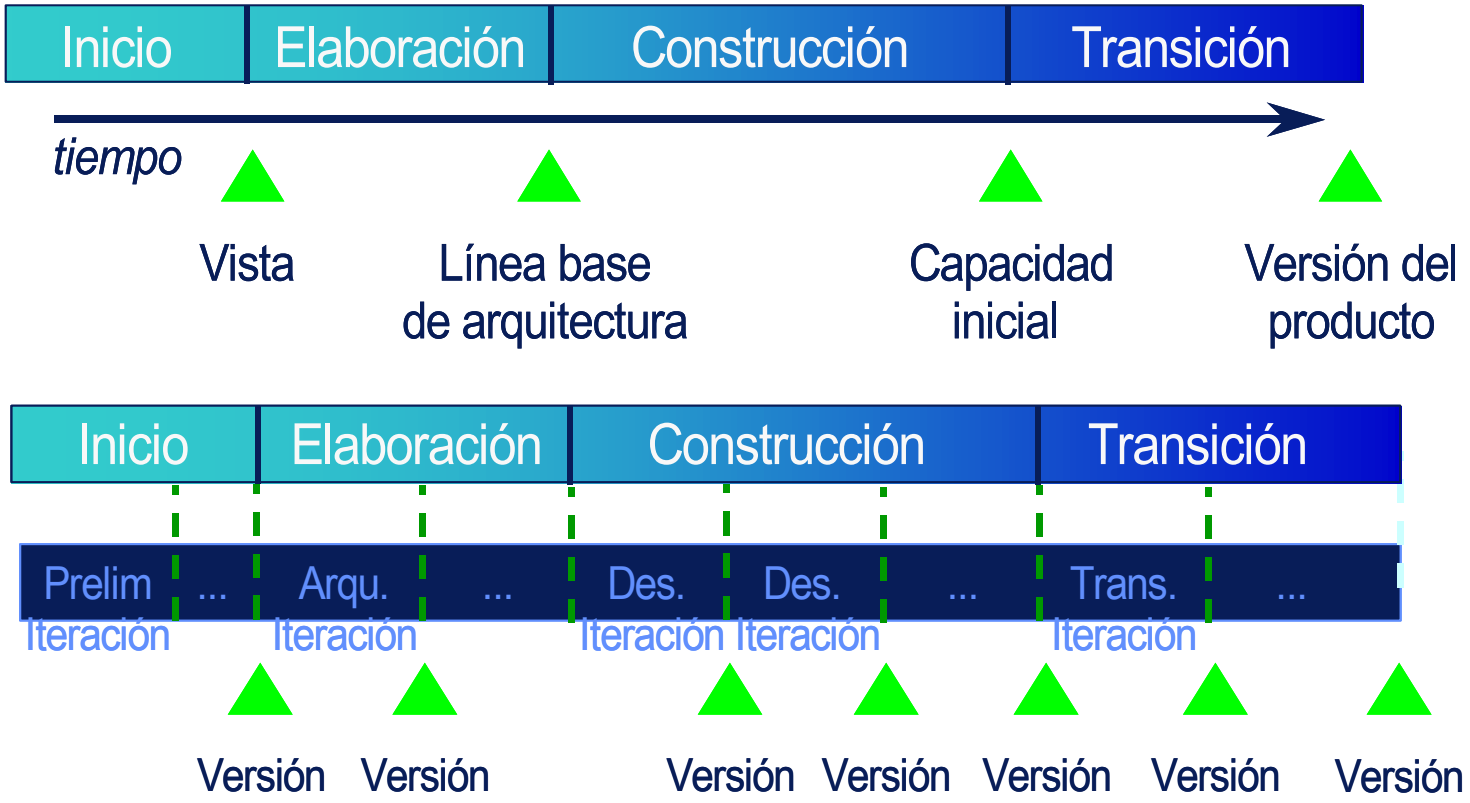
La vida del Proceso Unificado (iii)

- Hitos
  - Los hitos son puntos de control en los cuales los participantes en el proyecto revisan el progreso del proyecto
  - Se pretende
    - Sincronizar las expectativas y la realidad
    - Identificar los riesgos
    - Se evalúa la situación global del proyecto
  - Se necesitan
    - Resultados tangibles para comparar con las expectativas
  - Varios niveles
    - Hitos principales al final de cada fase
    - Hitos secundarios final de cada iteración

La vida del Proceso Unificado (iv)

- Una **iteración** es una secuencia de actividades con un plan establecido y unos criterios de evaluación, cuyo resultado es una **versión ejecutable no orientada a la entrega** (hito secundario)
- Dentro de cada fase se puede, a su vez, descomponer el trabajo en iteraciones con sus incrementos resultantes
- Cada fase termina con un hito, cada uno de los cuales se caracteriza por la disponibilidad de un conjunto de componentes de software
- Objetivos de los hitos
  - Toma de decisiones para continuar con la siguiente fase
  - Controlar el progreso del proyecto
  - Proporcionar información para la estimación de tiempo y recursos de proyectos sucesivos
- Las iteraciones discurren a lo largo de las disciplinas

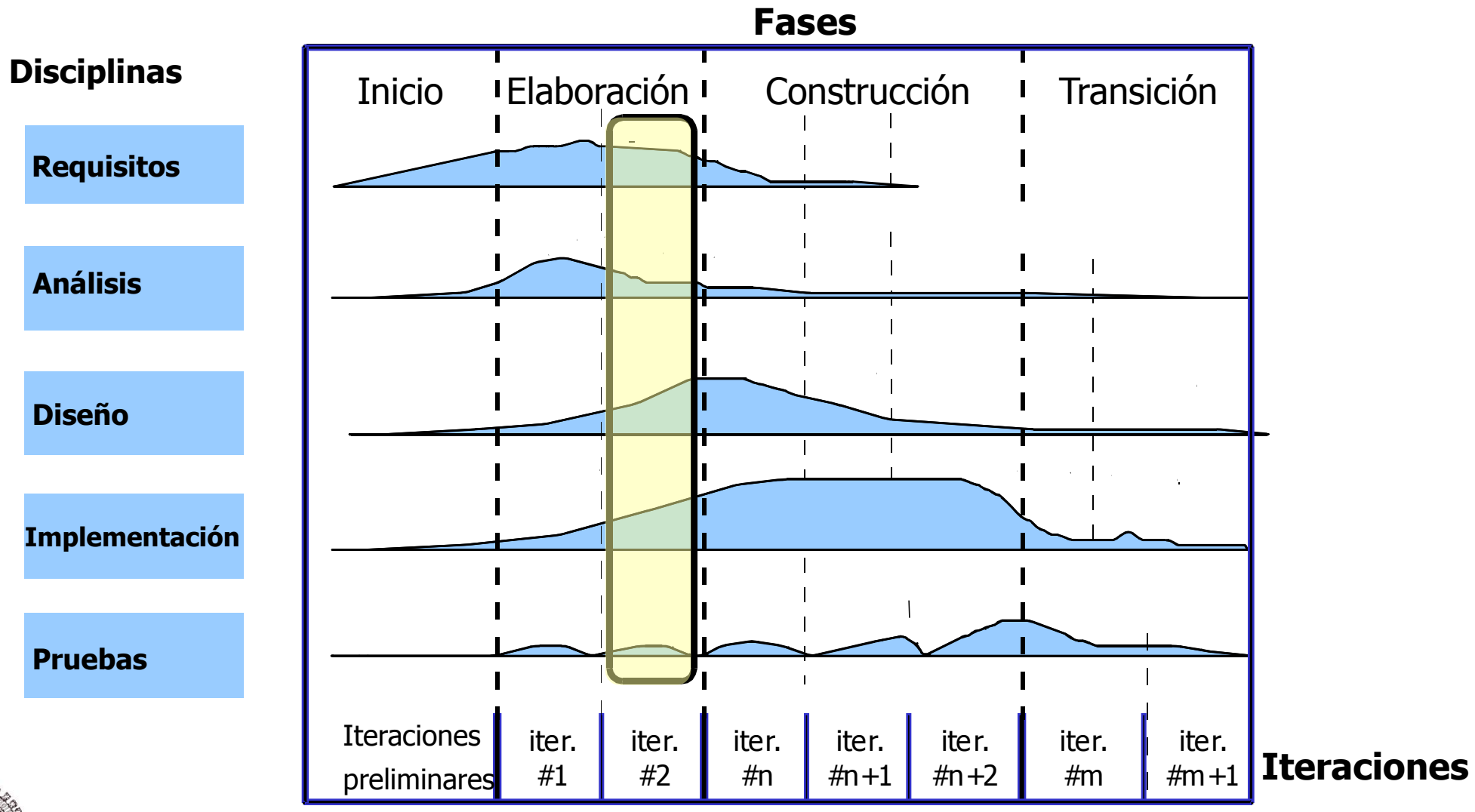
La vida del Proceso Unificado (v)



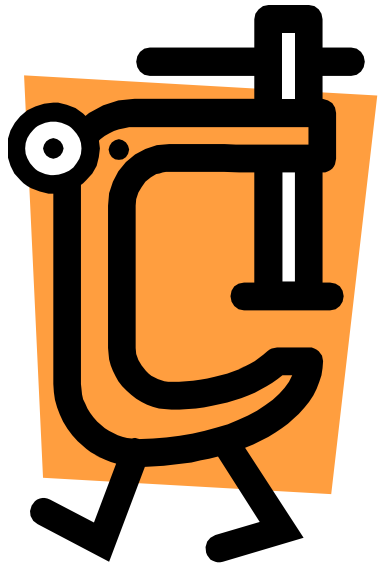
La vida del Proceso Unificado (vi)

- Las disciplinas o flujos de trabajo organizan las actividades fundamentales de gestión y desarrollo del proyecto
  - **Disciplinas de desarrollo**
    - Requisitos, análisis, diseño, implementación, pruebas...
  - **Disciplinas de gestión o soporte**
    - Gestión de proyecto, gestión de configuraciones, entorno, evaluación...
- Al contrario de lo que ocurre con las fases, las distintas actividades del equipo de desarrollo se pueden solapar en el tiempo

La vida del Proceso Unificado (vii)



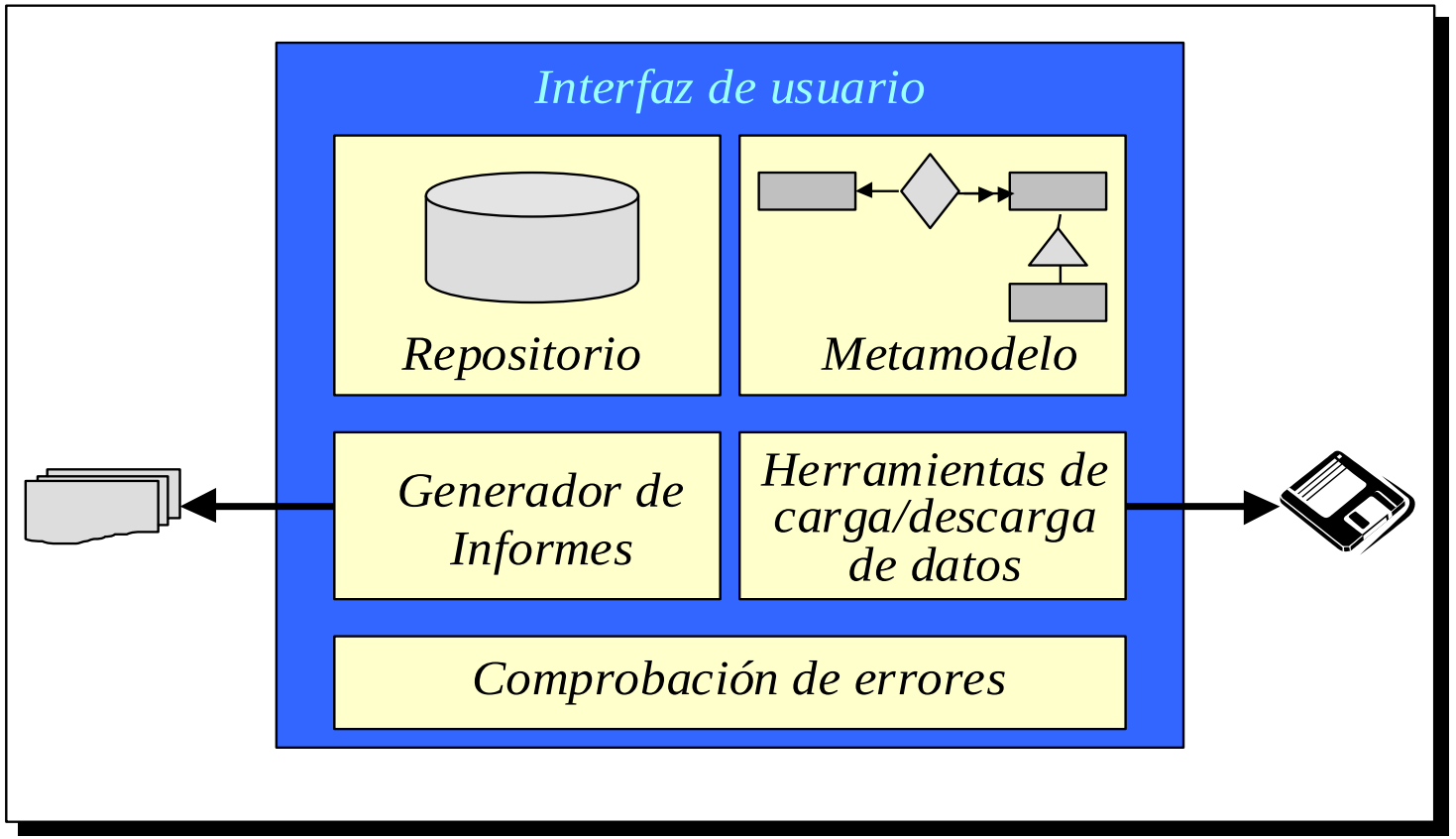
7. CASE



Introducción

- Tecnología CASE (*Computer Aided/Assisted Software/System Engineering*)
  - Es el nombre que se le da al software que se utiliza para dar soporte a las actividades del proceso software, tales como la ingeniería de requisitos, diseño, desarrollo y prueba [Sommerville, 2005]
  - Ofrece soporte al proceso software automatizando alguna de sus actividades, a la vez que da información sobre el software que está siendo desarrollado
- Algunas de las herramientas CASE son soluciones puntuales
  - Se utilizan para prestar apoyo a en una actividad del proceso software concreta, pero no se comunica directamente con otras
- Otras herramientas forman un entorno integrado I-CASE (*Integrated CASE*)

Componentes de una herramienta CASE

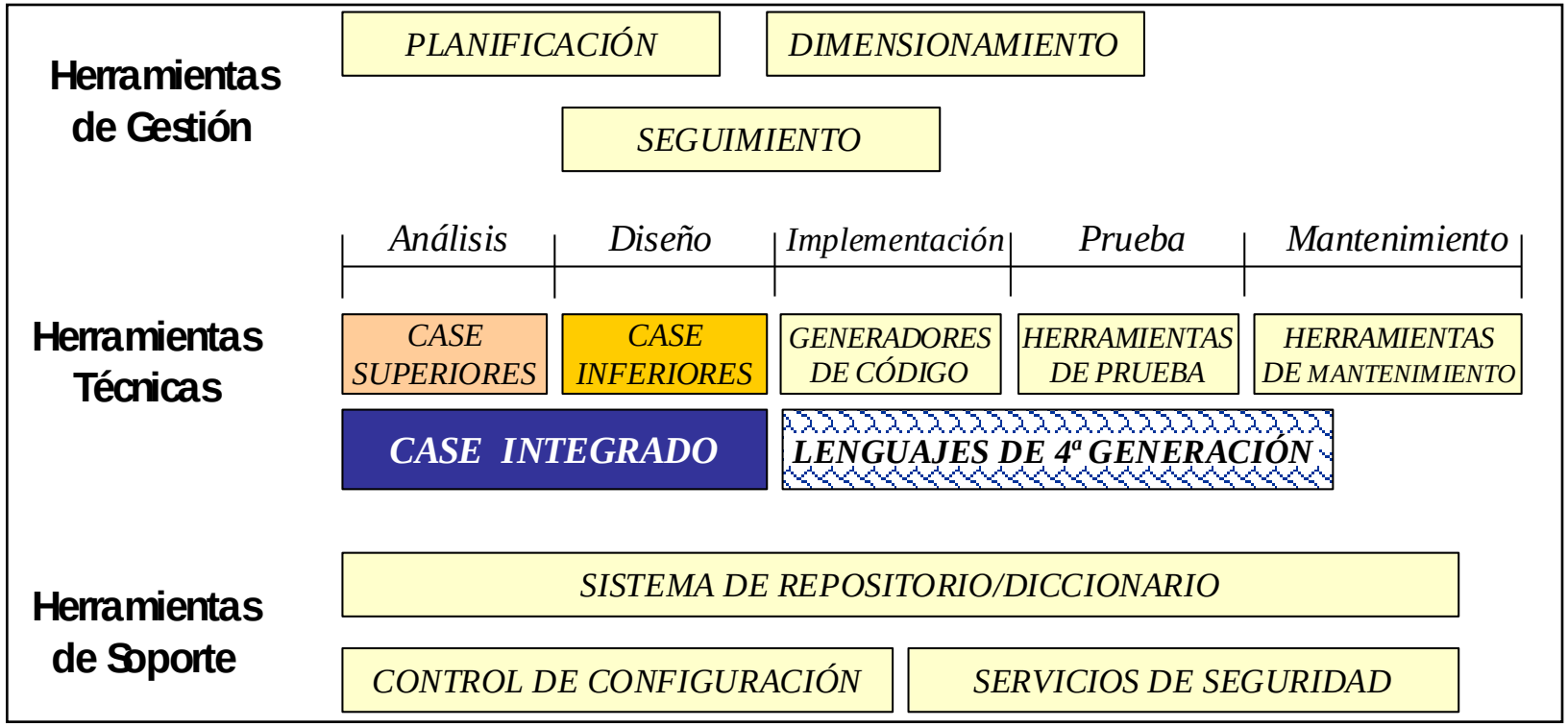


[Piattini et al., 2004]

Clasificación de las herramientas CASE (i)

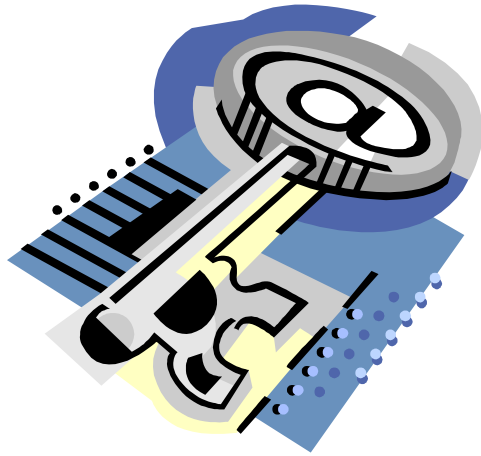
- Sommerville (2005) establece que se pueden realizar diferentes clasificaciones en función de diversas perspectivas, distinguiéndose
  - Perspectiva funcional
    - Se clasifican de acuerdo a su función específica
      - Herramientas de planificación, Herramientas de soporte de métodos, Herramientas de documentación…
  - Perspectiva de proceso
    - Se clasifican de acuerdo a las actividades del proceso software que soportan
  - Perspectiva de integración
    - Se clasifican de acuerdo a cómo se organizan en unidades de integración que ofrecen soporte a una o más actividades del proceso software

Clasificación de las herramientas CASE (ii)



[Piattini et al., 2004]

- Existen otras muchas clasificaciones de herramientas CASE, como por ejemplo [McClure, 1992], [Fuggetta, 1993] o [Pressman, 2002]



8. Aportaciones principales del tema

Aportaciones principales (i)

- El software es un producto con gran peso en la economía
- El desarrollo de software es una actividad compleja
- El software no se fabrica en el sentido clásico
- Un enfoque no de ingeniería provoca una profunda crisis en el sector del software
- Los objetivos de la Ingeniería del Software son desarrollar software de calidad, con mayor productividad, en menor tiempo y de forma lo más económica posible
- Resulta necesario establecer un enfoque sistemático y disciplinado para llevar a cabo un desarrollo software

Aportaciones principales (ii)

- Los procesos software son las actividades que intervienen en la producción de un sistema software
- Los modelos de proceso son representaciones abstractas de los procesos software
- Todos los modelos de proceso incluyen especificación de requisitos, diseño e implementación, validación y evolución
- La linealidad en el desarrollo del software no es realista. Normalmente se necesitan diversas iteraciones

9. Cuestiones y ejercicios

Cuestiones y ejercicios

- Buscar, documentar y comentar dos situaciones problemáticas recientes cuyas causas se puedan achacar al software
- Seleccionar diferentes aplicaciones software e indicar la categoría de aplicación software
- Realizar una comparativa entre el ciclo de vida clásico, el ciclo de vida en espiral y el ciclo de vida del Proceso Unificado
- Muchas organizaciones compran software comercial, pensando que es más barato que desarrollar y mantener software propio. Discutir las ventajas y desventajas de la utilización de paquetes comerciales
- Documentarse sobre las denominadas metodologías ágiles y realizar una comparación razonada con las denominadas metodologías pesadas, tipo Proceso Unificado
- Considerar los modelos de procesos presentados, ¿cuáles permiten una mayor capacidad de reacción ante requisitos cambiantes?
- Evaluar diferentes herramientas CASE, especialmente aquéllas que den un soporte a las técnicas usadas en las fases de análisis y diseño

10. Lecturas complementarias

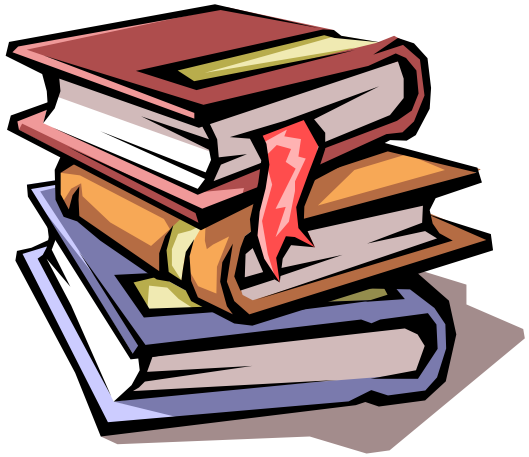
Lecturas complementarias (i)

- **Boehm, B. W., Egyed, A., Kwan, J., Port, D., Shah, A., Madachy, R.** "*Using the WinWin Spiral Model: A Case Study*". IEEE Computer, 31(7):33-44, July 1998
  - En este artículo se presenta la aplicación práctica del modelo de ciclo de vida en espiral WinWin, una extensión del ciclo de vida definido por Boehm, al que se le ha añadido las actividades de la *Teoría W* al comienzo de cada ciclo
- **Chandra, J., March, S. T., Mukherjee, S., Pape, W., Ramesh, R., Rao, H. R., Waddoups, R. O.** "*Information Systems Frontiers*". Communications of the ACM, 43(1):71-79. January 2000
  - Artículo que trata sobre el crecimiento y aplicación de los sistemas de información, junto a las Tecnologías de la Información, en dominios de aplicación que nunca habían sido considerados
- **Fayad, M. E., Laitinen, M., Ward, R. P.** "*Software Engineering in the Small*". Communications of the ACM, 43(3):115-118. March 2000
  - Artículo donde se defiende que las compañías que desarrollan proyectos software de pequeño tamaño también deben utilizar técnicas de Ingeniería del Software
- **Fuggetta, A.** "*A Classification of CASE Technology*". IEEE Computer, 26(12):25-38. December 1993
  - Artículo que realiza un informe clasificatorio sobre la tecnología CASE
- **Gage, D.** "*Consumer Products: When Software Bugs Bite*". Baseline. http://www.baselinemag.com/article2/0,3959,833424,00.asp [Última vez visitado, 20-9-2006]. January 2003
  - Pone de manifiesto el desprestigio de la industria del software
- **Glass, R. L.** "*The Software Criris... Not?*". IEEE Computer, 27(4):104. April 1994
  - Opinión personal del autor del artículo sobre la crisis del software
- **Gutiérrez, I., Medinilla, N.** "*Contra el Arraigo de la Cascada*". En las Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISBD'99. P. Botella, J. Hernández, F. Saltor (Eds.). (24-26 de noviembre de 1999, Cáceres - España). Páginas 393-404. 1999
  - Trabajo crítico con el modelo de ciclo de vida en cascada, realizado desde la perspectiva de la complejidad de la incertidumbre en los proyectos software

Lecturas complementarias (ii)

- **Henderson-Sellers, B., Edwards, J. M**. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September 1990
  - En este artículo se describe el modelo de ciclo de vida fuente para desarrollos orientados a objetos
- **Jacobson, I**. "*Building Without Blueprints*". Object Magazine, 7(9): 71-72. November 1997
  - Artículo que resalta la importancia de los modelos software
- **Lions, J. L.** "*ARIANE 5 Flight 501 Failure*". Report by the Inquiry Board. http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html. [Última vez visitado, 20-9-2006]. París, 19 Julio 1996
  - Informe de las causas del fallo del lanzamiento de la lanzadera espacial Ariane 5 el 4 de Junio de 1996
- **Ministerio de Obras Públicas**. "MÉTRICA. Versión 3". http://www.csi.map.es/csi/metrica3/index.html. [Última vez visitado, 19-9-2006]. 2001
  - Documentación en línea sobre la metodología Métrica v3
- **Raccoon, L. B. S.** "*Fifty Years of Progress in Software Engineering*". Software Engineering Notes (SEN), 22(1):88-104. January 1997
  - Presenta una visión de la evolución de la Ingeniería del Software. Presenta también el modelo de ciclo de vida **Caos**, comparándolo con otros modelos
- **Risk Forum.** http://catless.ncl.ac.uk/Risks. [Última vez visitado, 20-9-2006]
  - Foro donde se describen diversas situaciones problemáticas causadas por fallos informáticos
- **Sharma, S., Rai, A.** "*CASE Deployment in IS Organizations*". Communications of the ACM, 43(1):80-88. January 2000
  - Informe de la presencia de la tecnología CASE en los sistemas de información de las empresas
- **Singh, R.** "*The Software Life Cycle Processes Standard*". IEEE Computer, 28(11):89-90. November 1995
  - Visión esquemática del estándar ISO/IEC 12207 sobre los procesos que componen el ciclo de vida del software
- **Yourdon, E.** "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana. 1993
  - En el capítulo 5 describe el ciclo de vida estructurado

11. Referencias



Referencias (i)

[AECC, 1986] Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986

[Ashworth y Goodland, 1990] Ashworth, C., Goodland, M. "*SSADM: A Practical Approach*". McGraw-Hill, 1990

[Balzer, 1981] Balzer, R. "*Transformational Implementation: An Example*". IEEE Transactions on Software Engineering, SE7(1): 3-14, 1981

[Baresi et al., 2000] Baresi, L., Garzotto, F., Paolini, P. "*From Web Sites to Web Applications: New Issues for Conceptual Modeling*". En Proceedings of ER'2000 Workshop on Conceptual Modeling and the Web. Series: Lecture Notes in Computer Science. Vol. LNCS 1921, pp. 89-100. Springer-Verlag, 2000

[Bauer, 1972] Bauer, F. L. "*Software Engineering Information Processing*". Amsterdam: North Holland, 1972

[Basili y Turner, 1975] Basili, V., Turner, A. "*Iterative Enhancement: A practical Technique for Software Development*". IEEE Transactions on Software Engineering, 1(4):390-396, 1975

[Beck, 2000] Beck, K. "*Extreme Programming Explained. Embrace Change*". Addison-Wesley, 2000

[Blum, 1992] Blum, B. I. "*Software Engineering, A Holistic View*", Oxford University Press, New York, p. 20, 1992

[Boehm, 1986] Boehm, B. W. "*A Spiral Model of Software Development and Enhancement*". ACM Software Engineering Notes, 11(4):22-42, 1986

[Booch et al., 1999] Booch, G., Rumbaugh, J., Jacobson, I. "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999

[Brooks, 1987] Brooks, F. P. Jr. "*No Silver Bullet. Essence and Accidents of Software Engineering*". IEEE Computer, April 1987

Referencias (ii)

[CERN, 1996] CERN. "*STING Software Engineering Glossary*". CERN. http://www.apl.jhu.edu/Classes/Notes/Hausler/web/glossary.html. September 1996 [Última vez visitado 26-9-2007]

[Clark, 2000] Clark, G. "*Fatal Error: Buggy Software May Have Crashed Mars Polar Lander*". http://www.space.com/businesstechnology/technology/mpl_software_crash_000331.html. [Última vez visitado 26-9-2007]

[Coleman et al., 1994] Coleman, D., Arnold, P., Bodoff, S. "*Object-Oriented Development: The Fusion Method*". Prentice-Hall, 1994

[Curtis et al., 1987] Curtis, B., Krasner, H., Shen, V., Iscoe, N. "*On Building Software Process Models under the Lamppost*". En Proceedings of the 9th International Conference on Software Engineering, pp. 96-103, IEEE CS Press, 1987

[Davis, 1982] Davis, A. M. "*Rapid Prototyping Using Executable Requirements Specifications*". ACM Software Engineering Notes, 7(5):39-44, 1982

[Davis, 1992] Davis, A. M. "*Operational Prototyping: A new Development Approach*". IEEE Software, 9(5):70-78, 1992

[Davis, 1993] Davis, A. M. "*Software Requirements. Objects, Functions and States*". Prentice-Hall International, 1993

[De Troyer y Leune, 1997] De Troyer, O., Leune, C. "*WSDN: A User-Centered Design Method for Web Sites*". En Proceedings of the 7th International World Wide Web Conference WWW6. Elsevier, pp. 85-94. 1997

[DoD, 1995] DOD. "*SRI Reuse Glossary*". DOD, December 1995

[Dyer, 1992] Dyer, M. "*The Cleanroom Approach to Quality Software Development*". Wiley, 1992

Referencias (iii)

[Finkelstein y Dowell, 1996] Finkelstein, A., Dowell, J. "*A Comedy of Errors: The London Ambulance Service Case Study*". En Proceedings of the 8th International Workshop on Software Specification & Design IWSSD-8, pp. 2-4, IEEE CS Press, 1996

[Frakes et al., 1991] Frakes, W. B., Fox, C., Nejmeh, B. A. "*Software Engineering in the UNIX/C Environment*". Prentice Hall, 1991

[Fuggetta, 1993] Fuggetta, A. "*A Classification of CASE Technology*". IEEE Computer, 26(12):25-38. December 1993

[Garvin, 1984] Garvin, D. "*What Does 'Product Quality' Really Mean?*". Sloan Management Review. 25-45. Autumn 1984

[Garzotto et al., 1993] Garzotto, F., Paolini, P., Schwabe, D. "*HDM – A Model-Based Approach to Hypermedia Application Design*". ACM Transactions on Information Systems, 11(1), 1-26, 1993

[Gilb, 1988] Gilb, T. "*Principles of Software Engineering Management*". Addison-Wesley, 1988

[Ginige y Murugesan, 2001] Ginige, A., Murugesan, S. "*Web Engineering-An Introduction*". IEEE Multimedia, 8(1):14-18. 2001

[Gómez et al., 1998] Gómez, A., Juristo, N., Montes, C., Pazos, J. "*Ingeniería del Conocimiento*". Madrid. Ceura, 1998

[Gómez et al., 2000] Gómez, J., Cachero, C., Pastor, O. "*Extending a Conceptual Modeling Approach to Web Application Design*". En Proceedings of Conference on Advanced Information Systems Engineering (CAiSE). Series: Lecture Notes in Computer Science. Vol. LNCS 1789, pp. 79-93. Springer Verlag, 2000

[GMD, 1992] German Ministry of Defense. "*V-Model: Software Lifecycle Process Model*". Bundesminister des Innern, Koordinierungs-und Beratungstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung, 1992

Referencias (iv)

[Ghezzi et al., 1991] Ghezzi, C., Jazayeri, M., Mandrioli, D. "*Fundamentals of Software Engineering*". Prentice-Hall International Inc. 1991

[Hatley y Pirbhai, 1987] Hatley D. J., Pirbhai, A. "*Strategies for Real-Time System Specification*". Dorset House, 1987

[Henderson-Sellers y Firesmith, 1999] Henderson-Sellers, B., Firesmith, D. "*Comparing OPEN and UML: The Two Third-Generation OO Development Approaches*". Information and Software Technology, 41:139–156, 1999

[Horan, 1995] Horan, P. "*Software Engineering - A Field Guide*". Deakin University. http://www3.it.deakin.edu.au/~peter/SEweb/field_gu.html**.** December 1995 [Última vez visitado 26-9-2007]

[Humphrey, 1989] Humphrey, W. S. "*Managing the Software Process*". Addison-Wesley, 1989

[IEEE, 1999a] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 1: Customer and Terminology Standards*". IEEE Computer Society Press, 1999

[IEEE, 1999b] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 2: Process Standards*". IEEE Computer Society Press, 1999

[ISO/IEC, 1995] ISO/IEC. "*Information Technology – Software Life Cycle Processes*". Technical ISO/IEC 12207:1995(E), 1995

[Jackson, 1975] Jackson, M. A. "*Principles of Program Design*". Academic Press, 1975

[Jackson, 1983] Jackson, M. A. "*System Development*". Prentice-Hall, 1983

[Jacobson et al., 1992] Jacobson, I., Christerson, M., Jonsson, P., Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992

Referencias (v)

[Jacobson et al., 1999] Jacobson, I., Booch, G., Rumbaugh, J. "*The Unified Software Development Process*". Object Technology Series. Addison-Wesley, 1999

[Karlsson, 1995] Karlsson, E.-A. (Ed). "*Software Reuse. A Holistic Approach*". Wiley Series in Software Based Systems. John Wiley and Sons Ltd., 1995

[Kruchten, 1995] Kruchten, P. "*The 4+1 View Model of Architecture*". IEEE Software, 12(6):42-50. November 1995

[Kruchten, 2001] Kruchten, P. "*Agility with the RUP*". Cutter IT Journal, 14(12):27-33. 2001

[Leaney, 2004] Leaney, J. "*Software Engineering - An Introductory Tutorial*". University of Technology, Sydney. http://services.eng.uts.edu.au/~jrleaney/SEWeb/index.html. July 2004 [Última vez visitado 26-9-2006]

[Leveson, 1995] Leveson, N. G. "*Safeware: System Safety and Computers*". Addison-Wesley, 1995

[Leveson y Turner, 1993] Leveson, N. G., Turner, C. S. "*An Investigation of the Therac-25 Accidents*". IEEE Computer, 26(7):18-41. July 1997

[Maddison, 1983] Maddison, R. N. "*Information System Methodologies*". Wiley Henden, 1983

[MAP, 1995] MAP – Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995

[MAP, 2001] MAP – Ministerio de Administraciones Públicas. "*MÉTRICA 3.0, Volúmenes 1-3*". Ministerio de Administraciones Públicas, 2001

[McClure, 1992] Mc Clure, C. "*CASE: La Automatización del Software*". Ra-ma, 1992

[McCracken y Jackson, 1981] McCracken, D. D., Jackson, M. A. "*A Minority Dissenting Opinion*". En W. W. Cotterman, J. D. Couger, N. L. Enger, F. Harold (Eds.). Systems Analysis and Design: A Foundation for the 1980s, pp. 551-553, New York: Elsevier, 1981

[McDermid y Rook, 1993] McDermid, J., Rook, P. "*Software Development Process Models*". En McDermid, J. (Ed.) The Software Engineer's Reference Book. CRC Press, Páginas 15-28. 1993

Referencias (vi)

[Mills et al., 1987] Mills, H. D., Dyer, M., Linger, R. "*Cleanroom Software Engineering*". IEEE Software, 4(5): 19-25, 1987

[Minasi, 2000] Minasi, M. "*Software Conspiracy*". McGraw Hill, 2000

[Moreno-Navarro, 2005] Moreno-Navarro, J. J. "*De la Arquitectura Software al Urbanismo Software: Hacia Nuevas Formas de Concebir los Sistemas de Software Intensivo*". En las Actas de las X Jornadas sobre Ingeniería del Software y Bases de Datos, JISBD'2005 (Granada, 13-16 de septiembre de 2005), pp. 179-186. Thomson

[Muller, 1997] Muller, P. A. "*Modelado de objetos con UML*". Eyrolles-Ediciones Gestión 2000, 1997

[Nierstrasz, 1992] Nierstrasz, O. M. "*Component Oriented Software Development*", Communications of the ACM, 35(9):160-165. September 1992

[NIST, 1994] National Institute of Standards and Technology (NIST). "*Glossary of Software Reuse Terms*". NIST, December 1994

[Olsina, 1998] Olsina, L. "*Building a Web-Based Information System Applying the Hypermedia Flexible Process Modeling Strategy*". En Proceedings of the 1st International Workshop on Hypermedia Development, Hypertext'98. 1998

[OMG, 2003] OMG. "*OMG Unified Modeling Language Specification. Version 1.5*". Object Management Group Inc. Document formal/03-03-01. March 2003. http://www.omg.org/docs/formal/03-03-01.pdf [Última vez visitado, 26-9-2007]

[Orr, 1977] Orr, K. T. "*Structured System Development*". Yourdon Press, 1977

[Palvia y Nosek, 1993] Palvia, P., Nosek, J. "*A Field Examination of System Life Cycle Techniques and Methodologies*". Information and Management, 25(2):73-84, 1993

[Parnas y Weiss, 1987] Parnas, D. L., Weiss, D. M. "*Active Design Reviews: Principles and Practices*". Journal of Systems and Software, 7(4):259-265, December 1987

Referencias (vii)

[Pastor et al., 2001a] Pastor, O., Abrahão, S. M., Fons, J. J., Ramón, S. "*An Object-Oriented Approach to Automate Web Applications Development*". En Proceedings of the 2nd International Conference on Electronic Commerce and Web Technologies (EC-Web'01). Series: Lecture Notes in Computer Science, LNCS 2115, pp. 16-28. Springer Verlag, 2001

[Pastor et al., 2001b] Pastor, O., Gómez, J., Insfrán, E., Pelechano, V. "*The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming*". Information Systems, 26(7): 507-534, 2001

[Piattini et al., 2004] Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 2004

[Pfleeger, 2002] Pfleeger, S. L. "*Ingeniería del Software. Teoría y Práctica*". Prentice Hall, 2002

[Pressman, 2002] Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill. 2002

[Pressman, 2006] Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 6ª Edición. McGraw-Hill. 2006

[Raccoon, 1995] Raccoon, L. B. S. "*The Chaos Model and the Chaos Life Cycle*". ACM Software Engineering Notes, 20(1):55-66, 1995

[RAE, 2001] Real Academia Española "*Diccionario de la Lengua Española*". Vigésimo segunda edición. http://www.rae.es. [Última vez visitado, 26-9-2007]. 2001

[Rossi, 1996] Rossi, G. "*OOHDM: Object-Oriented Hypermedia Design Method*". PhD Thesis, PUC-Rio, Brazil. 1996

[Royce, 1970] Royce, W. W. "*Managing the Development of Large Software Systems: Concepts and Techniques*". En Proceedings of IEEE WESCON. Los Angeles, CA – USA, August 1970

Referencias (viii)

[Rubin y Goldberg, 1992] Rubin, K. S., Goldberg, A. "*Object Behavior Analysis*". Communications of the ACM 35(9): 48-62. September 1992

[Rumbaugh, 1992] Rumbaugh, J. "*Over the Waterfall and into the Whirlpool*". Journal of Object-Oriented Programming (JOOP), 5(2):23-2, 1992

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991

[Scacchi, 1987] Scacchi, W. "*Models of Software Evolution: Life Cycle and Process*". SEI Curriculum Module SEI-CM-10-1.0, Pittsburgh (EEUU), Software Engineering Institute (Carnegie Mellon University). 1987

[Shaw y Garlan, 1996] Shaw, M., Garlan, D. "*Software Architecture: Perspectives on a Emerging Discipline*". Prentice-Hall, 1996

[Shlaer y Mellor, 1992] Shlaer, S., Mellor, S. "*Object Life Cycles: Modeling the World in States*". Prentice-Hall, 1992

[Smith, 2001] Smith, J. "*A Comparison of RUP and XP*". White Paper. TP-167 5/01. Rational Software Corporation. 2001

[Sommerville, 2005] Sommerville, I. "*Ingeniería del Software*". 7ª Edición, Addison-Wesley. 2005

[Tardieu et al., 1986] Tardieu, H., Rochfeld, A., Coletti, R. "*La Méthode Merise. Tomo 1. Principes et Outils*". Editions d'Organisation, 1986

[Turk et al., 2002] Turk, D., France, R., Rumpe, B. "*Limitations of Agile Software Processes*". Proceedings of 4th International Conference on eXtreme Programming and Agile Processes in Software Engineering, XP2002. (Alghero, Sardinia, Italy, April 2002). Páginas 43-46. 2002

Referencias (ix)

[Ward y Mellor, 1985] Ward, P., Mellor, S. "*Structured Development for Real-Time Systems. Vols. 1-3*". Yourdon Press, 1985

[Warnier, 1974] Warnier, J. "Logical Construction of Programs". Van Nostrand Reinhold, 1974

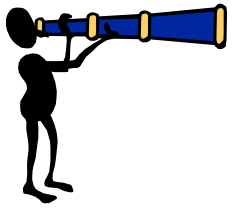
[Wielinga et al., 1991] Wielinga, B. J., Schreiber, A. T., Breuker, J. A. "*KADS: A Modeling Approach to Knowledge Engineering*". Technical Report ESPRIT Project P5248 KAD-II. 1991

[Wirfs-Brock et al., 1990] Wirfs-Brock, R., Wilkerson, B., Wiener, L. "*Designing Object-Oriented Software*". Prentice-Hall, 1990

[Wolff, 1989] Wolff, J. G. "*The Management of Risk System Development: 'Project SP' and the 'New Spiral Model*". Software Engineering Journal, May 1989

[Yourdon Inc., 1993] Yourdon Inc. "*Yourdon™ Systems Method. Model-Driven Systems Development*". Prentice Hall International Editions, 1993

Ingeniería del Software



Tema 1: Introducción a la Ingeniería del Software

Dr. Francisco José García Peñalvo
(fgarcia@usal.es)
Miguel Ángel Conde González
(mconde@usal.es)
Sergio Bravo Martín
(ser@usal.es)

3º I.T.I.S.
Fecha de última modificación: 26-9-2007

Universidad de Salamanca – Departamento de Informática y Automática