Introducción a la Ingeniería del Software

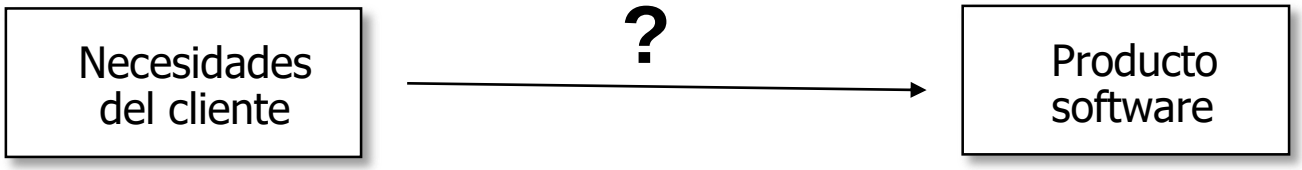
Indice

- 1. Motivación
- 2. ¿Qué es la ingeniería del software?
- 3. Metodologías
- 4. Ciclo de vida del software

1. Motivación

- Críticas a los sistemas software
  - Retrasos no previstos
  - Desbordamiento de costes
  - Software no acorde con las necesidades del cliente
  - Errores en los programas
  - Sensibilidad a los errores humanos
  - Dificultad de puesta en marcha
  - Dificultad de evolución
  - Mantenimiento ruinoso

Busquemos el problema …



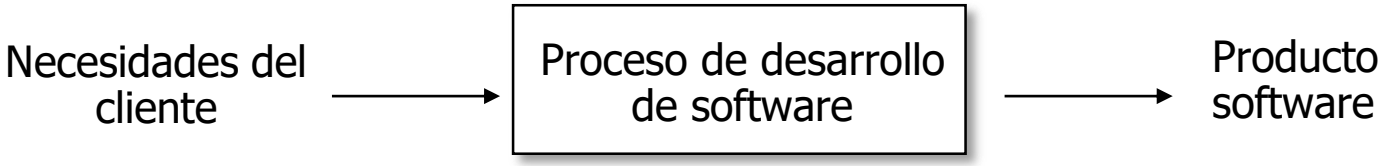
- El producto surge espontáneamente después de un largo periodo de oscuridad
- Existencia de un vacío de información entre la especificación del producto software y su entrega
  - ¿Qué pasa si alguien se va a mitad de un proyecto? ¿y si se va todo un equipo?
- No hay ingeniería …

¿Donde está la solución?

- Producir software de calidad a bajo coste y a tiempo
- La calidad debe medirse desde el punto de vista del cliente, usuarios finales, mantenedores del software, etc.

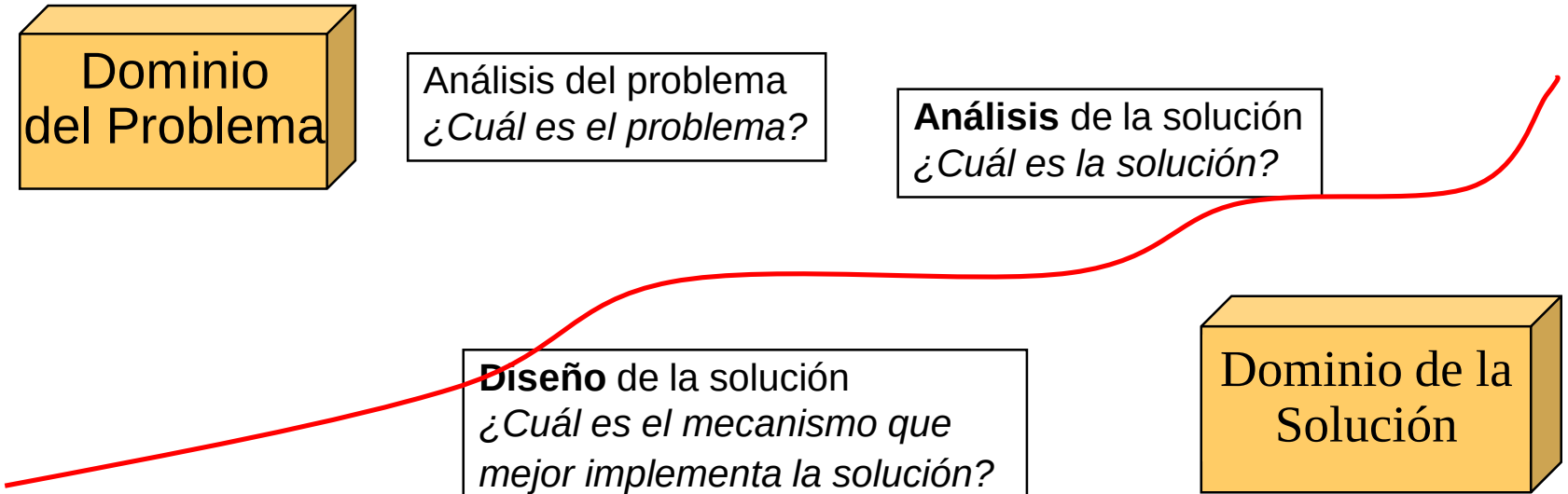
Ingeniería del software

¿Cómo hacer software de calidad?



- Ciclo de vida del software
  - Sucesión de pasos a través de los cuales el producto software va progresando
  - Estos pasos abarcan desde el planteamiento del problema a resolver con el producto software, hasta la retirada del producto una vez finalizada su vida útil

¿Cómo abordar el proceso de desarrollo?



2. ¿Qué es la ingeniería del software?

- Ingeniería del software
  - Aplicación de una aproximación sistemática, disciplinada y cuantificable al desarrollo, funcionamiento y mantenimiento del software.
  - Establecimiento y uso de sólidos principios de ingeniería y buenas prácticas de gestión, así como la evolución de herramientas y métodos aplicables y su uso cuando sea apropiado para obtener, dentro de las limitaciones de recursos existentes, software que sea de alta calidad en un sentido explícitamente definido.
    - *F.L.Bauer. "Software Engineering", Information Processing, 71, North Holland Publishing Co., Amsterdam 1972*

Fases del proceso de desarrollo (I)

- Requisitos del software
  - Características con las que deberá contar el software a desarrollar
  - Tipos de requisitos: funcionales y no funcionales
- Determinación de requisitos (Fase de requisitos)
  - Definición del sistema en términos entendidos por el cliente ("Descripción del problema")
  - Tareas: obtención de requisitos; representación de requisitos; validación de requisitos
- Análisis (Fase de análisis)
  - Especificación técnica del sistema en términos entendidos por el desarrollador ("Especificación del problema")

Fases del proceso de desarrollo (II)

- Diseño software
  - Definir la arquitectura, componentes, interfaces y otras características de un sistema o componente.
- Construcción del software (Codificación, implementación)
  - Traducción de las especificaciones de diseño a un lenguaje de programación determinado
- Pruebas del software
  - Comprobar el comportamiento real del software frente al especificado
  - Pruebas de desarrollo (inspección, pruebas unitarias, pruebas de integración, depuración del código), pruebas de sistema, pruebas de aceptación

Actividades relacionadas con la gestión del proceso de desarrollo (I)

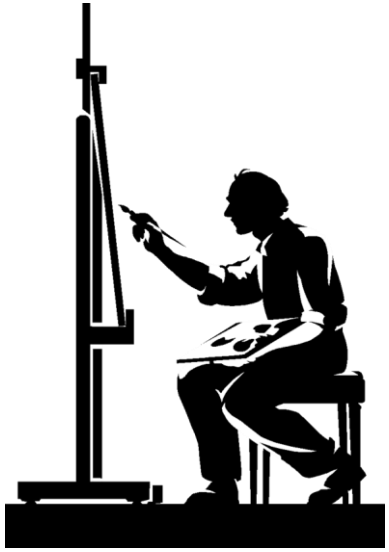
- Gestión del proyecto software
  - Gestionar y medir el proyecto software. Planificar el proceso, estimar costes, asignar recursos, gestionar riesgos, evaluar prestaciones, etc.
  - Riesgo: problema que puede surgir en un proyecto afectando a su correcto desarrollo
    - Los riesgos deberían estar identificados y se deberían determinar estrategias que permitan mitigarlos
- Mantenimiento del software
  - Detectar anomalías y cambios en el entorno de ejecución, nuevos requisitos
  - Hay que estimar y medir los costes de mantenimiento

Actividades relacionadas con la gestión del proceso de desarrollo (II)

- Gestión de configuraciones
  - Gestionar sistemáticamente los cambios que se producen en el software a lo largo del tiempo
  - Se debe mantener la integridad y seguimiento de la configuración durante el ciclo de vida
  - Hay herramientas y técnicas para gestionar las versiones del software
- Calidad del software
  - Calidad: aptitud de un producto para satisfacer las necesidades del usuario
  - Existen modelos de calidad a nivel de procesos: *Capability Maturity Model Integration* (CMMI), *Software Process Improvement Capability Determination* (SPICE), …
  - La calidad de un proyecto se revisa y audita para asegurarla y mejorarla

Conclusión

El software es una industria,...



... no es un arte.

- La ingeniería del software la aplicamos para desarrollar cualquier tipo de software:
  - Videojuegos
  - Software de control
  - Software de tiempo real
  - Software de gestión
  - Software para sistemas empotrados
  - etc.

3. Metodologías

- Método
  - Procedimiento formal para producir resultados utilizando alguna notación bien definida.
    - Ejemplos: Diagramas de estados (modelar el comportamiento), diagramas de clases (modelar la estructura)
- Tipos de métodos
  - No formales, usando lenguaje natural
  - Semi-formales, notaciones (en parte graficas) con ciertas reglas y cuyas construcciones tienen una semántica más o menos precisa
  - Formales, usando una notación gráfica o textual basada en un sistema formal (soporte matemático)

Metodología

- Colección de métodos, aplicados a lo largo del proceso de desarrollo del software, unificado por alguna aproximación filosófica general
- Una metodología es una colección de:
  - Métodos: técnicas que ayudan a construir modelos
  - Notaciones: lenguajes (normalmente gráficos) que sirven para expresar los modelos
  - Procesos: guías para construir los modelos de manera ordenada
  - Herramientas: dan soporte a la construcción y comprobación de los modelos (completitud, consistencia, no ambigüedad, etc.)

Metodologías

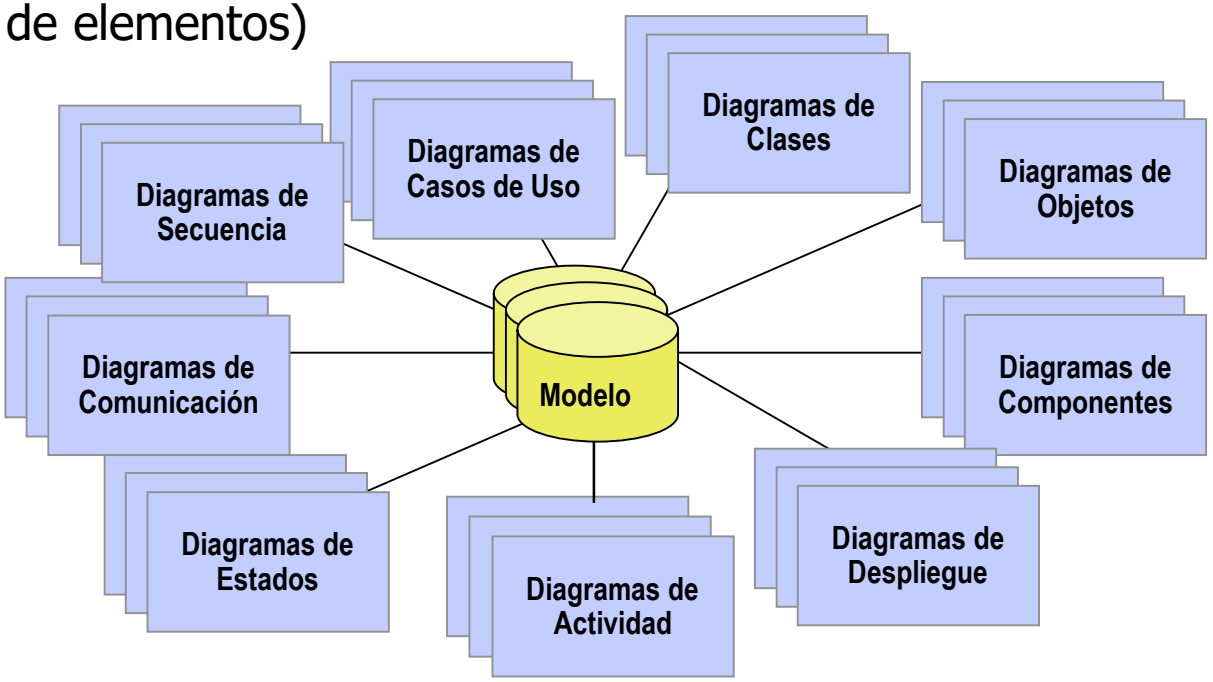
- Tipos de metodologías
  - Estructuradas
    - Basadas en la descomposición funcional de sistemas
    - Ejemplos:
      - Merise (metodología pública francesa)
      - Métrica 2 (metodología pública española, estructurada)
  - Orientadas a objetos
    - Basadas en técnicas orientadas a objeto
    - Ejemplos:
      - Object Modeling Technique (OMT)
      - Métrica 3 (metodología pública española, OO)
- En este curso vemos las actividades de requisitos, análisis y diseño utilizando un enfoque orientado a objetos

Notaciones: UML

- UML = Unified Modeling Language
- Un lenguaje de propósito general para el modelado orientado a objetos
- UML *no* es una metodología de desarrollo de software
- Podemos utilizar UML con cualquier metodología (orientada a objetos o no)

UML

- Un modelo es una abstracción de un sistema para comprenderlo mejor
- Una vista es una proyección del modelo centrado en algún aspecto concreto:
  - Vista de casos de uso (comportamiento), vista lógica, vista de realización, vista de despliegue
  - Para mostrar una vista utilizamos diagramas (representación de un conjunto de elementos)



UML

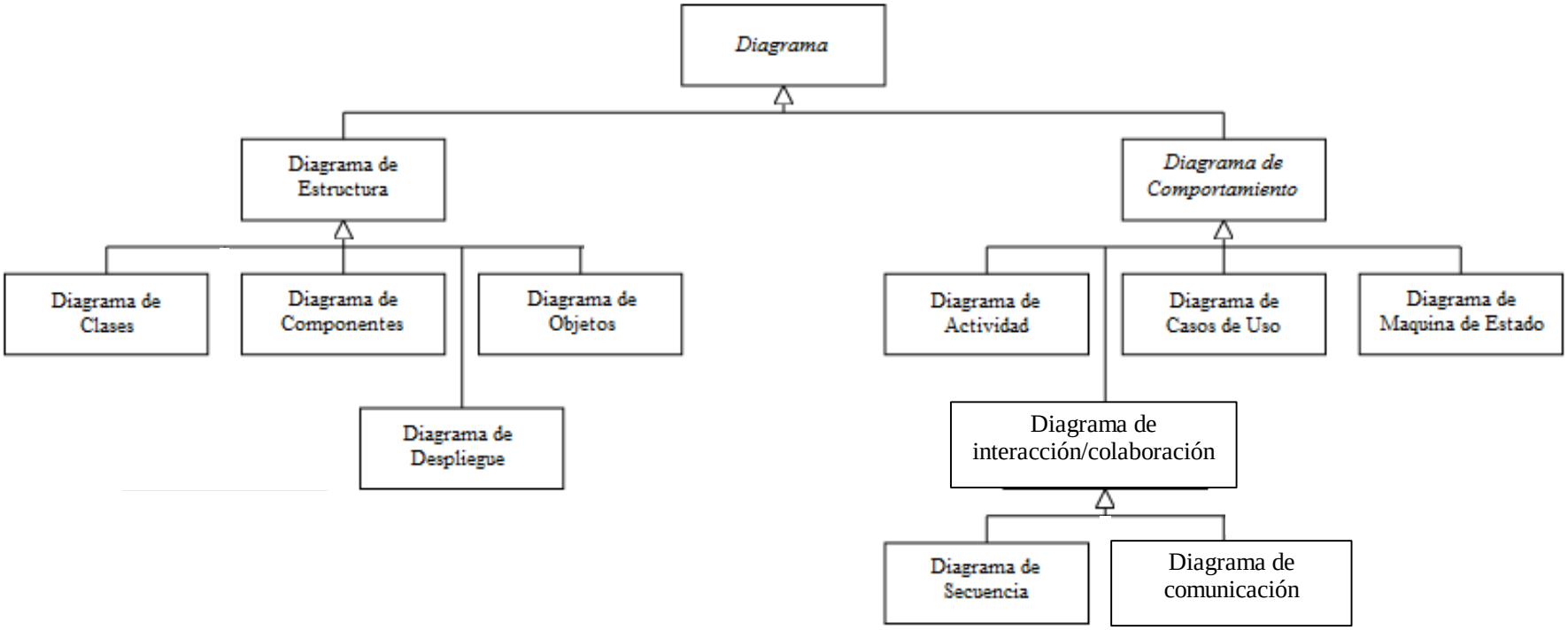


Diagrama
Diagrama de Estructura
Diagrama de Comportamiento
Diagrama de Clases
Diagrama de Componentes
Diagrama de Objetos
Diagrama de Despliegue
Diagrama de Actividad
Diagrama de Casos de Uso
Diagrama de Maquina de Estado
Diagrama de interacción/colaboración
Diagrama de Secuencia
Diagrama de comunicación

Herramientas

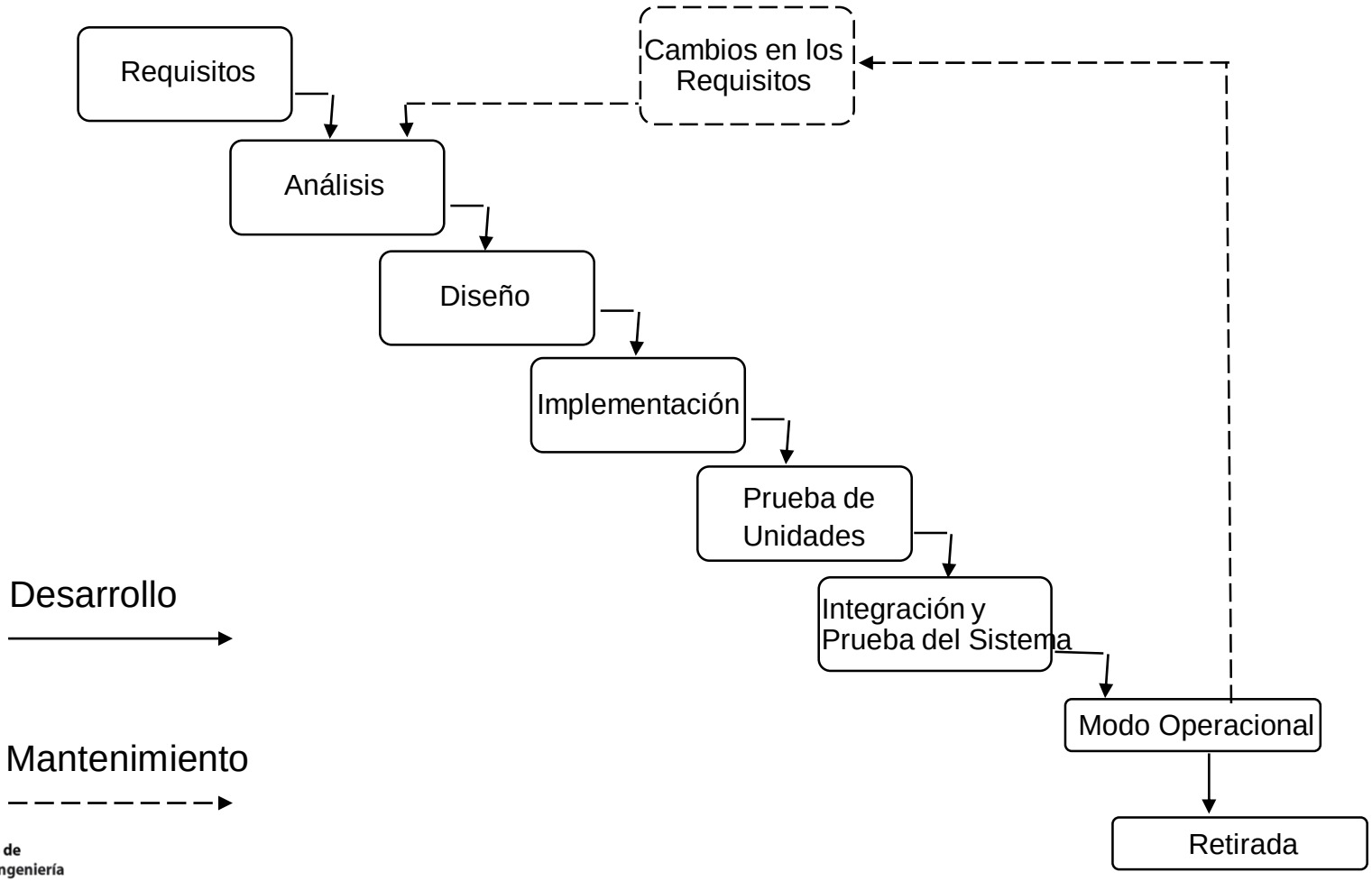
- Definición de herramienta CASE (Computer Aided Software Engineering):
  - Software que proporcione ayuda, más o menos automática, en cualquiera de las etapas del ciclo de vida del software:
    - Dibujar diagramas: crear y mantener de forma gráfica los modelos del sistema
    - Prototipado de la interfaz de usuario
    - Generar automáticamente código y/o documentación
    - Verificar y corregir especificaciones
    - Permitir reingeniería de los sistemas actuales
- Este curso utilizaremos *Visual Paradigm* para crear y mantener modelos

4. Modelos de ciclo de vida

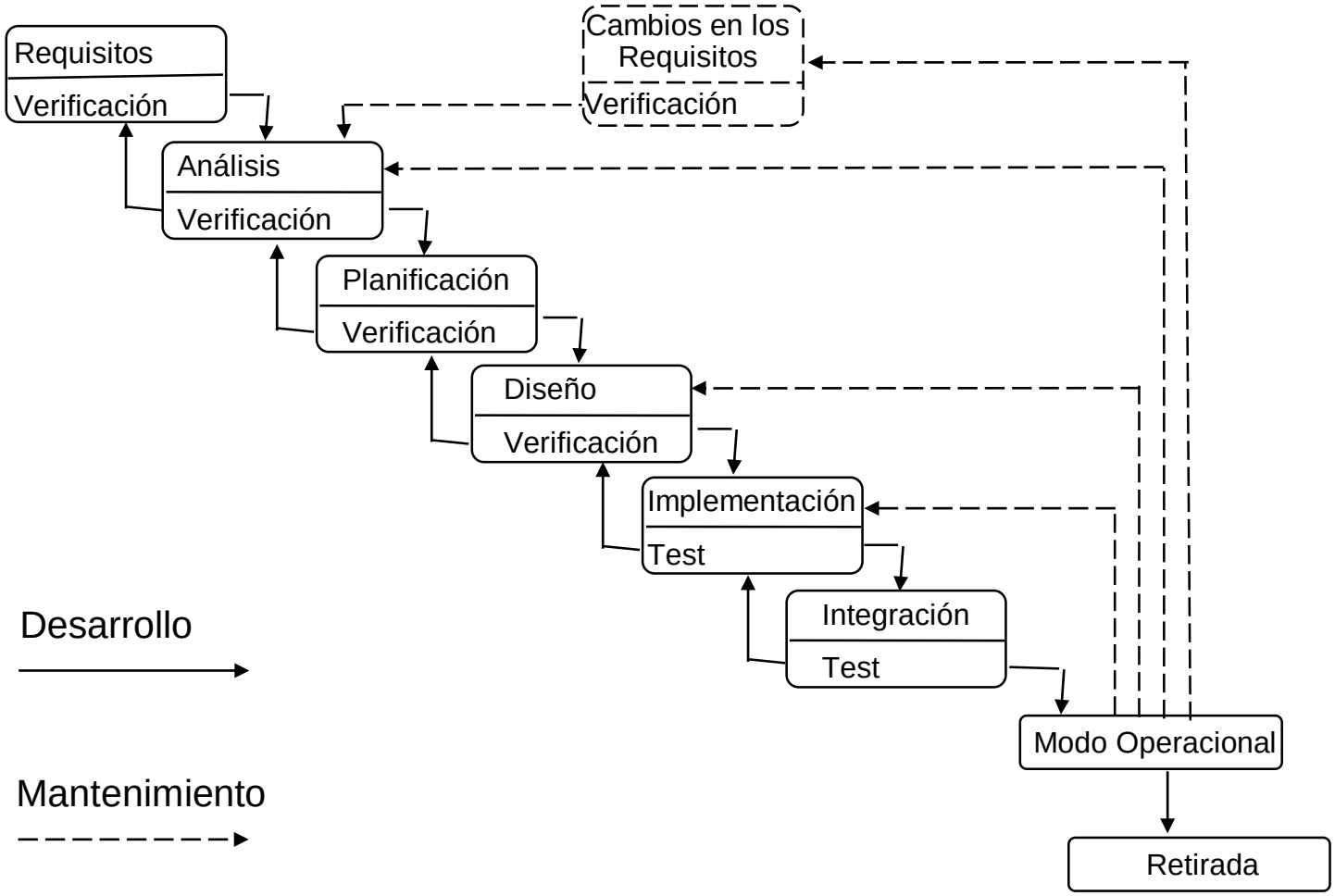
Ciclo de vida = Sucesión de pasos a través de los cuales el producto software va progresando

- Modelos secuenciales
  - Ciclo de vida en cascada clásico
  - Ciclo de vida en cascada mejorado
  - Ciclo de vida en V
- Modelos iterativos
  - Ciclo de vida con prototipado
  - Ciclo de vida incremental
  - Ciclo de vida en espiral
  - Proceso Unificado
- Metodologías ágiles

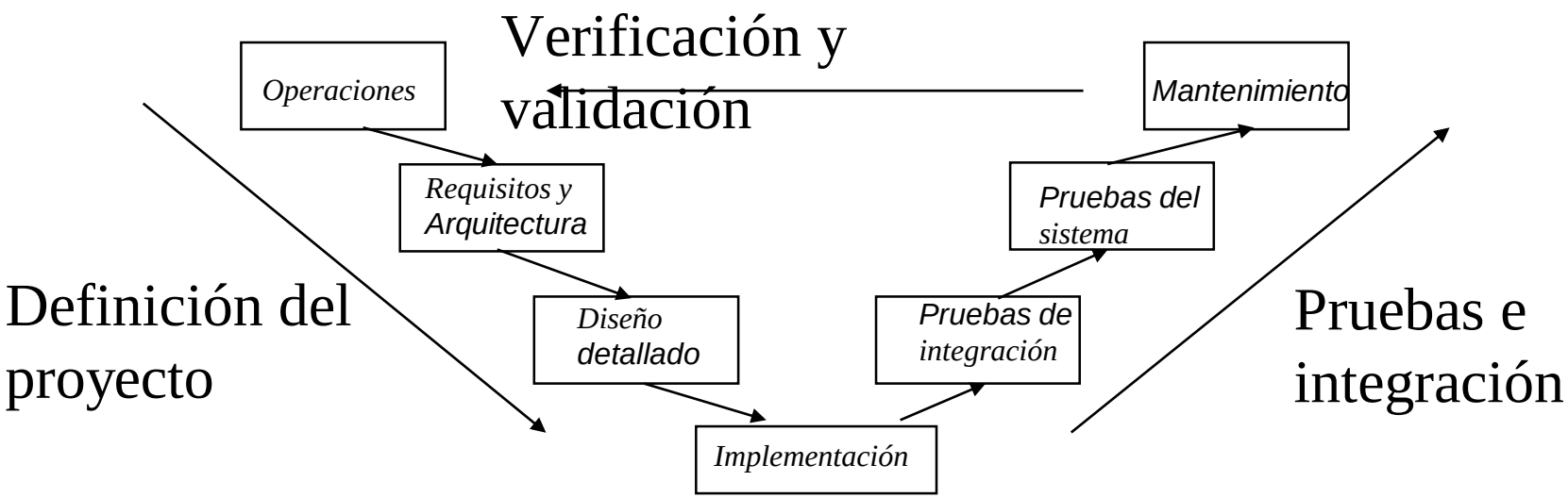
Cascada clásico



Cascada mejorado

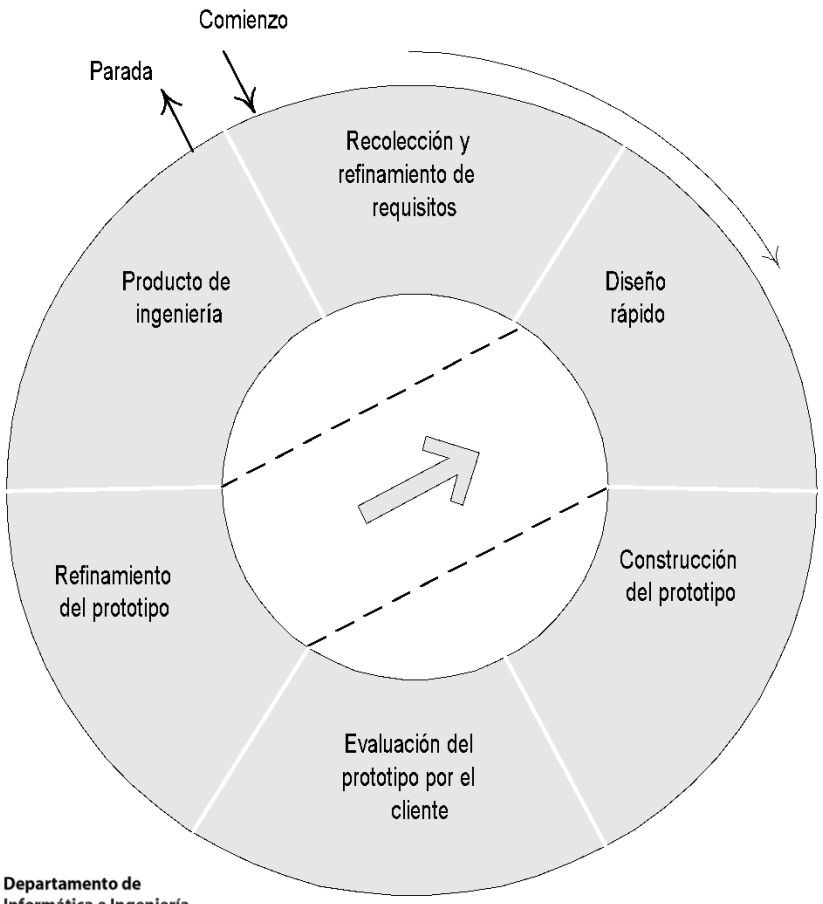


Ciclo de vida en V



- Se verifica cada salida con la especificación de su entrada
- Ventajas:
  - Supone una mejora respecto al modelo en cascada en cuanto a la verificación y la validación
- Problemas:
  - Prácticamente los mismos que los del modelo en cascada

Ciclo de vida con prototipado



Objetivo:

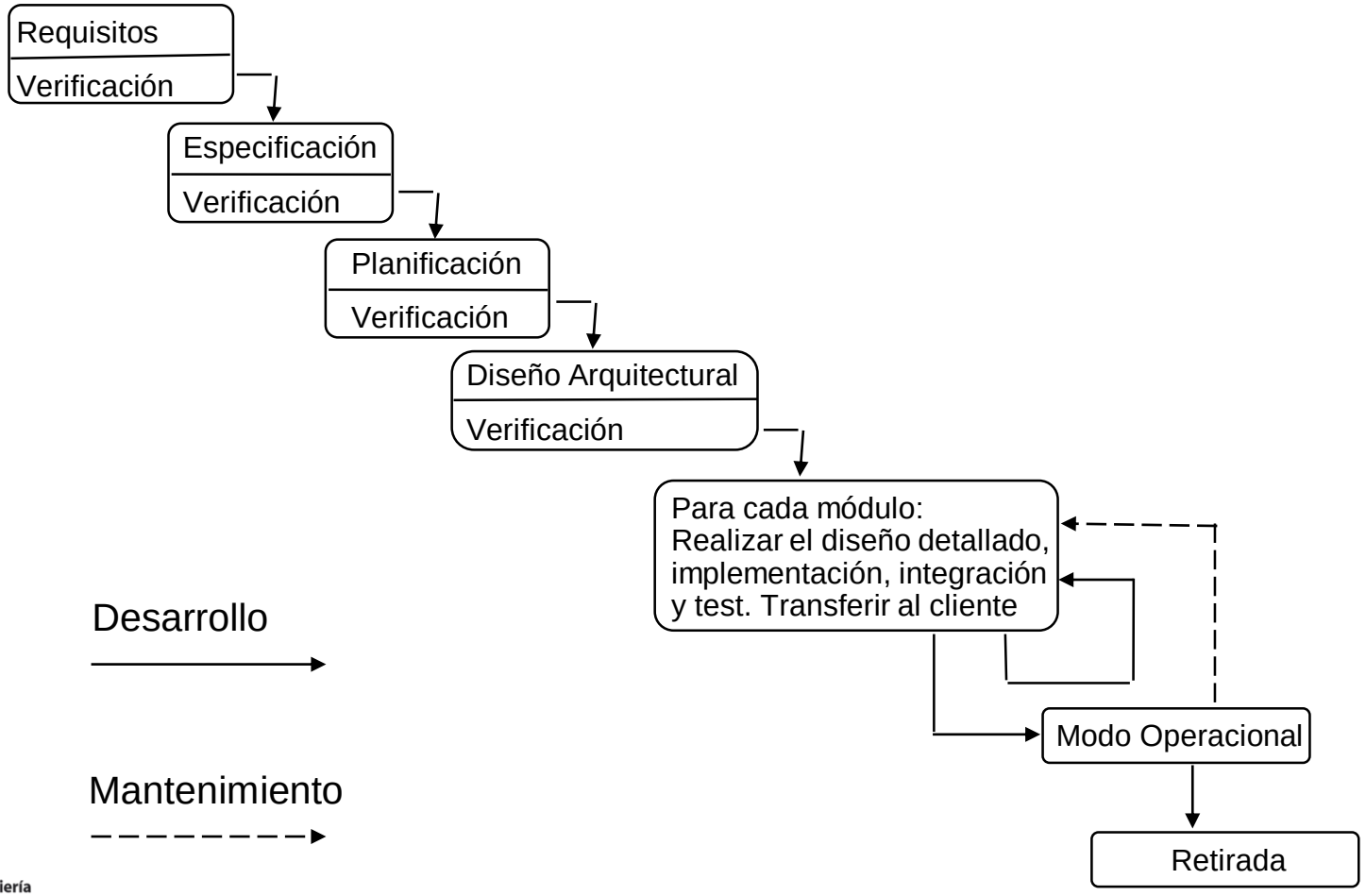
Obtener rápidamente un prototipo De la aplicación que permita al cliente interactuar con ella con el fin de detectar deficiencias en las especificaciones.

Construimos un *boceto* rápido de la aplicación

Tipos de prototipado

- Revolucionario
  - Crear versiones de usar y tirar para determinar correctamente los requisitos
  - Desventaja: los usuarios tienen que aceptar que algunas de las características no estarán en el sistema final debido a su coste
- Evolutivo
  - El prototipo se usa como base de la implementación del sistema final
  - Solo es posible si el sistema final se puede construir con la misma tecnología (lenguaje) que el prototipo
- Funcional
  - Implementar un sistema operacional con la mínima funcionalidad. Después añadir más funcionalidad
- Exploratorio
  - Implementar parte del sistema para saber un poco más acerca de los requisitos

Modelo incremental



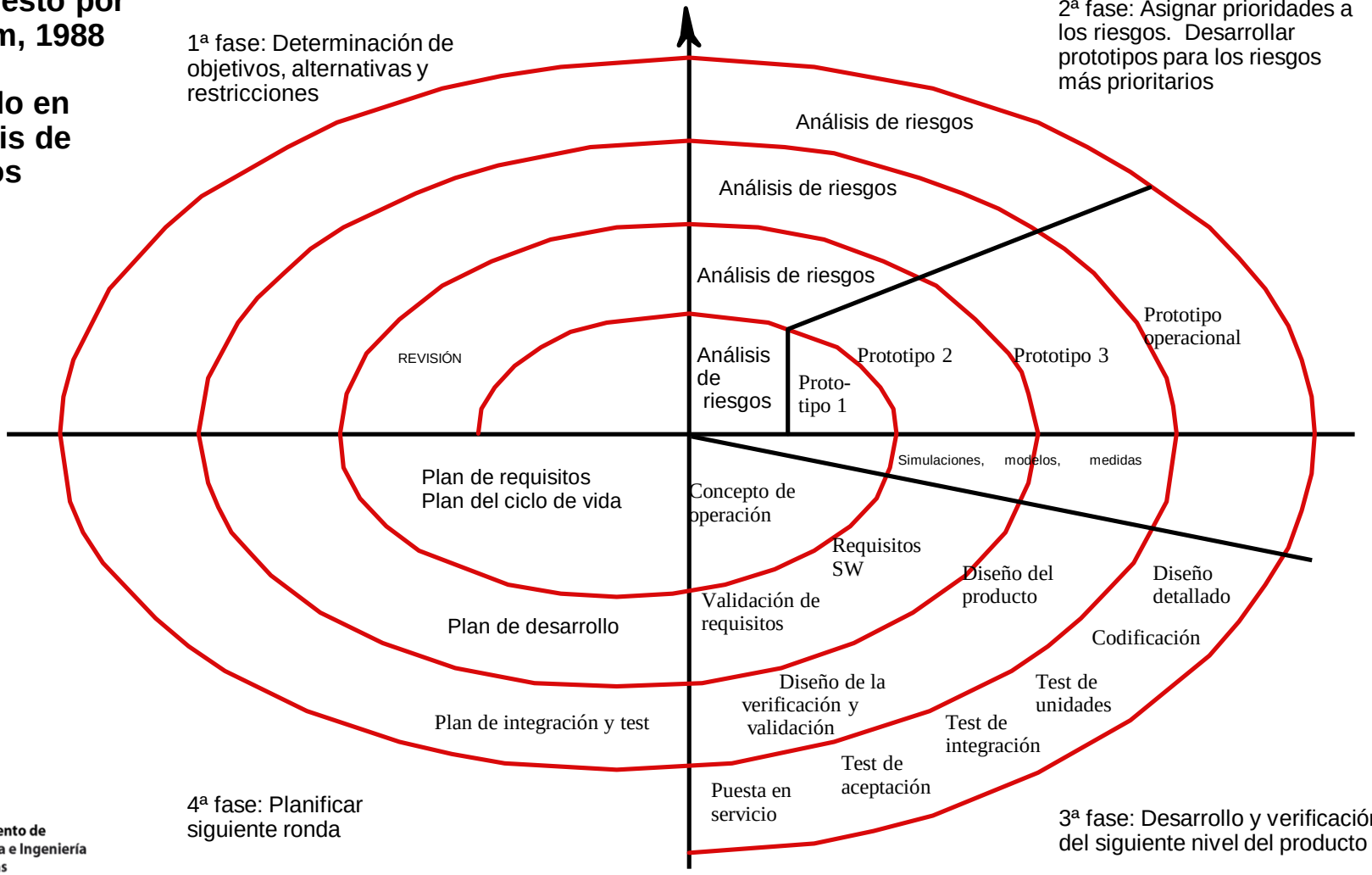
Modelo incremental

- Ventajas
  - Satisface la curiosidad del cliente
  - El cliente puede ir "viendo" la aplicación real, no un prototipo
  - Posibilidad de detener el proyecto sin perder todo lo realizado
  - Mayor flexibilidad ante cambios en los requisitos durante el desarrollo
- Inconvenientes
  - Cada nuevo módulo se debe integrar en el sistema sin afectar a lo que ya está funcionando (datos, interfaces,...)
  - El diseño y desarrollo de los diferentes módulos debe ser coherente entre sí
  - Gran tendencia a la degeneración del control del proyecto ante la imposibilidad de verlo como un todo

Ciclo de vida en espiral

Propuesto por Boehm, 1988

Basado en análisis de riesgos

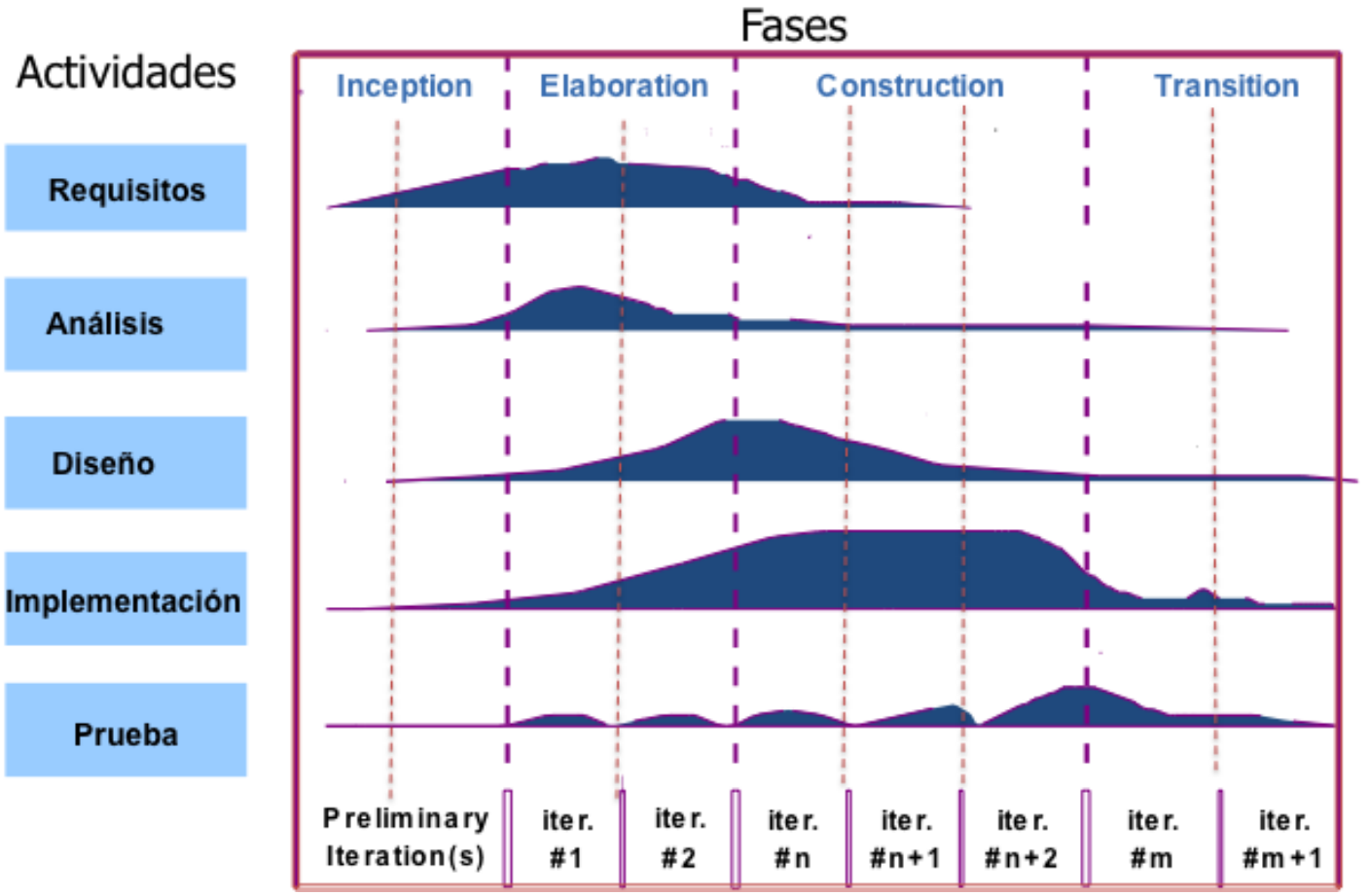


Proceso Unificado (Rational Unified Process)

- Axiomas:
  - Dirigido por los requisitos y por los riesgos
  - Centrado en la arquitectura
  - Iterativo e incremental
- Iterativo e incremental:
  - Dividir el proyecto en subproyectos (iteraciones), que proporcionan la funcionalidad en "trozos"
  - Cada iteración genera una versión parcialmente completa del sistema final, y tiene todas las actividades de un proyecto de desarrollo de software normal

  - A su vez las iteraciones se agrupan en fases

Proceso Unificado: Estructura



Concepción (los clientes aceptan requisitos, riesgos y arquitectura establecidos)

Versión ejecutable de arquitectura

El producto es estable, se puede instalar

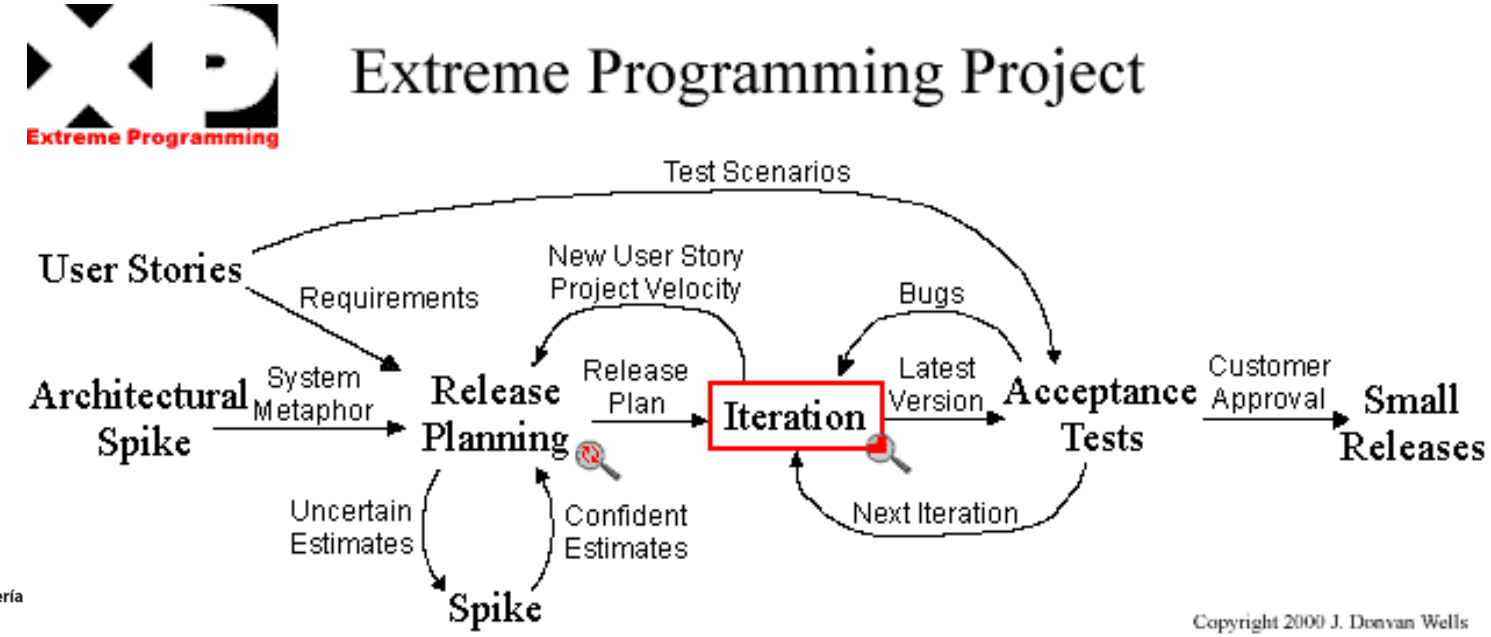
Se ha probado, cliente acepta producto, plan de mantenimiento

Metodologías ágiles

- Desarrollos iterativos e incrementales
  - Entrega temprana de software incremental
  - Las iteraciones duran de una a cuatro semanas
  - Después de una iteración se evalúan prioridades
- Equipos de proyecto pequeños y con alta motivación
- Métodos informales, un mínimo de productos de trabajo de la IS:
  - minimizar la documentación
  - comunicación cara a cara
- Simplicidad general de desarrollo
- Colaboración con el cliente sobre la negociación contractual
- Respuesta ante el cambio sobre seguir un plan

Ejemplo: Programación Extrema (eXtreme Programming, XP)

- Se distingue por dos ideas radicales
  - Programación por pares: Todo el desarrollo es realizado por dos personas en lugar de individualmente
    - Más que mejorar la velocidad de producción, se mejora la calidad del software (la compartición de ideas facilita la aceleración del crecimiento)
  - Desarrollo dirigido por las pruebas
    - La prueba continua es tan importante que no solo se debería hacer por los desarrolladores, sino que los tests se deberían escribir previamente



http://www.extremeprogramming.org/map/iteration.html

Ejemplo: Scrum

- Desarrollo incremental en lugar de planificación y ejecución completa del producto
- Basar más la calidad en el acuerdo tácito entre personas que en la calidad de los procesos (documentación)
- Solapamiento de fases de desarrollo, en lugar de desarrollo en cascada
- Incrementos de software mediante *sprints* (iteraciones entre 1 y 4 semanas)