UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

Proyectos de Desarrollo Software

Capítulo 5

Dr. Daniel Tapias
daniel.tapias@uam.es

Curso 2014/ 15

PROYECTOS

Escuela Politécnica Superior

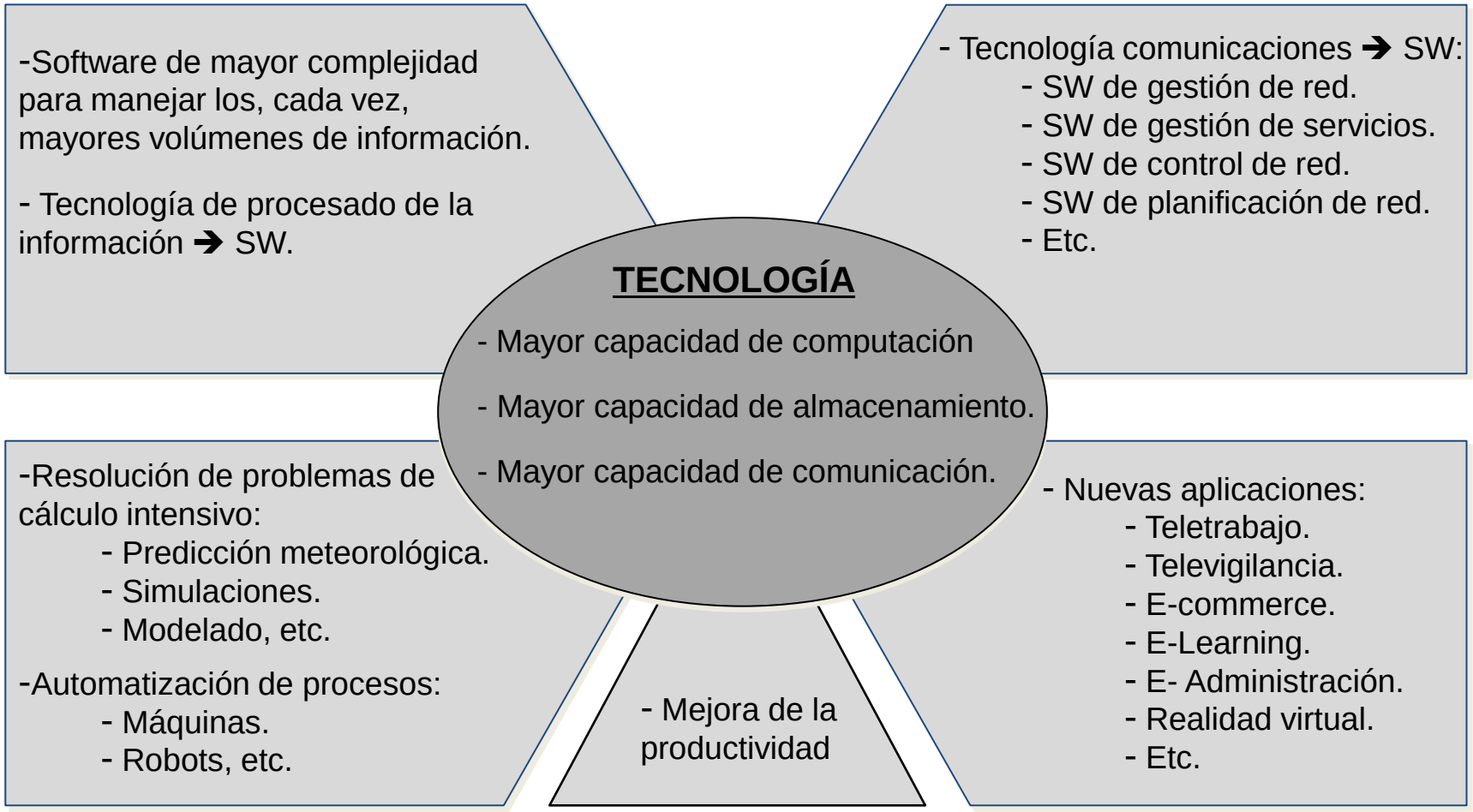
PROGRAMA DE LA ASIGNATURA

- Capítulo 1: Introducción.
- Capítulo 2: ¿Qué es un proyecto?
- Capítulo 3: Tipos de proyectos.
- Capítulo 4: Ciclo de vida de los proyectos.
- **Capítulo 5: Proyectos de desarrollo software.**
- Capítulo 6: Organización empresarial y proyectos.
- Capítulo 7: La Calidad.
- Capítulo 8: La usabilidad y la accesibilidad.
- Capítulo 9: El Riesgo.
- Capítulo 10: Ingeniería Económica. Estudios de viabilidad económica.
- Capítulo 11: Técnicas para la planificación y control de proyectos.
- Capítulo 12: Toma de decisión.
- Capítulo 13: Proyecto: Búsqueda de empleo.

ÍNDICE

EL SOFTWARE EN LA ACTUALIDAD (I)

El desarrollo de la Sociedad de la Información y el paso a la Sociedad del Conocimiento requieren un importante esfuerzo inversor en Desarrollo Software.



EL SOFTWARE EN LA ACTUALIDAD (II)

Sin embargo, a pesar de la importancia del Software en la sociedad actual y futura, hay múltiples estudios que indican que:

- Cerca de un tercio de los proyectos de desarrollo software falla.
- Más de la mitad de los proyectos superan el presupuesto inicialmente estimado.
- Sólo el 16% de los proyectos terminan en el plazo fijado y dentro de presupuesto.

CARACTERÍSTICAS DEL SOFTWARE

Los proyectos de desarrollo software son distintos del resto de los proyectos de ingeniería tradicional por la propia naturaleza del software:

- Se desarrolla. No se fabrica en el sentido tradicional de la palabra.
- No se rompe ni se deteriora con el uso, el paso del tiempo o el entorno.
- No es algo “tangible”, lo que hace que muchas veces se minusvalore.

ELEMENTOS DEL SOFTWARE

PROBLEMAS EN LA PLANIFICACIÓN DE UN PROYECTO SOFTWARE

Especificaciones

- Incorrectas y/o incompletas.
- Inestables y sujetas a cambios, muchas veces, importantes.

Planificación

- No se planifica: “Como los planes cambiarán, es una pérdida de tiempo.”
- Los costes y plazos no se actualizan cuando hay cambios.
- Los costes y plazos los marca el mercado y no el cumplimiento de unos objetivos y de unos niveles de calidad.
- Es difícil estimar la complejidad de un proyecto software.
- No se tienen en cuenta factores de riesgo.
- Bajo presión, se pasa de un trabajo planificado a otro errático de codificar y corregir, sin metodología ni documentación.
- Se escatiman recursos para usabilidad, accesibilidad y calidad.

Aprendizaje

- La mayor parte de las empresas de desarrollo software no recogen ni analizan datos de proyectos anteriores. Consecuentemente, en los nuevos proyectos se omiten tareas o se cometen errores de estimación.
- La mayor parte de las empresas no disponen de procesos específicos de desarrollo software.

PROBLEMAS DEL EQUIPO DE UN PROYECTO SOFTWARE (I)

Relacionados con los miembros del equipo

- **Motivación:** El director de proyecto no crea un ambiente que invite a la motivación o, incluso, toma medidas que minan la moral del equipo.
- **Mala selección de personal:** El criterio de selección no está relacionado con los resultados que se esperan de la persona. Por ejemplo: criterios económicos, de inmediatez de la incorporación, etc.
- **Miembros del equipo problemáticos e incontrolados:** Todo el mundo sabe que esa persona sólo da problemas pero el jefe no interviene.
- **Oficinas saturadas, ruidosas e incómodas.**
- **Desarrolladores desenfocados:** Encuentran fascinante una nueva tecnología, prestación, lenguaje, etc. y se ponen a trabajar en temas innecesarios pero interesantes para ellos.

PROBLEMAS DEL EQUIPO DE UN PROYECTO SOFTWARE (II)

Relacionados con la dirección del equipo

- **Fomento del comportamiento heróico:** El jefe valora que los miembros del equipo “sean capaces de” en lugar de valorar progresos bien fundamentados y en línea con la planificación. Llevan la planificación al límite, no reconociendo ni informando de las desviaciones hasta el último minuto.
- **Asignar más recursos a los proyectos retrasados:** Puede reducir la productividad de los miembros del equipo existente.
- **Ante problemas urgentes, más reuniones:** Reducen el tiempo disponible y desconcentra a las personas que están resolviendo el problema.
- **Imposiciones poco realistas:** La alta dirección o el cliente imponen un plazo de realización o unos costes irreales y el director de proyecto no lo corrige o no lo puede corregir (falta de un promotor en la alta dirección).
- **Planificación fantasiosa:** Se realiza una estimación optimista y fantasiosa de la planificación. Se espera que todo vaya bien sin tener unas bases razonables que así lo indiquen.
- **No participan todos los implicados:** Se excluye incluso a los usuarios.
- **Se valora poco el logro de los resultados en favor de otros temas** (buena relación personal, disponibilidad presencial, etc.)

PROBLEMAS DE LA TECNOLOGÍA EN UN PROYECTO SOFTWARE

- **Optimismo tecnológico:** Es frecuente que se pongan demasiadas esperanzas en las bondades proclamadas de las nuevas tecnologías (OOD, C++, Java, etc.)
  No se analiza lo buenas que pueden ser en el entorno de aplicación.

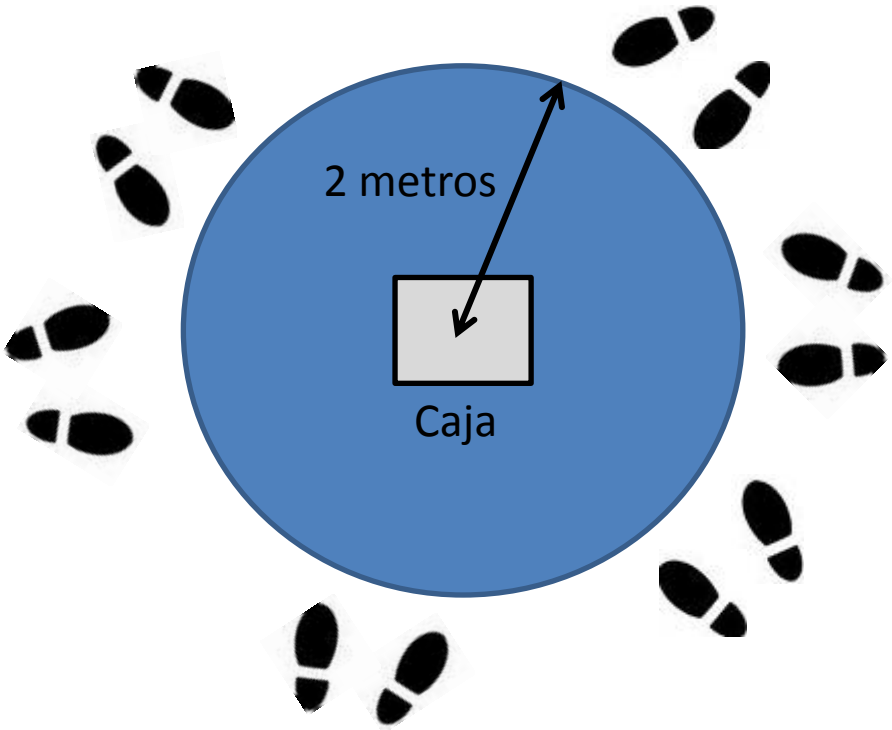
- **Sobreestimación de las nuevas herramientas y/o metodologías:** Los beneficios potenciales de las nuevas herramientas se ven afectados por el tiempo de aprendizaje, de implantación y de resolución de los problemas asociados a las mismas.

- **Cambio de herramientas durante el proyecto:** Si un cambio de versión de la herramienta ya supone un riesgo por el tiempo de aprendizaje, la resolución de los errores cometidos con la nueva versión, etc. un cambio de herramienta es un riesgo importante para el proyecto.

- **Control de versiones:** Si no se realiza de forma automática, hay muchas probabilidades de que dos desarrolladores modifiquen la misma parte del programa (sobreescribiendo el código del otro desarrollador) o que trabajen sobre versiones distintas del código.

EXPERIMENTO 6: Trabajo en Equipo.

El experimento que vamos a realizar consiste en intentar meter un conjunto de pelotas en una caja en el menor tiempo posible. La única regla que existe es que nadie puede poner los pies a menos de dos metros de la caja.



Para hacer el experimento, se formarán dos equipos: el equipo A y el equipo B.

¿Se pueden meter todas las pelotas en la caja en menos de 1 minuto?

¿Y en menos de 10 segundos?

CICLO DE VIDA DEL SOFTWARE (I)

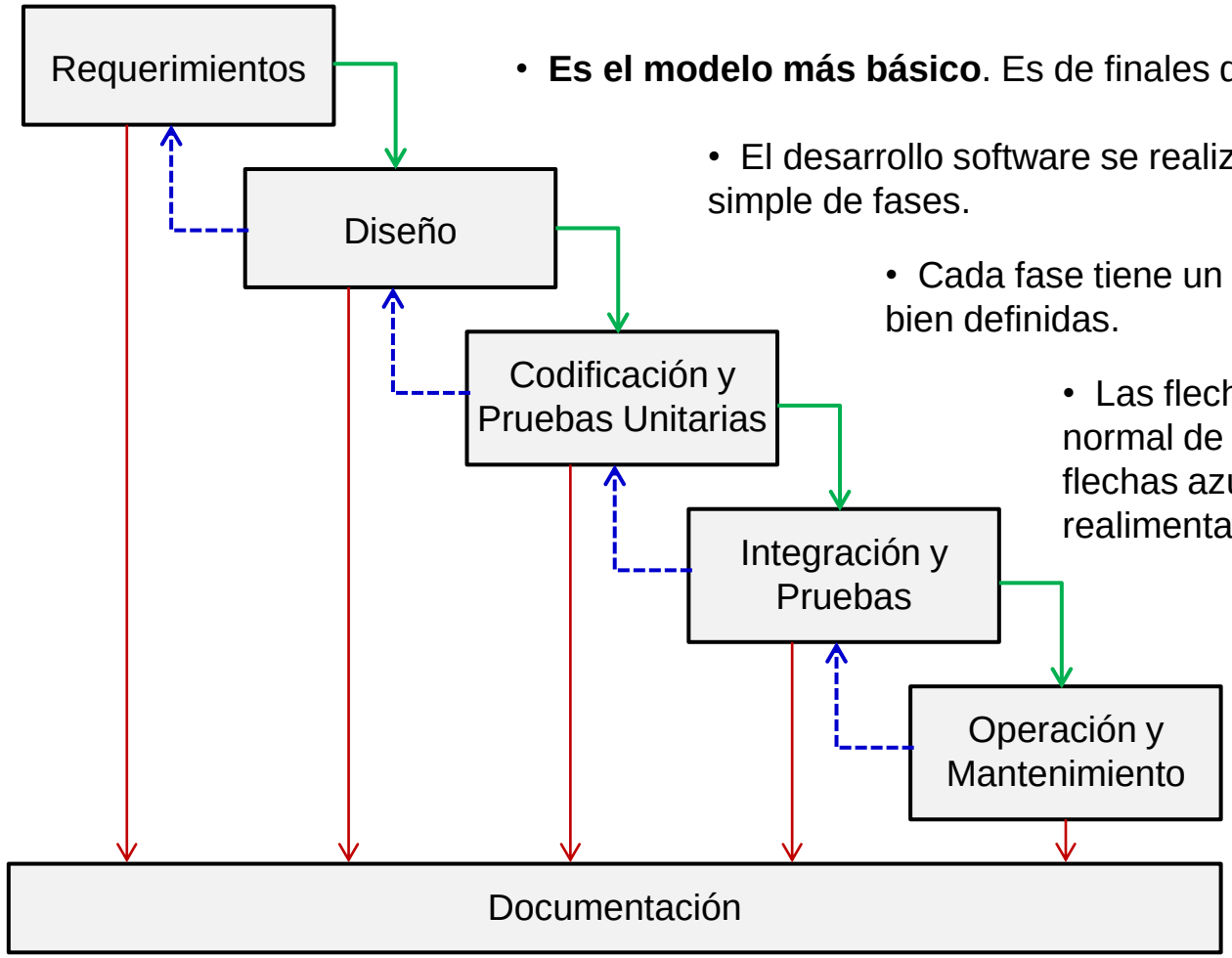
El ciclo de vida del software define las distintas etapas por las que transcurre un proyecto de desarrollo de software. Hay distintos modelos de ciclo de vida.

- Modelo en cascada.
- Modelo de Desarrollo Incremental.
- Modelo de Desarrollo Evolutivo.
- Modelo de Prototipado de Requerimientos.
- Modelo en Espiral.
- Modelo Concurrente.

El ciclo de vida del software permite describir las fases principales del desarrollo software y ayuda a administrar el progreso del desarrollo. Esto es: ayuda a ordenar las diversas actividades técnicas del proyecto y nos dota de un marco para estimar los recursos, definir los puntos de control, conocer el grado de avance, etc.

CICLO DE VIDA DEL SOFTWARE (II)

MODELO EN CASCADA MEJORADO

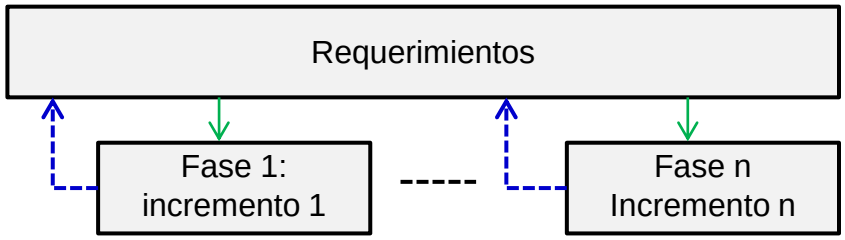


- **Es el modelo más básico**. Es de finales de los 70.
- El desarrollo software se realiza a través de una secuencia simple de fases.
- Cada fase tiene un conjunto de metas y actividades bien definidas.
- Las flechas verdes muestran el flujo normal de avance del ciclo de vida. Las flechas azules discontinuas muestran la realimentación entre fases.

CICLO DE VIDA DEL SOFTWARE (III)

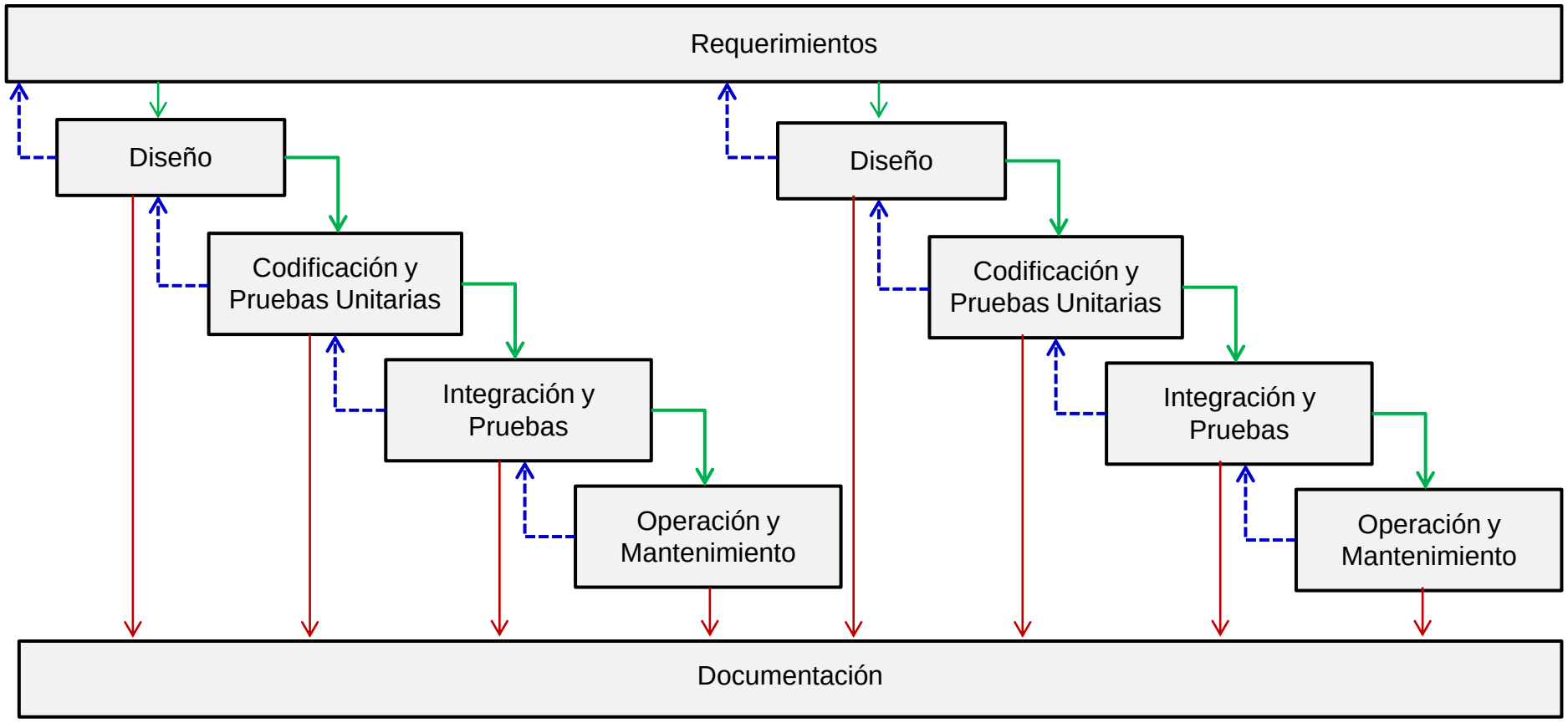
MODELO DE DESARROLLO INCREMENTAL

- Cuanto más complejo es un sistema software, mayor es el riesgo asociado al proyecto.
- Una forma de reducir el riesgo es desarrollar el sistema de forma incremental. Esto es: dividir el proyecto en fases y desarrollar una parte de los requerimientos en cada fase.
- El modelo incremental mantiene el modelo en cascada, pero lo repite “n” veces.
- Ventajas del modelo incremental:
  - Construir un sistema pequeño siempre es menos arriesgado que construir un sistema grande.
  - Si se comete un error importante, afecta a la última fase y siempre se puede ir a una versión anterior.
  - Se puede depurar cada fase (versión) antes de pasar a la siguiente.
  - Al desarrollar sólo parte de las funcionalidades y requerimientos en cada fase, es más fácil comprobar si los requerimientos de las siguientes fases son adecuados y correctos.
  - Se dispone de una primera versión más rápidamente y se controla mejor el avance del proyecto.



CICLO DE VIDA DEL SOFTWARE (IV)

MODELO DE DESARROLLO INCREMENTAL



CICLO DE VIDA DEL SOFTWARE (V)

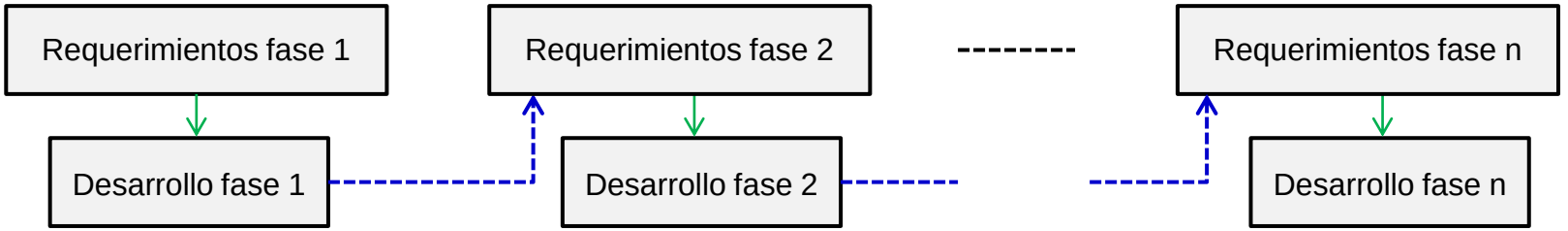
MODELO DE DESARROLLO EVOLUTIVO

- Es muy similar al modelo de desarrollo incremental. La principal diferencia está en que en el modelo incremental se presupone que todos los requisitos son conocidos al comienzo, en el modelo evolutivo se asume que **NO todos los requerimientos son conocidos** desde el primer momento.

- En una primera fase, se desarrolla la primera versión, que recoge los requisitos conocidos. Posteriormente, los usuarios utilizan el software desarrollado, generándose una nueva lista de requerimientos, con la que se desarrolla una nueva versión en la siguiente fase y así sucesivamente.

- El usuario ve la materialización del proyecto más rápido que si hubiera que hacer un estudio largo para concretar las especificaciones. Además, puede modificar/añadir requerimientos sin afectar al proyecto y conseguir algo que se ajuste mejor a sus necesidades.

- Es un modelo compatible con el modelo de cascada y puede ser combinado con el modelo incremental.



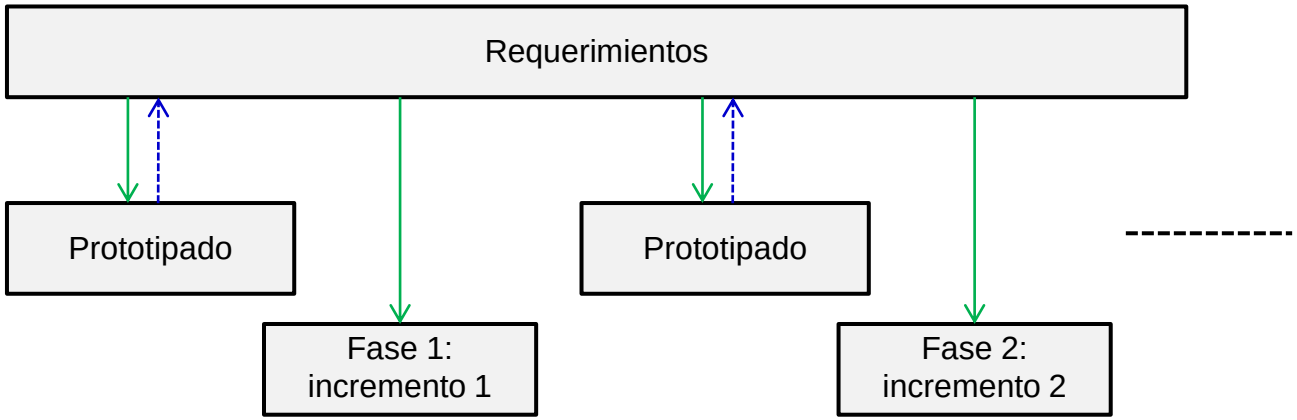
CICLO DE VIDA DEL SOFTWARE (VI)

MODELO DE PROTOTIPADO DE REQUERIMIENTOS

- Consiste en realizar implementaciones parciales del sistema para poder experimentar y validar o modificar los requerimientos del mismo.

- El prototipo se construye deprisa y se da a los usuarios para que lo prueben. Estos dan sus comentarios sobre lo bueno y lo malo, sobre lo que les gustó y lo que no. Los comentarios se recogen y sirven para hacer la especificación del sistema real.

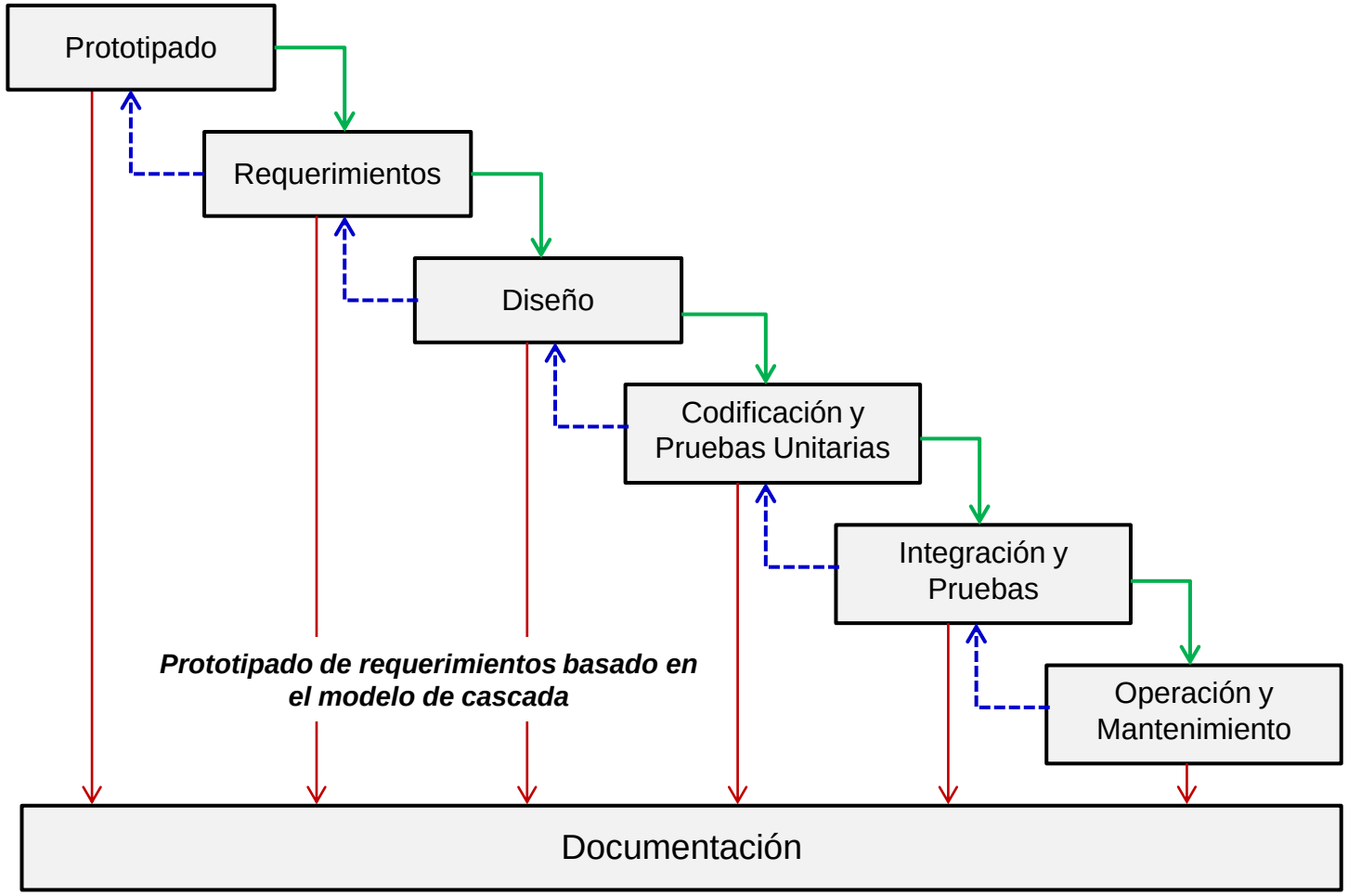
- Este modelo se suele usar como parte de la tarea de especificación de requisitos o justo antes de la misma.



Prototipado de requerimientos basado en el modelo de desarrollo incremental

CICLO DE VIDA DEL SOFTWARE (VII)

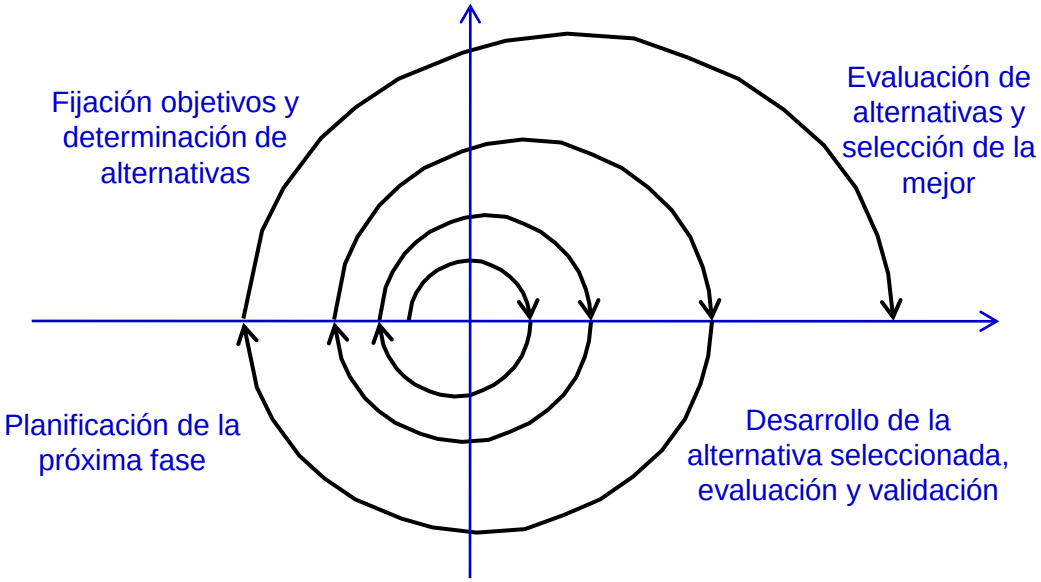
MODELO DE PROTOTIPADO DE REQUERIMIENTOS



CICLO DE VIDA DEL SOFTWARE (VIII)

MODELO ESPIRAL

- En este modelo el esfuerzo de desarrollo es iterativo. Esto es: tan pronto se completa un esfuerzo de desarrollo (una vuelta a la espiral), comienza el siguiente.
- En cada vuelta a la espiral se suelen seguir los siguientes pasos:
  1. Fijar objetivos y determinar alternativas.
  2. Evaluar las alternativas y elegir la mejor.
  3. Desarrollo de la alternativa elegida y evaluación y validación del resultado.
  4. Planificación de la próxima iteración.
- Es compatible también con el modelo en cascada.



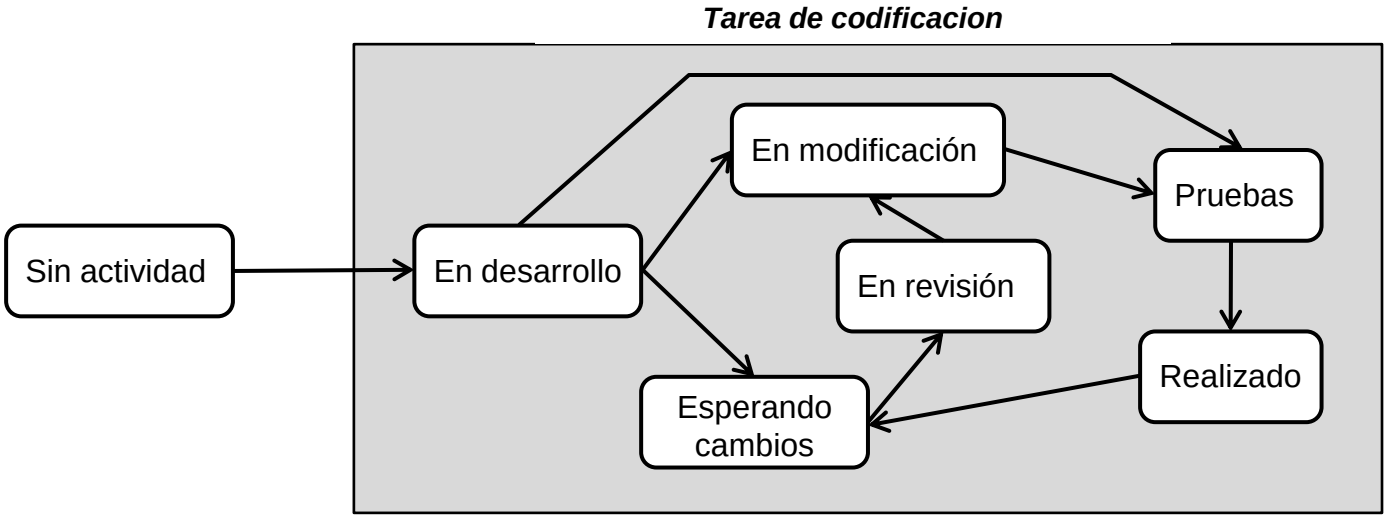
CICLO DE VIDA DEL SOFTWARE (IX)

MODELO CONCURRENTE

- Se representa de forma esquemática como una serie de tareas junto con sus estados asociados. Da respuesta a la situación habitual de los proyectos, en la que se realizan varias tareas simultáneamente, aunque se encuentran en distintos estados. Muchas veces, el mayor conocimiento del problema en la fase de diseño, de codificación, etc. pueden, por ejemplo, hacer que se replantee la especificación de requisitos mientras se sigue desarrollando.

- Este modelo define una serie de eventos que disparan transiciones de unos estados a otros para cada una de las tareas o actividades del proyecto de desarrollo software.

- Este modelo proporciona, por tanto, una visión exacta del estado actual del proyecto.



CICLO DE VIDA DEL SOFTWARE (X)

MODELO EN CASCADA MEJORADO

VENTAJAS

- Apropiado para problemas que se entienden y conocen bien.
- Produce resultados predecibles.

INCONVENIENTES

- Todos los requisitos del sistema se fijan al comienzo del desarrollo.
- Poca flexibilidad para introducir cambios.
- Hasta el final no hay nada terminado.
- Puede quedar obsoleto con rapidez.

MODELO DE DESARROLLO INCREMENTAL

VENTAJAS

- Proporciona resultados en plazos más razonables que el modelo en cascada.
- Reduce el riesgo del proyecto.

INCONVENIENTES

- Si no se seleccionan bien los requisitos a implementar en cada fase, podemos llegar a una fase incremental en la que haya que rehacer partes importantes del software y/o del diseño.

CICLO DE VIDA DEL SOFTWARE (XI)

MODELO DE DESARROLLO EVOLUTIVO

VENTAJAS

- Reconoce la naturaleza evolutiva de la mayoría de los proyectos de desarrollo software.
- Proporciona resultados en plazos más razonables que el modelo en cascada.
- Hay que implicar a los usuarios.

INCONVENIENTES

- Es difícil predecir el coste y duración de un proyecto, por lo que es conveniente limitar el número de fases, recursos y plazos si es un proyecto con principio y fin.

MODELO DE PROTOTIPADO DE REQUERIMIENTOS

VENTAJAS

- Útil cuando es difícil conocer los requisitos exactos.
- Hay que implicar a los usuarios.
- Reconoce la naturaleza evolutiva de la mayoría de los proyectos de desarrollo software.

INCONVENIENTES

- Puede resultar costoso si hay que reiniciar el desarrollo.
- Hay que mantener muy bien la documentación del proyecto para facilitar el control de versiones y su mantenimiento.

CICLO DE VIDA DEL SOFTWARE (XII)

MODELO DE ESPIRAL

VENTAJAS

- Reconoce la naturaleza evolutiva de la mayoría de los proyectos de desarrollo software.

INCONVENIENTES

- Es difícil predecir el coste y la duración del proyecto, por lo que es conveniente limitar el número de fases, recursos y plazos si es un proyecto con principio y fin.

MODELO CONCURRENTE

VENTAJAS

- Modela mucho mejor la naturaleza del proceso de desarrollo software.
- Proporciona una visión exacta de la situación del proyecto.

INCONVENIENTES

- Su implementación y gestión son complejas.