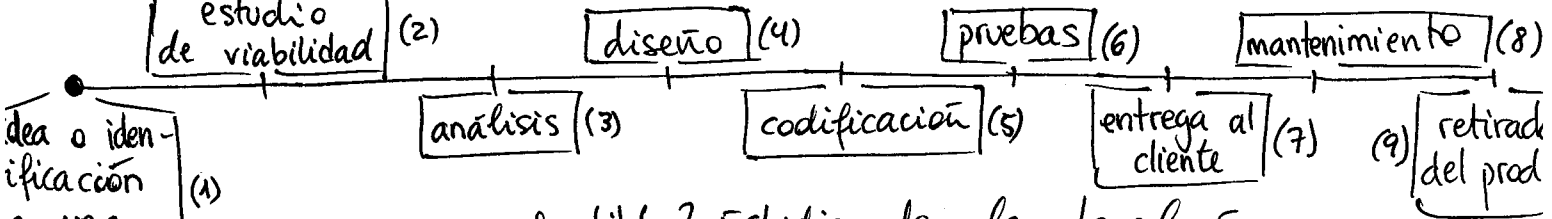
TEMA 1 - CICLO DE VIDA DEL SOFTWARE

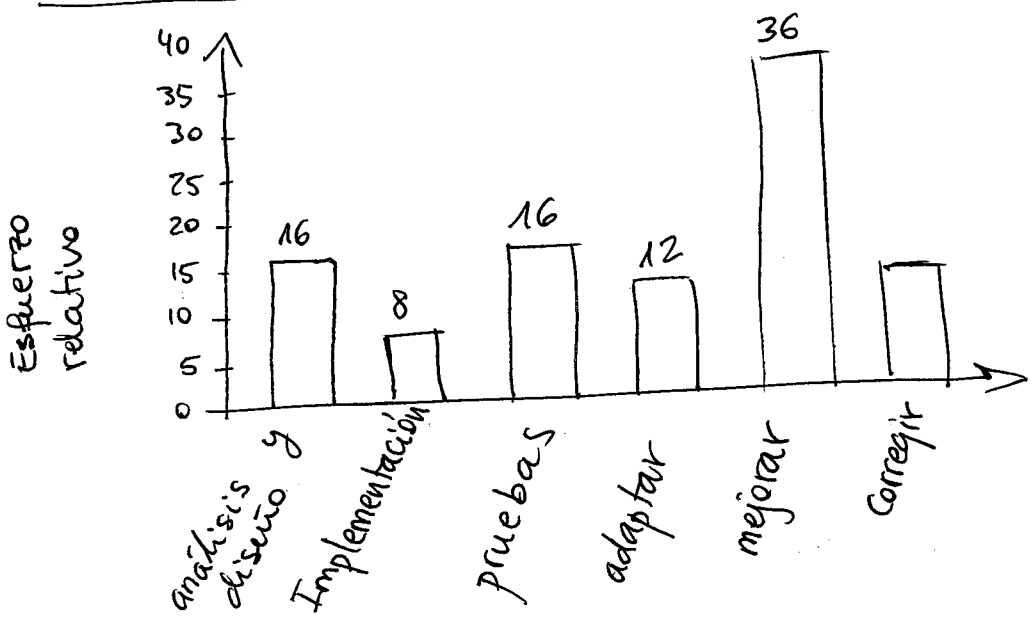
SOFTWARE = programas + datos + documentación

CICLO DE VIDA DEL SOFTWARE: fases por las que pasa el software desde el inicio de su construcción hasta su retirada.



(2) → ¿es factible? Estudio de la tecnología
(3) → ¿qué hay que hacer? ¿cuáles son los requisitos?
(4) → ¿cómo hay que hacerlo? diseño de alto y bajo nivel.
(5) → programar la aplicación de acuerdo al diseño.
(6) → probar el sistema
(8) → mantenimiento, soporte, mejorar, adaptarse, ...

DISTRIBUCIÓN DE ESFUERZO DURANTE EL CICLO DE VIDA



ESTUDIO DE VIABILIDAD

Anális técnico, operacional y económico previo a un proyecto para determinar si este es rentable. Decisión de continuidad o no del proyecto, así como los riesgos que conlleva la ejecución del mismo.

Viabilidad → técnica: funcionalidad, rendimiento y restricciones
Viabilidad → económica: costes y beneficios.
Viabilidad → legal: infracción, vidación o ilegalidad del sistema.

Alternativas → evolución de alternativas al desarrollo del sistema.

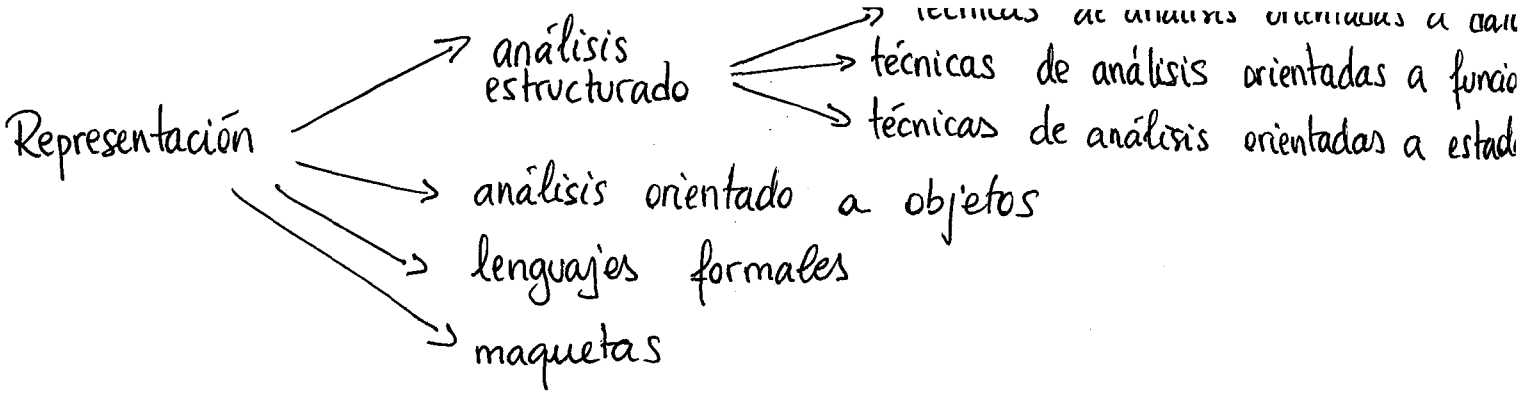
ANÁLISIS DE REQUISITOS

¿Qué hay que hacer? ¿Qué funcionalidad hay que implementar?

Tareas → captura de requisitos: que se obtienen de los usuarios.
Tareas → análisis del problema y de los requisitos: razonar sobre los requisitos, combinar relacionados, establecer prioridades entre ellos, determinar su viabilidad, etc.
Tareas → modelización: registrar los requisitos de alguna forma, incluyendo lenguaje natural, lenguajes formales, modelos, maquetas, etc.
Tareas → validación: examinar inconsistencias entre requisitos, determinar la corrección, ambigüedad, etc. Establecer criterios para asegurar que el software reúne los requisitos cuando se haya producido. El cliente, usuario y desarrollador se deben poner de acuerdo.

Tipos → Funcionales: acciones fundamentales que tienen que tener lugar en la ejecución del software.

Tipos → No funcionales:
- operacionales (backups, recuperación)
- seguridad (niveles de acceso, protección)
- mantenimiento y portabilidad (Windows & Linux)
- recursos (memoria, almacenamiento...)
- rendimiento (tiempo de respuesta, nº usuarios, ...)
- user interface & usability (size of visual elements)
- confiabilidad (fallos, situaciones anormales...)



ANÁLISIS ESTRUCTURADO: DIAGRAMA DE FLUJO DE DATOS (DFDs)

Muestra el flujo de información y las transformaciones de los datos al moverse desde la Entrada a la Salida.

Compuesto de:

- Flujos de datos: conjunto de valores del sistema en un momento determinado.
- Procesos: transformación de los flujos de entrada en los flujos de salida.
- Almacenes de datos: datos que se guardan para ser usados por uno o más procesos.
- Entidades externas: productor o consumidor de datos que reside fuera del sistema que refleja el DFD, pero que está relacionado con él.

Datos
- información que necesita la aplicación.
- diccionario de datos.
- diagramas entidad-relación.

ANÁLISIS ORIENTADO A OBJETOS

- Casos de Uso: conjunto de escenarios que describen distintas formas de usar el software, desde el punto de vista de cada tipo de usuario.

- Escenarios: secuencias de iteraciones que describen condiciones de éxito o fracaso (errores).

- Actores: elementos activos externos (usuarios, otro sistema) que interacciona con el sistema.

Maquetas → captura de requisitos en forma de interfaces que permitan un mejor entendimiento con el usuario.
→ desde programas de dibujo hasta apps especializa

DISEÑO

¿Cómo hacerlo? Del dominio del problema pasamos al dominio de la solución.

- Diseño de arquitectura: definición de los componentes del sistema y de sus interfaces.
- Diseño detallado: descripción detallada de la lógica de cada módulo, de las estructuras de datos que utilizan y de los flujos de control.
- Principios básicos → abstracción
  → refinamiento
  → modularidad
  → ocultación de información.

- diseño estructurado vs diseño orientado a objetos.

Diseño estructurado: jerarquía de llamadas entre funciones, y paso de parámetros.

Diseño orientado a objetos

→ Estructura: diseño de clases
→ Comportamiento: de cada clase, de interacciones entre objetos.
→ UML: unified modelling language.
→ Tipos → diagramas de clase
→ máquinas de estado
→ diagramas de secuencia

CODIFICACIÓN

Traducir las especificaciones de diseño a un lenguaje de implementación (C, Java, etc.)

Incluye también:
- pruebas de unidad (pruebas de cada función o método por separado para ver que funcionan).
- Manual técnico: Javadoc.
- Guías de estilo

Programación estructurada vs orientada a objetos.

PRUEBAS

- Ejecutar el programa para encontrar errores: maximizar la probabilidad de encontrar errores.
- Error de software: cuando el programa no hace lo esperado.

Tipos → caja blanca:
→ ejercitar el programa teniendo en cuenta su lógica.
→ Se ejecutan:
- todas las sentencias (al menos una vez).
- todos los caminos independientes de cada módulo.
- todas las decisiones lógicas
- todos los bucles.

Tipos → caja negra: considera el software como una caja negra. Se ejecutan: producidos por los datos de E/S.

Estrategias de pruebas

- pruebas unitarias: comprueba la lógica, funcionalidad y si es correcta la especificación de cada módulo.
- pruebas de integración: tiene en cuenta la agrupación de módulos y el flujo de información entre las interfaces.
- pruebas de validación: comprueban la concordancia respeto a los requisitos.
- pruebas del sistema: se integra con su entorno hardware y software.
- pruebas de aceptación: que el producto se ajusta a los requisitos del usuario.

MANTENIMIENTO

Actividades que la empresa desarrolladora realiza sobre el software una vez que este está operativo (después de la entrega).

Modificaciones necesarias para cumplir con nuevos o antiguos requisitos.

Tipos

- correctivo: corregir errores $\simeq$ 20%
- adaptativo: acomodar a nuevo entorno $\simeq$ 25%
- preventivo: prevenir errores $\simeq$ 5%
- perfectivo: mejorar, expandir requisitos implementac $\simeq$ 50%.

ACTIVIDADES DE GESTIÓN

- Predicción de duración, esfuerzo y costes para realizar el proyecto.
- Planificación: selección de una estrategia, coordinador, ...
- Negociación
- Seguimiento del proyecto.
- Gestión
- Coordinación del equipo de trabajo.

MODELOS DE CICLO DE VIDA

Salvo en casos muy especiales y sencillos, la construcción del software no sigue una distribución lineal de fases.

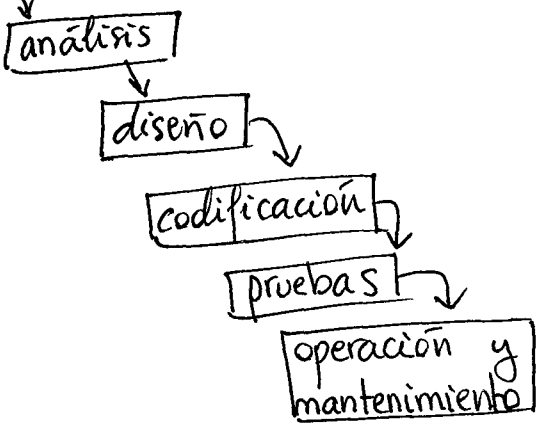
A veces es más conveniente realizar iteraciones, o incremento.

Un modelo de ciclo de vida se diferencia de los demás dependiendo de estas características:

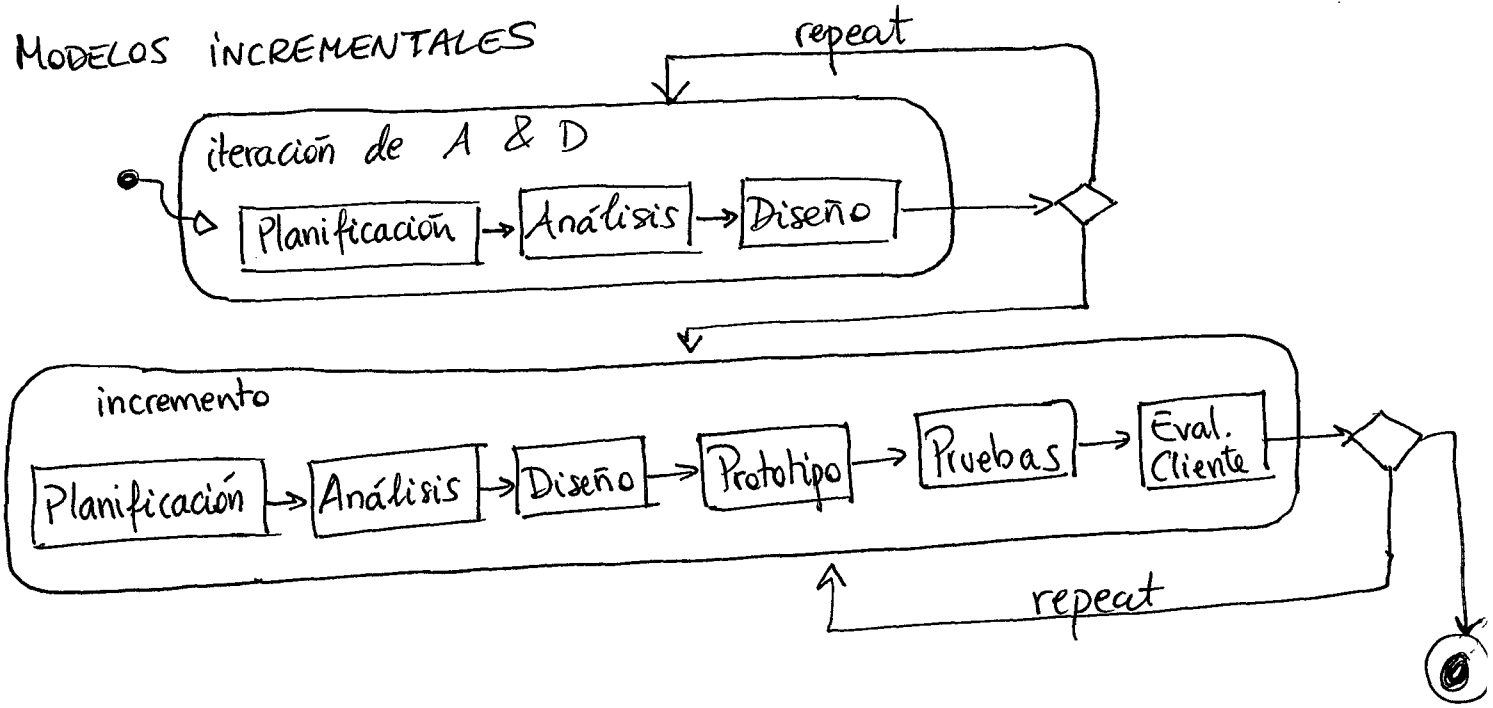
- fases por las que pasa el proyecto.
- criterios de transición de una fase a la siguiente.
- entradas y salidas de cada fase.

1. CASCADA

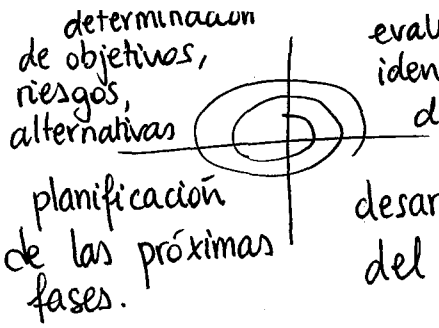
- Adecuado sólo para proyectos muy simples.
- Desventajas:
  - no se permiten las iteraciones.
  - los requisitos se congelan al principio del proyecto.
  - no existe un proyecto "enseñable" hasta el final.



2. MODELOS INCREMENTALES



3. ESPIRAL



4. PROTOTIPADO