Tema 2. Análisis y Diseño OO mediante UML.

Ciclo de Vida

Índice:

- Introducción
- Ciclo de vida en cascada
- Ciclo de vida basado en prototipos
- Ciclo de vida evolutivos
- Métodos ágiles
- Metodologías

- ¿Cómo desarrollarías un bingo o poker o una tienda de animales?
  - ¿Qué funcionalidades tiene?
  - ¿Qué clases voy a tener?
  - ¿Qué usuarios van a existir?
  - ¿Como consigo que el código sea fácilmente mantenible?
  - ¿Qué tipo de aplicación va a ser: local, web, cliente – servidor?
  - ¿Va a tener asociada una base de datos?¿qué datos voy a almacenar?
  - ¿Qué interfaz es la más intuitiva y sencilla para la aplicación?
  - ……

Objetivos generales

- Ingeniería del Software. Conceptos
- Ciclo de Vida. Conceptos
- Introducción a los distintos ciclo de vidas más usados
- Introducción a las metodologías básicas
- Estudiar los puntos débiles y fuertes e inconvenientes de las metodologías básicas

1. Introducción

- La Ingeniería de Software (IS) es
  - una disciplina de ingeniería
    - ✓ Aplicación de teorías, métodos, herramientas para hacer cosas que funcionen:
      - Software que sea fiable y trabaje en máquinas reales
    - ✓ Teniendo en cuenta restricciones financieras, organizacionales y técnicas
  - que comprende todos los aspectos de la producción de software
    - ✓ Desde la especificación inicial al mantenimiento del sistema
    - ✓ Administración y gestión del proceso de producción
      - Principios y metodologías para desarrollo y mantenimiento de sistemas de software
  - Abarca además, del proceso de desarrollo del software, etapas de planificación y gestión de proyectos

1. Introducción

- El **proceso de desarrollo de software** requiere por un lado un conjunto de conceptos, una metodología y un lenguaje propio. A este proceso también se le llama el **ciclo de vida** y esta formado:
  - Métodos: nos indican cómo desarrollar el software. Abarcan un conjunto de tareas:
    - ✓ Planificación o especificación de requisitos
    - ✓ Análisis de requisitos (software y hardware)
    - ✓ Diseño de estructuras de datos
    - ✓ Codificación
    - ✓ Pruebas
    - ✓ Mantenimiento
  - Herramientas: suministran un soporte automático o semiautomático para los métodos. Herramientas cuya información generada puede ser utilizada por otra herramienta => sistema de soporte CASE
  - Procedimientos: Definen la secuencia en la que aplicar los métodos, las entregas de documentos, informes, controles de calidad, herramientas, …

2. Ciclo de Vida de un Proyecto de Desarrollo de Software

➢ Definición de Ciclo de Vida:

Conjunto de tareas, etapas, fases o actividades que hay que llevar a cabo para desarrollar un proyecto software.

Distintos Ciclos de Vida constituyen los diferentes *paradigmas* (maneras de resolver un problema) de la Ingeniería del Software.

2. Ciclo de Vida de un Proyecto de Desarrollo de Software

- ¿Por qué utilizar un ciclo de vida?
  - Porque hay que organizar un gran número de actividades, que hay que hacer, y en cierto orden
  - Tener definidos los pasos a dar ayuda a resolver los problemas que se van presentando durante el desarrollo del sistema
  - Se va generando una documentación sobre el estado del proyecto y su seguimiento

2. Ciclo de Vida de un Proyecto de Desarrollo de Software

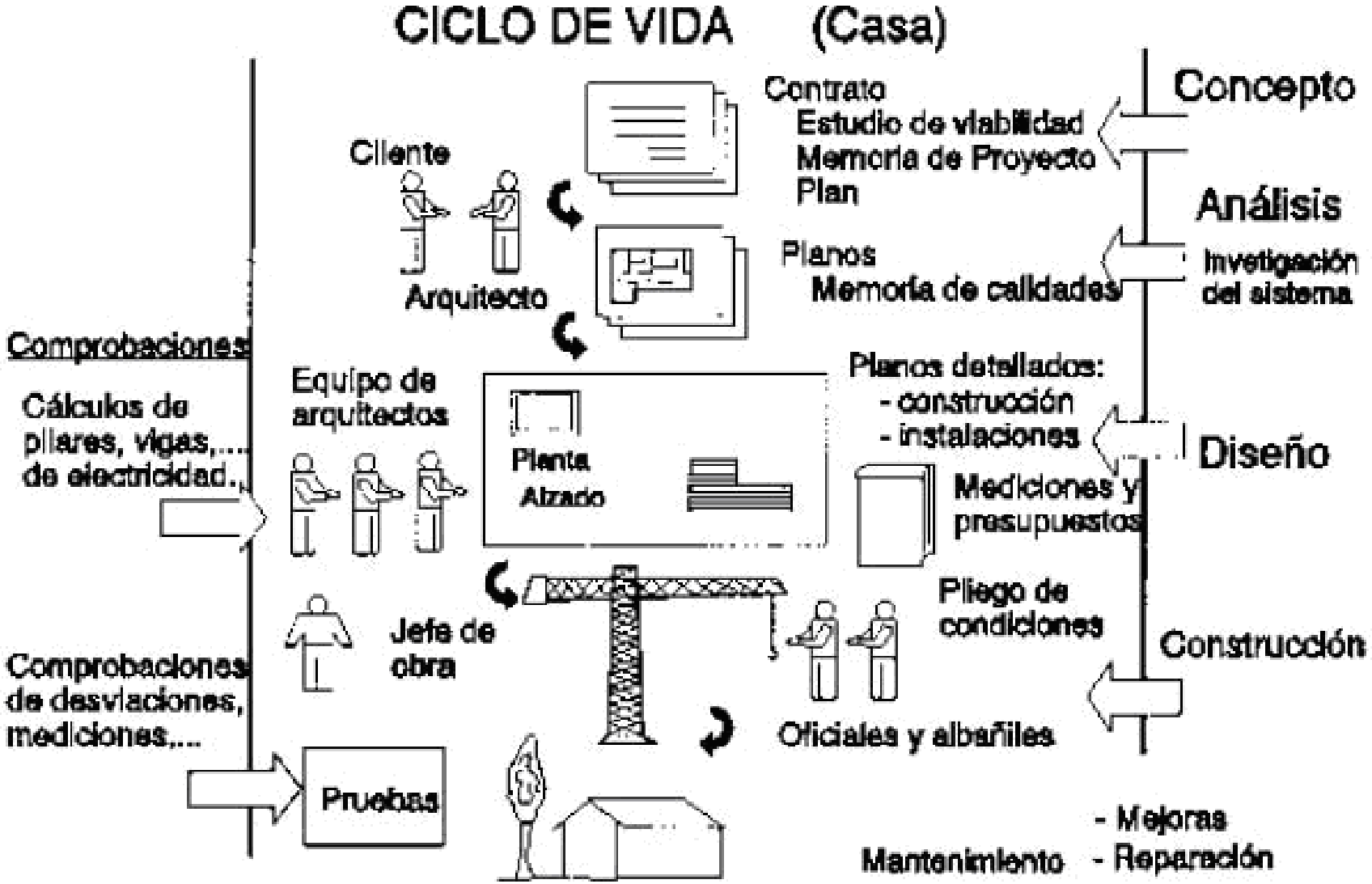
- ¿Cuáles serían a modo de ejemplo las principales fases de un proceso de ingeniería (software o no software)?
  - **1.- Planificación**
  - **2.- Análisis**
  - **3.- Diseño**
  - **4.- Implementación o Construcción**
  - **5.- Mantenimiento**

2. Ciclo de Vida de un Proyecto de Desarrollo de Software

- Principales fases de un proceso de ingeniería NO software
  - **1.- Planificación:** El ingeniero recoge información del problema, establece los requerimientos y las restricciones, esboza varias posibles soluciones genéricas y elige la más adecuada.
  - **2.- Análisis:** En esta fase se desarrolla la solución sin entrar en detalles constructivos. (Elaboración de los planos generales de distribución, tamaños aproximados, calidades genéricas, etc. - Maqueta del nuevo sistema-).
  - **3.- Diseño:** Desarrollo concreto de la solución. (Construcción de los planos a escala, generales y de detalle -columnas, vigas...-, cálculos, materiales, condiciones eléctricas, de agua, de gas,....)
  - **4.- Implementación o Construcción:** Realización del diseño. (El constructor lleva a cabo la obra de acuerdo al diseño, eligiendo en este caso marcas de materiales, colores, que cumplan las especificaciones)
  - **5.- Mantenimiento:** Durante esta fase se llevan a cabo las labores de reparación, modificación del sistema (la obra), según las necesidades de los usuarios.

2. Ciclo de Vida de un Proyecto de Desarrollo de Software

- Ciclo de vida de un proyecto NO software



2. Ciclo de Vida de un Proyecto de Desarrollo de Software

- Principales fases de un proceso de ingeniería software
  - 1.- Planificación
  - 2.- Análisis
  - 3.- Diseño
  - 4.- Implementación
  - 5.- Pruebas
  - 5.- Mantenimiento

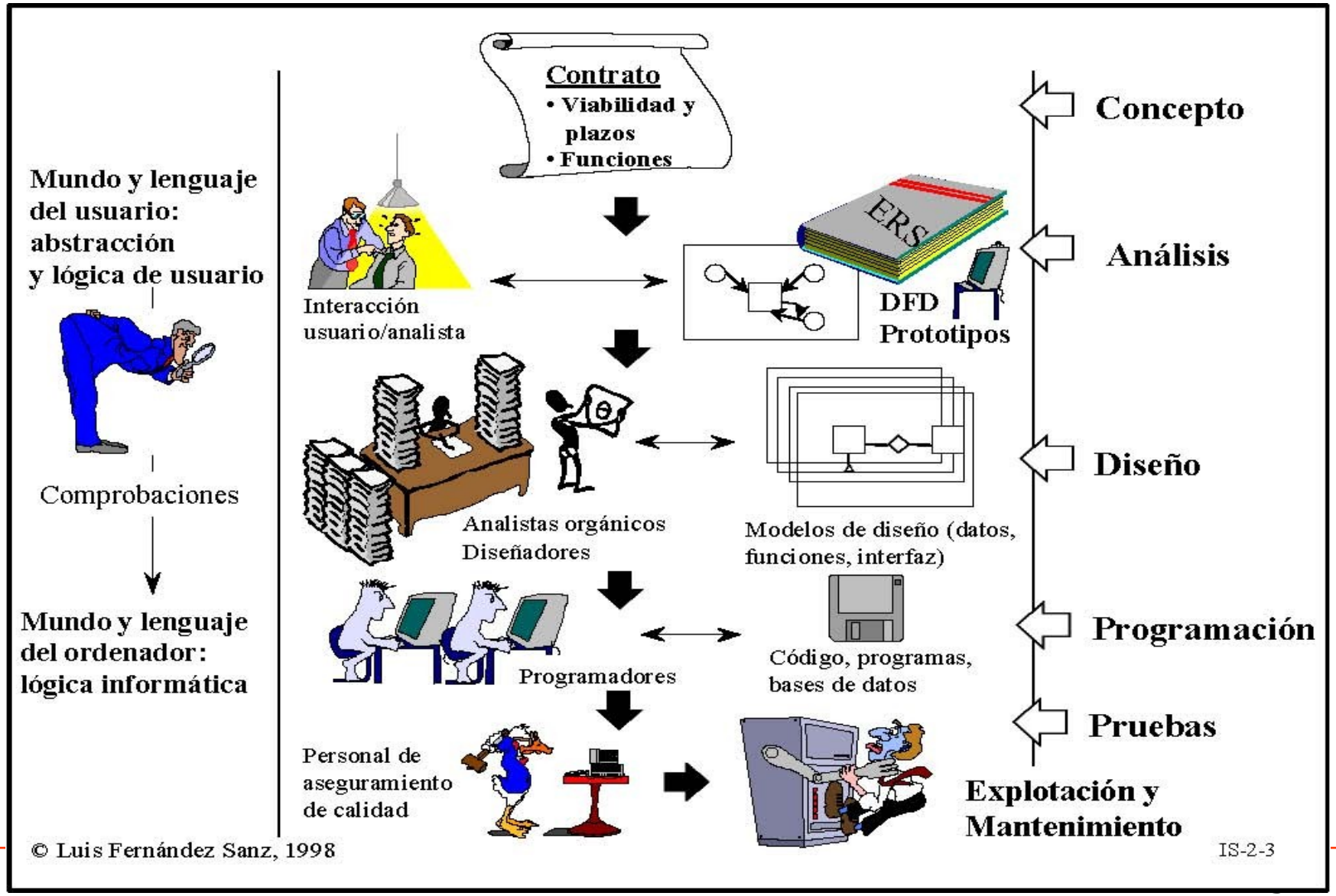
¿Qué pensáis que se debe hacer en cada fase?

2. Ciclo de Vida de un Proyecto de Desarrollo de Software

- Principales fases de un proceso de ingeniería software
  - 1.- Planificación
    - ✓ Ámbito del proyecto, Estudio de viabilidad técnica, Estudio operacional, Estudio de viabilidad económica (Análisis de riesgos, coste, Planificación temporal, Asignación de recursos.)
  - 2.- Análisis (Qué)
    - ✓ Requerimientos: funcionales y no funcionales
    - ✓ Modelado: de datos y de procesos
  - 3.- Diseño (Cómo)
    - ✓ Estudio de alternativas y diseño arquitectónico
      - Diseño de la base de datos y de las aplicaciones
  - 4.- Implementación o Construcción
  - 5.- Pruebas
    - ✓ Pruebas de unidad, integración, alfa, beta, Test de aceptación
  - 6.- Mantenimiento

2. Ciclo de Vida de un Proyecto de Desarrollo de Software

- Ciclo de vida de un proyecto software



2. Ciclo de Vida de un Proyecto de Desarrollo de Software

- Ciclo de vida en cascada
- Ciclo de vida basado en prototipos
- Ciclos de vida evolutivos
  - Modelo Iterativo Incremental
  - Modelo en Espiral
- Ciclos de vida ágiles
  - Extreme Programming (XP)
  - SCRUM

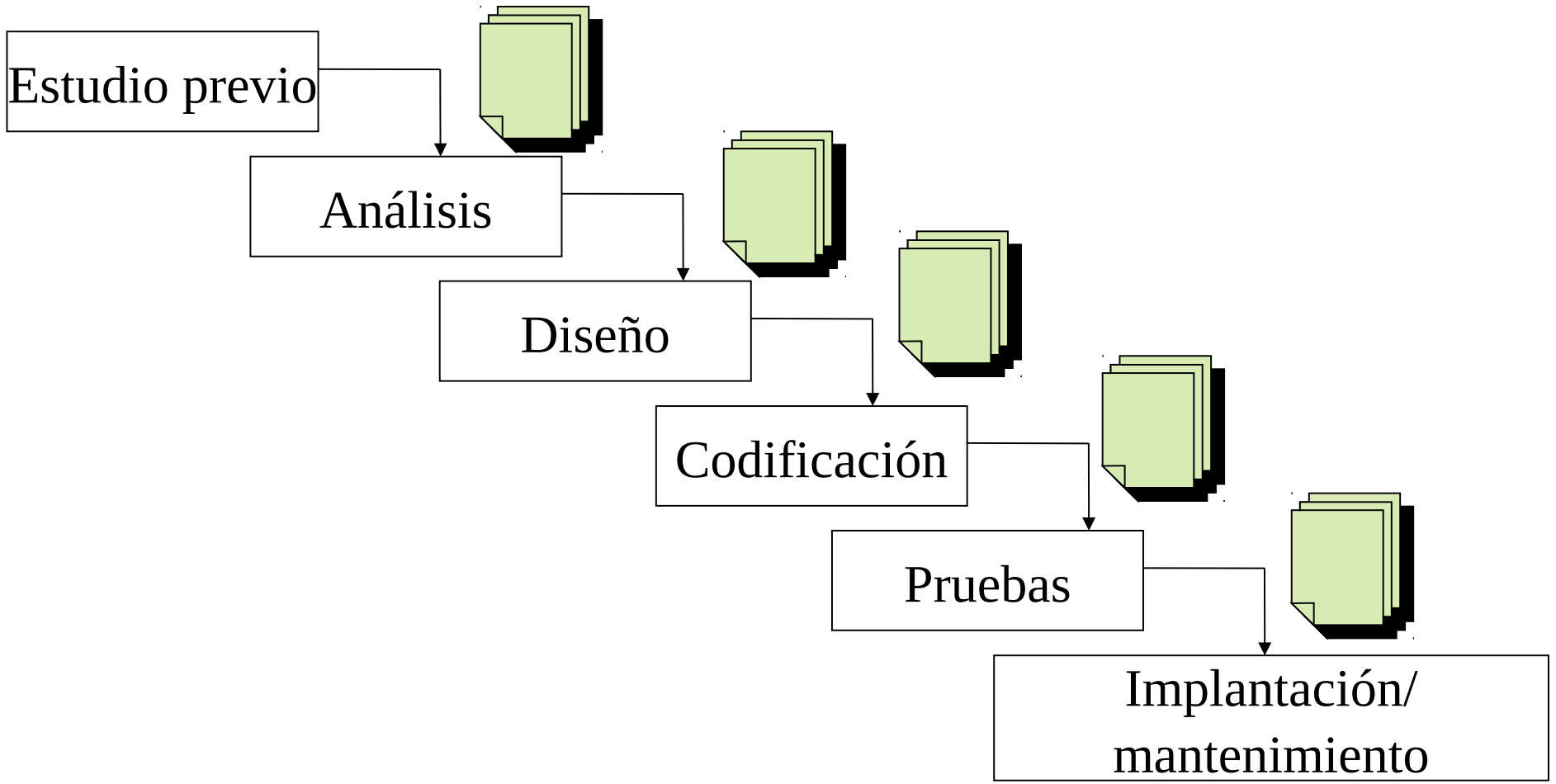
- Planificación de clase
  1. Formar 7 grupos
  2. Buscar información sobre el modelo asignado (30 minutos)
     1. Descripción
     2. Motivación
     3. Ventajas
     4. Inconvenientes
  3. Preparar una pequeña presentación (10 minutos)
  4. Discusión sobre algunos casos prácticos

- Casos prácticos
  - Sistema de control antibloqueo de frenos en un coche
  - Sistema de sensorización y control de una explotación agrícola
  - Sistema de contabilidad que debe reemplazar a un sistema existente

3. Modelo Ciclo de Vida en Cascada

- Características
  - Mas antiguo, clásico y más utilizado
  - Punto de partida para otras metodologías
  - Usuario expone todas sus necesidades con detalle al comienzo del proyecto
  - Las distintas etapas se realizan **secuencialmente,** es decir, cada etapa comienza cuando termina el anterior
  - También denominado en **Cascada**

3. Modelo Ciclo de Vida en Cascada: Etapas



3. Modelo Ciclo de Vida en Cascada: Etapas

Críticas:

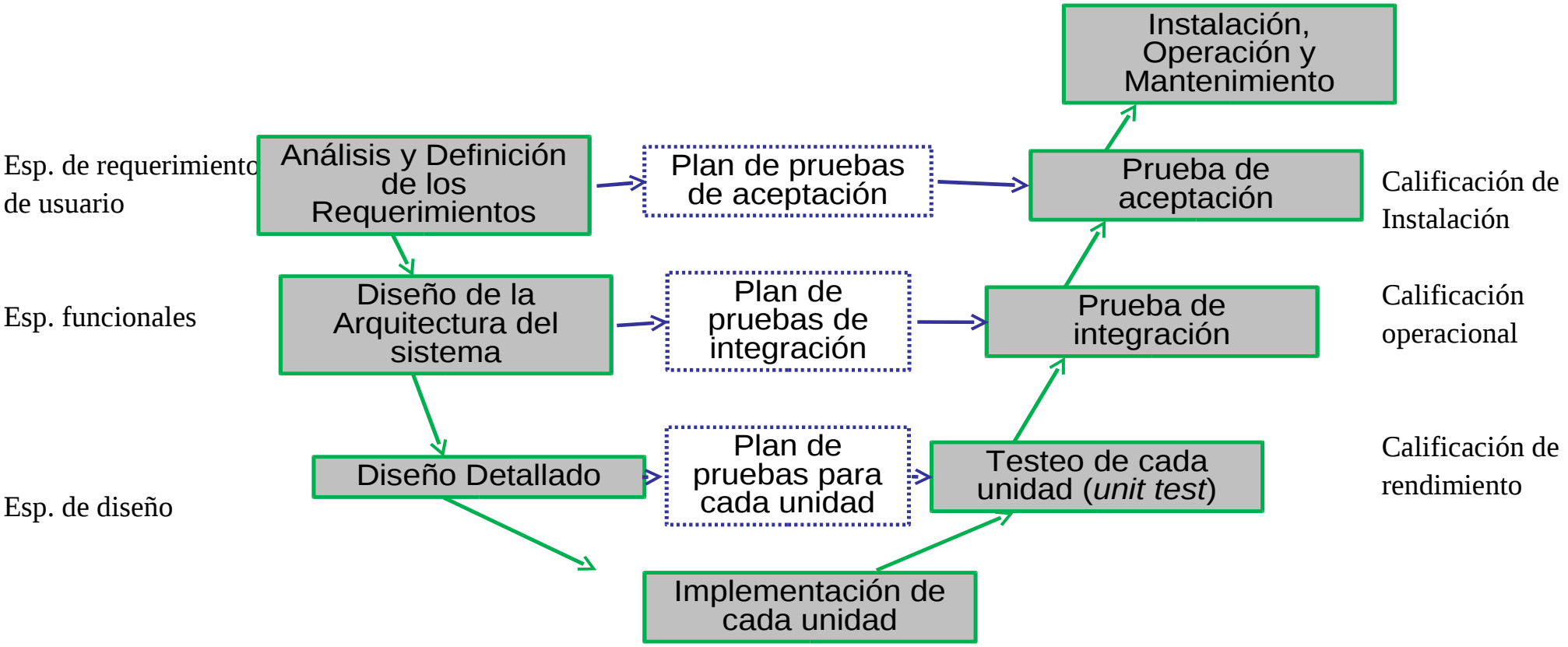
- Proyectos reales no suelen seguir y mantener la secuencialidad
- Es muy difícil que el usuario aporte toda la información al inicio
- Se tarda mucho tiempo en detectar errores
- Se tarda mucho y es muy costoso corregir errores
- El cliente tarda mucho tiempo en ver el producto

Ventajas:

- Es muy utilizado a pesar de las críticas
- Si se mantiene la secuencialidad, es el mejor

El modelo cascada, en algunas de sus variantes, es uno de los actualmente más utilizados

3. Modelo Ciclo de Vida V



El lado izquierdo de la V representa la descomposición de las necesidades, y la creación de las especificaciones del sistema.

El lado derecho de la V representa la integración de las piezas y su verificación. V significa «Verificación y validación»

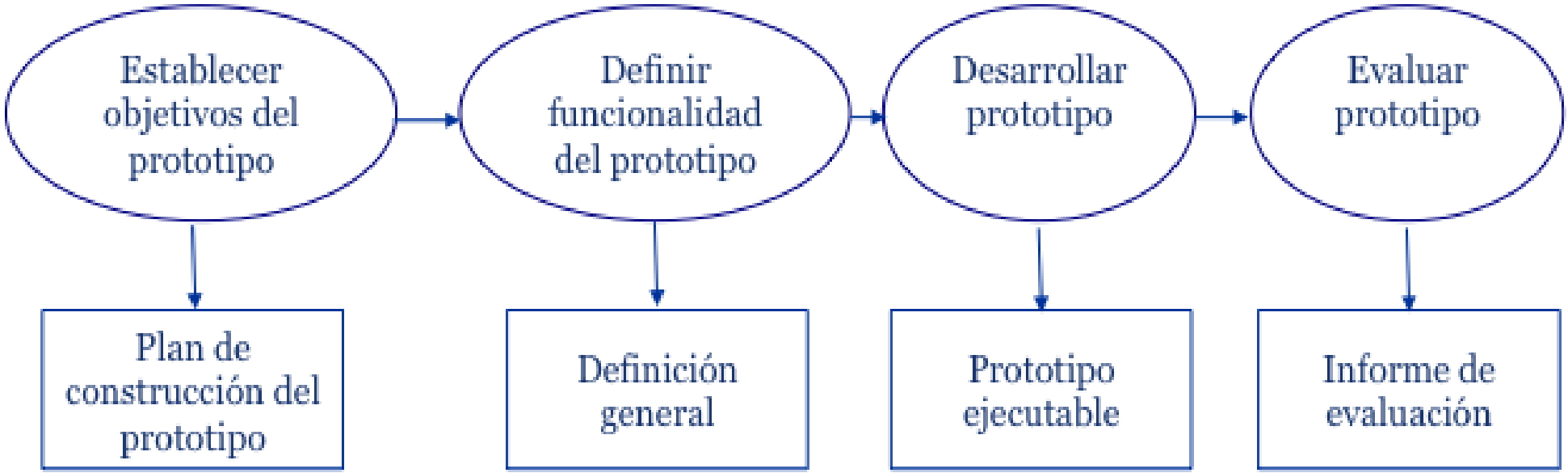
3. Modelo Ciclo de Vida V

- Los siguientes objetivos están destinados a ser alcanzados durante la ejecución del proyecto:
  - Minimización de los riesgos del proyecto: Permite una detección temprana de las desviaciones y los riesgos y mejora la gestión de procesos, reduciendo así los riesgos del proyecto.
  - Mejora y Garantía de Calidad: Los resultados provisionales definidos se puede comprobar en una fase temprana.
  - Reducción de los gastos totales durante todo el proyecto y sistema de Ciclo de Vida: El esfuerzo para el desarrollo, producción, operación y mantenimiento de un sistema puede ser calculado, estimado y control de manera transparente mediante la aplicación de un modelo de procesos estandarizados.

3. Modelo Prototipos

- Un prototipo es una versión inicial del software que se utiliza para informarse más del problema y sus posibles soluciones
  - Es un “Diseño rápido” centrado en los aspectos visibles para el cliente
- Un prototipo se puede usar de varias maneras en un proceso de desarrollo de software:
  - Un prototipo está disponible de forma rápida y se puede enseñar al cliente
  - Obtención y validación de requisitos
  - Explorar soluciones de diseño
  - Ejecutar pruebas

3. Modelo Prototipos



3. Modelo Prototipos

- Conceptos:
  - Se han expuesto las necesidades por parte del usuario pero sin detalle
  - No tiene muy claro lo que quiere
  - Aporta información pero no la suficiente para aplicar el modelo en Cascada
  - Se dispone de herramientas:
    - ✓ Generadores de pantalla
    - ✓ Generadores de informes
    - ✓ Administradores de Bases de Datos

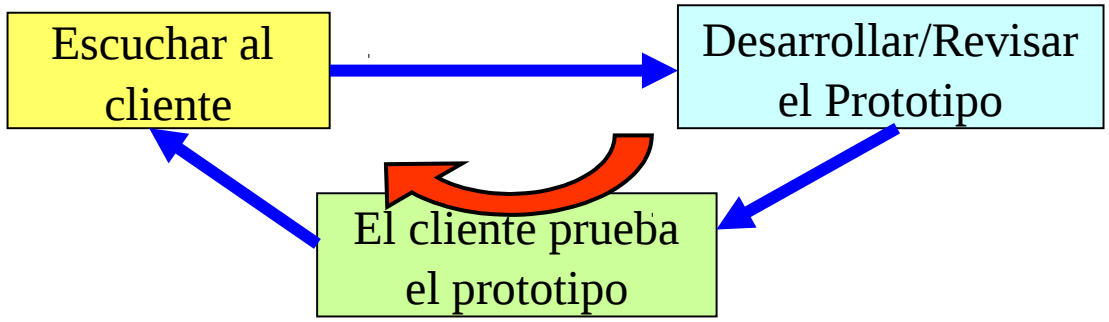
3. Modelo Prototipos

Cuando es bueno utilizarlo:

- El usuario no tiene muy claro lo que quiere:

  "*no sé qué es lo que quiero, pero lo sabré cuando lo vea*"

- El usuario se preocupa mas de la interfaz que de los procesos y cálculos que realiza el sistema.



- Tipos
  - Prototipo sobre papel
  - Funcionamiento (secuencia de ventanas)
  - Prototipo existente

4.1. Prototipo sobre papel: ejemplos

Figura 6: Ejemplo de un prototipo de papel "desplegable" perteneciente al caso de estudio de la reserva del hotel

3. Modelo Prototipos

Consideraciones:

- El cliente ve un software funcionando, pero es sólo una fachada
- Lo que ve no reúne los requisitos que desea
- A veces el cliente exige que sea lo que está viendo el producto final
- Por requisitos de tiempo se modifica y se desarrolla el sistema sobre el propio prototipo

Críticas

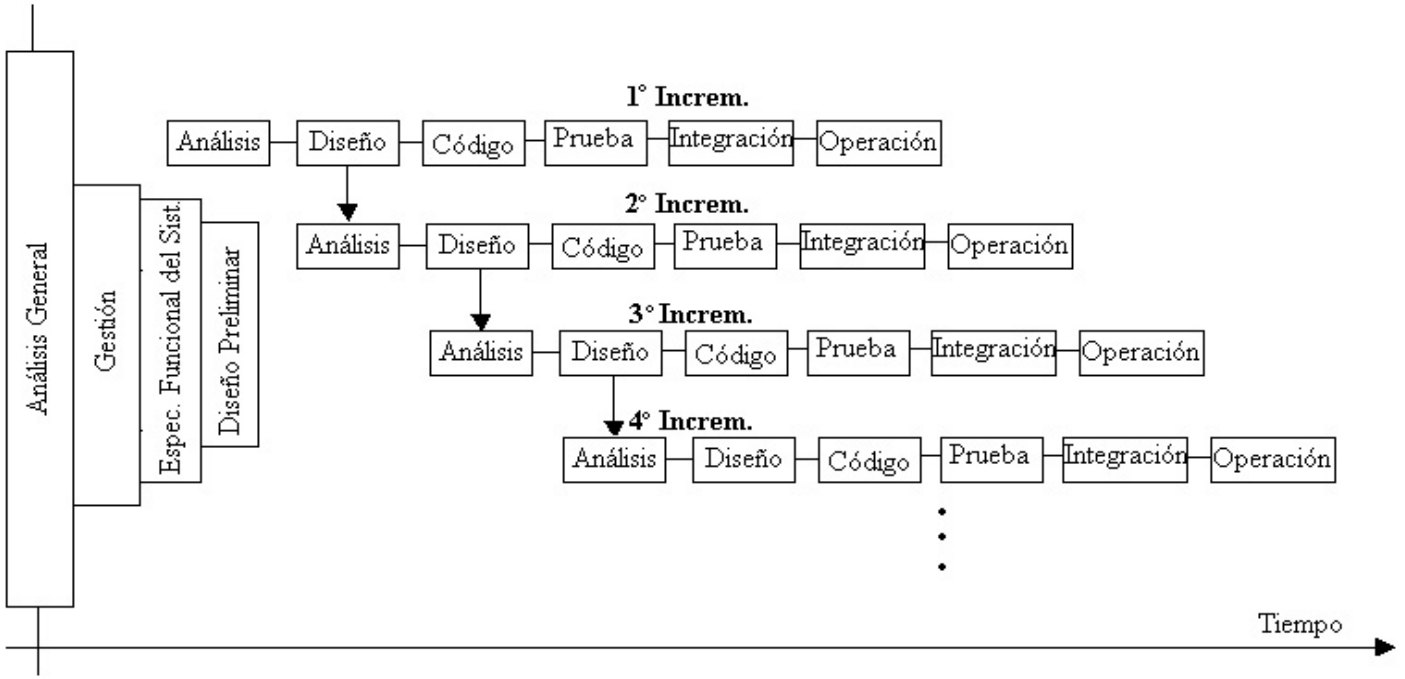
- Exige disponer de las herramientas adecuadas que permitan desarrollar prototipos en fases tempranas del desarrollo.
- Los prototipos no presentan calidad ni robustez, ya que exige tomar decisiones de diseño en fases tempranas del análisis
- Presentar un prototipo a los usuarios puede suponer presiones en la entrega que hipotequen la calidad del diseño

4. Modelo Evolutivo

- El software evoluciona con el tiempo
  - Los requisitos del usuario y del producto suelen cambiar conforme se desarrolla el mismo
- En el modelo Cascada **no tiene en cuenta** la naturaleza evolutiva del software
- Los evolutivos son modelos **iterativos**, permiten desarrollar versiones cada vez **más completas y complejas**, hasta llegar al objetivo final deseado

4.1. Modelo Evolutivo. Modelo iterativo incremental

- Permite mucha interactividad con el cliente, que va viendo versiones parciales y puede detectar problemas antes de que finalice el trabajo
- No es necesario que se termine una versión antes de comenzar la siguiente. Distintos grupos de trabajo pueden comenzar una nueva versión mientras se termina la anterior
- A diferencia de prototipos, la primera versión es perfectamente funcional y válida



4.1 Modelo Evolutivo. Modelo iterativo incremental

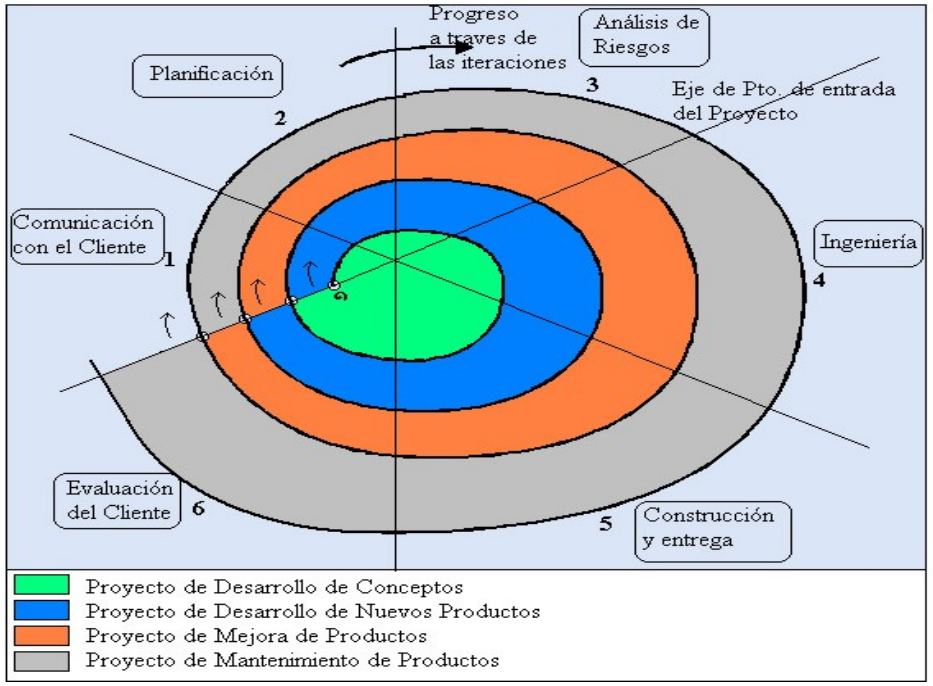
- Ventajas: la posibilidad de realizar entregas sucesivas de un producto es una idea atractiva para los desarrolladores y para los usuarios finales
  - Se evitan proyectos largos, y se van realizando entregas a los usuarios que permitir detectar errores
  - El usuario se involucra más en el desarrollo que en los modelos de desarrollo lineal
  - En este tipo de desarrollos, de equipos de trabajo en paralelo, el aumento de recursos puede hacer disminuir sensiblemente los tiempos de entrega de una manera más sencilla que en otros modelos
  - Este modelo es adecuado cuando se dispone de poco personal para el desarrollo, ya que permite acomodar las entregas a los usuarios en función de los recursos disponibles

4.1 Modelo Evolutivo. Modelo iterativo incremental

- Inconvenientes:
  - Es difícil de aplicar en algunos sistemas integrados, pues no es posible realizar diferentes entregas del producto final
  - La coordinación de los diferentes grupos de desarrollo requiere de gestores experimentados en este tipo de desarrollos
  - Es difícil poder evaluar el coste cada uno de los incrementos por el movimiento de personal y tareas de cooperación existentes de cada una de las entregas

4.2 Modelo Espiral

- Modelo evolutivo que se diferencia del incremental en que no hay solapamiento entre revisiones
- En cada vuelta se reajusta el proyecto (coste, planificación, etc.) en función de la reacción del cliente
- Cada vuelta completa representa una versión, cada vez más grande y completa



Flexibilidad ante cambios de requerimientos = más valor para el usuario

Corrección más temprana de errores = menor costo

Mejores métricas de progreso del proyecto = más control (ej., se puede modificar el plan en función del ritmo de avance logrado) = menor riesgo

4.2 Modelo Espiral:

Críticas:

- Evaluación del riesgo incorrecta => problemas
- ¿Cuándo terminamos?
  - ✓ Usuarios incapaces de definir sus necesidades
  - ✓ Entornos dinámicos en el que se mueven las organizaciones
    - ❑ Sistemas abiertos, adaptativos

- Manifiesto del desarrollo ágil de software:
  - http://agilemanifesto.org/iso/en/

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

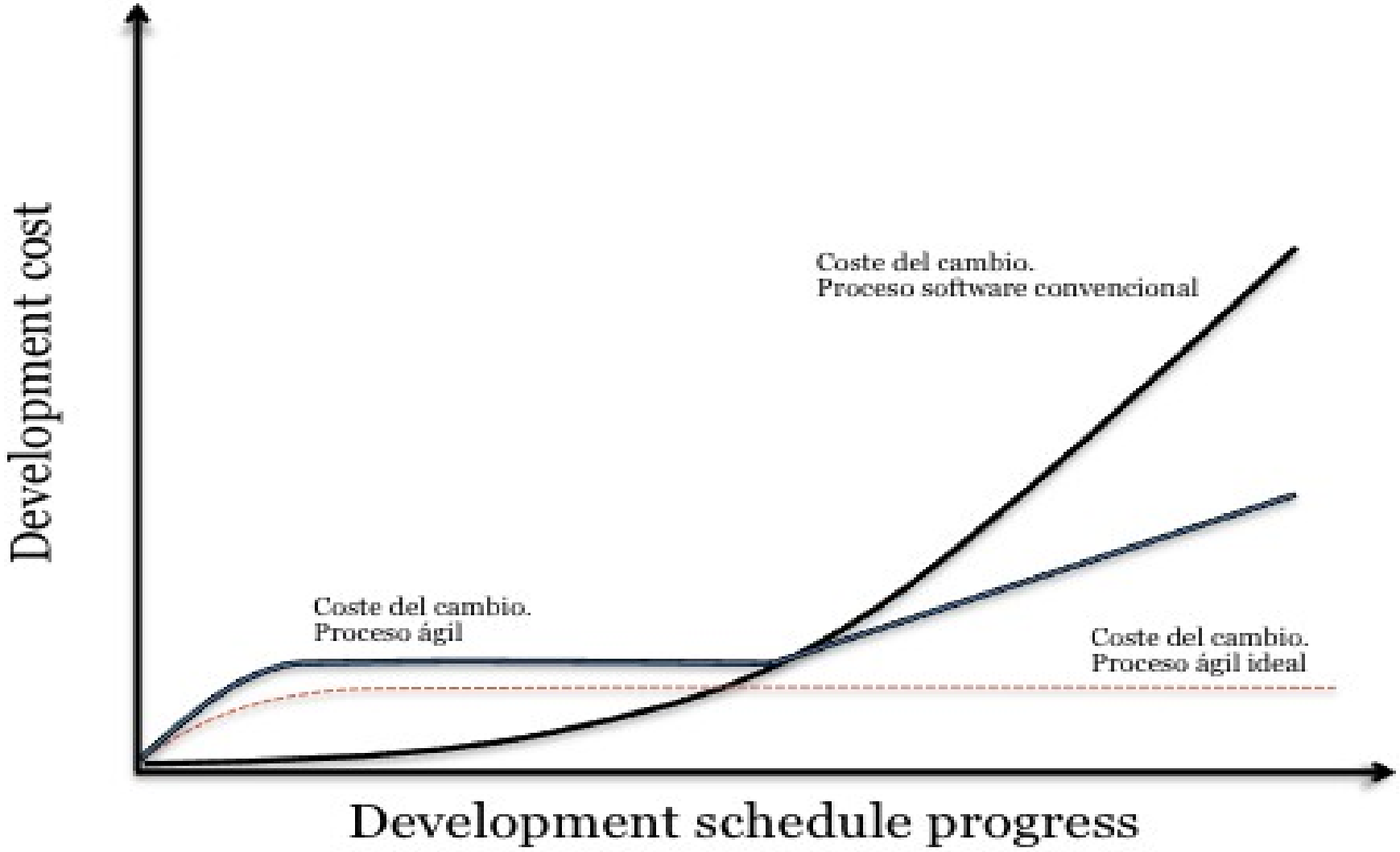
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

5. Métodos ágiles

- Los métodos ágiles combinan:
  - Filosofía
  - Fomento de la satisfacción del cliente
  - Desarrollo incremental
  - Equipos de pequeño tamaño y muy motivados
  - Métodos informales
  - Simplicidad
- Guías de desarrollo
  - Comunicación continua y activa con el cliente
  - Fomento de la entrega de partes del producto frente a análisis y diseño

- Agilidad y el coste del cambio



- Ideas sobre las que sustentan los métodos ágiles:
  - Es difícil predecir inicialmente qué requisitos software se mantendrán y cuáles cambiarán
  - Para muchos tipos de software, diseño e implementación son tareas que se entremezclan
  - Análisis, diseño, implementación y pruebas son tareas con un grado de impredictibilidad desde un punto de vista de la planificación
- Solución para abordar la impredictibilidad
  - El proceso software debe ser adaptable

5.1. Métodos ágiles. Programación extrema (XP)

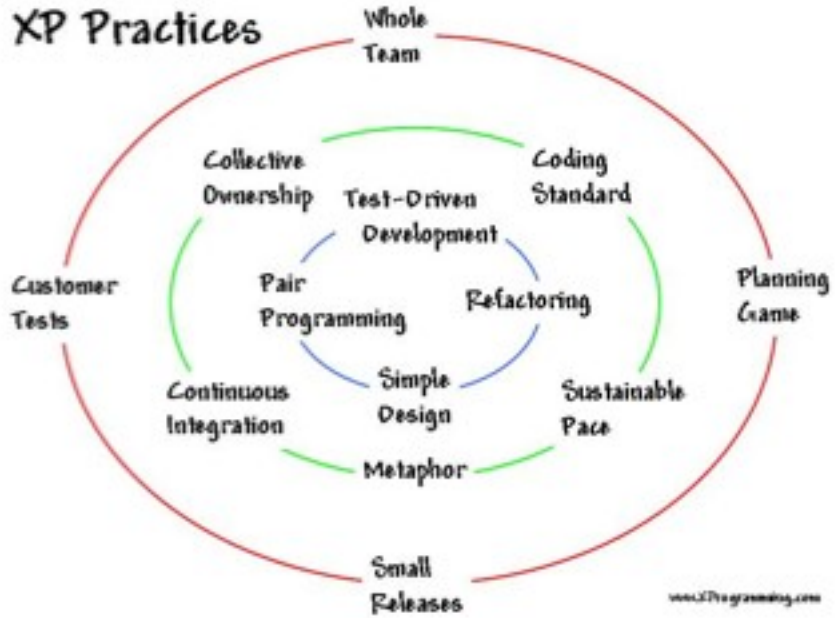
- Proceso software ágil más conocido
  - Desarrollado en 1999
  - Por Kent Beck, Ward Cunningham and Ron Jeffries

- Indicado para proyectos
  - Con requisitos cambiantes
  - De alto riesgo
  - Con equipos de desarrollo pequeños (2 – 12 personas)

- Las cuatro claves de XP
  - Comunicación
    - ✓ Entre miembros del equipo, gestores de proyecto y clientes
  - Simplicidad
    - ✓ “Do the simplest thing that could possibly work”
    - ✓ “Never implement a feature you don’t need now”
  - Retroalimentación (feedback)
    - ✓ Por el sistema (tests unitarios), por el cliente, por el equipo
  - Coraje
  - Perder el miedo a refactorizar, quitar código obsoleto, persistencia para resolver los problemas que se presentan, etc.
- Idea de funcionamiento
  - Poner a todo el equipo unido mediante prácticas simples, con suficiente feedback para hacer que vean lo que están haciendo y permitan ajustar estas prácticas a cada situación

5.1. Métodos ágiles. Programación extrema (XP)

- Características principales
  - Planificación incremental
  - Pequeñas versiones (releases)
  - Diseño simple
  - Test-driven development
  - Refactorización
  - Pair programming
  - Integración continua
  - El cliente es miembro del equipo



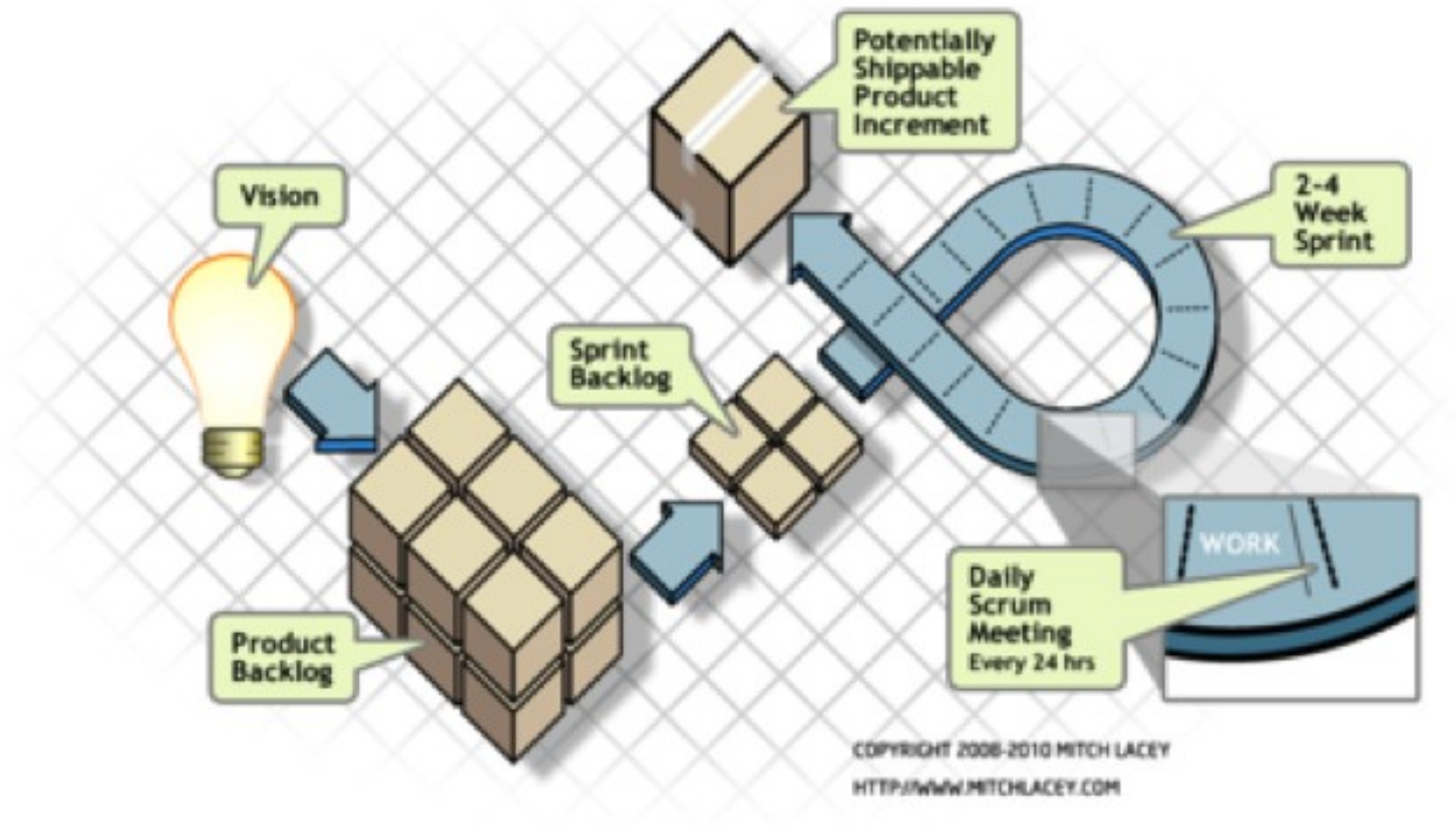
http://xprogramming.com/what-is-extreme-programming/

- Críticas al modelo
  - Modelo poco realista: muy centrado en el programador
  - No se escriben las especificaciones con detalle
  - Refactorizaciones constantes (consume tiempo)
  - Las actividades que caracterizan al proceso son demasiado interdependientes

5.2. Métodos ágiles. Scrum

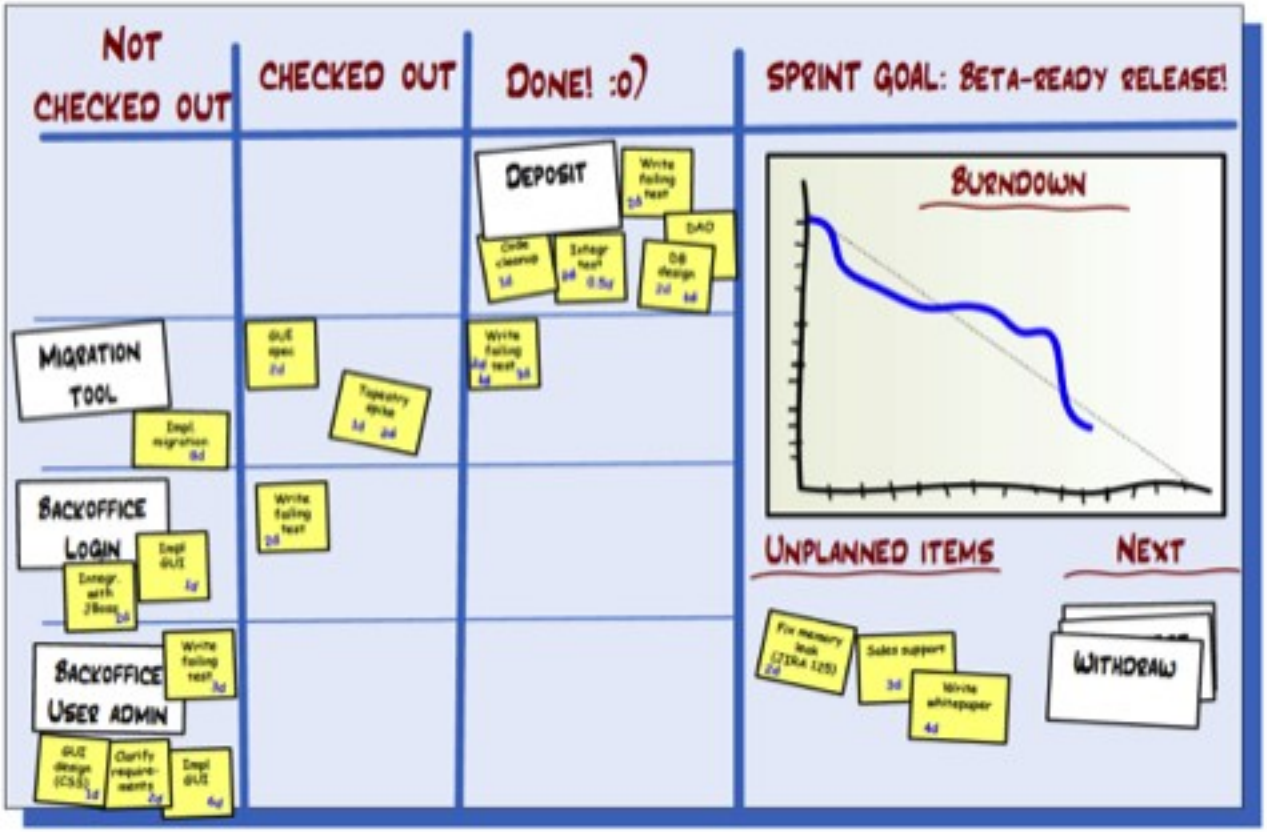
- Scrum es una metodología ágil de gestión de proyectos que apareció en los 90
- Se basa en un proceso iterativo que define un conjunto de prácticas y roles
  - Roles: scrum master, team, product owner
  - Prácticas: backlogs, sprints, meetings, demos
- Ha mostrado ser muy útil en proyectos de tamaño pequeño y mediano
- http://www.scrum.org/Resources/What-is-Scrum

5.2. Métodos ágiles. Scrum



http://www.scrumalliance.org/pages/what_is_scrum

5.2. Métodos ágiles. Scrum: Burndown chart



http://www.crisp.se/bocker-och-produkter/scrum-and-xp-from-the-trenches

6. Metodologías

- Una metodología es una colección de métodos aplicados a lo largo del ciclo de vida del desarrollo de software y unificados por alguna aproximación general
- Las metodologías se basan en una combinación de los modelos de proceso genéricos (cascada, incremental…). Para ello definen **artefactos, roles y actividades, junto con prácticas y técnicas recomendadas**
- Los enfoques más generales, que se desarrollan en varias metodologías son los siguientes:
  - Modelo en cascada:
  - Prototipado:
  - Incremental: Combinación de lineal e iterativo.
  - Espiral: Combinación de lineal e iterativo.
  - RAD: Rapid Application Development, iterativo.

- Clasificación de las metodologías
  - Diseño estructurado descendente: Yourdon y Constantine, Wirth, Dahl, Dijkstra y Hoare
  - Diseño dirigido por datos: Jackson, Warnier y Orr
  - Diseño orientado a objetos son las que siguen el modelo de objetos: Booch, OMT (Rumbaugh et al.), Objectory (Jacobson et al.), Schlaer-Mellor, Coad/Yourdon, Fusion (Coleman et al.)
  - Diseño Ágil: Scrum o Extreme Programming (XP)
  - ….

- Las metodologías (tanto comerciales como en el ámbito académico y de investigación) pueden ser agrupadas en dos grandes corrientes: Metodologías Estructuradas y Metodologías Orientadas a Objetos

6. Metodologías

- Características de los desarrollos estructurados:
  - Enfocada al proceso
  - Programación estructurada
  - Diseño estructurado (DFD)
  - Análisis estructurado
  - Especificaciones funcionales

- Características de los desarrollos orientados a objetos:
  - Trata procesos y datos de forma conjunta
  - Abstracción, ocultación de información y modularidad
  - La gran mayoría utilizan UML para la representación
  - Las técnicas estructuradas han influido en estas metodologías

Bibliografía

- Preguntas tipo test
  - http://tests.rincondelvago.com/informatica_ingenieria_del_software/ (pueden que todas no estén correctas)
- Ingeniería de software. Ian Sommerville. Editorial Addison Wesley.
- Ingeniería del software. Un enfoque práctico, R. S. Pressman. Editorial McGraw Hill Higher Education
- Bertrand Meyer. “Construcción de Software Orientado a Objetos”. 2ª edición. Editorial Prentice, 1999. ISBN: 84-8322-040-7.
- Jacobson, Booch, Rumbaugh. “El lenguaje unificado de modelado”. Editorial Addison Wesley, 1999. ISBN: 0-201-57168-4.
- A. Weitzenfeld. "Ingeniería de Software Orientada a Objetos con UML. Java e Internet. Editorial Thomson 2004. ISBN: 970-686-190-4.
- Henrik Kniberg, Scrum and XP from the Trenches, Enterprise Software Development Series, online version at http://www.crisp.se/bocker-och-produkter/scrum-and-xp-from-the-trenches