



Tema 1

Ciclo de Vida del Software

Análisis y Diseño de Software
2º Ingeniería Informática
Universidad Autónoma de Madrid



Indice

- Introducción.
- Fases y Ciclo de Vida del Software.
- Modelos de Ciclo de Vida.
- Metodologías.



¿Qué es el Software?

- Software =
programas+datos+documentación.
- Todo ello se ha de desarrollar **y mantener.**



Ciclo de Vida del Software

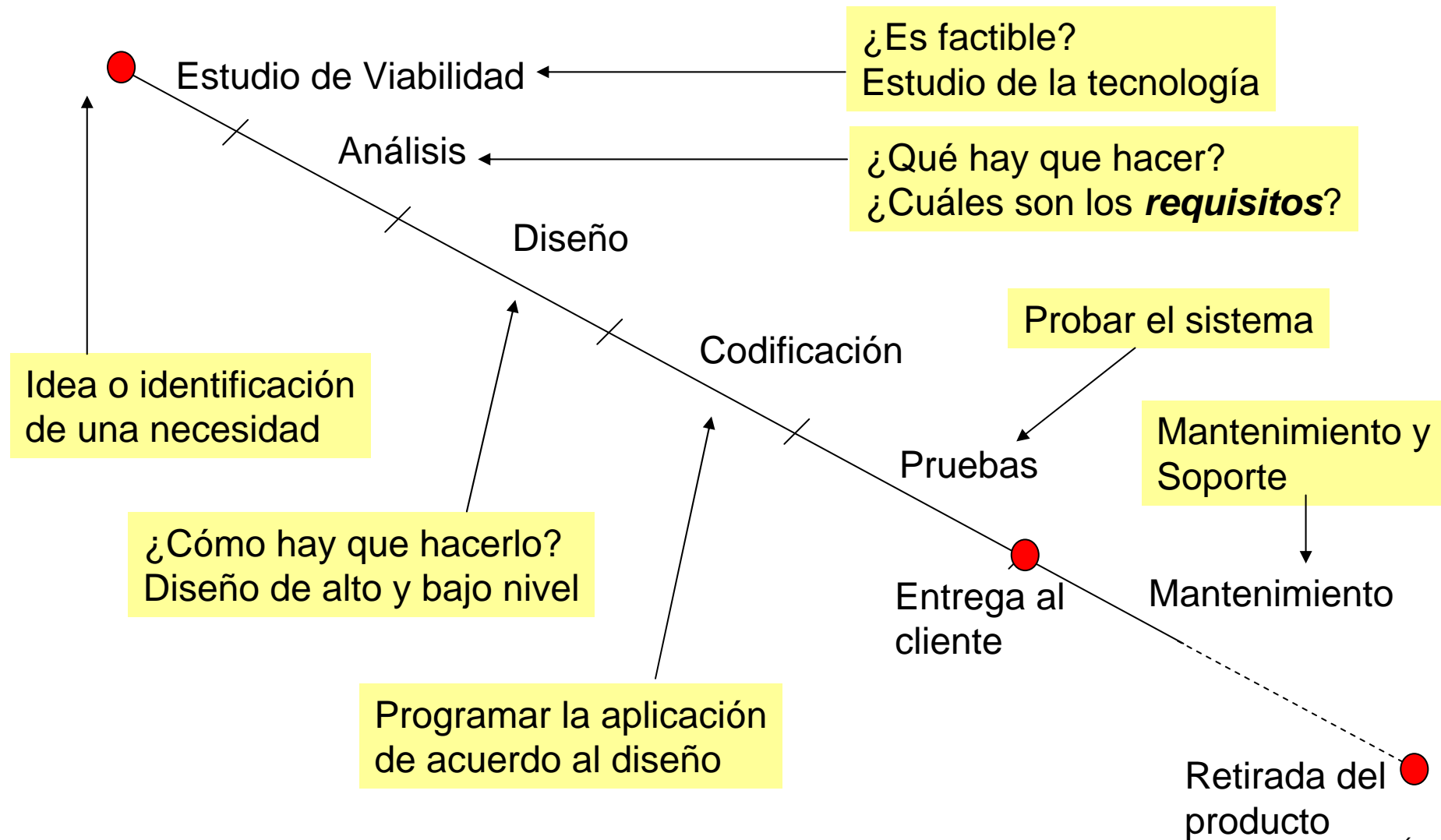
- Construir Software no consiste sólo en programar
- Fases adicionales: estudio de viabilidad, requisitos, análisis, diseño, codificación, pruebas, mantenimiento.
- Todas estas fases se desarrollan conforme a un plan de proyecto.
- Similitud con otras disciplinas:
 - construir un coche no consiste sólomente en soldar chapa y apretar tornillos,
 - construir una casa no es sólo poner ladrillos.
 - ...



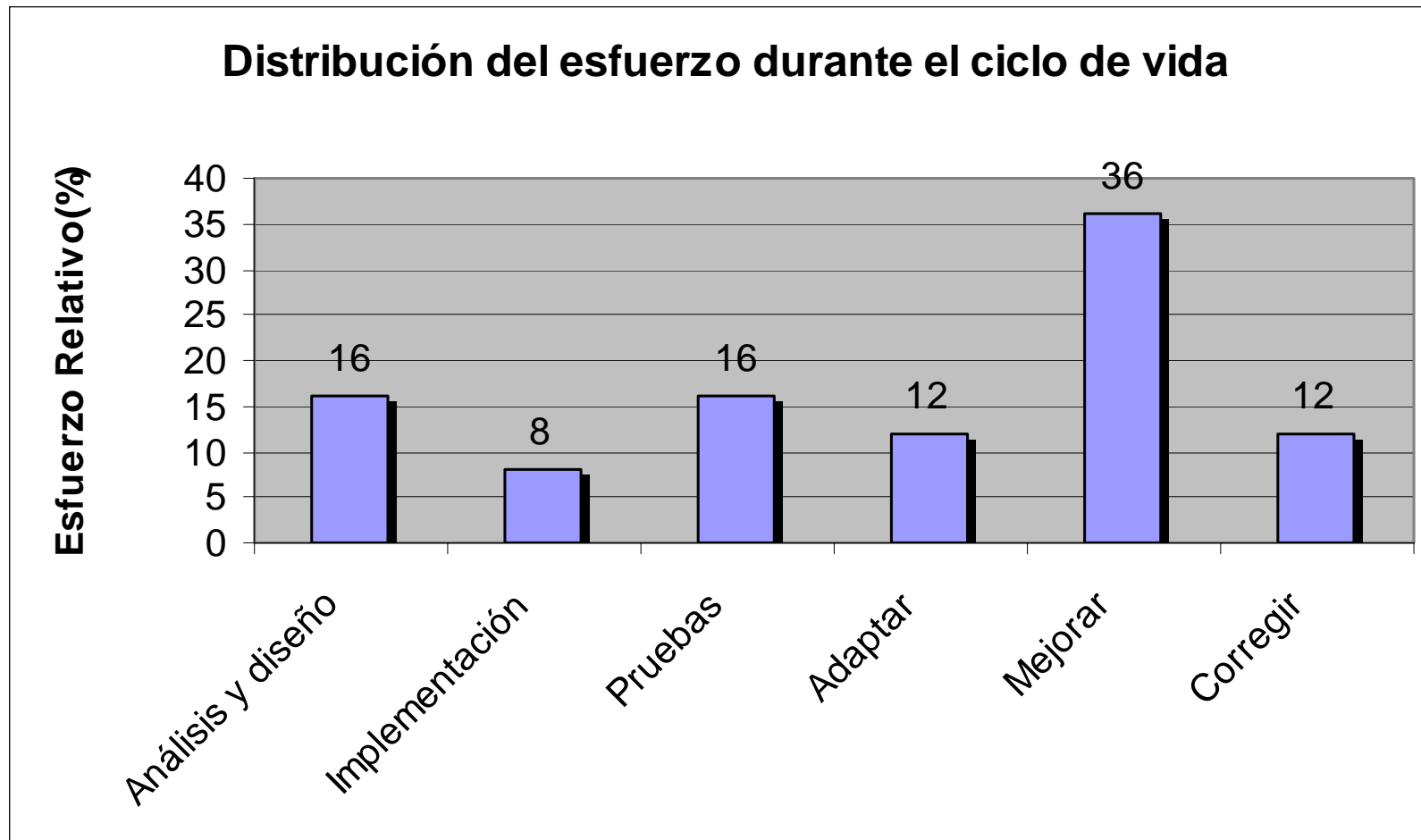
Ciclo de Vida del Software

- ***Ciclo de Vida***= Fases por las que pasa el software desde el inicio de su construcción hasta su retirada.

Ciclo de Vida del Software



Ciclo de Vida del Software





Estudio de Viabilidad

- Análisis técnico, operacional y económico previo a un proyecto para determinar si éste es rentable y decidir si el problema es abordable.
- Fundamento para la toma de decisión acerca de la continuidad o no del proyecto, así como los ***riesgos*** que conlleva la ejecución del mismo.



Estudio de Viabilidad

- Viabilidad **técnica**.
 - Estudio de la funcionalidad, rendimiento y restricciones.
- Viabilidad **económica**.
 - Análisis de costes/beneficios.
- Viabilidad **legal**.
 - Infracción, violación o ilegalidad resultado del desarrollo del sistema.
- **Alternativas**.
 - Evaluación de posibles alternativas al desarrollo del sistema.



Análisis de Requisitos

- Responder a las preguntas:

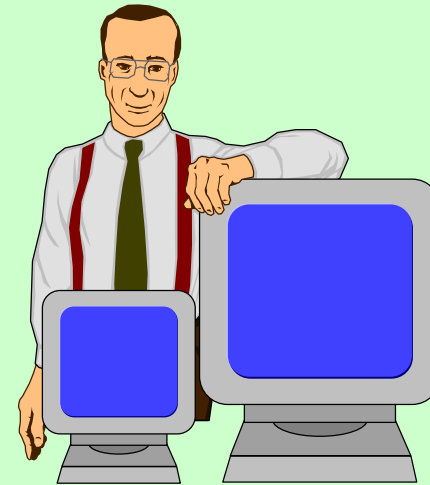
- ☐ ¿Qué hay que hacer?
- ☐ ¿Qué funcionalidad hay que implementar?
- ☐ ¿Cuáles son los requisitos no funcionales (rendimiento, fiabilidad, etc)?

“Análisis del problema y especificación completa del comportamiento externo que se espera del sistema software que se va a construir, así como de los flujos de información y control.”

Análisis de Requisitos



Los clientes y usuarios plantean el problema actual, el resultado que esperan obtener y las condiciones que esperan.



El ingeniero del software pregunta, analiza, asimila y presenta la solución adecuada.



Análisis de Requisitos

Tareas

- **Captura** de requisitos.
 - Identificar los requisitos que se obtienen de los usuarios y clientes.
- **Análisis** del problema y de los requisitos.
 - Razonar sobre los requisitos, combinar requisitos relacionados, establecer prioridades entre ellos, determinar su viabilidad, etc.
- Representación (**modelización**).
 - Registrar los requisitos de alguna forma, incluyendo lenguaje natural, lenguajes formales, modelos, maquetas, etc.
- **Validación**.
 - Examinar inconsistencias entre requisitos, determinar la corrección, ambigüedad, etc. Establecer criterios para asegurar que el software reúna los requisitos cuando se haya producido. El cliente, usuario y desarrollador se deben poner de acuerdo.



Análisis de Requisitos

Captura. Tipos de requisitos

- Funcionales y no funcionales.
- Requisitos funcionales
 - Acciones fundamentales que tienen que tener lugar en la ejecución del software.
- No funcionales:
 - Operacionales (ej., recuperación, back-ups)
 - Seguridad (niveles de acceso, protección,...)
 - Mantenibilidad y Portabilidad.
 - Recursos (memoria, almacenamiento, etc)
 - Rendimiento (tiempo de respuesta, nº usuarios,...)
 - ...



Análisis de Requisitos

Representación.

- Análisis estructurado.
 - Técnicas de análisis orientadas a datos.
 - Técnicas de análisis orientadas a funciones.
 - Técnicas de análisis orientadas a estados.
- Análisis orientado a objetos.
- Lenguajes formales.
- Maquetas.



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.

Diagrama de Flujo de Datos

Nivel 0.

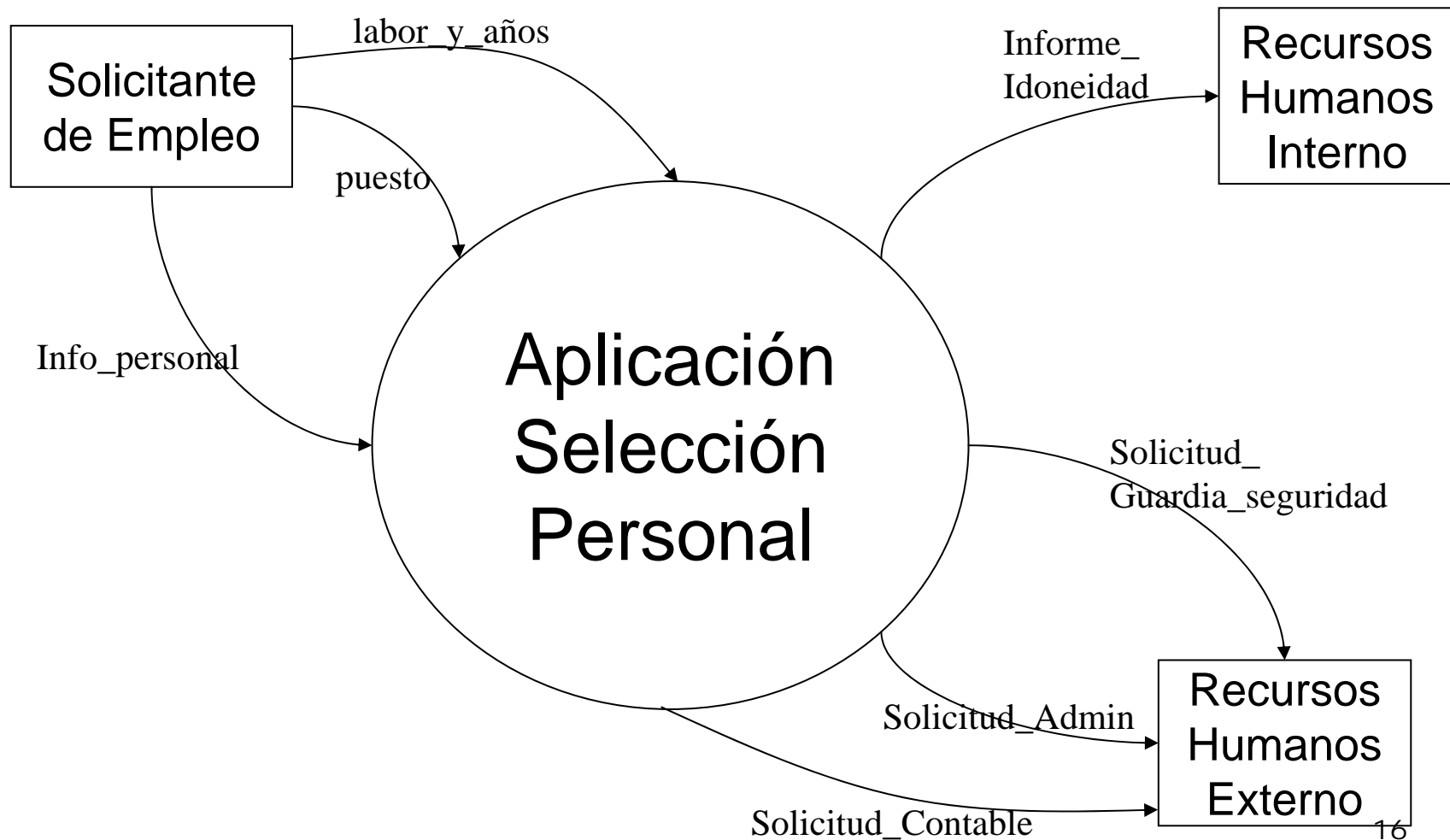
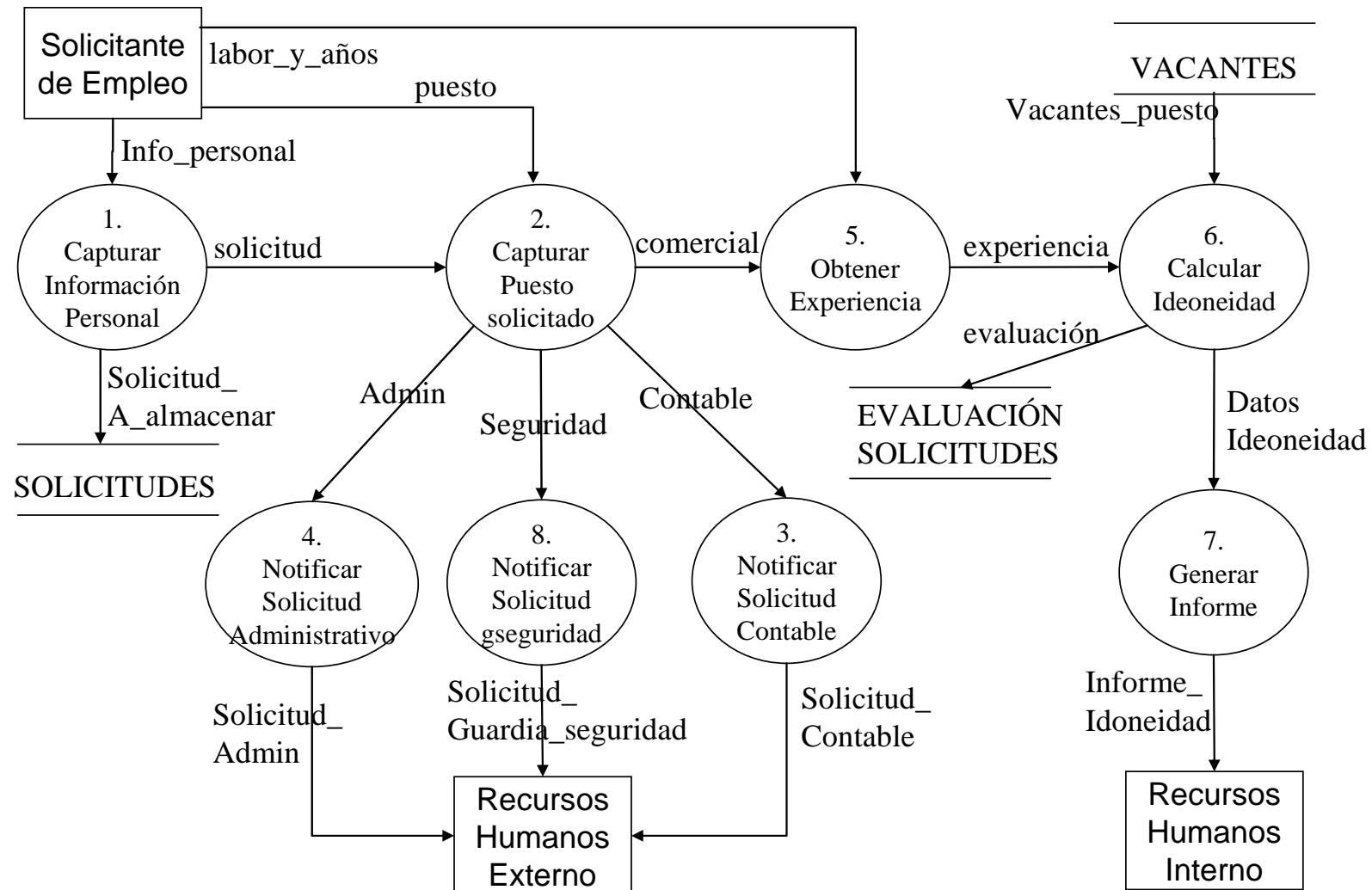


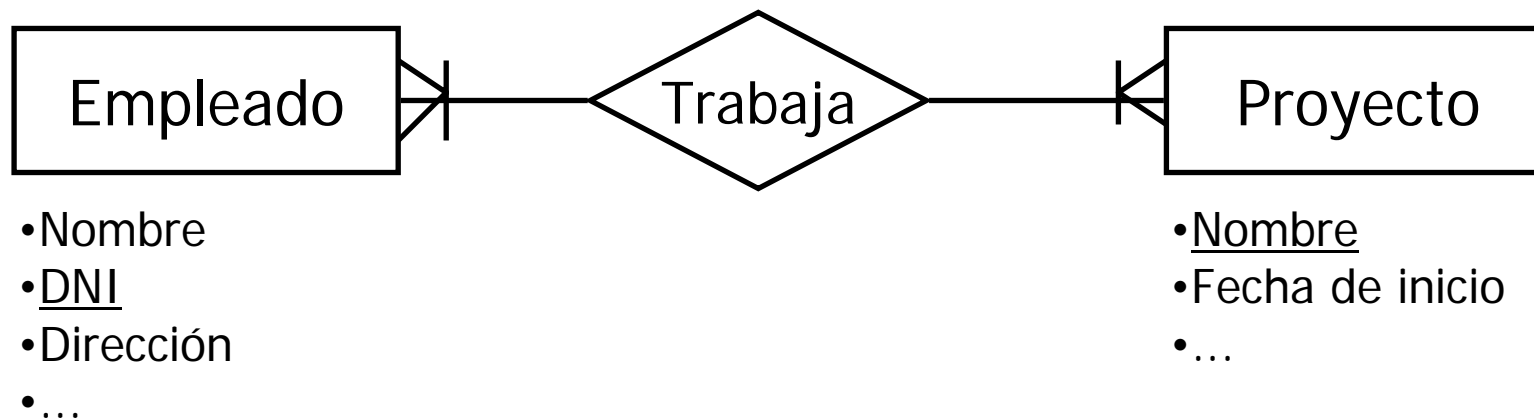
Diagrama de Flujo de Datos

Nivel 1



Datos

- Qué información necesita la aplicación.
- Diccionario de datos.
- Diagramas Entidad Relación.



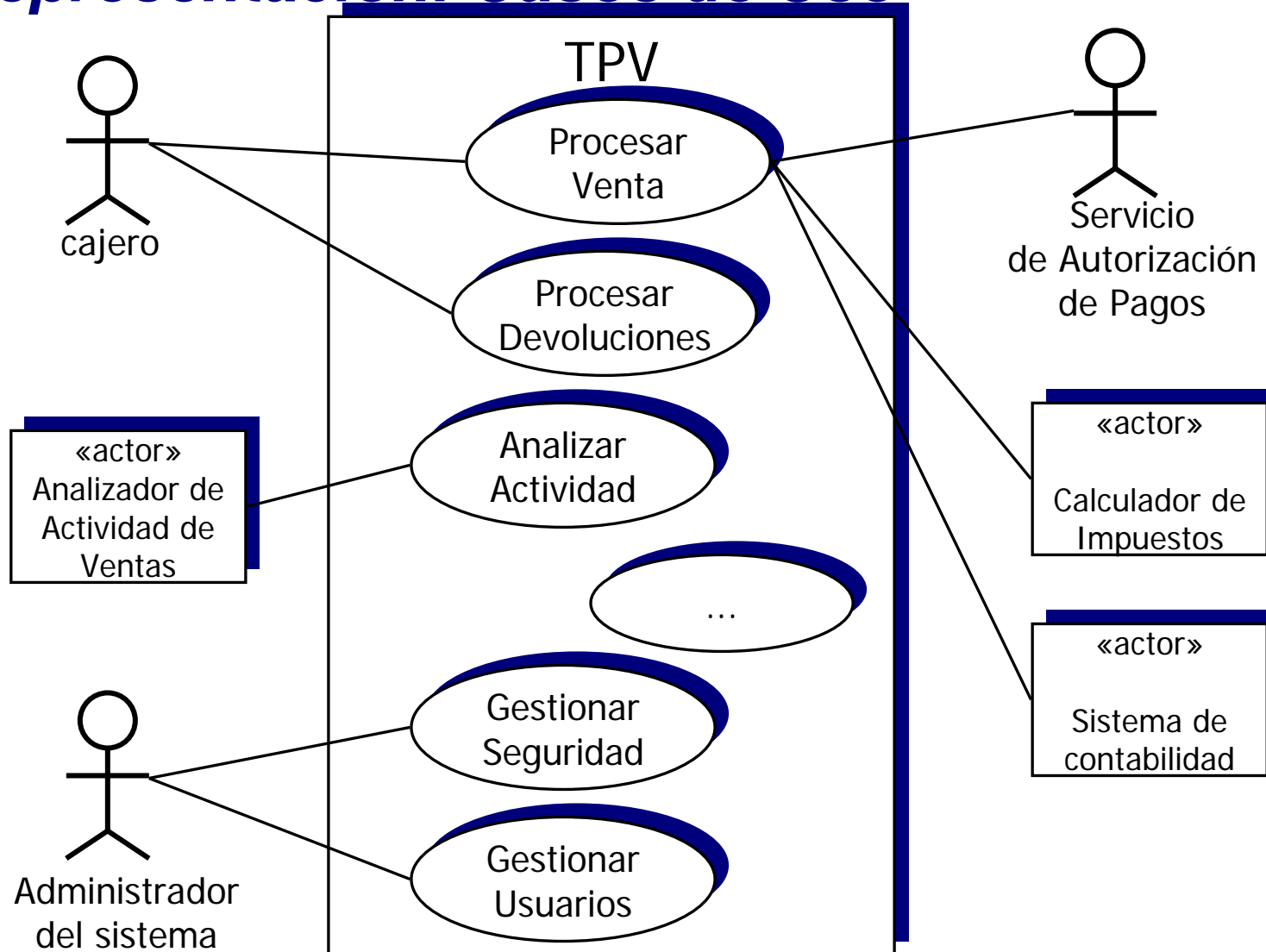


Análisis Orientado a Objetos

- **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 interacciones que describen condiciones de éxito o fracaso (e.j. errores)
- **Actores:** Elementos activos externos (usuarios, otro sistema) que interacciona con el sistema.

Análisis de Requisitos

Representación: Casos de Uso





Análisis de Requisitos

Representación: Casos de Uso

CASO DE USO 1: Procesar venta

Actor Primario: Cajero.

Precondiciones: El cajero se ha identificado y autenticado.

Garantía de éxito (Postcondiciones): Se registra la compra en el sistema. Se calcula el impuesto aplicable. Se actualizan los sistemas de inventario y de contabilidad. Se registran las comisiones. Se genera un recibo. Se registran las aprobaciones de pago por tarjeta.

Escenario principal de Exito:

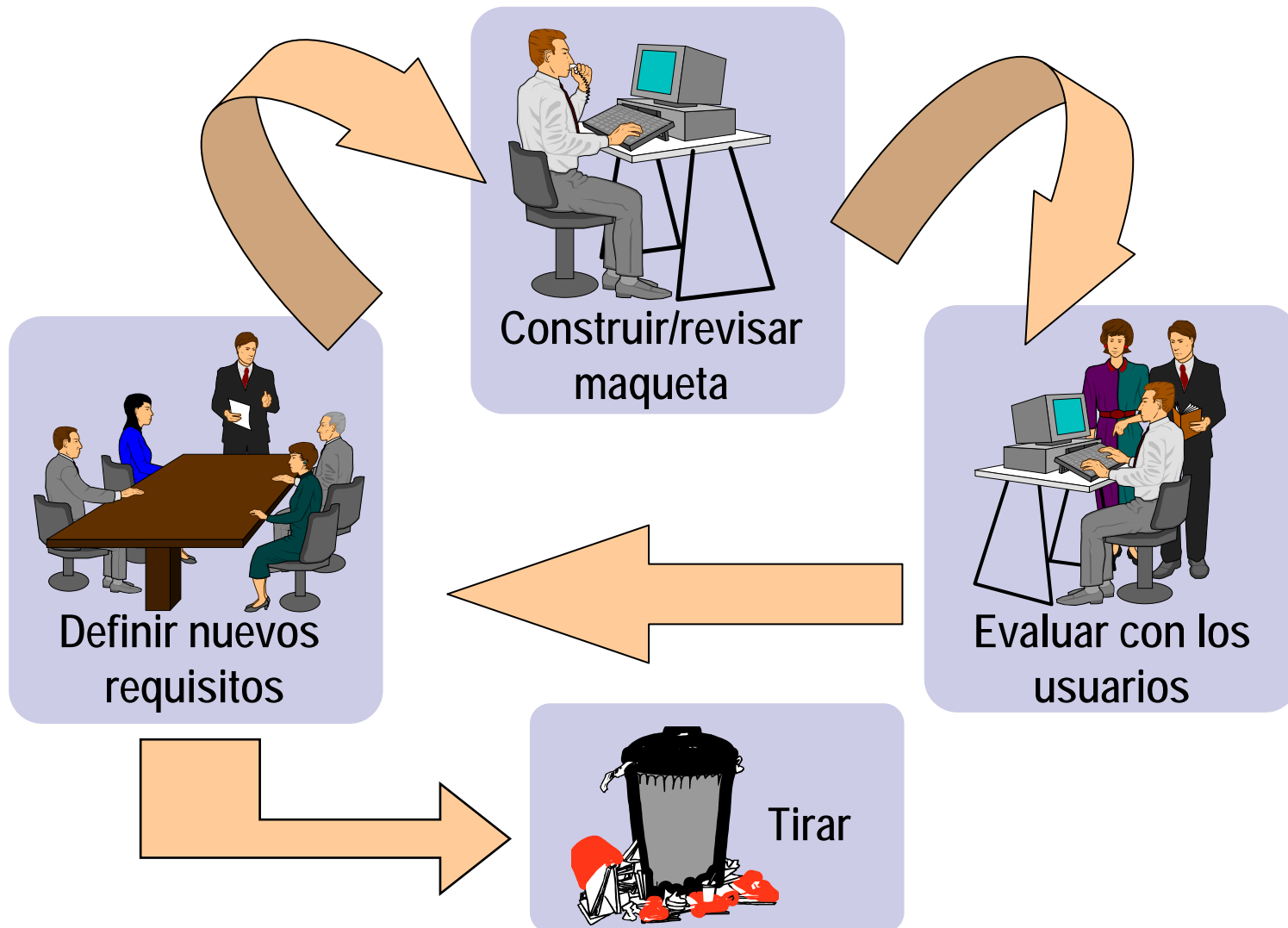
1. Llega un cliente al TPV con bienes o servicios que comprar.
 2. El cajero comienza una nueva compra.
 3. El cajero introduce un identificador de producto.
 4. El sistema registra el elemento y presenta una descripción del mismo, su precio y total actual. Se calcula el precio de una lista de reglas.
El cajero repite los pasos 3-4 hasta que no hay más elementos.
 5. El sistema presenta el total con los impuestos calculados.
 6. El cajero le dice el total al cliente, y le pide que pague.
 7. El cliente paga y el sistema procesa el pago.
 8. El sistema registra la venta completada y manda la información a los sistemas externos de inventario y contabilidad.
 9. El sistema genera el recibo.
 10. El cliente se va.
- (además hay que especificar excepciones y flujos alternativos)

Maquetas

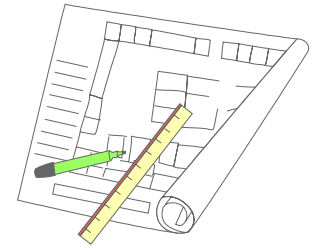
- Captura de requisitos en forma de interfaces que permitan un mejor entendimiento con el usuario.
- Desde programas de dibujo (powerpoint, visio) hasta aplicaciones especializadas (ej.: Mockupscreens, Balsamiq, etc).



Maquetas: Ciclo de vida



Diseño



- Cambiar la atención del qué hay que hacer al cómo hay que hacerlo.

- Del dominio del problema al dominio de la solución.

“Es el proceso de definición de la arquitectura, componentes, módulos, interfaces, procedimientos de prueba y datos de un sistema software para satisfacer unos requisitos especificados.”

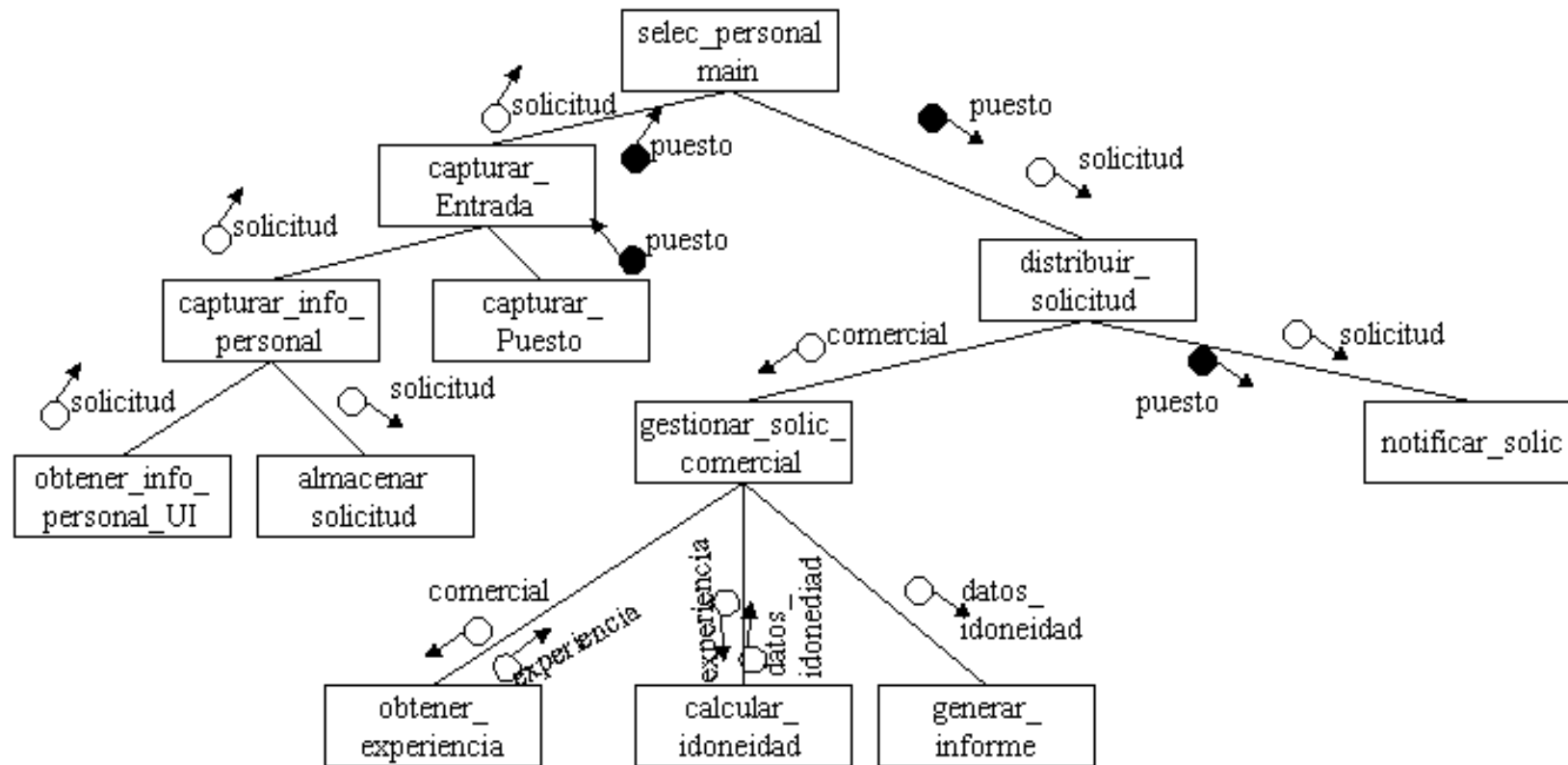


Diseño

- Diseño de **arquitectura**
 - Definición de los componentes del sistema y 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. Orientado a Objetos.

Diseño Estructurado

- Jerarquía de llamadas entre funciones, y paso de parámetros.





Diseño Orientado a Objetos

- Estructura:
 - Diseño de clases.

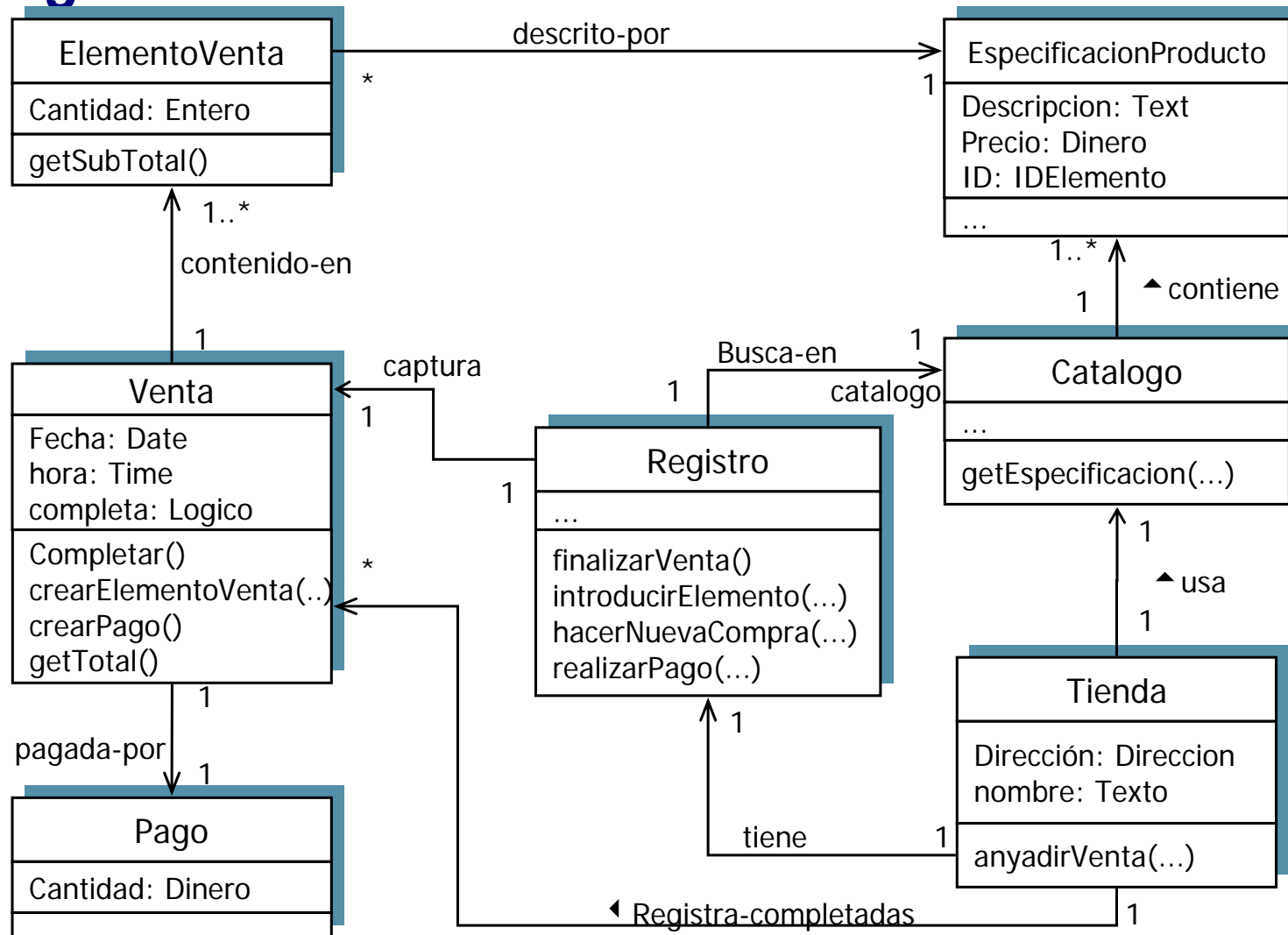
- Comportamiento:
 - De cada clase.
 - De las interacciones entre objetos.

- Una notación muy común es UML (Unified Modelling Language).
 - <http://www.uml.org>



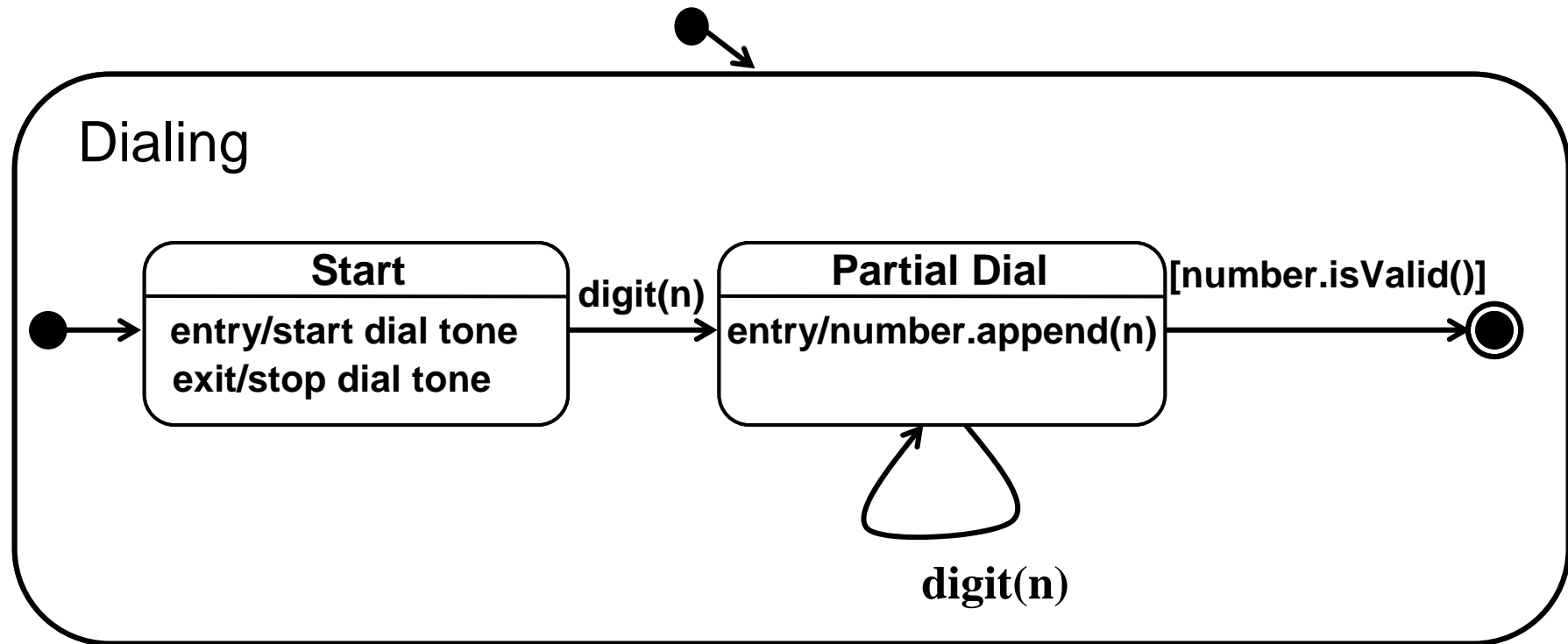
Diseño Orientado a Objetos

Diagrama de clases



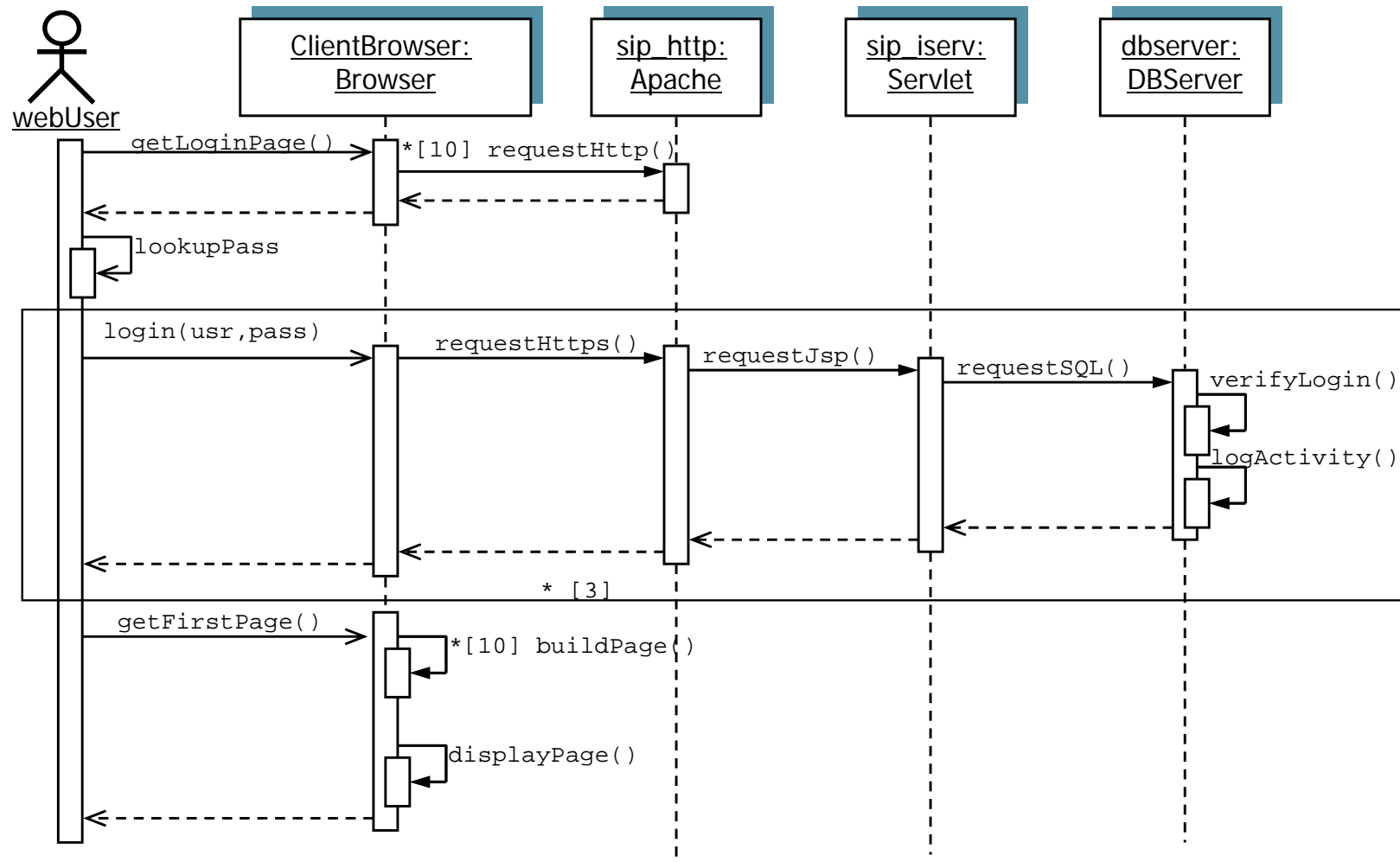
Diseño Orientado a Objetos

Máquinas de estados.



Diseño Orientado a Objetos

Diagrama de Secuencia.

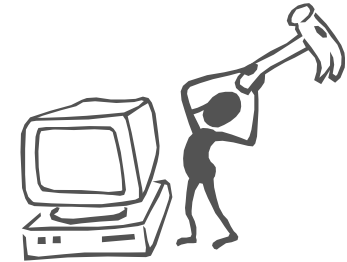




Codificación

- Traducir las especificaciones de diseño a un lenguaje de implementación (C, Java, etc).
- Incluye también:
 - ☐ Pruebas de unidad (e.j.: pruebas de cada función o método por separado para ver que funcionan).
 - ☐ Manual técnico (e.j.: Javadoc)
 - ☐ Manual de usuario.
- Guías de estilo.
- Programación estructurada vs. orientada a objetos.

Pruebas



- Ejecutar el programa para encontrar errores.
 - ☐ NO es demostrar que no hay errores, o demostrar que el programa funciona.
 - ☐ Ya que asegurar la corrección es prácticamente imposible para programas grandes.
 - ☐ Diseñar casos que maximicen la probabilidad de encontrar errores.

- Error de software: cuando el programa no hace lo que se espera que haga, acordado en los requisitos.
 - ☐ Error de programación.
 - ☐ Problemas de comunicación con los usuarios.



Pruebas

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

■ Caja negra.

- ☐ Conducidas por los datos de entrada/salida.
- ☐ Considera el software como una caja negra sin tener en cuenta los detalles procedimentales de los programas



Pruebas

Estrategias

■ 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.

- ☐ Se comprueba la concordancia respecto a los requisitos sw.

■ 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 éste está operativo (después de la entrega).
- Modificaciones necesarias para cumplir con nuevos o antiguos requisitos.

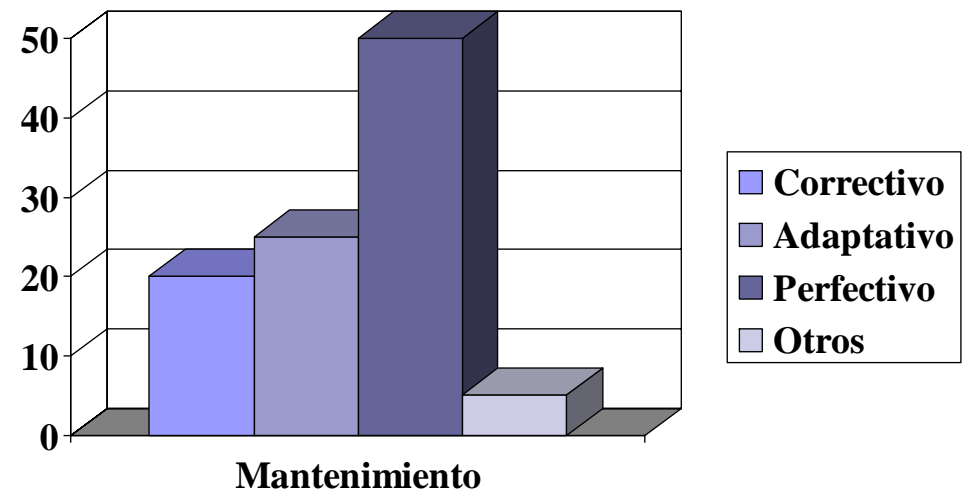
‘La modificación de un producto software después de su entrega, para corregir errores, mejorar su rendimiento u otros atributos, o adaptar el producto a modificaciones de su entorno operativo’

Mantenimiento

Tipos

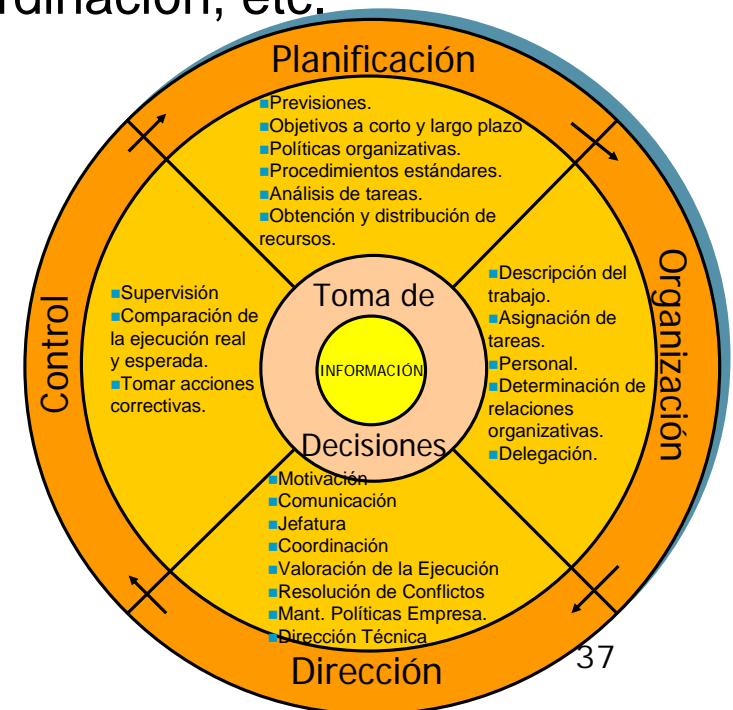
- **Correctivo.** ($\cong 20\%$)
 - Corregir errores
- **Adaptativo.** ($\cong 25\%$)
 - Acomodar a nuevo entorno.
- **Preventivo.** ($\cong 5\%$)
 - Prevenir errores.
- Estructural (tipo de mant. preventivo)
 - Modificar la arquitectura interna.
- **Perfectivo.** ($\cong 50\%$)
 - Mejorar, expandir requisitos implementados.

Porcentaje de Esfuerzo de
Mantenimiento



Actividades de Gestión

- Como en cualquier proyecto, el desarrollo de software requiere de una serie de actividades de gestión:
 - Estimación: predicción de duración, esfuerzo y costes para realizar el proyecto.
 - Planificación: selección de una estrategia, definir actividades, asignarles un calendario y recursos, coordinación, etc.
 - Negociación.
 - Seguimiento del proyecto.
 - Gestión.
 - Coordinación del equipo de trabajo.
 - Dirección técnica.
 - ...

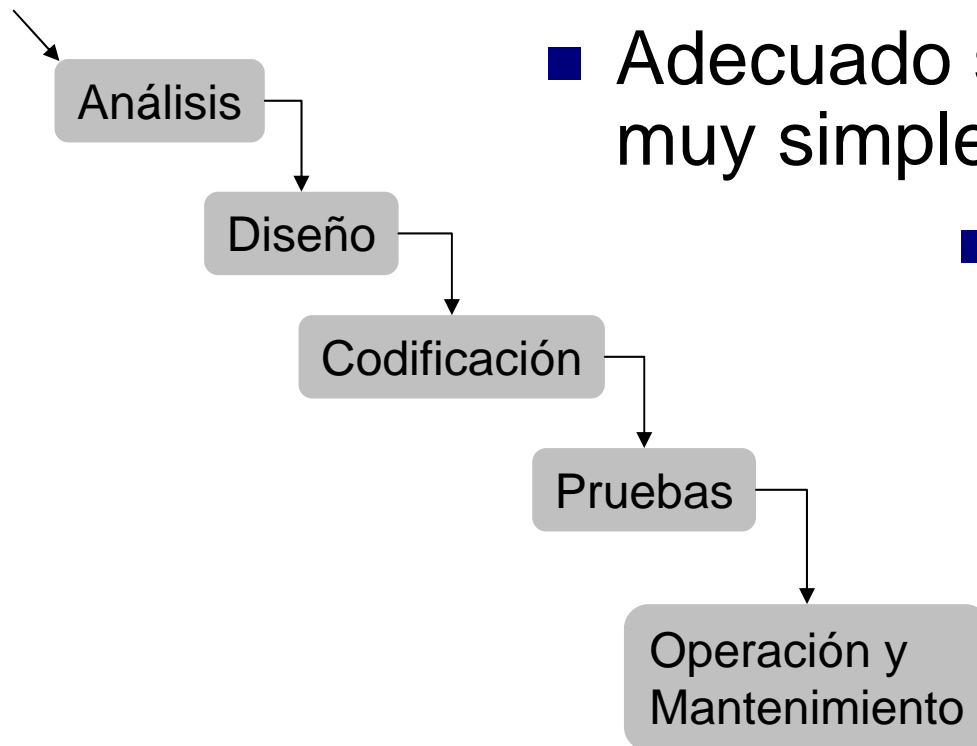




Modelos de Ciclo de Vida

- Salvo en casos muy sencillos, la construcción del software no sigue una distribución lineal de fases.
- A veces es más conveniente realizar iteraciones, o incrementos.
- ***Modelo de ciclo de vida.*** Esquema que describe:
 - Fases por las que pasa el proyecto.
 - Criterios de transición de una fase a la siguiente.
 - Entradas y Salidas de cada fase.

Modelo de Ciclo de Vida en Cascada

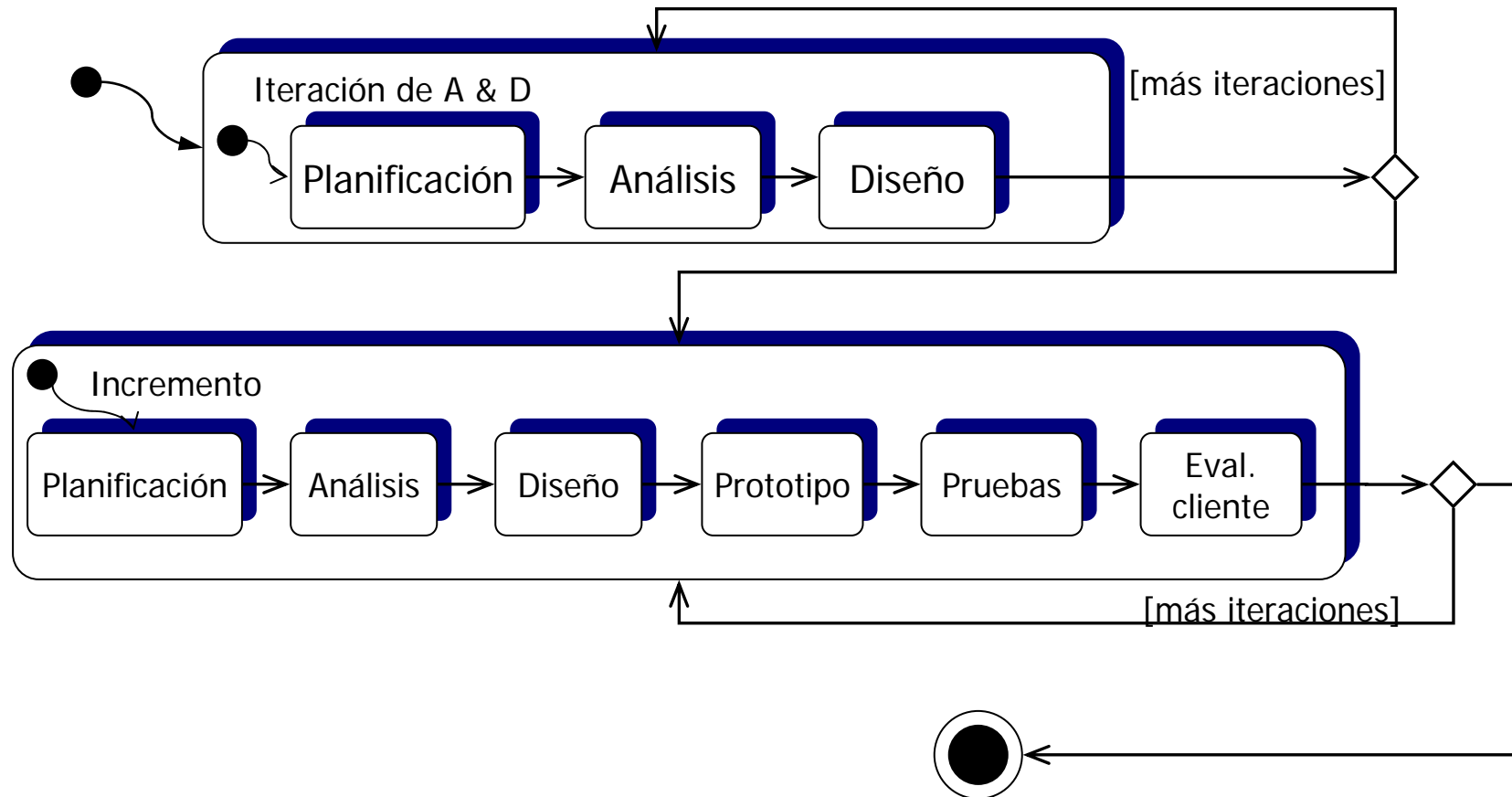


- Adecuado sólo para proyectos muy simples.

■ Desventajas:

- ☐ No se permiten las iteraciones.
- ☐ Los requisitos se congelan al principio del proyecto.
- ☐ No existe un proyecto “enseñable” hasta el final del proyecto.

Modelos incrementales

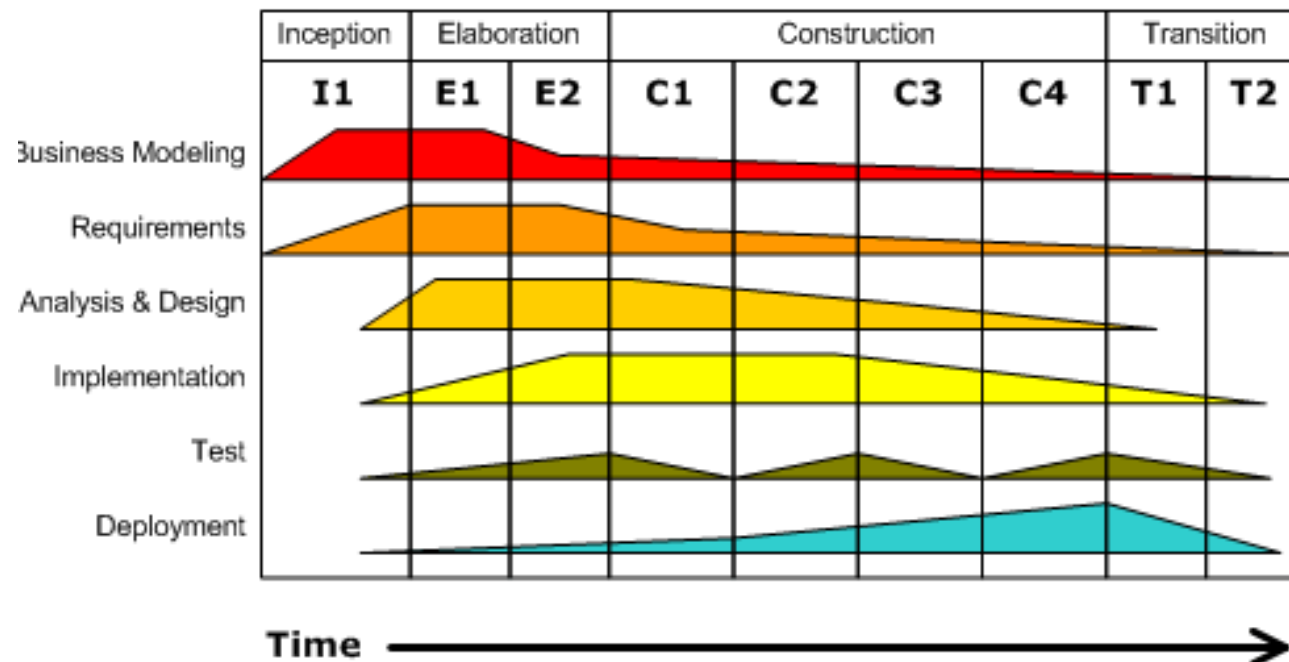


- Rational Unified Process (<http://www-01.ibm.com/software/awdtools/rup/>).
- Otras “metodologías ágiles”:
 - Extreme Programming (Xp): www.extremeprogramming.org

Modelos Incrementales

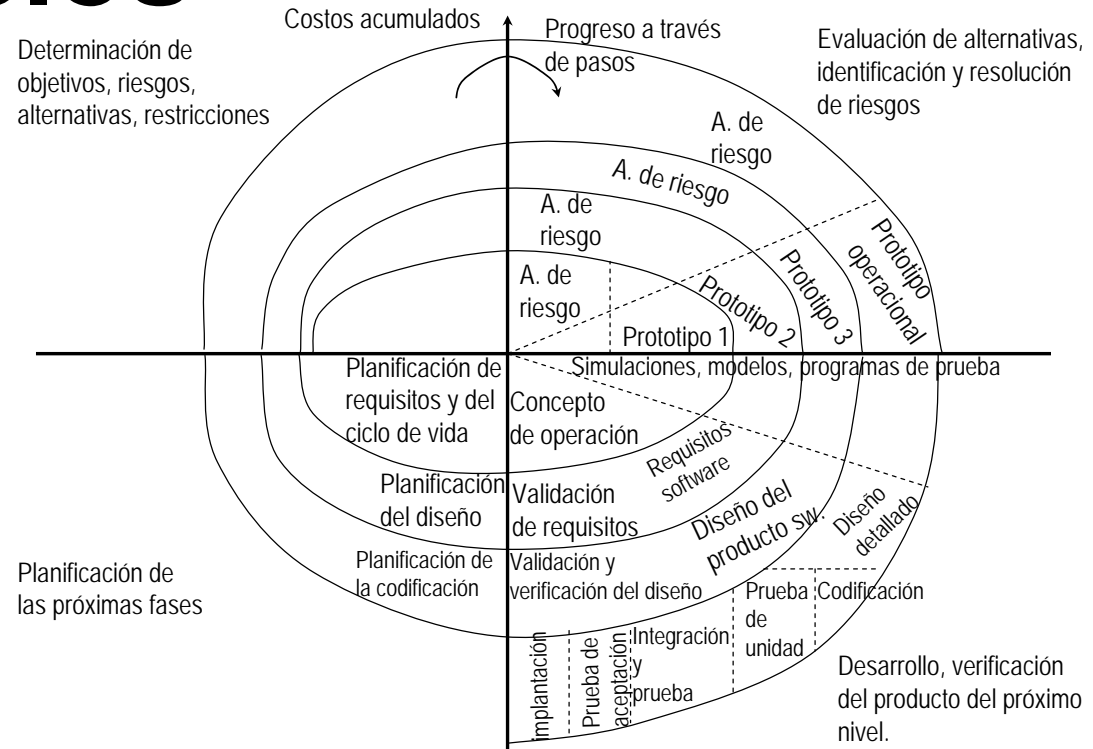
Iterative Development

Business value is delivered incrementally in time-boxed cross-discipline iterations.

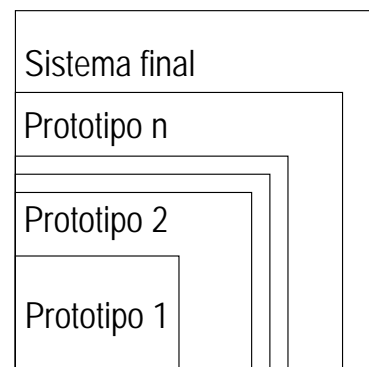


Otros modelos

- Ciclo de vida en espiral.



- Modelos de desarrollo de productos
 - Prototipado.





Metodologías

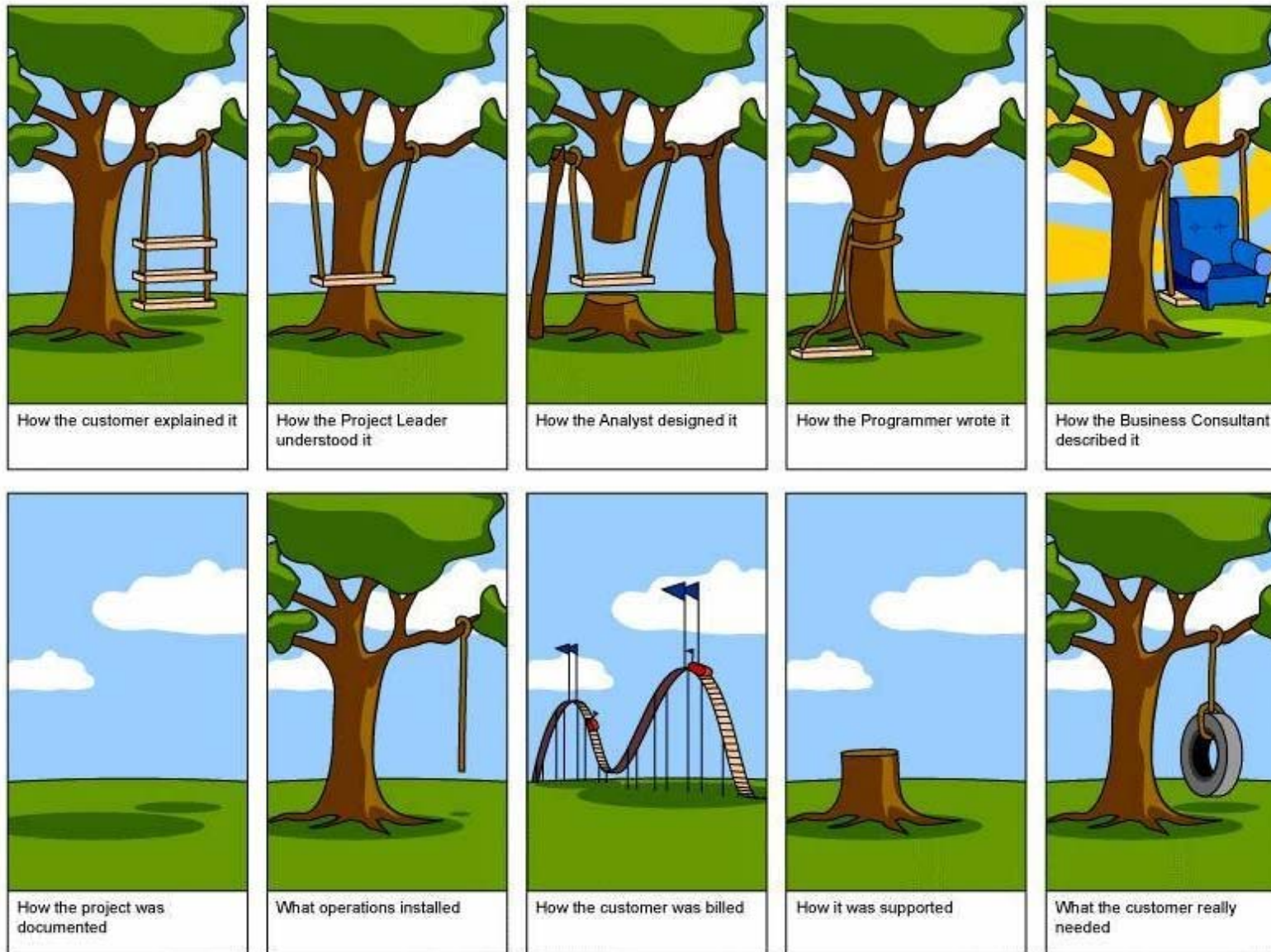
- Además de las fases, nos dicen exactamente que métodos tenemos que usar en cada una de ellas.
- Ejemplos:
 - Metodología METRICA
(<http://www.csae.map.es/csi/metrica3/>)
 - Para empresas que trabajan con el ministerio.



Resumen

- Desarrollar software no es sólo programar.
- Otras actividades:
 - Viabilidad, Requisitos, Análisis, Diseño, Codificación, Pruebas y Mantenimiento.
- Organización del desarrollo:
 - Modelos de ciclo de vida.
 - Metodologías.

En la vida real...





Bibliografía

- Bibliografía básica:

- Software engineering a practitioner's approach, 7ªed. Roger Pressman. McGraw Hill Higher Education, 2010. INF/681.3.06/PRE. También disponible en castellano.
- Software engineering, 9ª ed. Addison Wesley. Ian Sommerville. INF/681.3.06/SOM. También disponible en castellano.

- Lecturas recomendadas:

- Mary Shaw. “*Prospects for an engineering discipline of software*”. IEEE Software, Vol.7(6), Noviembre 1990, pag. 486-495.
- Mary Shaw. “*Continuing Prospects for an Engineering Discipline of Software*”. IEEE Software. Vol. 26(6). Noviembre 2009. pag. 64-67.
- IEEE Standard 1074-2006. IEEE Standard Software Life-Cycle Processes. 2006.