



INTRODUCCIÓN A LA INGENIERÍA DEL SOFTWARE

Bloque I. Tema 1.

Ingeniería del Software
2º curso de ITIS
Paloma Cáceres



1.1. Introducción

1.2. La complejidad inherente del software

1.3. El ciclo de vida del software

1.4. Introducción a la OO



¿Qué es software?

????



¿Qué es software?

- Software son los programas, la documentación y la configuración de los datos asociados.
 - Genéricos: Productos de uso genérico (procesadores, bases de datos, etc.)
 - A medida: Productos específicos para un cliente particular.

¡¡Son diferentes productos y más su especificación!!



¿Qué es Ingeniería del Software?

“La Ingeniería del Software es la disciplina de ingeniería encargada de todos los aspectos relacionados con la producción de software desde sus etapas más tempranas de la especificación del sistema hasta el mantenimiento del sistema tras su puesta en marcha.”



¿Qué es Ingeniería del Software?

- “Disciplina de ingeniería”

- Uso apropiado de todas las herramientas, métodos y teorías para solucionar los problemas que aparecen. Incluso teniendo en cuenta restricciones económicas y de organización.

- “Todos los aspectos de la producción de software”

- Procesos técnicos de desarrollo, dirección de proyectos, generación de herramientas específicas para aplicar los métodos y teorías apropiados...



Diferencia entre ingeniería del software y ciencia de la computación

- Ciencia de la computación: Teorías y métodos subyacentes a las computadoras y los sistemas software.
- Ingeniería del software: Resolución de problemas prácticos para producir software.



Diferencia entre ingeniería del software e ingeniería de sistemas

- Los sistemas, hoy en día, incorporan software. Siendo un sistema algo más complejo que el software. Un sistema es un avión, una planta química.
- El software es una parte de casi todos los sistemas.



¿Qué es el proceso software?

- Proceso software es el conjunto de actividades y resultados asociados para producir un producto software
- ¿¿ Procedimiento para matricularse en ITIS ??
- 4 actividades comunes:
 - Especificación software
 - Desarrollo software
 - Validación software
 - Evolución software



Especificación software

- **Los clientes e ingenieros definen el software a producir y las restricciones para su operación.**
 - **Especificad cómo se hace una sangría**



Desarrollo software

- **Relacionado con el diseño y programación del software.**

- Realizad la sangría**



Validación del software

- **El software se valida (comprueba) para asegurar que es lo que el cliente quiere y espera.**
 - **Probad la sangría solos y luego dádsela a probar a quien os la encargó.**



Evolución del software

- **El software se modifica para adaptarlo a los cambios solicitados por el cliente y/o a los requisitos del mercado.**
 - **Echad más fruta o más azúcar a gusto del cliente. Y si hay diabéticos en la fiesta, haced una parte de sangría sin azúcar.**



¿Qué es un modelo de proceso software?

- Modelo de proceso software es una descripción simplificada de un proceso software (actividades, productos, roles,...)
- Modelos de proceso software
 - Modelo en cascada
 - Modelo incremental
 - Modelo de construcción de prototipos
 - Modelo en espiral

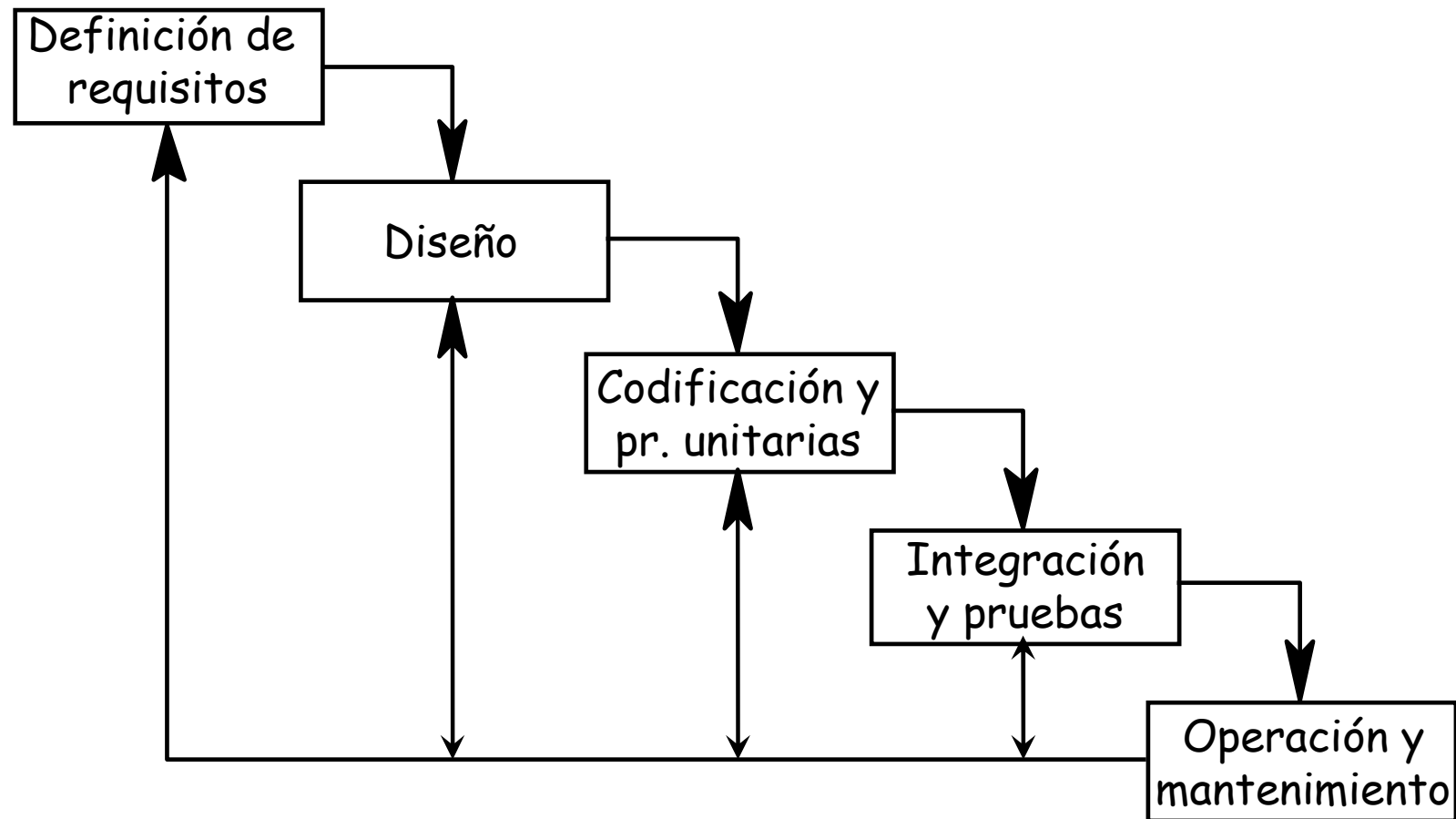


Modelo en cascada

- Fases: Análisis, diseño, codificación y pruebas unitarias, integración y pruebas, explotación y mantenimiento
- Inicio de fase ↻ Fin de la anterior
- Resultados

- Requisitos bien entendidos

Modelo en cascada





Modelo en cascada

■ Críticas

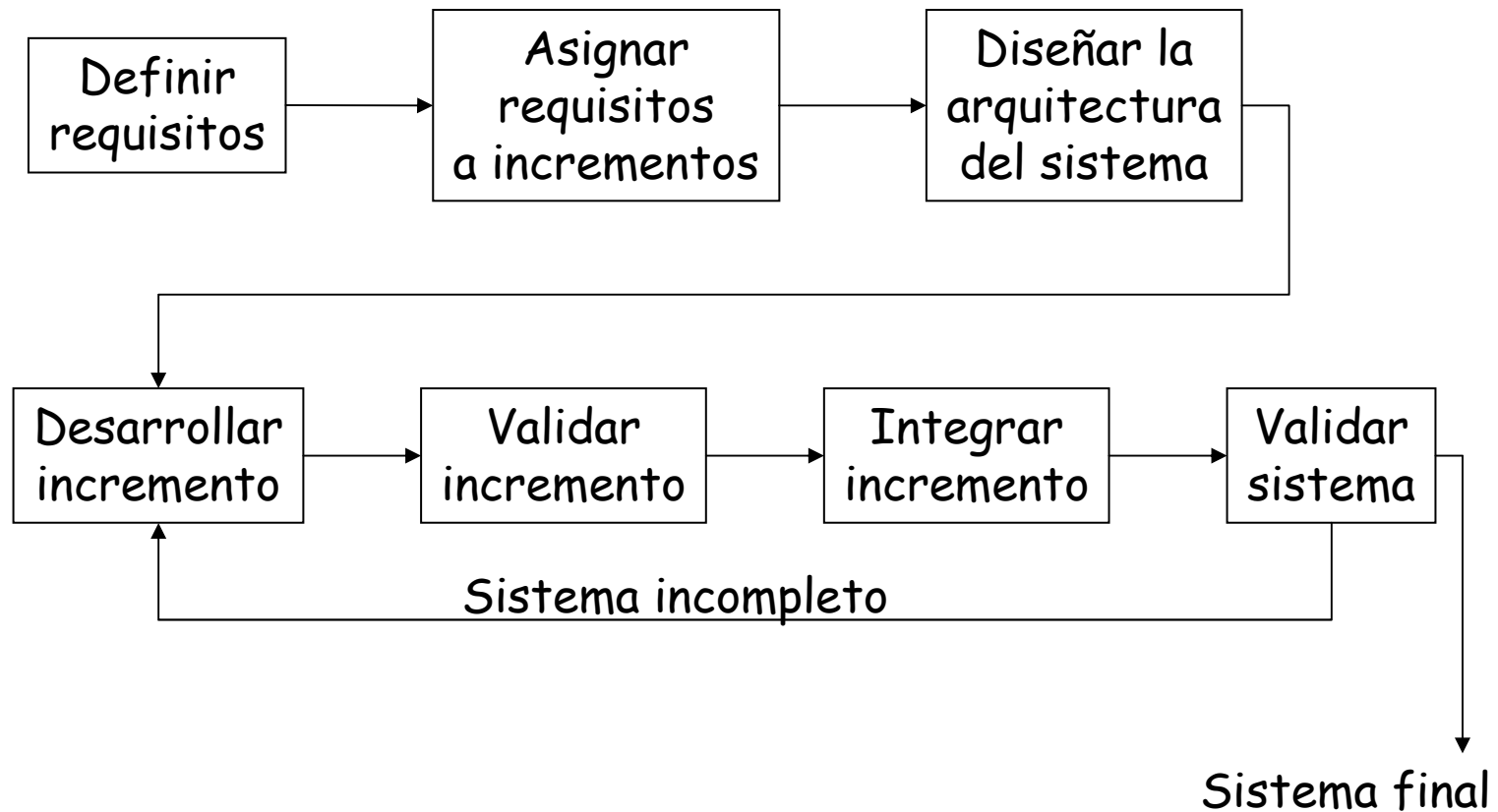
- No refleja realidad (hay iteraciones)
- Inversión de mucho tiempo
- Producto funcionando en fase final



Modelo incremental

- No lineal
- Cada versión implica nuevas funcionalidades
- Integración de resultados
- Ajuste a proyectos de alta incertidumbre

Modelo incremental





Modelo incremental

■ Ventajas

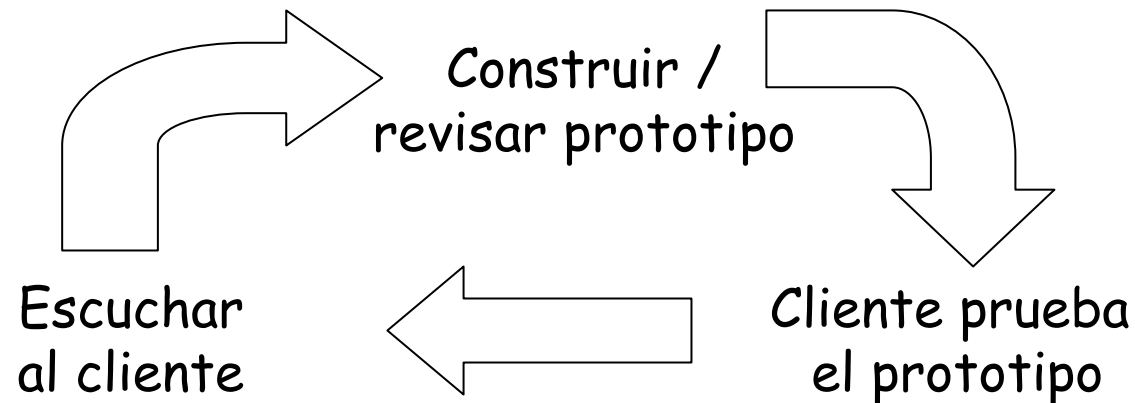
- Los clientes no tiene que esperar
- Los incrementos iniciales como prototipos
- Requisitos críticos muy probados

■ Problemas

- ¿Cómo deben ser los incrementos?

Modelo construcción de prototipos

- Clientes no muy explícitos
- Responsable técnico no seguro



- Sistemas pequeños o de tiempo de vida corto



Modelo construcción de prototipos

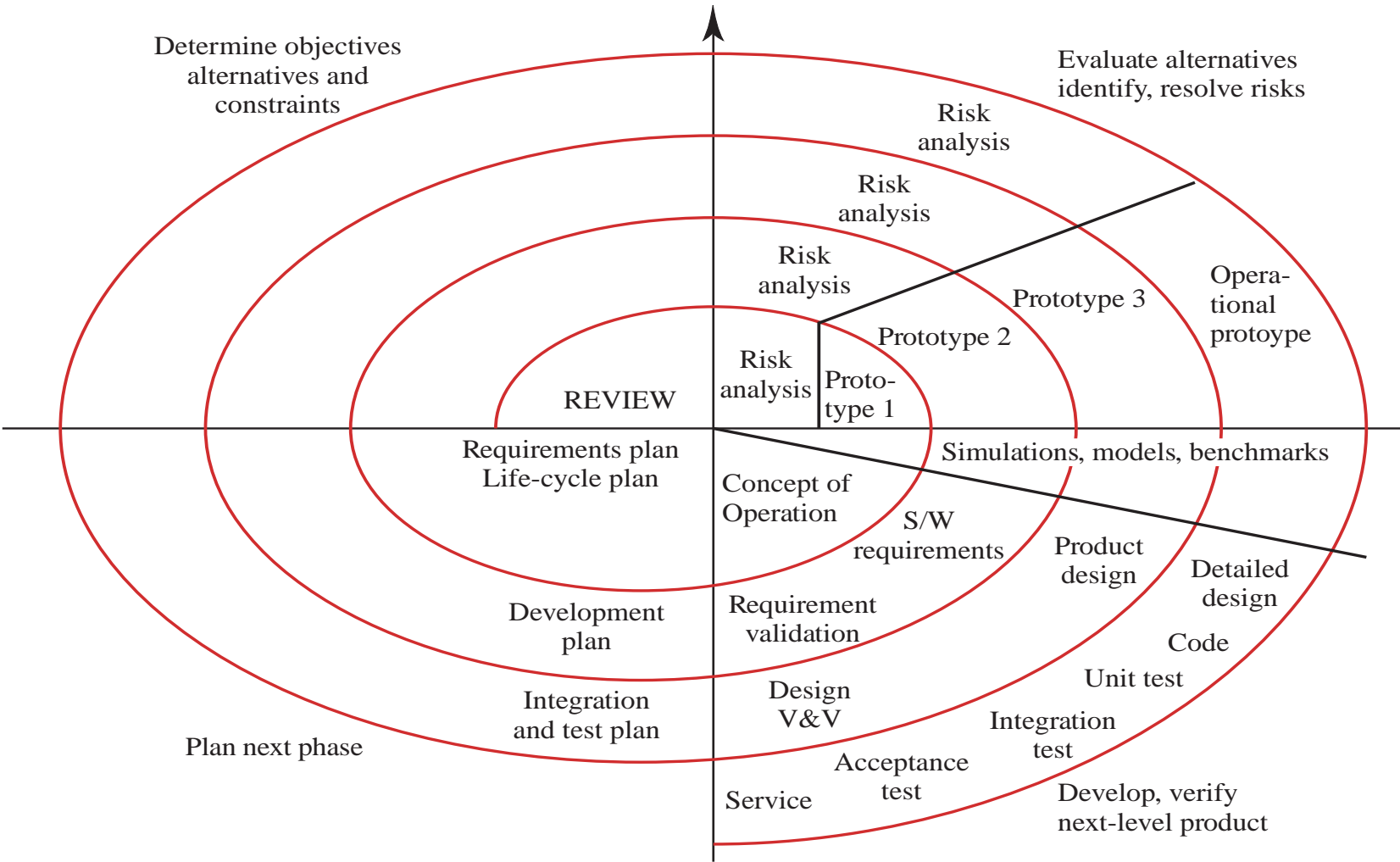
- La especificación se desarrolla de forma creciente y el cliente va viendo funcionalidades
- Pero:
 - estructura deficiente
 - herramientas y técnicas específicas



Modelo en espiral

- División del proyecto en ciclos
- División en actividades estructurales
 - Objetivos
 - Análisis de riesgos
 - Desarrollo y validación
 - Planificación
- Cada ciclo se completa con una revisión

Modelo en espiral





Modelo en espiral

- Enfoque realista del desarrollo
- Construcción de prototipos
- Reduce los riesgos
- Pero:
 - Precisa expertos en riesgos



¿Qué es CASE?

- CASE - Computer-Aided Software Engineering
 - Editores gráficos
 - Generadores de código
 - Generadores de casos de prueba...



Atributos de un buen software

■ Mantenibilidad

- Debe poder evolucionar para adecuarse a los cambios del cliente o del negocio
- Generadores de código

■ Confiabilidad

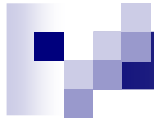
- Fiabilidad, protección y seguridad, por ejemplo. No daños físicos ni económicos.

■ Eficiencia

- No malgastar recursos, como memoria y buenos tiempos de respuesta

■ Usabilidad

- Fácil de usar, sin esfuerzo adicional por parte del usuario. Buena interfaz de usuario y documentación adecuada.



1.1. Introducción

1.2. La complejidad inherente del software

1.3. El ciclo de vida del software

1.4. Introducción a la OO



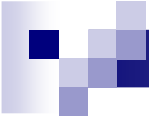
¿Por qué es complejo el software?

- Brooks: “La complejidad del software es una propiedad esencial y no accidental”
- Tres motivos:
 - La complejidad del dominio del problema.
 - La dificultad de controlar el proceso de desarrollo.
 - Los problemas para caracterizar sistemas discretos.



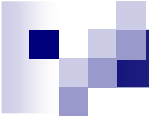
La complejidad del dominio del problema (I)

- Los problemas del mundo real son complejos (miles de requisitos compitiendo y quizás contradictorios).
- Ahora, añada requerimientos no funcionales: eficiencia, coste, fiabilidad...
- Diferente perspectiva del mismo problema entre los usuarios y los desarrolladores.




La complejidad del dominio del problema (y II)

- Los requerimientos cambian durante el desarrollo.
- Los grandes sistemas evolucionan a lo largo del tiempo:
 - Mantenimiento: corregir errores.
 - Evolución: cambios de requerimientos.
 - Conservación: uso continuo de medios extraordinarios para mantener operativo un sistema viejo.
- Ejemplo: ¿Podéis resumir qué pasos hay que dar para matricularos en la UNIVERSIDAD?
- Ejemplo: ¿Podéis resumir qué pasos hay que dar para solicitar una hipoteca bancaria?



La dificultad de controlar el proceso de desarrollo (I)

- Hay que dar impresión de simplicidad. ¡Imposible!
- Miles de líneas de código. Atajar con reutilización (de código, diseños reutilizables-patrones)
- Abordar complejidad por “divide y vencerás” ⇒ obtener módulos
- Aún así, cientos o miles de módulos.



La dificultad de controlar el proceso de desarrollo (y II)

- Se necesita un equipo de desarrolladores
- Cuanto más cercanos entre sí, ¡¡mejor!!



Los problemas para caracterizar sistemas discretos (I)

- El software trabaja con sistemas con estados discretos:
 - Conjunto de variables.
 - Flujos de control.
 - Eventos externos.
- Los valores de cada uno dan el estado del sistema.
- El mundo real es analógico se puede describir por funciones continuas.



Los problemas para caracterizar sistemas discretos (y II)

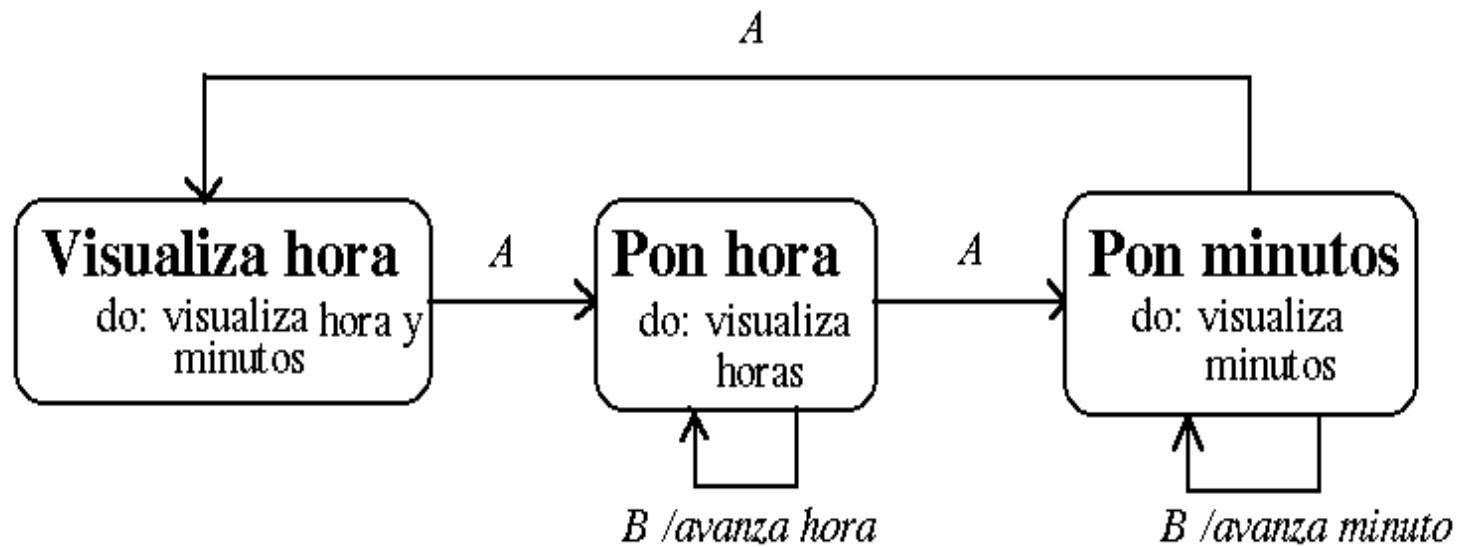
- En sistemas discretos la transición entre estados no puede modelarse con funciones continuas
- Cada evento externo puede cambiar el estado del sistema. ¿Al estado correcto y previsible?
- Necesidad de pruebas
 - complejo, costoso y en ocasiones imposible que sean exhaustivas
- **Ejemplo:** Identificar estados (situaciones diferentes) en un reloj digital



Ejercicio

Un reloj digital tiene una pantalla y dos botones para accionarlo, el botón A y el botón B. El reloj tiene dos modos de operación, visualizar la hora y establecerla. En el modo de visualización aparecen las horas y los minutos separados por dos puntos (:) intermitentes. El modo de establecer la hora tiene dos submodos: poner las horas y poner los minutos. El botón A se utiliza para seleccionar el modo de operación. Cada vez que se aprieta, el modo avanza en secuencia: visualizar la hora, poner hora, poner minutos, visualizar la hora, etc. Dentro de los submodos, el botón B se utiliza para avanzar una hora o un minuto cada vez que se aprieta. Prepare un diagrama de estados del reloj.

Solución





Ejercicio

Un semáforo de circulación, que tiene un visor para vehículos y otro para peatones, tiene el siguiente funcionamiento: El visor para vehículos muestra de forma alterna el color rojo, verde y ámbar. El visor para peatones muestra igualmente el color rojo, verde y verde intermitente. El semáforo cada cierto tiempo cambia de estado para permitir el paso tanto de peatones como de vehículos, siendo estos tiempos programables por agentes de la circulación.

Representar el diagrama de estados de funcionamiento de dicho semáforo.

Modificar el diagrama, incluyendo un botón que permitirá el paso a los peatones cuando lo soliciten. Considere que el semáforo garantiza el paso de vehículos durante un tiempo mínimo.



1.1. Introducción

1.2. La complejidad inherente del software

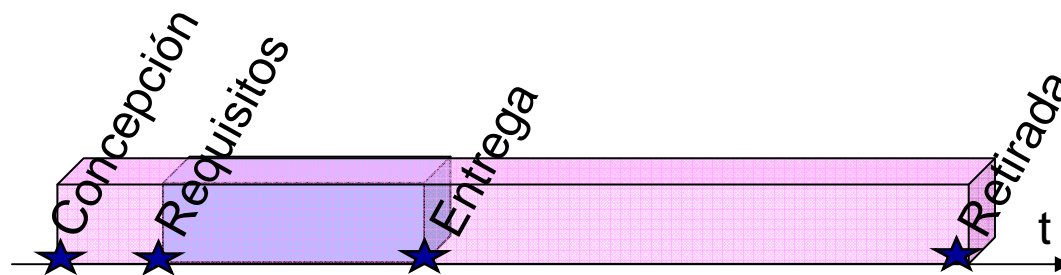
1.3. El ciclo de vida del software

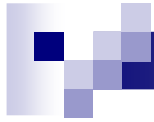
1.4. Introducción a la OO

Introducción al Ciclo de vida

- Procesos: Conjunto de actividades para la producción de software
- Actividades: Conjunto de tareas
- Tarea: Acción que transforma E en S

- Ciclo de desarrollo (Requisitos, análisis, diseño, implementación y pruebas).
 - Requisitos <-> Entrega
- Ciclo de vida (Adquisición, suministro, desarrollo, explotación y mantenimiento del sw (IEEE 1074)).
 - Concepción <-> Retirada





1.1. Introducción

1.2. La complejidad inherente del software

1.3. El ciclo de vida del software

1.4. Metodología PU (e Introducción a la OO)



UN EJEMPLO: El Proceso Unificado de Desarrollo

The unified software development process, Ivar Jacobson, Grade Booch, James Rumbaugh, Ed. Addison Wesley, 1999

El proceso unificado de desarrollo, Ivar Jacobson, Grade Booch, James Rumbaugh, Ed. Addison Wesley, 1999



Introducción

- Sistemas más complejos y más grandes.
- Objetivo:
 - Desarrollo más rápido y software de calidad.
- Es necesario un proceso que integre todas las facetas de un desarrollo software:
 - De una guía para ordenar las actividades del equipo.
 - Dirija las tareas individuales y del equipo.
 - Especifique los productos que hay que desarrollar.
 - Ofrezca criterios para monitorizar y medir los productos y actividades.



El Proceso Unificado (UP)

- Unificación de tres metodologías de desarrollo basadas en el paradigma orientado a objetos.
 - OOSE: Object Oriented Software Engineering (Casos de Uso) Jacobson, I.
 - Booch (Diseño) Booch, G.
 - OMT: Object Modeling Technique (Análisis) Rumbaugh, J.



El Proceso Unificado (UP)

- Es un proceso de desarrollo software.
- Un pds es un conjunto de actividades para transformar los requisitos de usuario en un sistema software.
- Usa UML - Unified Modeling Language
- Está caracterizado por:
 - 📄 Dirigido por **casos de uso**.
 - 📄 Centrado en la **arquitectura**.
 - 📄 **Iterativo e incremental**.



Dirigido por casos de uso

- Actor: alguien o algo.
- Si hay una interacción del sistema con alguien o algo, tiene que haber un caso de uso. (ejemplo cajero)
- Un caso de uso:
 - Es una función del sistema que da al actor un resultado útil o de interés.
 - Captura los requisitos funcionales.
- ¿Qué debe hacer el sistema *para cada actor*?
- Modelo de casos de uso.



Dirigido por casos de uso

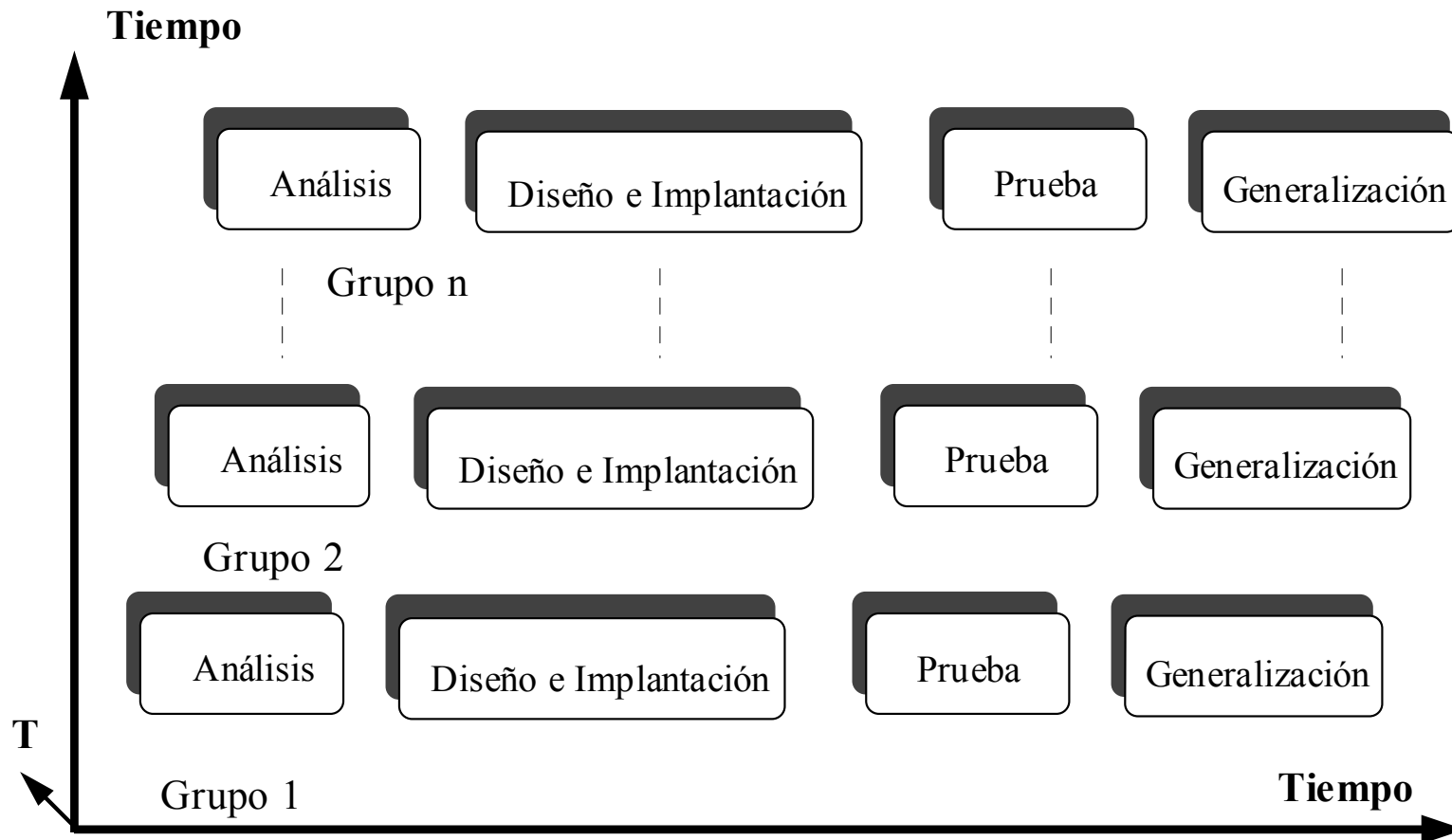
- Conducen el proceso de desarrollo:
 - Los desarrolladores crean modelos de diseño e implementación que realizan los casos de uso.
 - Los encargados de pruebas aseguran que los componentes implementan los casos de uso.
- Los casos de uso se especifican, se diseñan y sirven de base para construir los casos de prueba.
- Se desarrollan junto a la arquitectura del sistema.
- Ambos evolucionan paralelamente.



Iterativo - Incremental

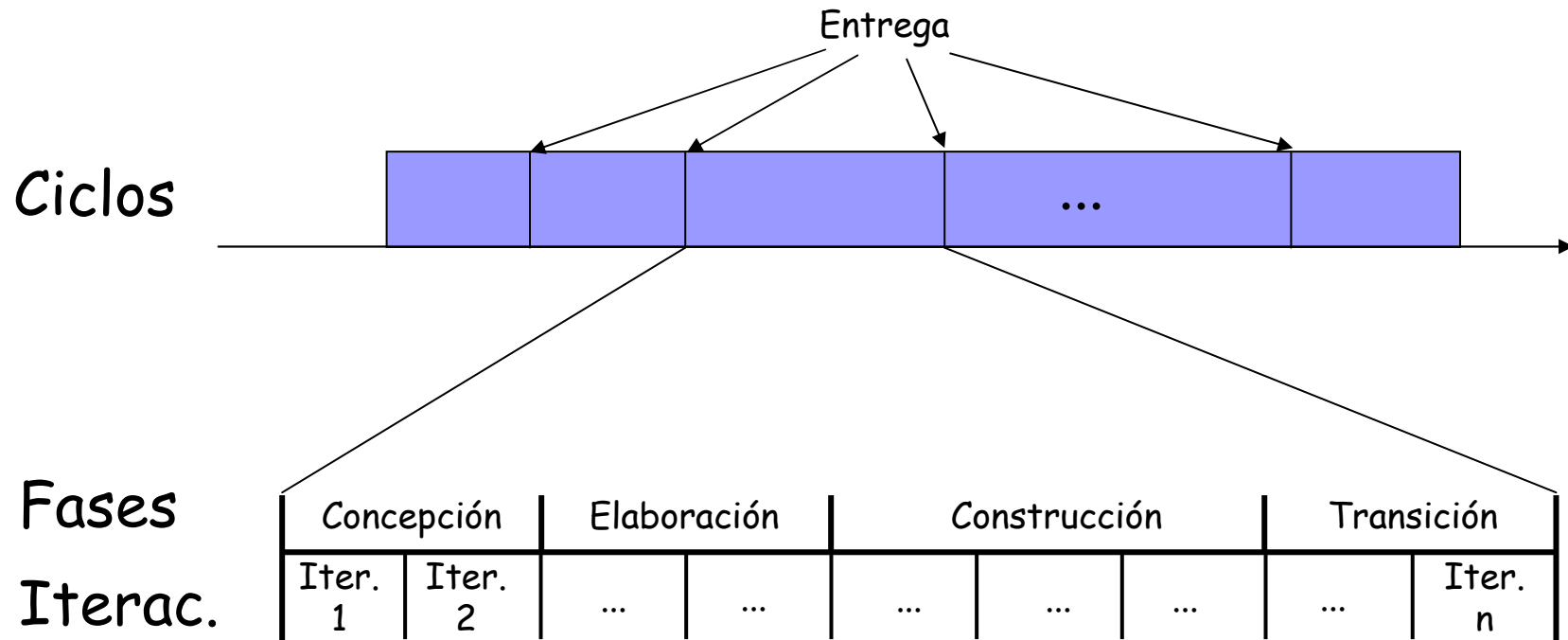
- División del proyecto.
- Una iteración produce un incremento.
- Iteraciones controladas.
- Factores para la selección en una iteración:
 - La iteración trata un grupo de casos que extienden la funcionalidad.
 - La iteración trata los riesgos más importantes.
- Cada iteración:
 - casos relevantes-diseño guiado por
 - arquitectura- implementar-verificar
- Beneficios.

Modelo Incremental

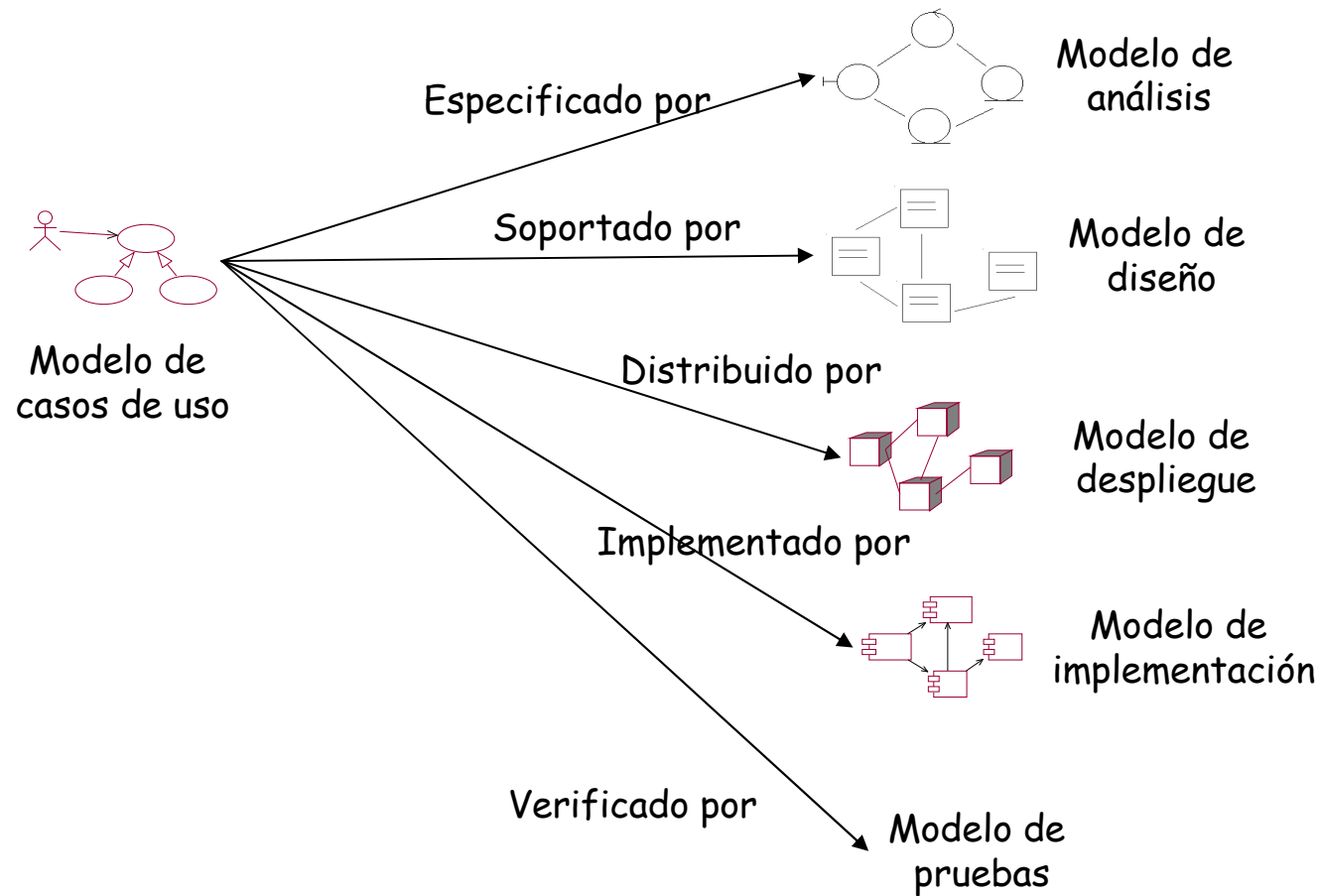


Ciclo de vida

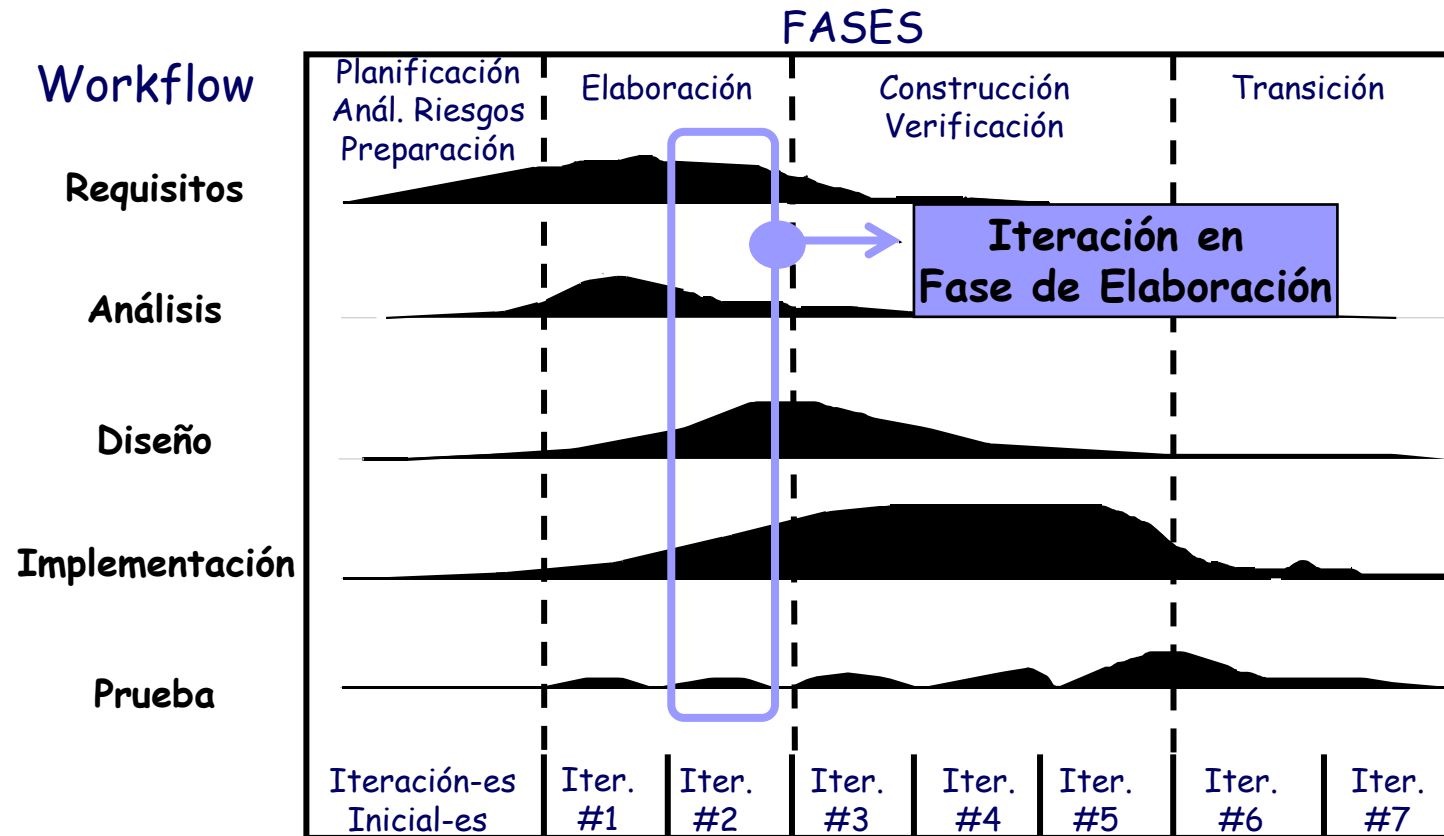
- Varios ciclos que concluyen con un producto.
- Código fuente, manuales y documentos.
- Hitos por fases



El producto



Fases en un ciclo



(Adaptado de Jacobson, 1999)



1.1. Introducción

1.2. La complejidad inherente del software

1.3. El ciclo de vida del software

1.4. Introducción a la OO



Qué es Orientado a Objetos

- Organizar el software como una colección de objetos que contiene tanto estructuras de datos como comportamiento.
- Características
 - Identidad
 - Clasificación
 - Polimorfismo
 - Herencia

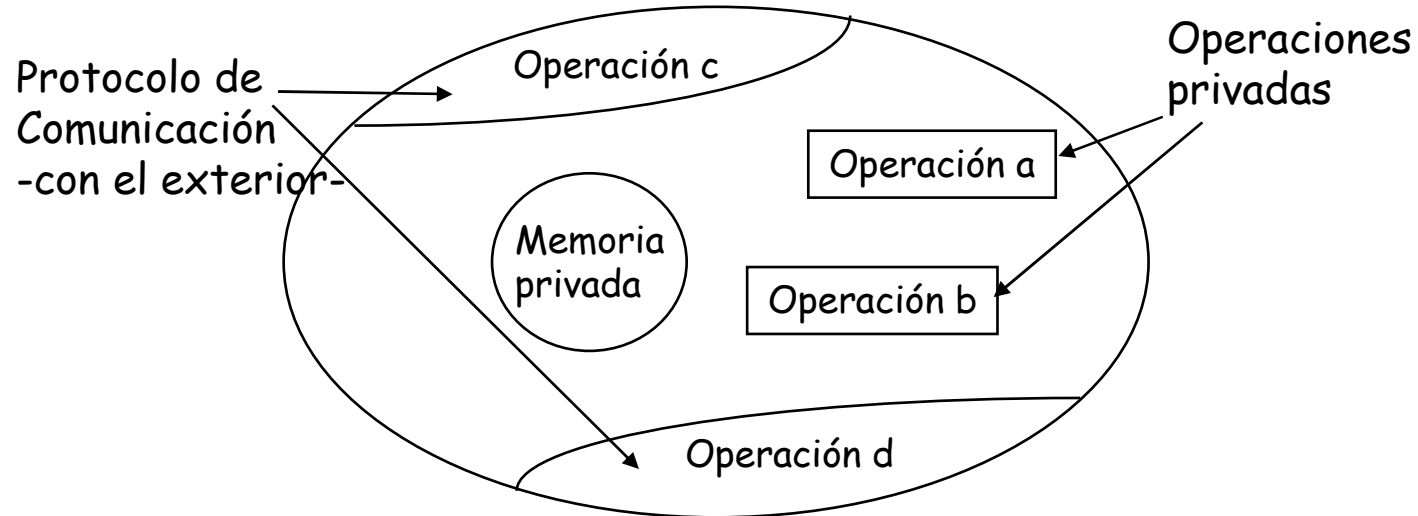


Características de los Objetos

- Identidad
 - Los datos se cuantifican en objetos
- Clasificación
 - Objetos con la **misma** estructura de datos y comportamiento se aglutinan y forman una clase
- Polimorfismo
 - Una misma operación puede comportarse de **distinto modo** en clases distintas (Calcular perímetro – Cuadrado, Círculo-, Mover Ficha –parchís, ajedrez-)
- Herencia
 - Compartir atributos y operaciones entre clases que tienen una relación jerárquica

Objeto

- Objeto: atributos + procedimientos (métodos, operaciones, servicios)

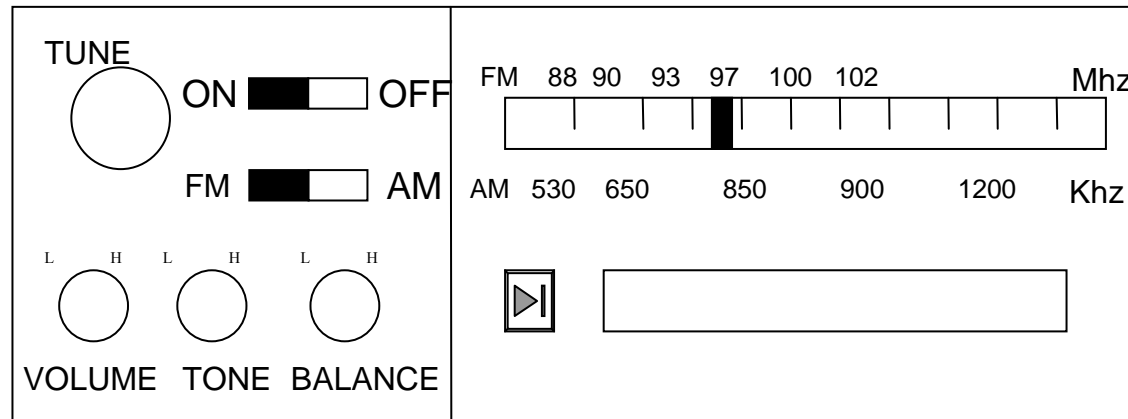




Objeto

- Los objetos son módulos software que contienen datos y funciones
- Cada objeto tiene su responsabilidad
- Acceso a la memoria privada: solo mediante operaciones en el protocolo
- Protocolo: define el comportamiento del objeto

Objeto. Ejemplo



memoria privada:

estado del volumen
estado del balance
apagado/encendido
cassette introducido

estado del tono
AM/FM
frecuencia seleccionada



Objeto. Ejemplo

■ protocolo:

acciona volumen

acciona balance

apaga/enciende

avanza cinta

acciona tono

selecciona AM/FM

selecciona frecuencia

■ operaciones privadas:

reproducir otra cara cuando cinta al final



Mensajes

- Un **mensaje** es una solicitud de un servicio a un objeto. Llamada a una rutina. Formado por :
 - Objeto.Método
- **Método** es una descripción formal de cómo la clase implementa un mensaje
- Participantes
 - Objeto Emisor (Agente activo)
 - Objeto Receptor (Agente pasivo). El método debe estar definido y ser de la interfaz.
- Los mensajes aparecen en el código del que llama.
- Un objeto siempre devuelve otro objeto (a veces él mismo) como respuesta a un mensaje

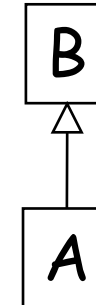


Clases

- Una clase es una abstracción de un objeto. Una especificación para un n^o arbitrario de objetos similares (profesor, nómina, coche, ordenador)
- Describe el comportamiento y los atributos de los objetos que representa la clase.
- Un objeto es una instancia de una clase. Todas las instancias de una misma clase tienen la misma memoria privada (seguramente con diferentes valores).
- Es la unidad modular en el paradigma OO

Herencia

- Es un mecanismo que permite definir una clase a partir de otra. La nueva clase hereda todos los atributos y métodos de la clase antecesora
- Si A es subclase de B:
 - A hereda de B
 - A es un refinamiento de B
 - A está más especializada que B
 - A es una subclase de B
 - B es superclase de A
- “es un”: A es un B
- Tipos de herencia: simple y múltiple



Herencia simple

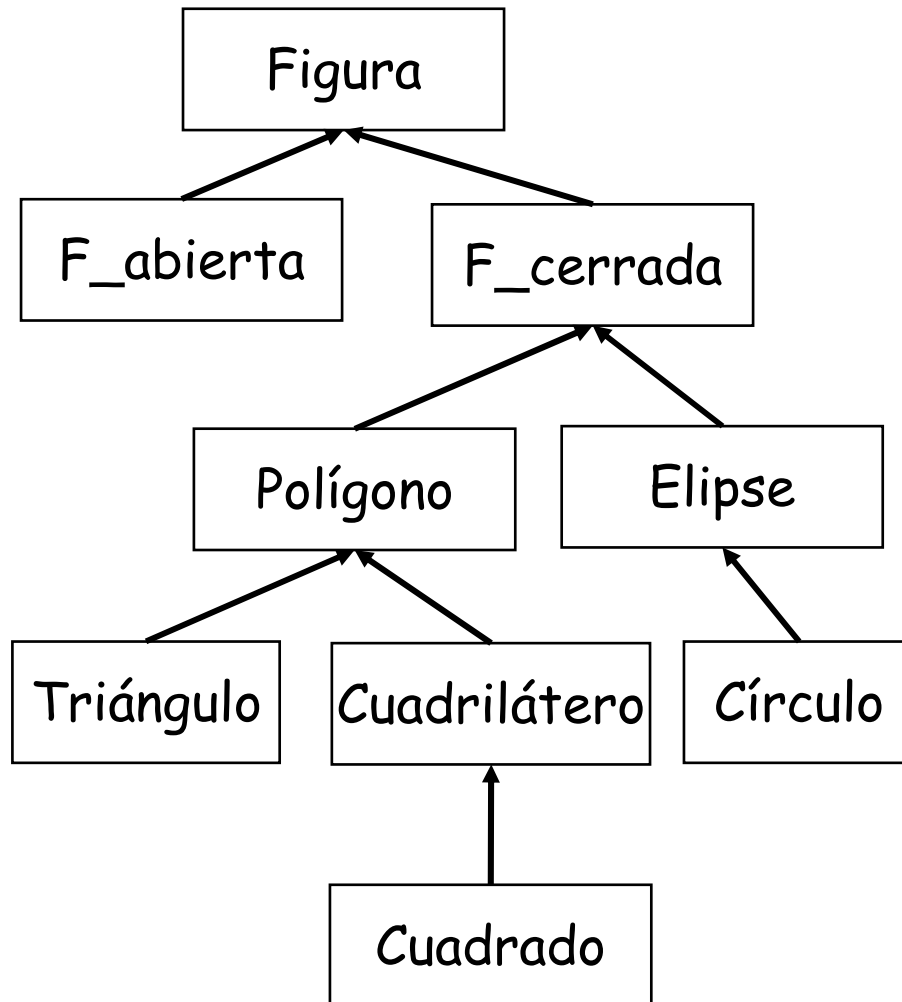


Figura:

color

rota, mueve, dibuja

F_cerrada:

perímetro, área

Polígono:

número_lados

Cuadrilátero:

perímetro, área

Cuadrado:

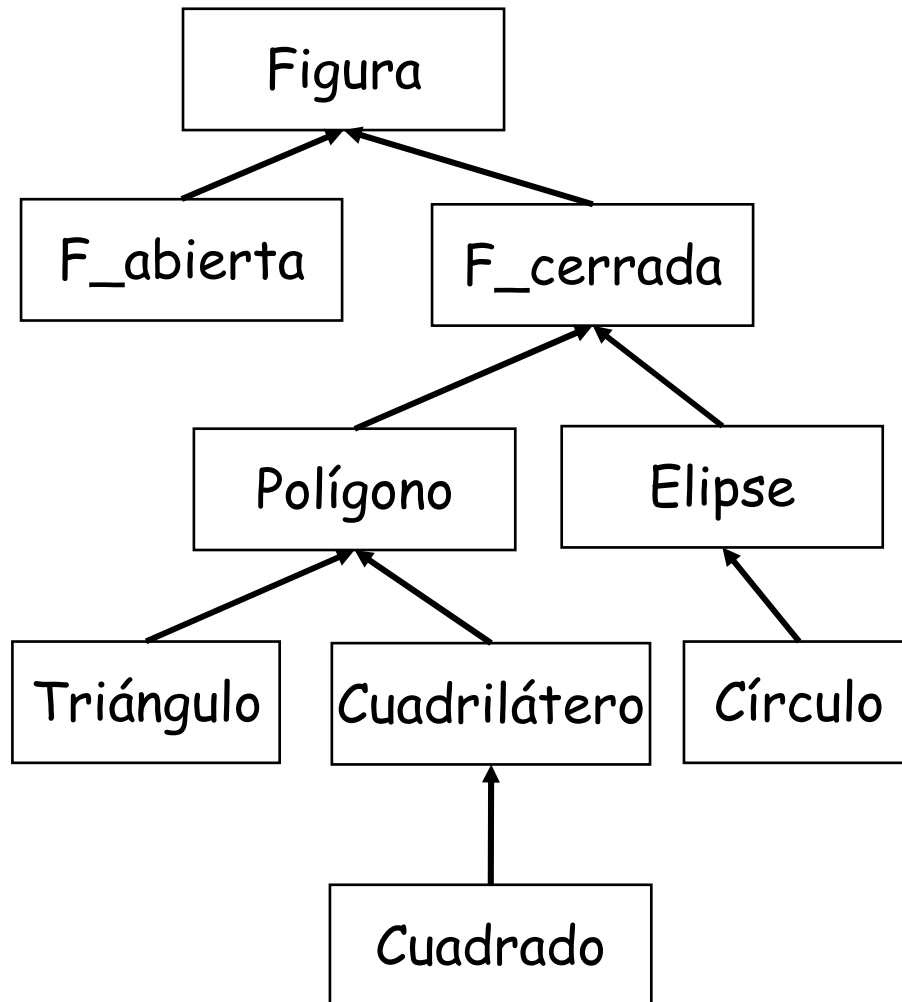
perímetro, área

Círculo:

radio

perímetro

Herencia simple



Todas las figuras se pueden:
rotar
escalar

Todas tienen un color

Clases abstractas:
Figura
F_Cerrada
F_Abierta
Polígono



Polimorfismo

- Es la habilidad de dos o más clases de objetos de responder al mismo mensaje de forma diferente
- Por ejemplo: Perímetro
- En las clases *Cuadrado*, *Circulo*, *Triangulo* se puede definir el método *Perimetro* cada uno especializado para su clase



Bases del Paradigma OO

- **Abstracción:**

Proceso mental por el que el ser humano extrae las características esenciales de algo, e ignora los detalles superfluos

- **Encapsulación:**

Proceso por el que se ocultan los detalles de las características de una abstracción



Bases del Paradigma OO

- **Modularización:**

Proceso de descomposición de un sistema en un conjunto de elementos con un índice bajo acoplamiento (independientes) y alto índice de cohesión (con significado propio)

- **Jerarquización:**

Proceso de estructuración por el que se organizan un conjunto de elementos en distintos niveles, atendiendo a determinados criterios