



INGENIERÍA DEL SOFTWARE I

Tema 4

Diseño de Software

Univ. Cantabria – Fac. de Ciencias

Francisco Ruiz



Objetivos

- Tener una **visión general** de los principios, características y métodos de **diseño del software**.
- Comprender la importancia de tener definida una correcta y adecuada **arquitectura** del sistema.
- Conocer las características generales de los principales **estilos arquitecturales**.
- Tener una visión general de los distintos tipos de **notaciones** gráficas y textuales para artefactos de diseño software.
- Conocer las características generales de las principales **estrategias y métodos** de diseño.



Contenido

- Introducción
 - Definición
 - Diseño Arquitectural vs Detallado
- Principios del Diseño de Software
 - Descomposición
- Principales Retos
 - Aspectos
- Arquitectura del Software
 - Vistas Arquitecturales
 - Estilos Arquitecturales
 - Estilos de Control
 - Patrones de Diseño
- Notaciones
 - Descripciones Estructurales
 - Descripciones de Comportamiento
- Tipos de Modelos
 - Arquitecturales
- Estrategias y Métodos
 - Diseño Estructurado
 - Diseño OO
 - Diseño Centrado en los Datos
 - Diseño con Componentes



Bibliografía

- Básica
 - IEEE Computer Society (2004)
 - SWEBOK - Guide to the Software Engineering Body of Knowledge, 2004 Version.
 - Capítulo 3.
 - <http://www.swebok.org/>
 - Caps. 8 y 11 del libro de Sommerville (2005).
- Complementaria
 - Cap. 14 del libro de Sommerville (2005).
 - Caps. 8 y 9 del libro de Pressman (2005).
 - Cap. 6 y 7 del libro de Piattini (2007).
 - Cap. 5 del libro de Pfleeger (2002).



Introducción - Definición

- En sentido general, **diseñar** es una forma de **resolución de problemas**.
- Por ello, al diseñar se utilizan nociones como
 - Objetivos
 - Restricciones
 - Alternativas
 - Representaciones
 - Soluciones



Introducción - Definición

- Juega un papel clave en el **desarrollo de software** porque permite a los ingenieros de software producir diversos **modelos** que:
 - Caracterizan la solución a implementar.
 - Pueden ser analizados y evaluados con el fin de determinar si se satisfacen los requisitos.
 - Facilitan el examen y evaluación de alternativas.
 - Sirven para planificar las siguientes actividades del desarrollo.



Introducción - Definición

- **Perspectiva del Proceso**
 - Diseñar es el **esfuerzo** para definir la arquitectura, componentes, interfaces y otras características de un sistema o componente [IEEE 610-1990].
 - El **Diseño de Software** es la actividad del ciclo de vida del software en la cual se analizan los **requisitos** para producir una **descripción de la estructura interna** del software que sirva de base para su **construcción**.
 - La **salida** es un conjunto de modelos y artefactos que registran las principales decisiones adoptadas.



Introducción - Definición

- **Perspectiva del Resultado**
 - Un Diseño es el **resultado** de dicho esfuerzo.
 - Un **Diseño Software** describe:
 - La **arquitectura** del software (cómo está descompuesto y organizado en **componentes**),
 - La **interfaces** entre dichos componentes, y
 - Los componentes a un nivel de **detalle** que permita su construcción.



Introducción – Diseño Arquitectural vs Detallado

- El estándar ISO 12207 identifica **dos tipos de Diseño Software**:
 - **Arquitectural** [alto nivel]
 - Describe la estructura y organización de alto nivel, es decir, los sub-sistemas o componentes y sus relaciones
 - **Detallado**
 - Describe cada componente y su comportamiento específico, de forma que puede procederse a su construcción



Introducción – Diseño Arquitectural vs Detallado

- **Diseño Arquitectural**
 - Es el primer paso en el diseño de un sistema, previo al diseño detallado.
 - Su resultado se conoce como **arquitectura del software**. *[se presenta después]*
 - Representa el enlace entre la especificación de requisitos y el diseño.
 - Puede llevarse a cabo en paralelo con actividades de especificación de requisitos.
 - Implica un esfuerzo creativo, de forma que las actividades a realizar pueden cambiar según la naturaleza del sistema a desarrollar.



Introducción – Diseño Arquitectural vs Detallado

- Durante el **diseño arquitectural** es necesario adoptar algunas **decisiones**:
 - ¿Existe una arquitectura genérica que pueda ser usada?
 - ¿Cómo será distribuido el sistema?
 - ¿Qué estilos arquitectónicos son apropiados?
 - ¿Qué aproximación se utilizará para estructurar el sistema?
 - ¿Cómo se descompondrá el sistema en módulos?
 - ¿Qué estrategia de control se utilizará?
 - ¿Cómo se evaluará el diseño arquitectural resultante?
 - ¿Cómo se documentará la arquitectura?



Principios del Diseño de Software

- **Principios**
 - Verdades básicas o leyes generales que se utilizan como base de razonamiento o como guía para actuar.
- Los **Principios del Diseño Software** son nociones clave consideradas fundamentales en muchas aproximaciones y conceptos de diseño diferentes.



Principios del Diseño de Software

- **Abstracción**
 - Olvidar información que diferencia ciertas cosas y así poder tratarlas como si fueran similares.
- En Software los **mecanismos básicos de abstracción** son:
 - Parametrización
 - Especificación
 - Abstracción Procedural
 - Abstracción de Datos
 - Abstracción de Control (iteración)



Principios del Diseño de Software

- **Acoplamiento y Cohesión**
 - **Acoplamiento:** Fortaleza de las relaciones entre módulos
 - INTER
 - **Cohesión:** cómo están relacionados los elementos de un mismo módulo.
 - INTRA



Principios del Diseño de Software

- **Descomposición**
 - Descomponer un software en diversas unidades más pequeñas, habitualmente con fines de situar diferentes funcionalidades o responsabilidades en diferentes componentes.
- **Encapsulamiento** [ocultamiento de información]
 - Consiste en agrupar y empaquetar los elementos y detalles internos de una abstracción y hacer que dichos detalles sean inaccesibles desde fuera.



Principios del Diseño de Software

- **Separación de Interfaz e Implementación**
 - Definir un componente especificando un interfaz público, conocido por otros componentes o clientes, separado de los detalles de cómo dicho componente es realizado (implementado).
- **Suficiencia y Completitud**
 - Asegurar que un componente software captura todas las características importantes de una abstracción y ninguna más.



Principios del Diseño de Software - Descomposición

- Hay dos tipos de **Descomposición**
 - **Estructuración/Organización del sistema:**
 - El sistema en sub-sistemas
 - **Descomposición modular:**
 - Sub-sistemas en módulos.
- Sub-sistemas vs Módulos
 - No siempre hay una diferenciación clara
 - **Sub-sistema:** Un sistema en sí mismo, cuyo funcionamiento es independiente de los servicios provistos por otros sub-sistemas.
 - **Módulo:** Componente de un sistema que provee servicios a otros componentes y que no se considera un sistema separado.



Principios del Diseño de Software - Descomposición

- Aproximaciones para **Descomposición Modular:**
 - **Objetos**
 - El (sub)-sistema se decompone en objetos que interactúan.
 - **Tubería o Flujo de Datos** [orientado a funciones]
 - El (sub)-sistema se decompone en módulos funcionales que transforman entradas en salidas.



Principales Retos

- Al diseñar software es necesario enfrentarse a varios problemas o dificultades importantes.
 - Atributos de **Calidad** que deben satisfacerse.
 - Ej: Rendimiento.
 - Cómo descomponer, organizar y empaquetar **componentes**.
 - **Aspectos** del comportamiento del software que no son del dominio del problema o aplicación, sino de dominios laterales que afectan de manera transversal a la funcionalidad del sistema.
 - Estos **aspectos** no suelen suponer unidades de descomposición funcional, sino que son propiedades que afectan a diversos componentes (en su rendimiento o semántica) de forma sistemática.



Principales Retos

- En los últimos años se está abordando el problema del Diseño de **Sistemas Software Genéricos** mediante **Líneas de Producto Software**
- Su objetivo es el diseño de **familias de programas**, es decir, colecciones de programas que tienen muchas cosas en común.
 - Ej: Gestión de Tiendas de Venta al Público
- Haciendo reutilización al máximo, pero permitiendo adaptación y variabilidad en cada producto particular.
 - Gestión de Tiendas de Ropa / Videoclubs / Supermercados



Principales Retos - Aspectos

- Los principales **Aspectos Software** a enfrentarse son:
 - **Concurrencia.**
 - Cómo repartir el software en procesos, tareas o hilos de ejecución y abordar los problemas de eficiencia, atomicidad, sincronización y planificación asociados.
 - **Control y Manejo de Eventos.**
 - Cómo organizar datos y flujo de control.
 - Cómo manejar eventos reactivos y temporales (mediante mecanismos como invocación implícita o llamadas hacia atrás).



Principales Retos - Aspectos

- Los principales **Aspectos Software** a enfrentarse son (cont.):
 - **Distribución de componentes.**
 - Cómo distribuir el software en el hardware.
 - Cómo comunicar los componentes.
 - Cómo utilizar el middleware para tratar con software heterogéneo.
 - **Manejo de Errores y Excepciones y Tolerancia a Fallos.**
 - Cómo prevenir y tolerar fallos y tratar condiciones excepcionales (no previstas).



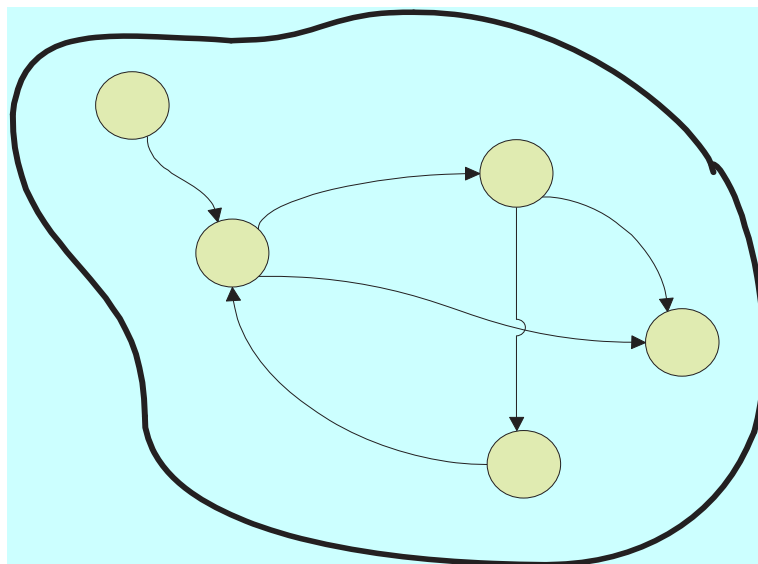
Principales Retos - Aspectos

- Los principales **Aspectos Software** a enfrentar son (cont.):
 - **Interacción y Presentación.**
 - Cómo estructurar y organizar las interacciones con el usuario y la presentación de información.
 - Ej: separando presentación de la lógica de negocio usando la aproximación MVC (Modelo-Vista-Controlador).
 - **Persistencia de Datos.**
 - Cómo se manejan los datos que tienen una vida superior e independiente a las ejecuciones del software.



Arquitectura del Software

- La **arquitectura** de un sistema es la descripción de los elementos que lo forman y de las interrelaciones entre ellos.





Arquitectura del Software

- **Arquitectura**
 - Estructura interna de algo
 - Forma en que algo es construido u organizado
- **Arquitectura Software**
 - *Descripción de los subsistemas / componentes de un sistema software y de las interrelaciones entre ellos*



Arquitectura del Software

- Disponer de la Arquitectura de forma explícita supone las siguientes **ventajas**
 - **Comunicación con interesados**
 - Puede utilizarse para la discusión sobre cómo será el sistema.
 - **Análisis del Sistema**
 - Permite el análisis del cumplimiento de los requisitos no funcionales.
 - **Reutilización a gran escala**
 - Puede servir para un grupo de sistemas parecidos.



Arquitectura del Software

- Los **atributos de calidad del sistema** (requisitos no funcionales) se ven afectados por la arquitectura:
 - **Rendimiento**
 - Utilizar componentes grandes en vez de grano fino para concentrar las operaciones críticas y minimizar las comunicaciones.
 - **Seguridad**
 - Usar una arquitectura por capas con los activos críticos en las capas más internas.
 - **Protección**
 - Localizar las características de protección críticas en un pequeño número de componentes.
 - **Disponibilidad**
 - Incluir componentes redundantes y mecanismos de tolerancia a fallos.
 - **Mantenibilidad**
 - Usar componentes de grano fino más fácilmente sustituibles.



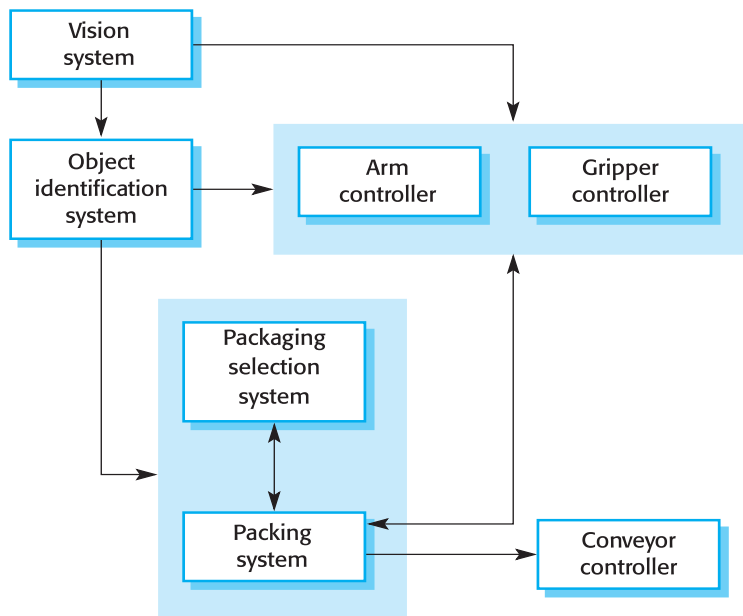
Arquitectura del Software

- Pero pueden aparecer **conflictos**:
 - Rendimiento vs Mantenibilidad
 - Usar componentes grandes mejora el rendimiento pero reduce la mantenibilidad.
 - Disponibilidad vs Seguridad
 - Introducir datos redundantes mejora la disponibilidad pero hace más difícil la seguridad.
 - Protección vs Rendimiento
 - Localizar las características de protección en diversos componentes suele significar más comunicación y por tanto, un rendimiento peor.



Arquitectura del Software

- Se suele expresar mediante un **diagrama de bloques** que **resume** la estructura del sistema.



Arquitectura de un sistema de control de un robot de empaquetar



Arquitectura del Software - Vistas

- Un Diseño Software puede/debe describir y documentar diferentes **vistas**.
 - Una Vista representa un aspecto parcial de un arquitectura software que muestra propiedades específicas de un sistema software.
- Un Diseño Software es un artefacto multi-perspectiva generalmente compuesto de vistas relativamente independientes y ortogonales.



Arquitectura del Software - Vistas

- Ejemplos de vistas son:
 - Lógica (requisitos funcionales)
 - De Proceso (conurrencia)
 - Física (distribución)
 - Desarrollo (descomposición del diseño en unidades de implementación)
- Otra clasificación también usada es:
 - Comportamiento
 - Funcional
 - Estructural
 - Modelado de Datos



Arquitectura del Software – Estilos Arquitecturales

- Un **estilo arquitectural** es un conjunto de restricciones que definen un conjunto o **familia de arquitecturas afines**, que satisfacen dichas restricciones.
- Puede ser visto como un modelo de modelos (meta-modelo) que enmarca a muy alto nivel la organización del software (**macro-arquitectura**).
- En sistemas grandes heterogéneos es común combinar varios estilos arquitecturales.



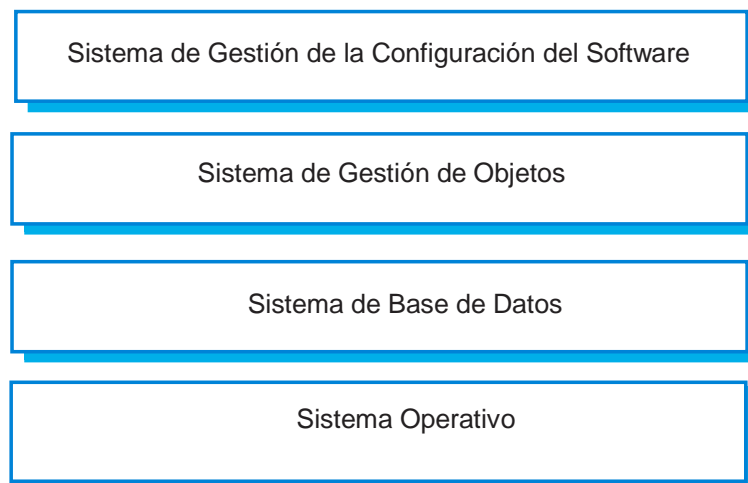
Arquitectura del Software - Estilos Arquitecturales

- Principales **estilos arquitecturales** :
 - **Estructura General**
 - **Capas**, tuberías y filtros, pizarra, **repositorio compartido**
 - **Sistemas Distribuidos**
 - **Cliente-servidor**, tres capas, broker
 - **Sistemas Interactivos**
 - MVC (Modelo-Vista-Controlador), PAC (Presentación-Abstracción-Control)
 - **Sistemas Adaptables**
 - Micro-núcleo, reflexión
 - **Otros**
 - Batch (lotes), intérpretes, control de procesos, basado en reglas



Arquitectura del Software - Estilos Arquitecturales

- **Capas** (Máquina Abstracta)
 - Organiza el sistema en un conjunto de capas, cada una de las cuales provee una serie de servicios a las capas superiores usando los de las capas inferiores.





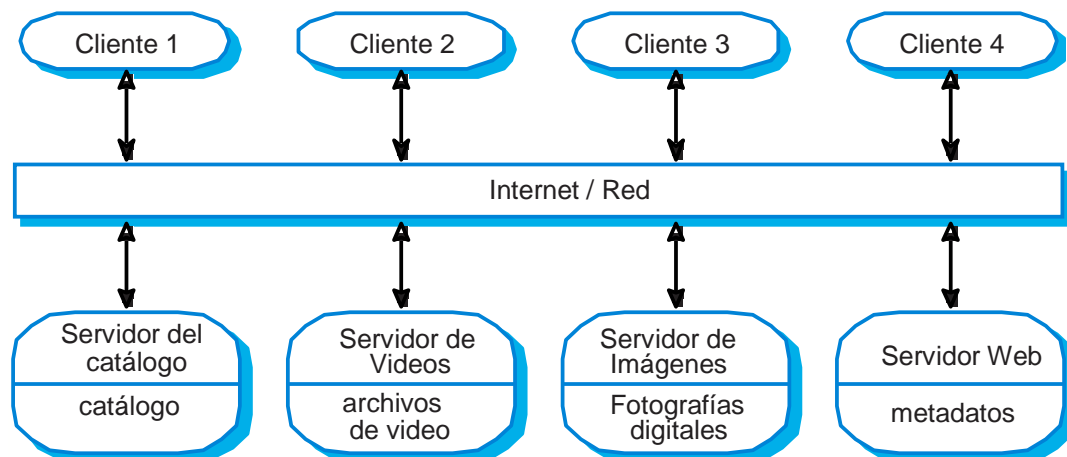
• Repositorio Compartido

- Datos comunes a los sub-sistemas se almacenan en una base de datos central o repositorio.
- Se emplea cuando se comparten muchos datos.
- **Ventajas**
 - Eficiente para compartir grandes cantidades de datos.
 - Sub-sistemas no necesitan “preocuparse” de la gestión (backups, seguridad, ...) de los datos.
 - Existe un modelo de datos compartido (esquema del repositorio).
- **Desventajas**
 - Todos los sub-sistemas deben usar el mismo modelo de datos.
 - La evolución de datos es difícil y costosa.
 - No caben políticas de gestión de datos específicas.
 - Es difícil hacer una distribución eficiente.



• Cliente Servidor

- Estructura un sistema distribuido en:
 - **Servidores** autosuficientes que proveen servicios (impresión, gestión de datos, ..).
 - **Clientes** que invocan dichos servicios.





- **Cliente Servidor**

- **Ventajas**

- La distribución de datos es real y directa;
- Se hace uso eficaz de sistemas en red. El hardware para ello puede ser barato;
- Fácil de añadir nuevos servidores o actualizar los existentes.

- **Desventajas**

- Los sub-sistemas usan diferentes modelos de datos. Esto puede hacer ineficiente el intercambio de datos;
- Gestión redundante en cada servidor;
- NO existe un registro central de nombres y servicios. Puede ser difícil averiguar qué servidores y servicios están disponibles.



- Estos estilos arquitecturales están enfocados al **flujo de control entre sub-sistemas**.

- **Control Centralizado**

- Un sub-sistema tiene toda la responsabilidad para controlar, iniciar y parar los otros sub-sistemas.
 - Llamada–Retorno (Call-Return)
 - Gestor (Manager)

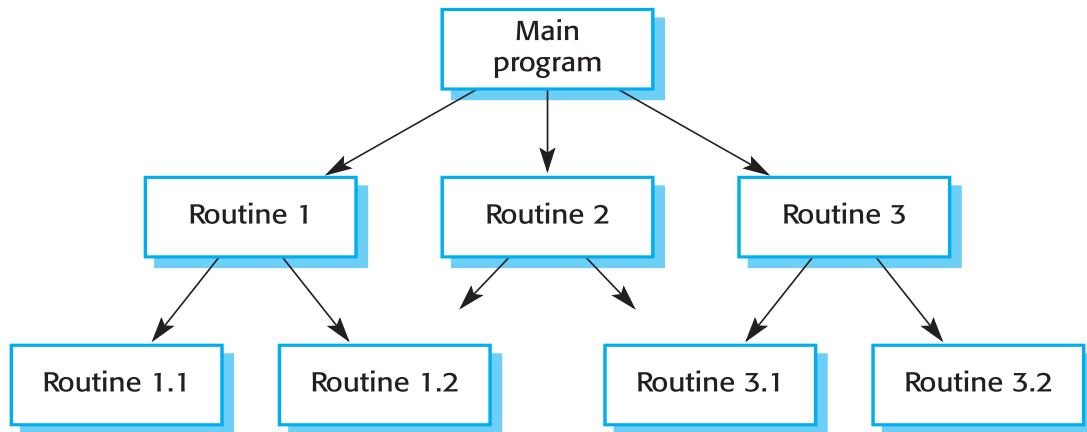
- **Control basado en Eventos**

- Cada sub-sistema puede responder a eventos generados externamente por los otros sub-sistemas o el entorno del sistema.
 - Difusión (Broadcast)
 - Guiado por Interrupciones



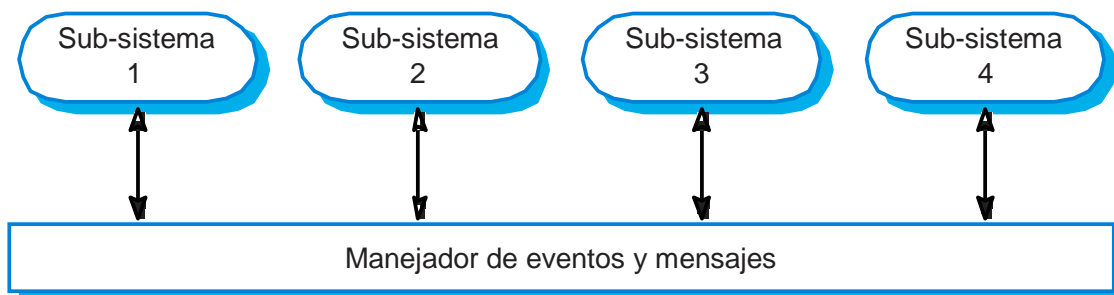
- **Control Centralizado**

- **Llamada-Retorno**
- Jerarquía Top-Down de Subrutinas



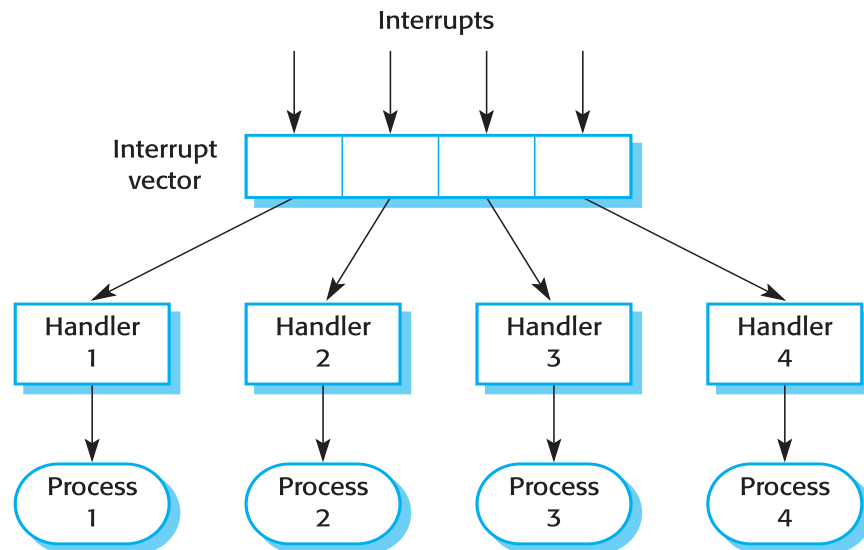
- **Control basado en Eventos**

- **Difusión (Broadcast)**
- Cuando ocurre un evento el control se transfiere al sub-sistema que puede tratarlo.
- Cada sub-sistema decide sobre los eventos que le interesan.





- **Control basado en Eventos**
 - **Guiado por Interrupciones**
 - Cada interrupción tiene un manejador.



- **Patrón**
 - Solución común a un problema común en un contexto dado.
- Los **estilos arquitecturales** pueden ser vistos como patrones para la organización de alto nivel del software (patrones macro-arquitecturales).
- Otros patrones de diseño sirven para describir a un nivel más bajo, más local (**patrones micro-arquitecturales**).



- Principales clases de **patrones de diseño** [micro-arquitecturales].
 - **Creacionales**
 - Constructor (builder), factoría (factory), prototipo (prototype), singleton
 - **Estructurales**
 - Adaptador (adapter), puente (bridge), compuesto (composite), decorador (decorator), fachada (facade), peso mosca (flyweight), delegado (proxy)
 - **De Comportamiento**
 - command, intérprete (interpreter), iterador (iterator), mediador (mediator), memento, observador (observer), estado (state), estrategia (strategy), plantilla (template), visitante (visitor)



- Existen muchas notaciones y lenguajes para representar los artefactos del diseño software.
 - Unas son para representar la estructura y otras el comportamiento
 - Unas sirven principalmente durante el diseño arquitectural, otras durante el diseño detallado, y algunas durante ambos.
 - Algunas se emplean principalmente en el contexto de métodos específicos.



Notaciones – Descripciones Estructurales

- Las siguientes notaciones describen **aspectos estructurales (estática)**, es decir, los componentes y sus interconexiones.
 - **Lenguajes de Descripción de Arquitecturas (ADLs)**
 - Lenguajes textuales formales ideados para describir una arquitectura software en términos de componentes y conectores.
 - **Diagramas de Clases y Objetos**
 - Para representar un conjunto de clases (y objetos) y sus interrelaciones.
 - **Diagramas de Componentes**
 - Para representar un conjunto de componentes (partes físicas y reemplazables de un sistema que son conformes a y proveen un conjunto de interfaces) y sus interrelaciones.



Notaciones – Descripciones Estructurales

- Las siguientes notaciones describen **aspectos estructurales (estática)**, es decir, los componentes y sus interconexiones (cont).
 - **Tarjetas CRC (Clase Responsabilidad Colaborador)**
 - Para denotar los nombres de los componentes (clases), sus responsabilidades, y los nombres de los componentes con los que colaboran.
 - **Diagramas de Despliegue**
 - Para representar un conjunto de nodos físicos y sus interrelaciones, modelando los aspectos físicos de un sistema.
 - **Diagramas Entidad-Interrelación**
 - Para representar modelos conceptuales de los datos almacenados en sistemas de información.



Notaciones – Descripciones Estructurales

- Las siguientes notaciones describen **aspectos estructurales (estática)**, es decir, los componentes y sus interconexiones (cont).
 - **Lenguajes de Descripción de Interfaces (IDLs)**
 - Similares a los lenguajes de programación normales, sirven para definir las interfaces (nombres y tipos de las operaciones exportadas) de los componentes software.
 - **Diagramas de Estructura de Jackson**
 - Para describir las estructuras de datos en términos de secuencia, selección e iteración.
 - **Grafo de Estructura**
 - Para describir la estructura de llamadas de los programas (qué módulos llaman y son llamados por qué módulos).



Notaciones – Descripciones del Comportamiento

- Las siguientes notaciones y lenguajes sirven para describir el **comportamiento (dinámica)** de un software y sus componentes.
 - **Diagramas de Actividad**
 - Para mostrar el flujo de control entre actividades (ejecuciones no atómicas dentro de una máquina de estados).
 - **Diagramas de Colaboración**
 - Para mostrar las interacciones entre un grupo de objetos, haciendo énfasis en los objetos, sus conexiones y los mensajes que intercambian en dichas conexiones.
 - **Diagramas de Flujo de Datos (DFDs)**
 - Para representar el flujo de datos entre un conjunto de procesos.



Notaciones – Descripciones del Comportamiento

- Las siguientes notaciones y lenguajes sirvan para describir el **comportamiento** (**dinámica**) de un software y sus componentes (cont).
 - **Tablas y Diagramas de Decisión**
 - Para representar combinaciones complejas de condiciones y acciones.
 - **Diagramas de Flujo [Estructurados]**
 - Para representar el flujo de control y las acciones asociadas que deben ser realizadas.



Notaciones – Descripciones del Comportamiento

- Las siguientes notaciones y lenguajes sirvan para describir el **comportamiento** (**dinámica**) de un software y sus componentes (cont).
 - **Diagramas de Secuencia**
 - Para mostrar las interacciones entre un grupo de objetos, con énfasis en la ordenación temporal de los mensajes.
 - **Diagramas de Transición de Estados y Grafos de Máquinas de Estados**
 - Para mostrar el flujo de control entre estados de una máquina de estados.



Notaciones – Descripciones del Comportamiento

- Las siguientes notaciones y lenguajes sirven para describir el **comportamiento (dinámica)** de un software y sus componentes (cont).
 - **Lenguajes de Especificación Formal**
 - Lenguajes textuales que usan nociones básicas matemáticas (lógica, conjuntos, secuencia) para definir de forma rigurosa y abstracta los interfaces y comportamiento de los componentes (habitualmente en términos de pre y post-condiciones).
 - **Pseudocódigo y Lenguajes de Diseño de Programas**
 - Lenguajes, al estilo de los tradicionales de programación estructurada, usados para describir, normalmente en la etapa de diseño detallado, el comportamiento de un procedimiento o método.



Tipos de Modelos

- Muchas de las notaciones anteriores son de tipo gráfico (**Modelos**).
- En una clasificación alternativa a la anterior (estructura-comportamiento), algunos de los principales **tipos de modelos del software** son
 - De Contexto
 - De Procesos
 - De Comportamiento
 - De Flujo de Datos
 - De Máquinas de Estados
 - De Datos
 - De Objetos

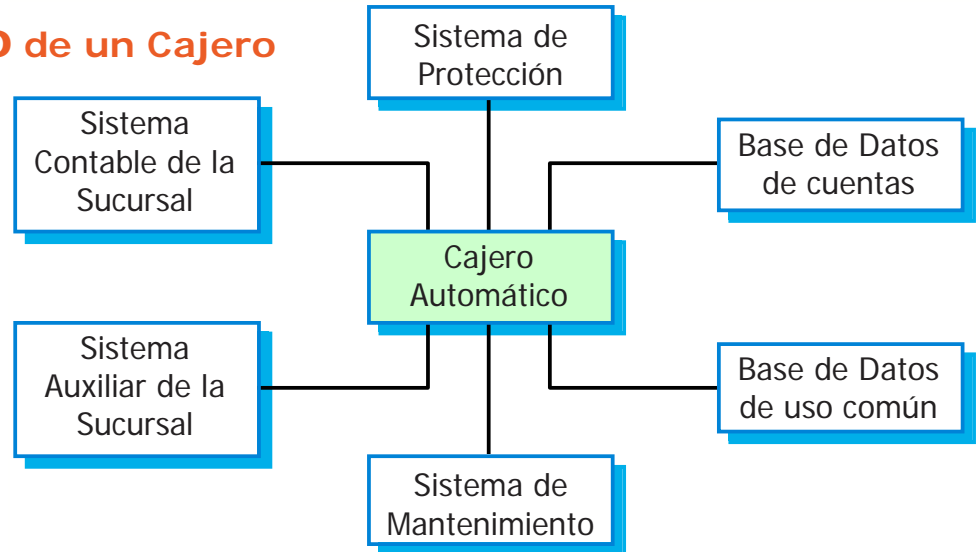


Tipos de Modelos

Modelos de Contexto

- Ilustran el contexto operacional de un sistema, mostrando los demás sistemas con los que se interactúa.

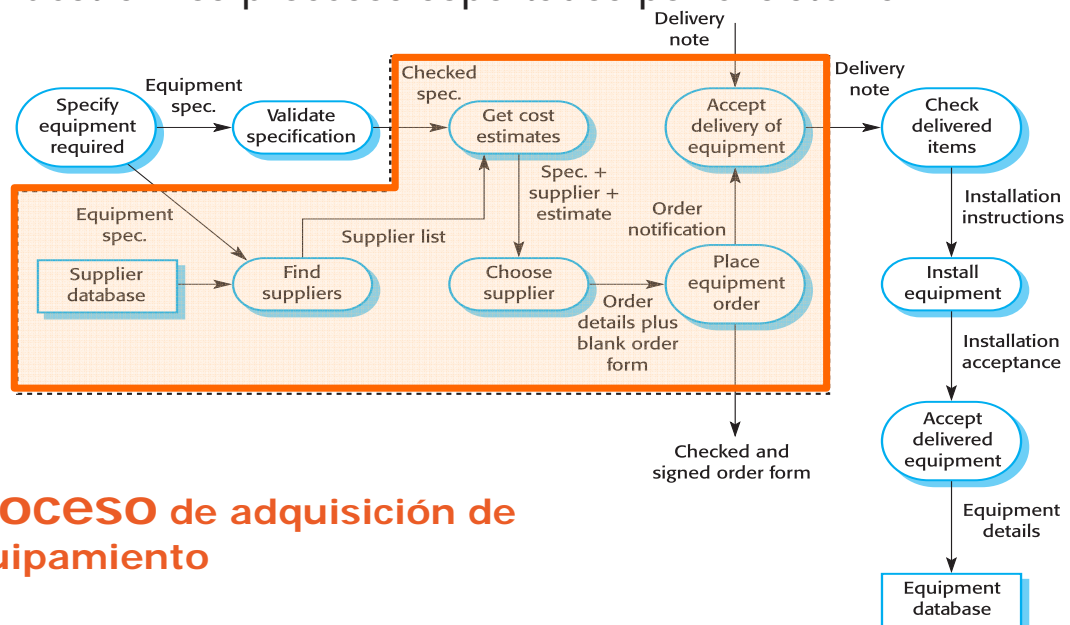
Contexto de un Cajero Automático



Tipos de Modelos

Modelos de Procesos (de Negocio)

- Muestran los procesos soportados por el sistema.



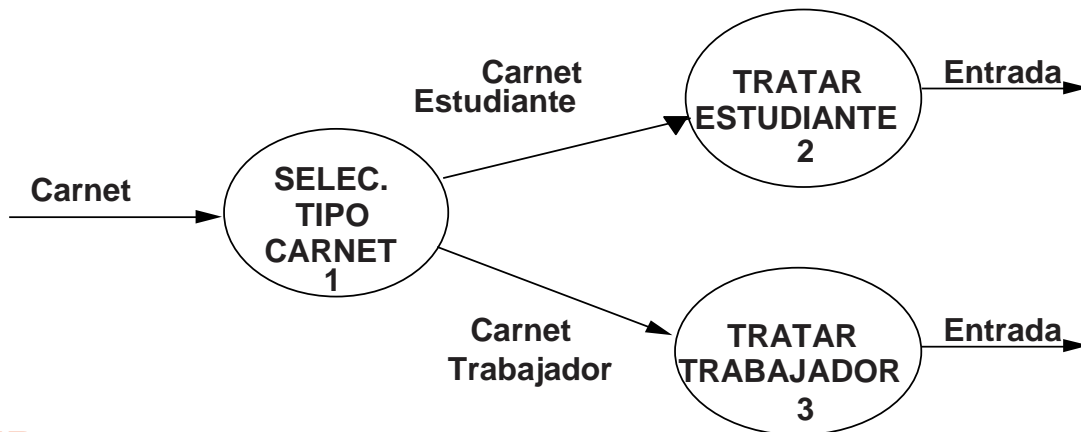
Proceso de adquisición de equipamiento



Tipos de Modelos

• Modelos de Comportamiento – Procesamiento de Datos

- Muestran cómo los datos son procesados por el sistema.



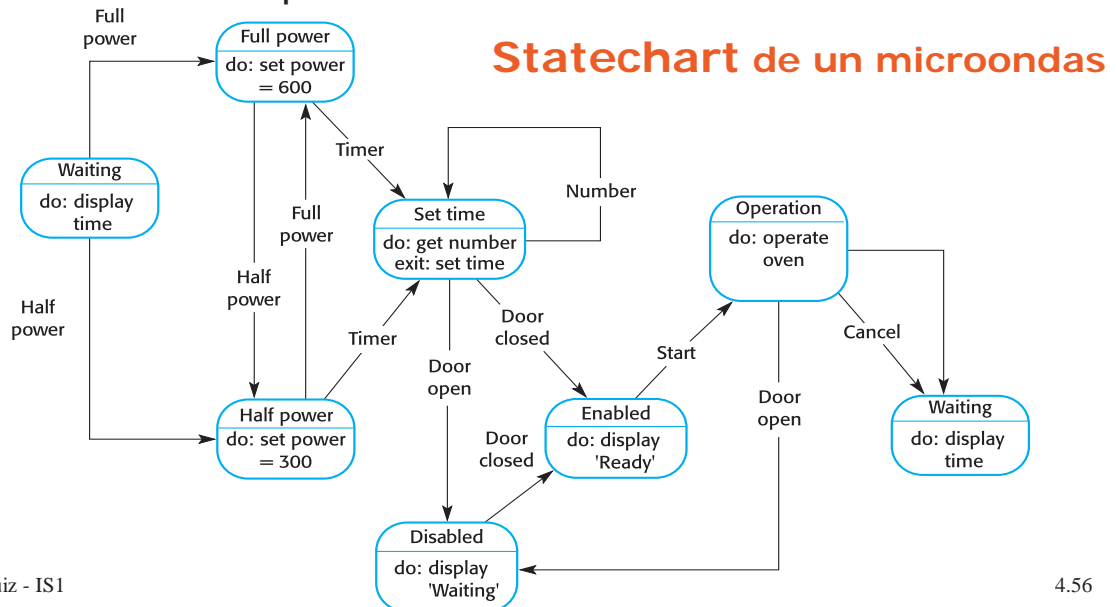
DFD



Tipos de Modelos

• Modelos de Comportamiento – Máquinas de Estado

- Muestran la respuesta del sistema ante eventos.



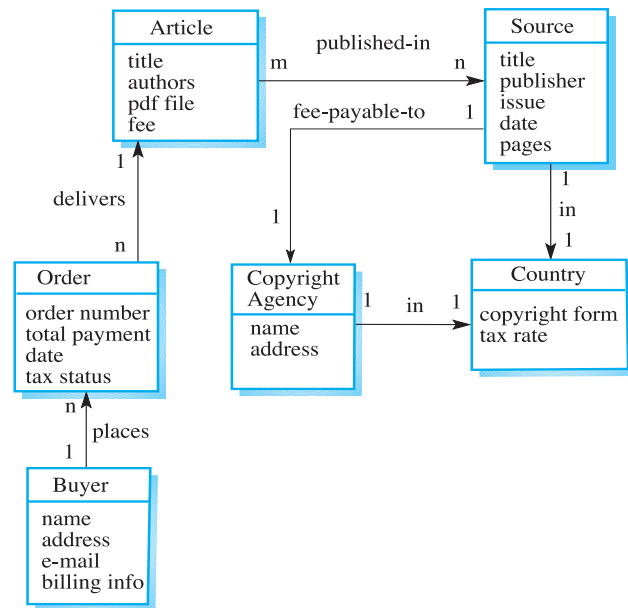


Tipos de Modelos

• Modelos de Datos

- Describen la estructura lógica de los datos procesados por el sistema.

E-R de un sistema de préstamo electrónico de artículos



Tipos de Modelos

• Modelos de Objetos

- Describen el sistema en términos de clases de objetos y sus asociaciones.
- Objetos/clases son una forma de reflejar las entidades del mundo real manipuladas por el sistema.
- Entidades más abstractas pueden ser difíciles de modelar con esta aproximación.
- Las clases reflejan entidades del dominio de aplicación que serán reutilizables en distintas partes del sistema.
- **UML** es el estándar para este tipo de modelos

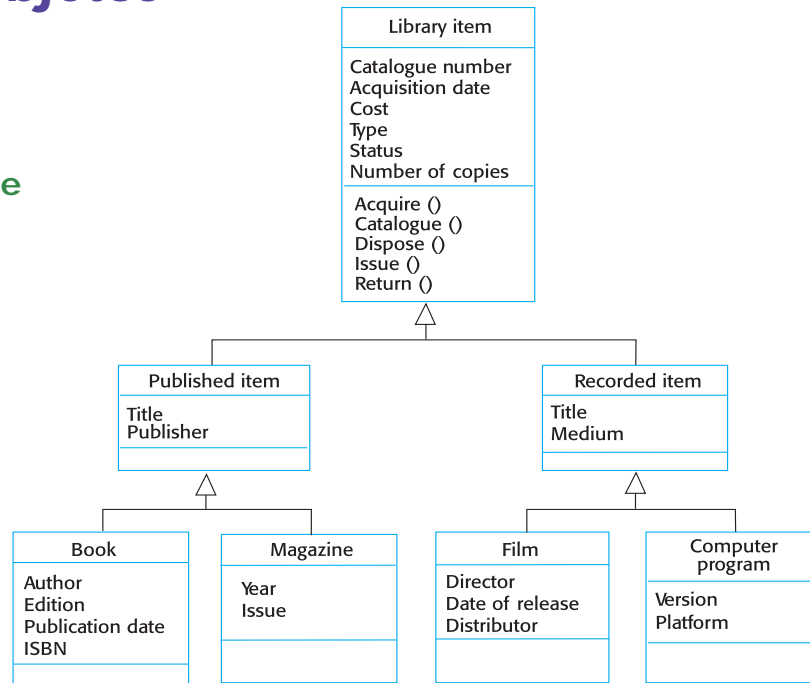


Tipos de Modelos

• Modelos de Objetos

Herencia

Jerarquía de clases de una biblioteca

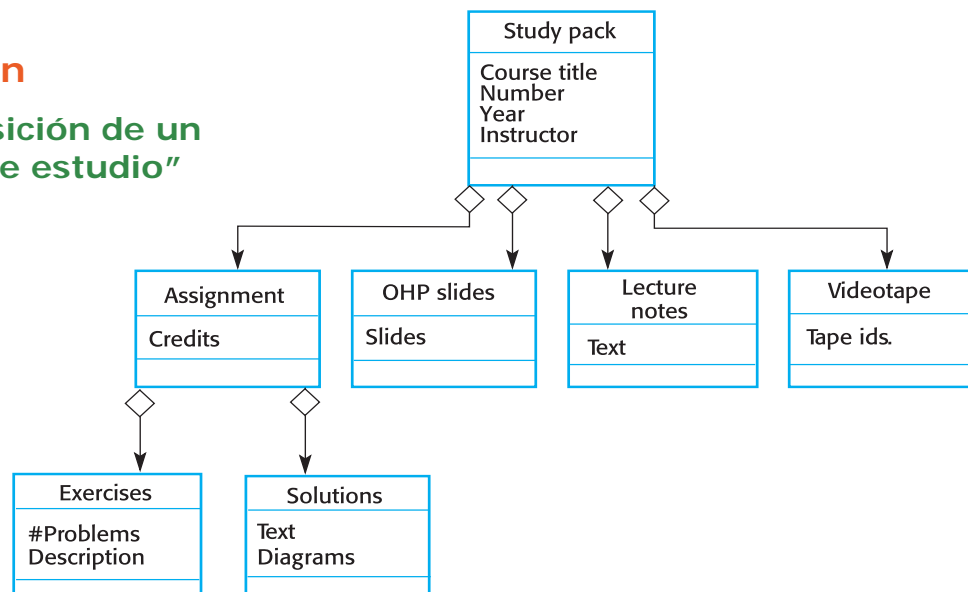


Tipos de Modelos

• Modelos de Objetos

Agregación

Descomposición de un "paquete de estudio"



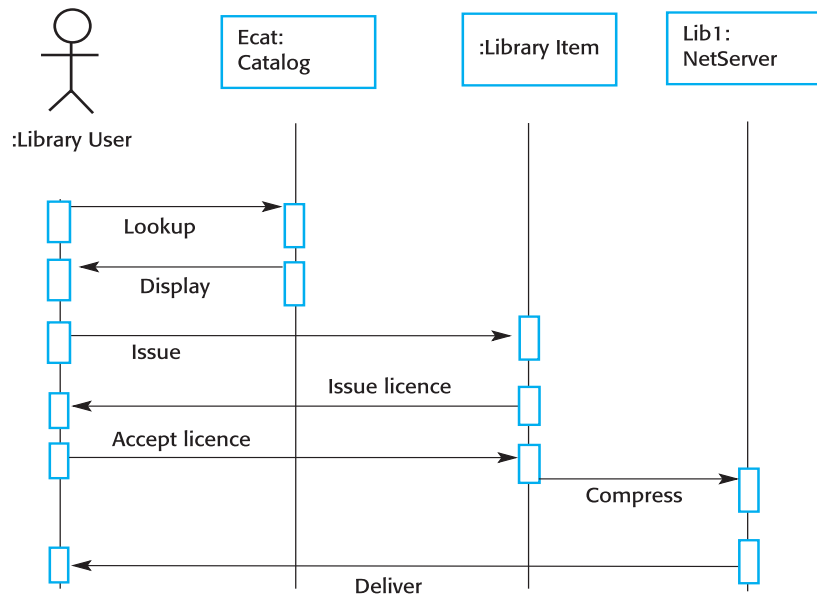


Tipos de Modelos

- **Modelos de Objetos**

Comportamiento

Diagrama de Secuencia del caso "préstamo electrónico de artículos"



Tipos de Modelos - Arquitecturales

- Se emplean para documentar la arquitectura del sistema
- Pueden ser
 - **Estructurales** estáticos, para mostrar los componentes principales o sub-sistemas.
 - De **Proceso** dinámicos, para mostrar la estructura de procesos del sistema.
 - De **Interfaces**, para definir las interfaces de los sub-sistemas.
 - De **Relaciones**, para mostrar las relaciones entre sub-sistemas de forma similar a un DFD.
 - De **Distribución**, para mostrar cómo los sub-sistemas se distribuyen entre diversas máquinas.



Estrategias y Métodos

- Las **estrategias generales** de diseño de software más conocidas son
 - Divide y vencerás
 - Refinamiento en pasos sucesivos
 - Top-down vs bottom-up
 - Abstracción de datos y ocultamiento de información
 - Uso de heurísticas
 - Uso de patrones
 - Aproximación iterativa e incremental

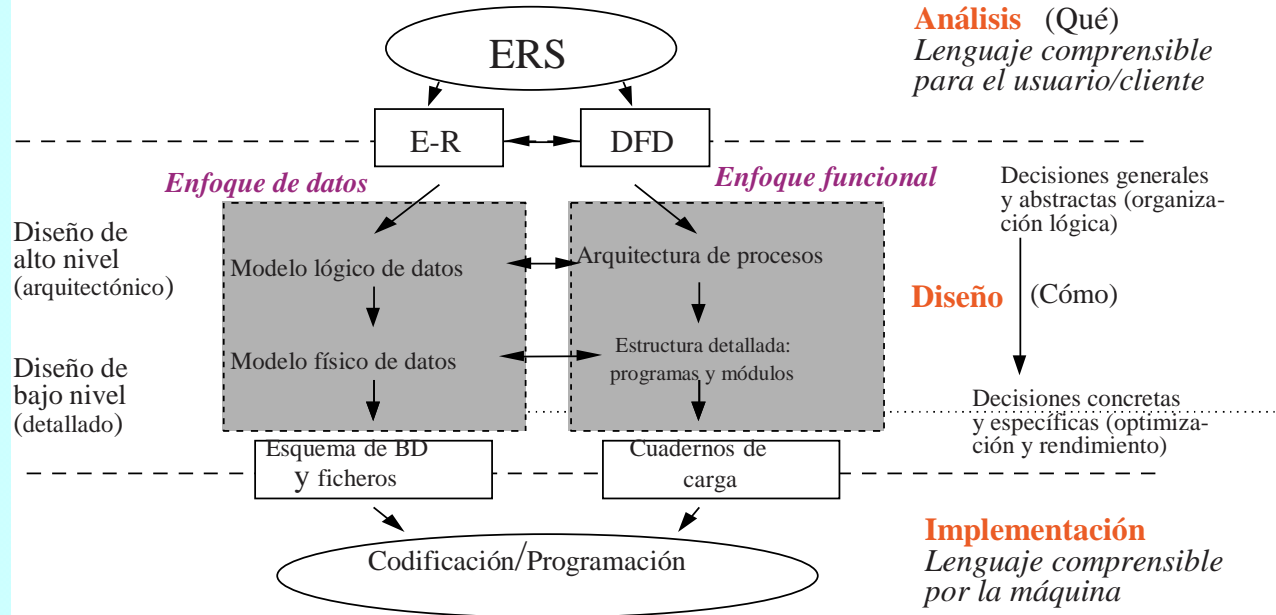


Estrategias y Métodos – Diseño Estructurado

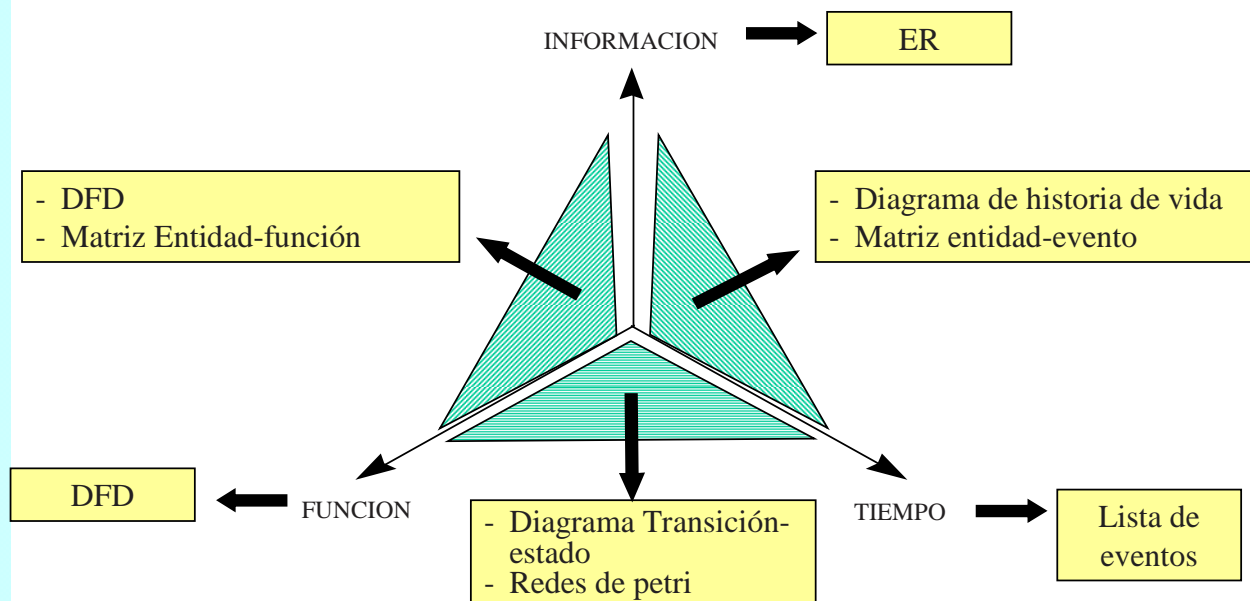
- Método clásico de diseño de software basado en
 - Identificar las funciones principales, y
 - Elaborarlas y refinarlas en un estilo top-down.
- Se realiza después del análisis estructurado, para producir, entre otros:
 - Diagramas de Flujos de Datos (DFDs)
 - Descripciones de los procesos asociados.



Actividades del Diseño Estructurado



Técnicas de Especificación según el enfoque de modelado





Estrategias y Métodos – Diseño Estructurado

- **Diagrama de Flujo de Datos (DFD):** Diagrama en forma de grafo dirigido que representa el flujo de datos y las transformaciones que se aplican sobre ellos al moverse desde la entrada hasta la salida del sistema.

- **Elementos:**

- Procesos
- Almacenes
- Entidades externas
- Flujos de datos

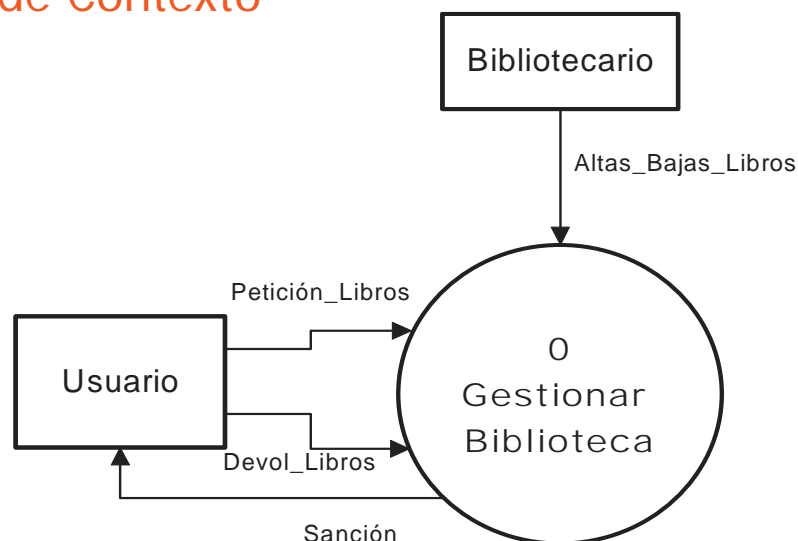
	Yourdon, DeMarco	Gane y Sarson	SSADM MÉTRICA
Flujos de datos			
Procesos			
Almacenes de datos			
Entidades externas			



Estrategias y Métodos – Diseño Estructurado

Ejemplos DFDs

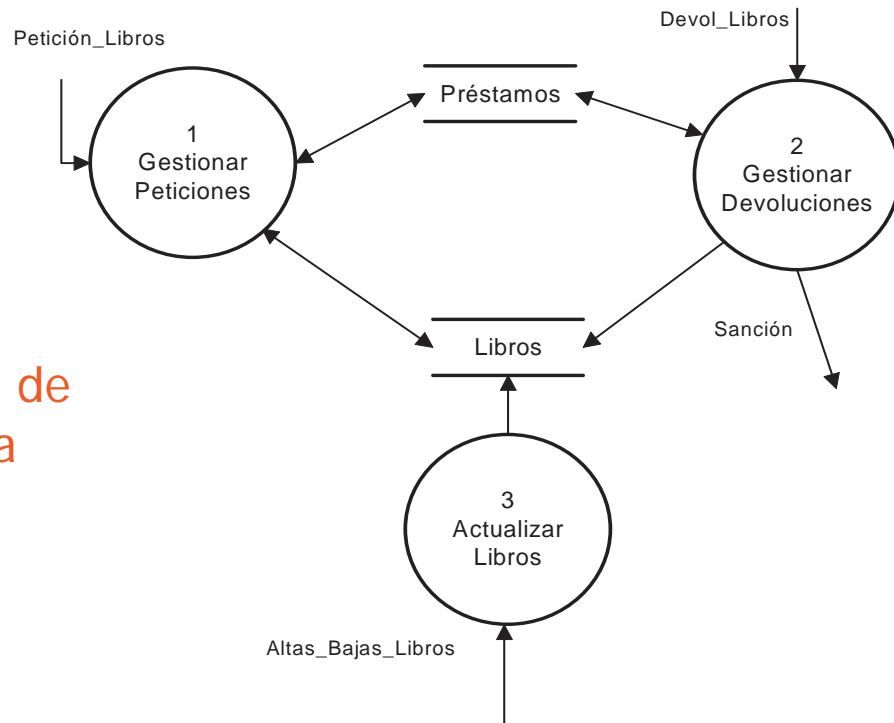
- **Diagrama de Contexto**





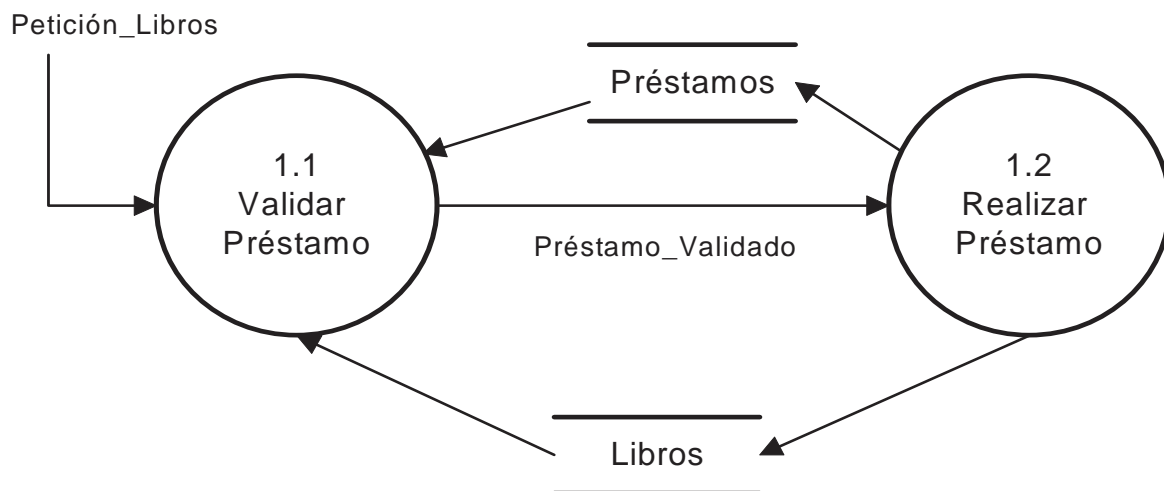
Ejemplos DFDs

- Diagrama de Sistema



Ejemplos DFDs

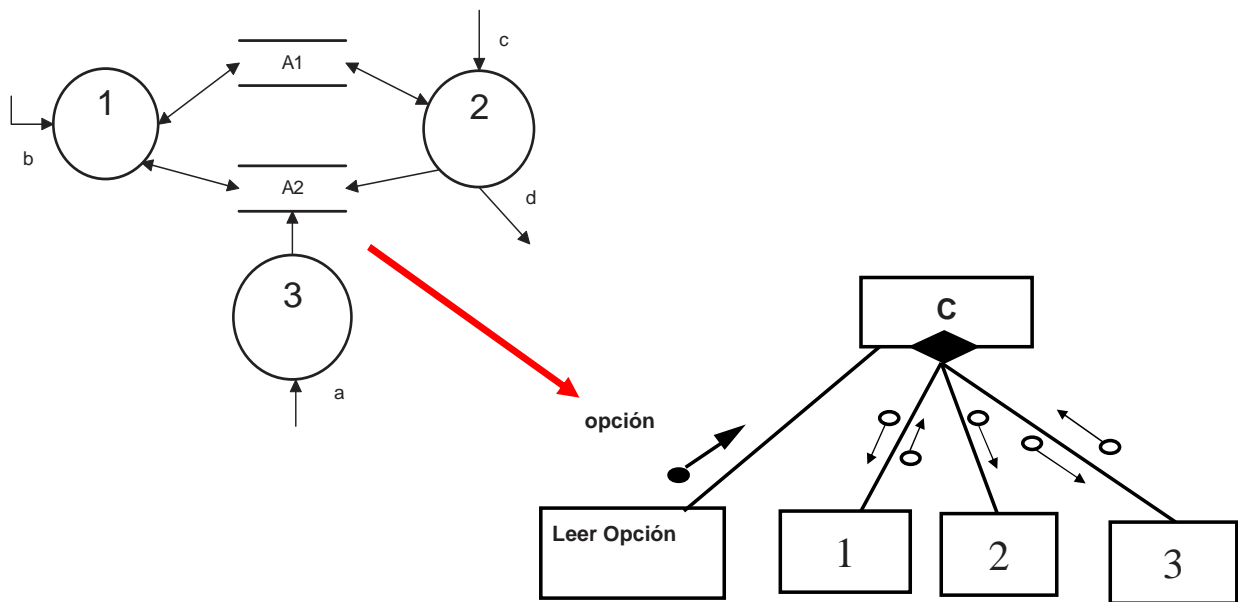
- Diagrama de Sistema detallado





Estrategias y Métodos – Diseño Estructurado

- Habitualmente, a partir de los **DFDs** (comportamiento) se genera un **Diagrama de Estructura** (DE)



Francisco Ruiz - IS1

4.71



Estrategias y Métodos – Diseño Estructurado

- Existen varios tipos de **Diagramas de Estructura** (cambian en la simbología usada):
 - Diagramas de Jackson
 - Diagrama de Cuadros de Constantine (usados en METRICA 3)
- Un DE muestra la **descomposición** de un **sistema** en **módulos** incluyendo
 - Jerarquía de Control → Llamadas entre Módulos
 - Parámetros que se intercambian en las llamadas
- Elementos Principales:
 - Módulos**
 - Conexiones** entre **Módulos**
 - Comunicación** entre **Módulos**

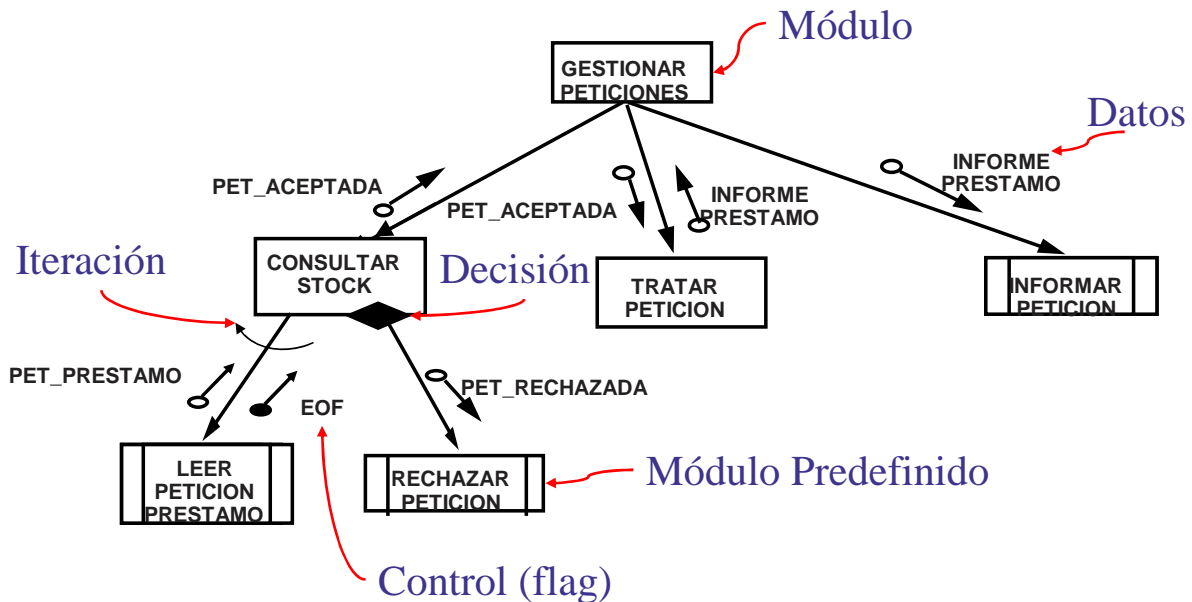
Francisco Ruiz - IS1

4.72



Estrategias y Métodos – Diseño Estructurado

- **Diagrama de Estructura**



Estrategias y Métodos – Diseño OO

- Obviamente el Diseño OO está relacionado con el Análisis y la Programación OO
- **Análisis OO**
 - Desarrollar un modelo de objetos del dominio de aplicación
- **Diseño OO**
 - Desarrollar un modelo del sistema orientado a objetos que satisfaga los requisitos.
- **Programación OO**
 - Implementar un diseño OO usando un lenguaje de programación OO.



- Existen muchos métodos de diseño basados en objetos
 - Desde los iniciales [80's]
 - Nombre->objeto, verbo->método, adjetivo->atributo
 - Centrales [90's]
 - Herencia y polimorfismo juegan un papel clave
 - Diseño basado en Componentes (CBD) [00's]
 - Meta-información puede ser definida y accedida (mediante reflexión)



- En general, en todos los métodos de Diseño OO se llevan a cabo las siguientes **actividades**:
 - Definir el **contexto y modos de uso** del sistema;
 - Diseñar la **arquitectura** del sistema;
 - Identificar los **objetos** principales del sistema;
 - Desarrollar los **modelos** de diseño;
 - Especificar las **interfaces** de los objetos.

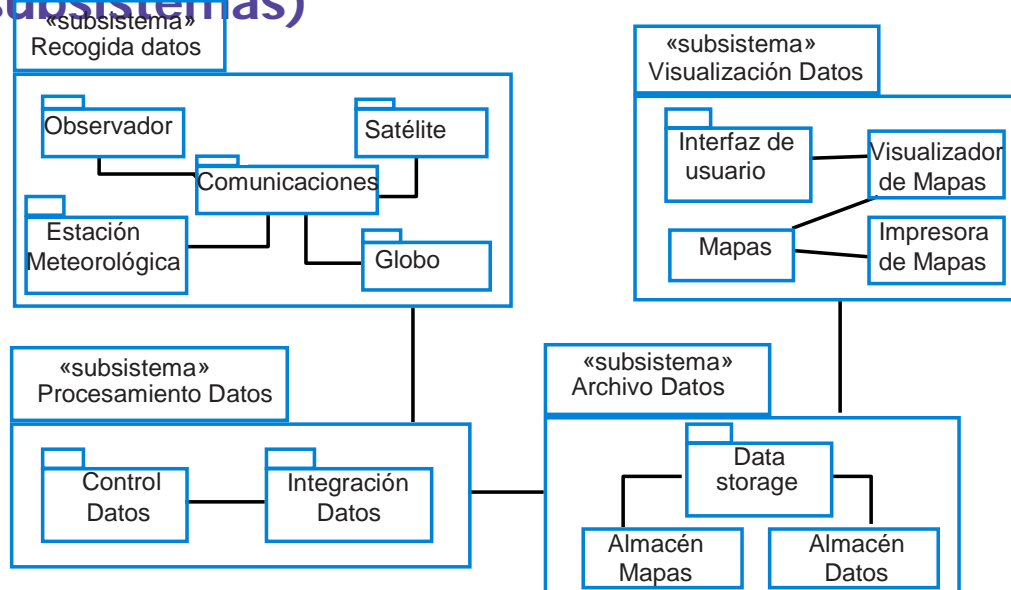


• Ejemplo

- Se necesita un **sistema de mapas climáticos** para generar mapas del tiempo de una forma regular usando los datos recopilados de estaciones meteorológicas remotas automáticas y otras fuentes como observadores, globos y satélites. Las estaciones meteorológicas transmiten sus datos a la computadora del sistema cuando reciben una petición al respecto desde dicha máquina.
- La computadora del sistema valida los datos recopilados y los integra con los datos de otras fuentes diferentes. Los datos integrados se archivan y, usando dichos datos y una base de datos de mapas digitalizados, se elabora un juego de mapas del tiempo locales. Los mapas pueden imprimirse para su distribución en una impresora de mapas especializada o pueden mostrarse en varios formatos diferentes.



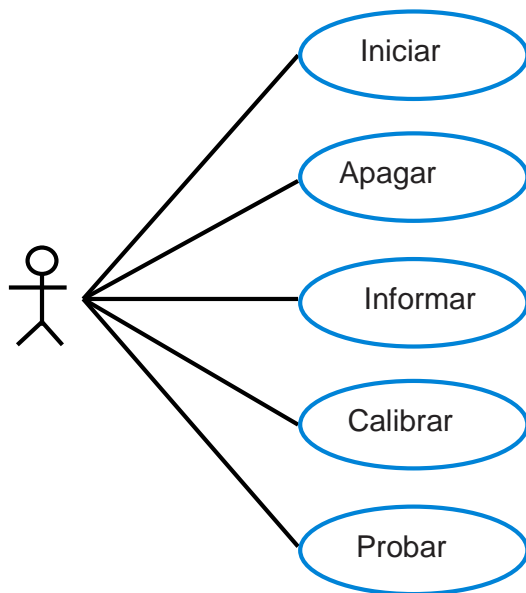
• Ejemplo - Contexto del Sistema (subsistemas)





• Ejemplo – Modelos de Uso

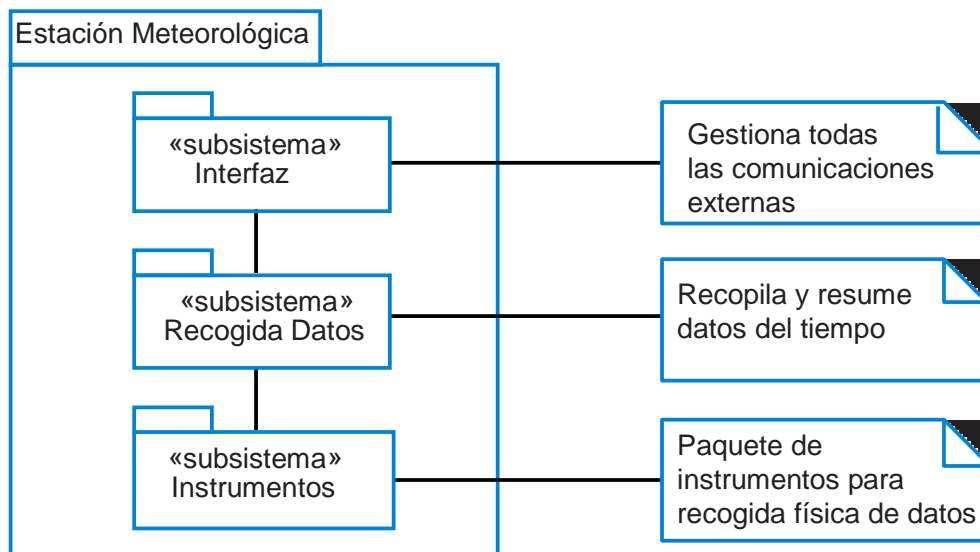
■ Casos de Uso



Sistema	Estación Meteorológica
Caso de Uso	Informar
Actores	Sistema de recolección de datos climáticos, Estación meteorológica
Datos	La estación meteorológica envía un resumen de los datos climáticos que han sido recogidos de los instrumentos en el período de recolección al sistema de recolección de datos climáticos. Los datos enviados son: temperaturas mínima, máxima y media de tierra y aire; presión atmosférica máxima, mínima y media; velocidad del viento máxima, mínima y media; total de lluvia caída; y dirección del viento. Se toman muestras de estos datos cada 5 minutos.
Estímulos	El sistema de recolección de datos climáticos establece una conexión por modem con la estación meteorológica y le requiere la transmisión de datos.
Respuesta	Los datos resumidos (agregados) se envían al sistema de recolección de datos.
Comentarios	Habitualmente las estaciones meteorológicas reciben la petición de informar una vez por hora, pero esta frecuencia puede diferir de una estación a otra y cambiar en el futuro.



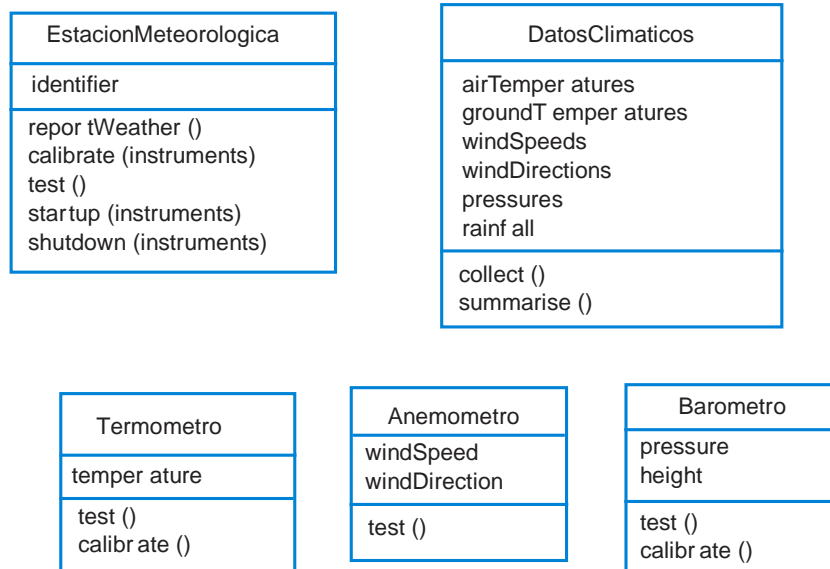
• Ejemplo – Arquitectura por capas



No más de 7 entidades en un modelo arquitectural

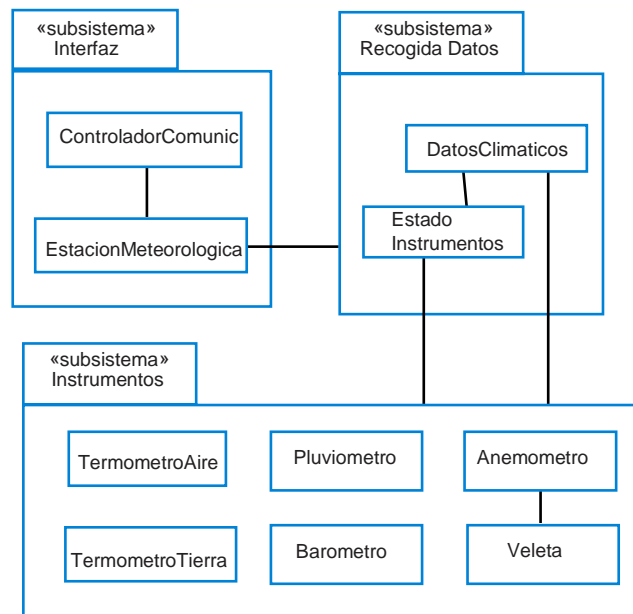


• Ejemplo – Identificar objetos (clases)



• Ejemplo – Modelos de Diseño

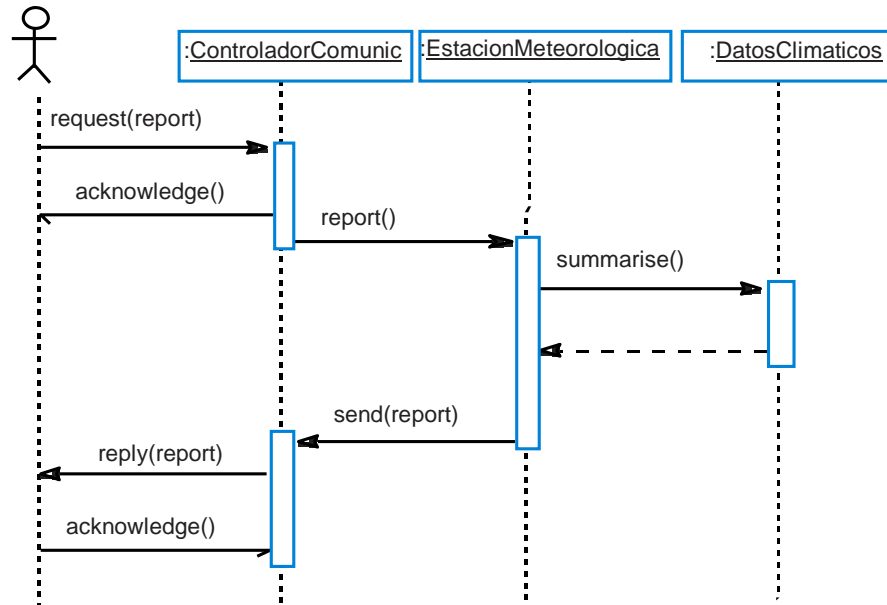
- Estático: Objetos por Subsistemas





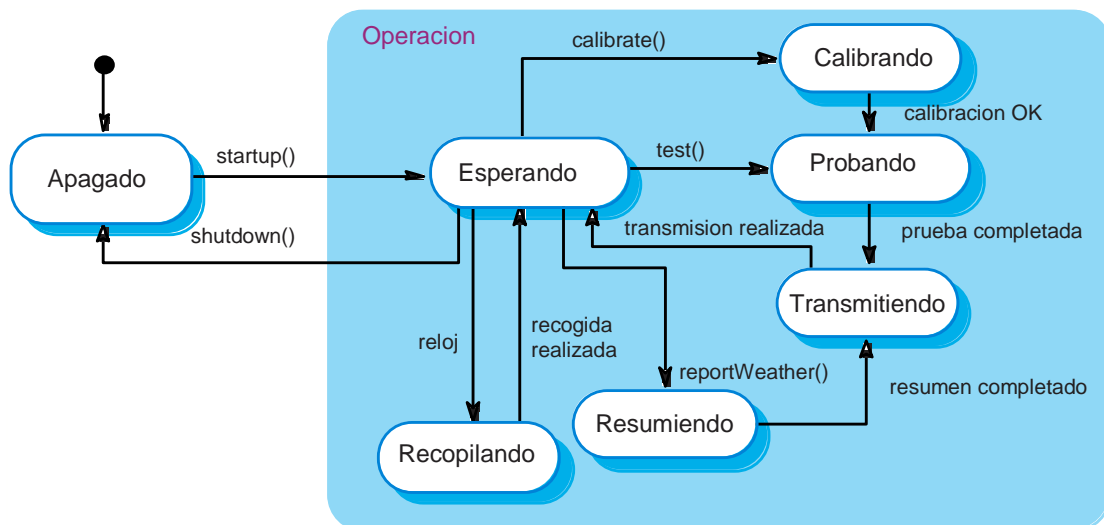
• Ejemplo – Modelos de Diseño

- Dinámico: Secuencia del caso de uso “Informar”



• Ejemplo – Modelos de Diseño

- Dinámico: Máquina de estados de “EstacionMeteorologica”





- **Ejemplo – Interfaz Java de “EstacionMeteorologica”**

```
interface WeatherStation {
    public void WeatherStation () ;
    public void startup () ;
    public void startup (Instrument i) ;
    public void shutdown () ;
    public void shutdown (Instrument i) ;
    public void reportWeather () ;
    public void test () ;
    public void test ( Instrument i ) ;
    public void calibrate ( Instrument i) ;
    public int getID () ;
} //WeatherStation
```



- En estos métodos las **estructuras de datos** guían el diseño.
 - Método de Jackson
 - Método de Warnier-Orr
- Se diferencian de los estructurados en que el foco son las estructuras de datos que manipula un programa y no las funciones que realiza con ellas.
- Hay dos pasos principales:
 - Describir las estructuras de datos de entrada y salida (Diagramas de estructura de Jackson)
 - Desarrollar las estructuras de control basadas en dichas estructuras de datos.



- **Diseño Basado en Componentes (CBD)**
 - Un **Componente Software** es una unidad independiente con interfaces y dependencias bien definidas, que pueda ser desarrollada y desplegada de forma independiente.
 - Principio de Caja Negra.
- CBD aborda problemas para proveer, desarrollar e integrar componentes.
- Su finalidad es mejorar la **reutilización**.