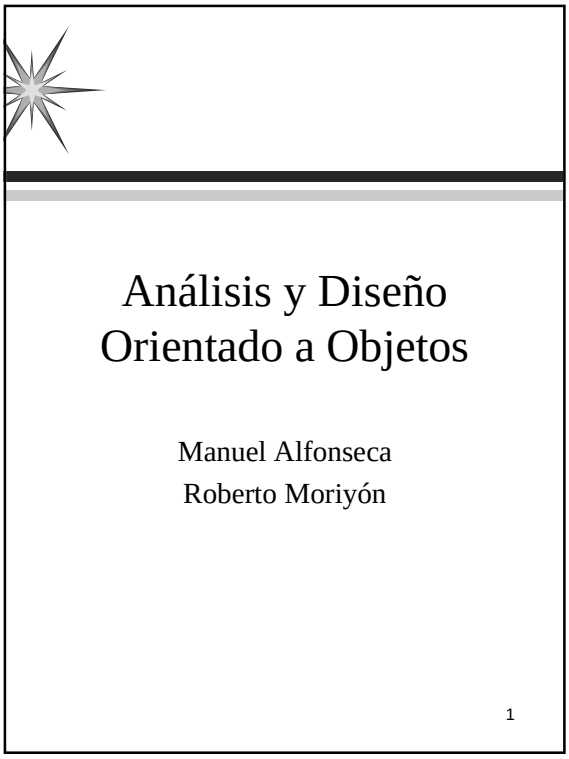
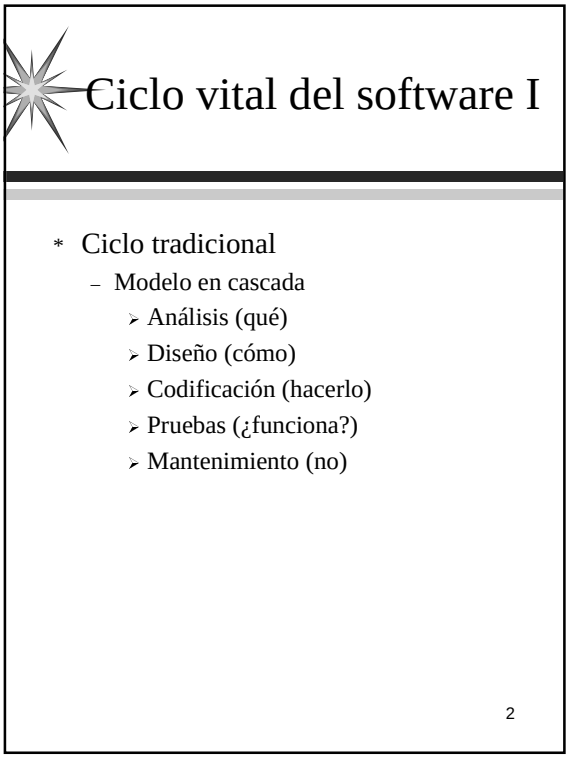


Análisis y Diseño Orientado a Objetos

Manuel Alfonseca
Roberto Moriyón

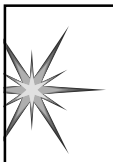


Ciclo vital del software I

* Ciclo tradicional
  - Modelo en cascada
    - Análisis (qué)
    - Diseño (cómo)
    - Codificación (hacerlo)
    - Pruebas (¿funciona?)
    - Mantenimiento (no)

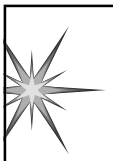
Ciclo vital del software II

- Ciclo evolutivo
  - Modelo en espiral
    - Definición de un subsistema (descomposición funcional)
    - Construcción de un modelo (qué clases se requieren)
    - Ciclo {
      - Análisis de clase(s)
      - Diseño de clase(s)
      - Codificación de clase(s)
      - Prueba de clase(s)
- Diseña un poco, programa un poco, prueba un poco...



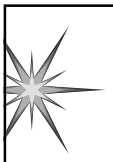
OOD es...

- Modular
- Escaso acoplamiento externo y fuerte cohesión interna
- Efectos laterales mínimos (encapsulamiento)
- Programación por extensión
- Orientado a datos
- Explota la herencia (jerárquico)
- Reutilización de clases
- Fácil de modificar



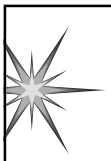
Encapsulamiento I

* Desarrollador
  - Objetivo: crear clase con interfaz clara y comprensible
  - Manera: ocultar detalles de implementación
  - Beneficios: cambio de estructuras/algoritmos sin afectar
  - Coste: clases reutilizables más caras a corto plazo



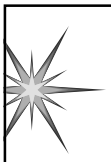
Encapsulamiento II

* Usuario de las clases
  - Objetivo: usar la clase con el mínimo esfuerzo
  - Manera: usar sólo las operaciones provistas
  - Beneficios: interfaz comprensible, bajo coste de programación
  - Coste: pérdida de eficiencia de ejecución



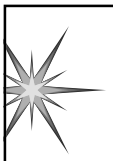
Descomposición funcional

- Módulos construidos alrededor de las operaciones
- Datos globales o distribuidos entre módulos
- Entrada/Proceso/Salida
- Organigramas de trabajo y flujo de datos



OOD

- Módulos construidos alrededor de las clases
- Clases escasamente acopladas, sin datos globales
- Encapsulamiento y mensajes
- Diagramas jerárquicos de clases



Ventajas de OOD

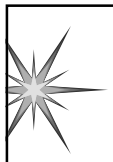
- Módulos con fuerte cohesión interna y escaso acoplamiento externo
- Facilita el funcionamiento en entorno multiprocesador
- Correspondencia directa con el mundo real
- Prototipos rápidos
- Herramientas y bibliotecas muy amplias
- Aplicaciones construidas enganchando objetos
- Mejor comprensión y mantenimiento

Inconvenientes de OOD

- Impactos desfavorables sobre espacio y tiempo de ejecución
- Forma de pensar diferente: curva de aprendizaje lenta
- Herencia y ligadura dinámica dificulta las pruebas

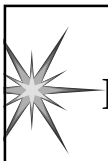
Definición de una clase

- Identificar y nombrar la clase
- Identificar sus componentes
- Identificar sus atributos
- Identificar los errores
- Identificar las conexiones funcionales (qué clases sirve/exige)
- Definir conexiones con superclase y subclases
- Identificar propiedades especiales (persistencia, concurrencia)
- Probar la clase en un prototipo



Identificación de atributos

- El conjunto de atributos de una clase debe ser:
  - Completo (contienen toda la información pertinente)
  - General (se aplican a todos los objetos de la clase)
  - Diferenciado (cada atributo representa un aspecto diferente de la clase)



Definición de atributos

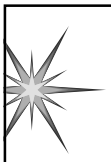
- Tipos de atributos
  - Atómicos predefinidos (entero, real, carácter, pixel...)
  - Atómico enumerativo (color, día de la semana...)
  - Colección
  - Composición (referencias objetos)
- Valor del atributo
  - Común a muchos objetos (variable de clase)
  - Propio de un objeto (variable de objeto)

Identificación de métodos

- Método: algoritmo que utiliza y modifica los atributos de una clase
- Un método es desencadenado por un mensaje
- Funcionalidad de la clase: conjunto de sus métodos
- El conjunto de métodos debe ser:
  - Completo (realizan toda la funcionalidad de la clase)
  - General (se aplican a todos los objetos de la clase)
  - Diferenciado (cada método debe ser simple y realizar una sola función)

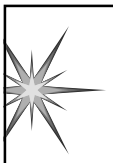
Definición de un método

- Tipos de métodos
  - Modificador (asigna valor a un atributo)
  - Selector (devuelve el valor de un atributo)
  - Aplicable a la clase (constructor)
  - Aplicable al objeto
- Parámetros del método
  - ¿Qué información necesita? (argumentos de entrada)
  - ¿Qué debe devolver? (resultado y argumentos de salida)



Identificación de los errores

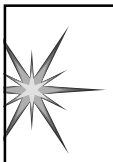
- ¿Qué puede salr mal durante la ejecución de un método?
- ¿Qué comprobaciones debe hacer cada método?
- ¿Cómo interceptar y corregir las condiciones de error?



UML

Unified Modeling Language

Manuel Alfonseca
Roberto Moriyón



Metodologías de Análisis y Diseño (OOA/OOD)

- Booch (OOAD)
- CASEIode (CCM)
- Coad-Yourdon-Nicola (OOA,OOD)
- NE University (Demeter)
- Object Engin. (Fresco)
- Hewlett-Packard (Fusion)
- Graham (SOMA)
- Texas Instruments (IE\O)
- ICL (MTD)
- ParcPlace (OBA)
- Jacobson (OOSE)
- Olivetti (OGROUP)
- Martin-Odell (OOIE)
- TASKON (OORAM)
- Winter (OSMOSYS)
- Rumbaugh (OMT)
- LBMS (SE/OT)
- Shlaer/Mellor (OOSA)
- CCTA (SSADM)
- Wirfs-Brock (RDD)
- Lloyds Register (Z++)

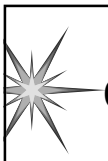
Metodologías de Análisis y Diseño (OOA/OOD)

* Booch (OOAD)
* Rumbaugh (OMT)
* Jacobson (OOSE)
* UML
  - Unión de las tres anteriores
  - Estándar internacional

OOAD (Booch)

Tipos de Relaciones

* Herencia o Generalización → * Metaclase →
* Agregación o Composición —● * Instanciación (templates) ⇢
* Asociación — * Cliente-Servidor (Using) —○



OOAD (Grady Booch)

Tipos de clases

* Clases ordinarias
* Metaclases
* Categorías de clases
* Clases parametrizadas (templates)
* Clases instanciadas (templates)
* Utilidades de clase: subprogramas libres y clases estáticas

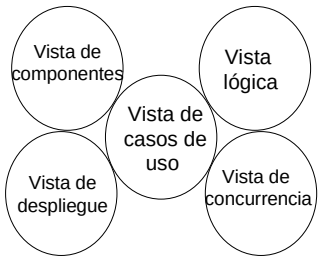
Partes de UML

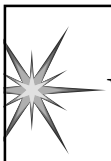
* Vistas
  - Conjunto de diagramas
* Diagramas
  - 9 tipos de grafos
  - Combinan los elementos del modelo
* Elementos del modelo
  - Clases, objetos, mensajes, relaciones
* Mecanismos generales
  - Comentarios, información, semántica, extensiones y adaptaciones

VISTAS

- Vista de Casos de Uso
  - Funcionalidad externa del sistema
- Vista Lógica
  - Estructura estática y conducta dinámica del sistema
- Vista de Componentes
  - Organización de las componentes
- Vista de Concurrencia
  - Comunicaciones y sincronización
- Vista de Despliegue (*deployment*)
  - Arquitectura física

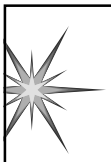
Las Vistas en UML





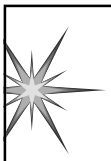
Vista de Casos de Uso

- Dirigida al Análisis de Requisitos
- Describe la funcionalidad del sistema, como la perciben los actores externos
- Dirige el desarrollo de las otras vistas
- Define los objetivos finales del sistema
- Permite validar el sistema
- Actor externo:
  - Usuario
  - Otro sistema
- Se plasma en diagramas
  - de Casos de Uso
  - de Actividad



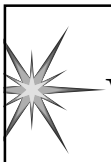
Vista Lógica

- Describe la funcionalidad interna
- Dirigida a diseñadores y desarrolladores
- Define la estructura estática
  - Clases, objetos y relaciones
- Define las colaboraciones dinámicas
  - Mensajes y funciones
- Propiedades adicionales
  - Persistencia y concurrencia
  - Interfaces y estructura interna de las clases



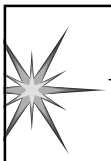
Vista Lógica

- Se plasma en diagramas
  - Estáticos
    - de Clases
    - de Objetos
  - Dinámicos
    - de Estado
    - de Secuencia
    - de Colaboración
    - de Actividad



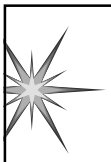
Vista de Componentes

- Describe los módulos del sistema y sus dependencias
- Dirigida a desarrolladores
- Se plasma en diagramas
  - de Componentes



Vista de Concurrencia

- Describe la división del sistema en procesos y procesadores
- Dirigida a desarrolladores e integradores
- Resuelve problemas de
  - uso eficiente de los recursos
  - ejecución en paralelo (hilos)
  - comunicación y sincronización de hilos
- Se plasma en diagramas
  - dinámicos
  - de Componentes
  - de Despliegue



Vista de Despliegue

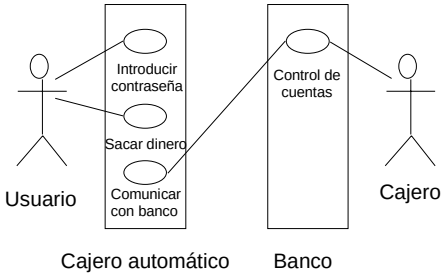
- Muestra la distribución física del sistema (ordenadores, dispositivos) y sus conexiones
- Dirigida a desarrolladores, integradores y probadores
- Se plasma en
  - el diagrama de Despliegue
  - el mapa de asignación de componentes a la arquitectura física

Tipos de Diagramas

- De Casos de Uso
- Estáticos
  - de Clases
  - de Objetos
- Dinámicos
  - de Estado
  - de Secuencia
  - de Colaboración
  - de Actividad
- De Componentes
- De Despliegue (deployment)

Diagramas de Casos de Uso

- Análisis de requisitos
- Describen la funcionalidad externa del sistema
- Pueden escribirse en lenguaje natural
- Se llaman también escenarios



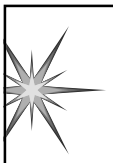
Escenario (secuencia de eventos)

Llamada telefónica

- Origen levanta receptor
- Suena tono de llamada
- Origen marca cifra (9)
- Para tono de llamada
- Origen marca cifra (1)
- Origen marca cifra (3)
- Origen marca cifra (9)
- Origen marca cifra (7)
- Origen marca cifra (4)
- Origen marca cifra (4)
- Origen marca cifra (6)
- Origen marca cifra (7)
- Suena alarma destino
- Suena señal origen
- Destino responde
- Para alarma destino
- Para señal origen
- Conexión
- Destino cuelga
- Origen cuelga

Ejemplo de escenario

•Mientras exploran un castillo, A y B descubren lo que parece ser la entrada de un corredor secreto. Mientras A examina la puerta, B retira una vela del candelabro. En ese momento, la puerta gira 180º, llevándose a A consigo.
•B vuelve a colocar la vela en el candelabro. La puerta gira 360º y A vuelve a quedar separado de B.
•B saca de nuevo la vela. La puerta gira 360º, pero esta vez A le impide cerrarse con su cuerpo. B pasa la vela a A. Los dos amigos fuerzan a la puerta a retroceder 180º y de nuevo quedan separados, pero ahora B está dentro y A junto al candelabro.
•A pone la vela en el candelabro.
•Cuando la puerta empieza a girar, A saca la vela. La puerta se detiene tras girar 90º.
•A y B penetran en el corredor para explorarlo.



ANALISIS: Modelo Dinámico

FASES:

* Preparar escenarios detallados a partir de casos de uso
  - Normales
  - Con problemas
* Identificar sucesos
* Construir diagramas de estados (uno/clase)
* Comprobar consistencia (iterar)
* Añadir métodos

ANALISIS: Modelo de Objetos

Cuarta Fase: Añadir herencia

• Resultado: Primer diagrama de clases

Quinta fase: Comprobar de los casos de uso

• Desde el punto de vista estático
• Iterar

Sexta fase: Modularizar

• Agrupar clases en módulos

Séptima fase: Añadir y simplificar métodos

• Todos los atributos se suponen accesibles
• Métodos para navegar de un objeto a otro

Diagramas estáticos

Elementos:

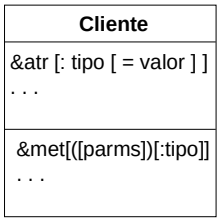
- Objetos
- Clases
- Herencia (Generalización)
- Agregación (Delegación)
- Relaciones Asociativas

Objetivos:

- Diagrama de clases
- Diccionario de datos

Símbolo de una clase

Una clase:



Nombre

Atributos

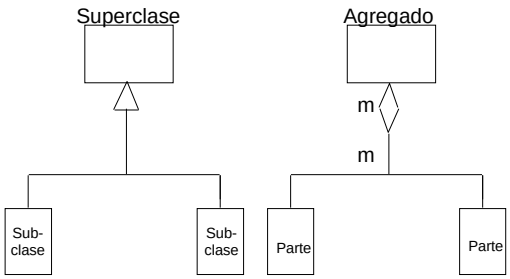
Métodos

&:Visibilidad opcional
+ público (por omisión para métodos)
- privado (por omisión para atributos)
protegido

Tipos de Relaciones

Dependencia:

Herencia (Generalización): Agregación:

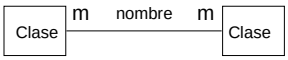


Implementación:

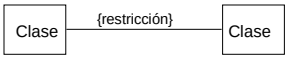
Multiplicidad: 1 1..* 0..1 0..* * 1-3,5 ...

Relaciones de Asociación

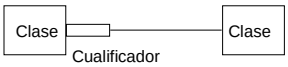
Asociación:



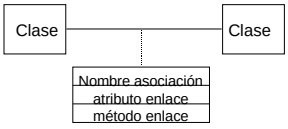
Asociación restringida:



Asociación cualificada:



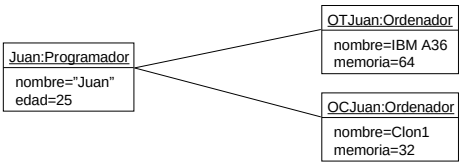
Clase Asociación:



Multiplicidad: 1 1..* 0..1 0..* * 1-3,5 ...

Diagramas de Clases y de Objetos

- Diagrama de clases:
  - Muestra la estructura estática de las clases (combina clases y relaciones)
  - Diagrama de objetos
- Diagrama de objetos:
  - Muestra objetos concretos con la misma notación
  - Nombres de objetos subrayados



Diagramas dinámicos

Elementos:

- Eventos
- Estados de un objeto
- Escenarios
- Traza de eventos
- Concurrencia
- Sincronización
- Acciones

Objetivos:

- Diagramas de estados
- Diagrama de colaboración

Descripción de un estado

Estado: *Suena la alarma*
Descripción: Suena la alarma del reloj para indicar que ha llegado la hora predeterminada
Se llega a él después de la secuencia:
- *set alarm* (objetivo)
- cualquier secuencia que no incluya *clear alarm*
- hora actual = objetivo

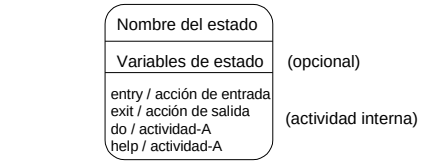
Condición:

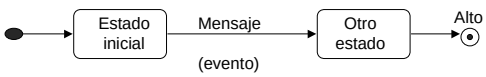
alarm=on && objetivo<=hora actual <= objetivo+20s && no se ha pulsado ningún botón desde objetivo

Acepta:
- *button pushed*(any) pasa al estado *normal*
- hora actual=objetivo+20 pasa a *normal*

Diagramas de Estados

* Autómata finito determinista
* Se aplica a una sola clase o al sistema
* Las entradas son mensajes
* Un mensaje recibido es un evento





Diagramas de Secuencia

- **También llamados trazas de eventos**
- **El tiempo crece verticalmente hacia abajo**

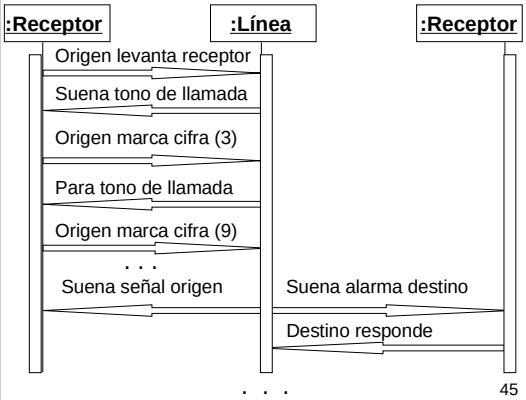
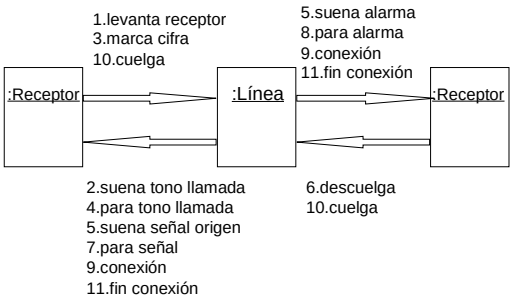


Diagrama de Colaboración

- **Combina un diagrama de objetos y uno de secuencia**
- **Se llama también diagrama de flujo de eventos**



Diagramas de Actividad

* Llamados también Diagramas de Flujo de Actividad
* Estados de Acción (se sale de ellos cuando termina su acción, sin esperar un evento)

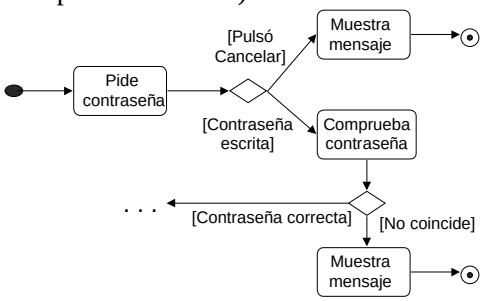


Diagrama de Componentes

* Estructura física del código
* Componentes fuente, objeto o ejecutables
* Pueden formar paquetes

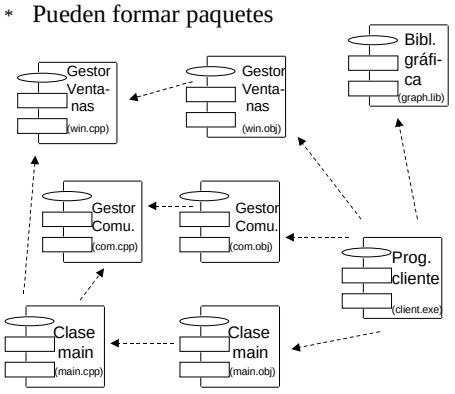
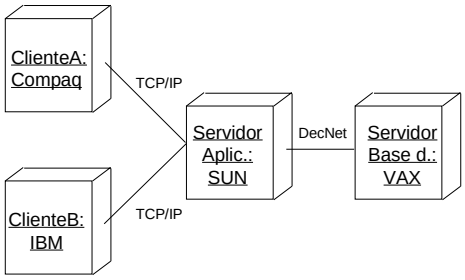
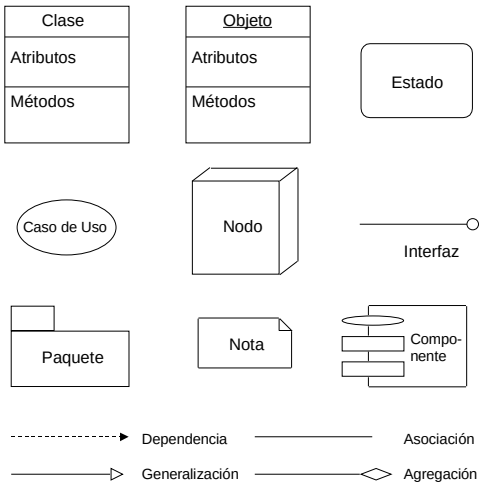


Diagrama de Despliegue

* Muestra la arquitectura física del *hardware* y del *software* del sistema
* Nodos: ordenadores y dispositivos
* Conexiones de diversos tipos



Símbolos



ANALISIS

REQUISITOS

Definidos conjuntamente por el usuario y el analista

- Lenguaje natural
- Casos de uso (I. Jacobson)

De ellos se sacan los tres modelos a nivel de análisis

- Modelo de Objetos
- Modelo Dinámico
- Modelo Funcional

ANALISIS: Modelo de Objetos

FASES:

- Identificar objetos y clases
- Identificar y depurar relaciones
- Identificar atributos de objetos y relaciones
- Añadir herencia
- Comprobar los casos de uso (iterar)
- Modularizar
- Añadir y simplificar métodos

ANALISIS: Modelo de Objetos

Primera Fase: Identificar objetos y clases

Resultado: Diccionario de clases

* Seleccionar nombres de los requisitos
* Añadir clases por conocimiento del tema
* Eliminar redundancias
* Eliminar clases irrelevantes
* Eliminar clases vagas
* Separar atributos
* Separar métodos
* Eliminar objetos de diseño

ANALISIS: Modelo de Objetos

Segunda Fase: Identificar y depurar relaciones

Resultado: Esqueleto del diagrama de clases sin herencia

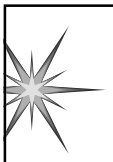
* Seleccionar verbos de relación en requisitos
* Añadir relaciones (conocimiento del tema)
* Eliminar relaciones de diseño
* Eliminar eventos transitorios
* Reducir relaciones ternarias
* Eliminar relaciones redundantes/derivadas
* Añadir relaciones olvidadas
* Definir la multiplicidad de cada relación

ANALISIS: Modelo de Objetos

Tercera Fase: Identificar atributos objetos/relaciones

Resultado: Esqueleto del diagrama de clases con atributos

- Distinguir objetos de atributos
- Distinguir atributos de objeto y de relación
- El identificador de objeto es siempre un atributo implícito
- Eliminar atributos privados (diseño)
- Eliminar atributos de detalle fino
- Localizar atributos discordantes (dividir clase)



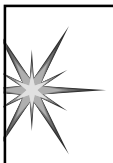
ANALISIS: Modelo Dinámico

Segunda Fase: Identificar sucesos

Resultado: Trazas de eventos

Resultado: Diagramas de flujo de eventos

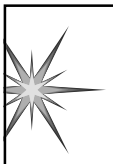
- Señales
- Entradas
- Decisiones
- Interrupciones
- Transiciones
- Acciones externas
- Condiciones de error



ANALISIS: Modelo Dinámico

Quinta Fase: Añadir métodos

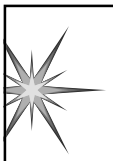
* Eventos como métodos
  (¿de quién?)
* Acciones y actividades en los estados



ANALISIS: Modelo Funcional

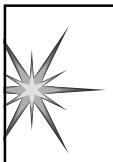
FASES:

* Identificar valores de entrada/salida
* Construir diagramas de flujo de datos
* Describir funciones
* Identificar restricciones y dependencias funcionales entre objetos
* Definir criterios de optimización (iterar)
* Añadir métodos (funciones de diagramas de flujo de datos)



Diseño del Sistema

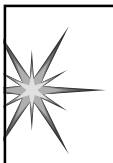
- Definición de subsistemas
- Indentificación de la concurrencia intrínseca y entre tareas
- Asignación de subsistemas a procesadores y tareas (diseño de "hardware")
- Gestión de datos (Bases de datos)
- Gestión de recursos globales
- Selección del control del "software"
- Condiciones límite
- Establecimiento de prioridades



Diseño del sistema

Definición de subsistemas

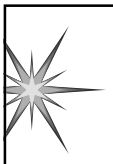
- Capas
- Particiones
- Flujo de información (data flow)



Diseño del sistema

Selección del control de “software”

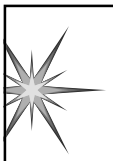
* “Procedure-driven”
* “Event-driven”
* Programación concurrente
* Programación lógica



Diseño del sistema

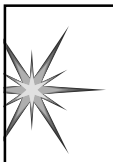
Condiciones límite

* Iniciación
* Terminación
* Fallos



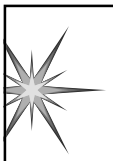
Grandes tipos de sistemas

- “Batch”
- Transformación continua
- Interfaz interactiva
- Simulación dinámica
- Tiempo real
- Gestión de transacciones



Diseño de objetos

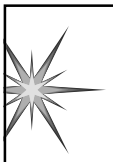
- Combinación de los tres modelos
- Diseño de algoritmos
- Optimización del acceso a los datos
- Implementación del control de “software”
- Ajuste de la herencia
- Diseño de relaciones
- Diseño de atributos
- Modularización



Diseño de objetos

Combinación de los tres modelos

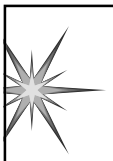
* Asignación de métodos a clases
  - Los procesos del modelo funcional son métodos del de objetos
  - Los eventos del modelo dinámico son métodos del de objetos



Diseño de objetos

Diseño de algoritmos

* Elección del algoritmo óptimo
* Elección de las estructuras de datos
* Definición de clases y métodos auxiliares
* Decidir a quién asignar los métodos dudosos



Diseño de objetos

Optimización del acceso a los datos

* Añadir relaciones redundantes
* Optimizar la eficiencia
* Guardar valores para su uso posterior