Tema 6: Introducción al Diseño

BLOQUE III: DISEÑO DE LOS SISTEMAS SOFTWARE

Ingeniería del Software

Grado en Ingeniería Informática

Curso 2024/2025

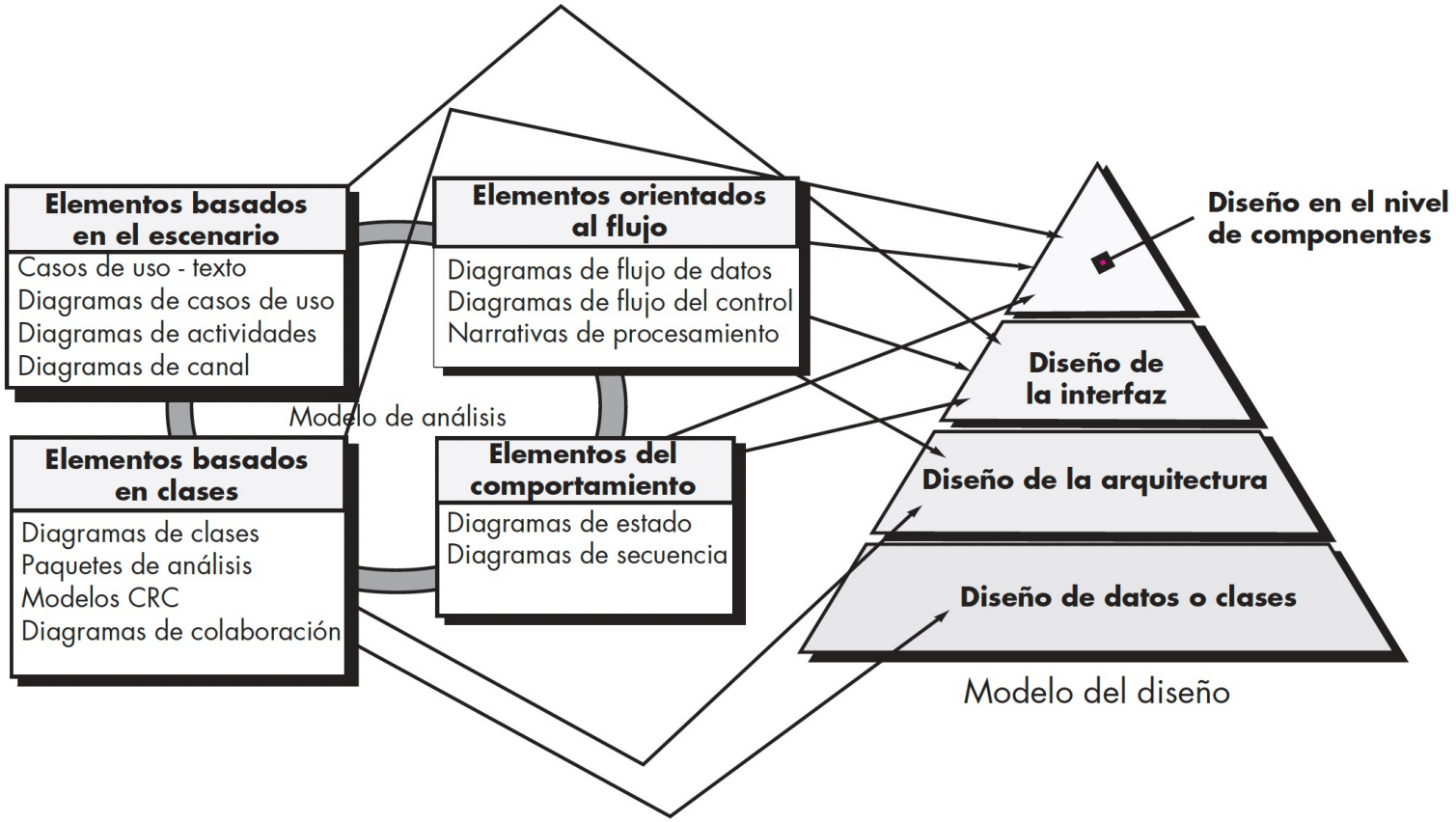
Índice

Índice

Introducción

- El diseño en ingeniería del software se enfoca en crear productos de software de calidad mediante principios y buenas prácticas
- Une “el mundo de los humanos” con “el mundo de la tecnología”
- Depende de la creatividad, experiencia y juicio de los ingenieros
- Buscar la mejor solución para cumplir con los requisitos
- El diseño es el núcleo de la ingeniería del software, independiente del modelo de proceso
- Tras análisis de requisitos se organiza la información en 4 niveles clave
- Antes de la construcción se establece las bases para el desarrollo exitoso (código y pruebas)

Introducción



Índice

Conceptos de Diseño

- Los siguientes conceptos son de aplicación general, con independencia del tipo de producto o paradigma de desarrollo:
  1. Abstracción
  2. Arquitectura
  3. Patrones
  4. Separación de intereses (*separation of concerns*)
  5. Modularidad
  6. Ocultación de la información
  7. Independencia funcional
  8. Refinamiento
  9. Refactorización

Conceptos de Diseño

1. Abstracción
   - Resolver problemas en diferentes niveles de detalle
   - Combina la terminología del problema con la técnica
   - A nivel de datos y de procesos

2. Arquitectura
   - Define la estructura del software y sus interrelaciones
   - Utiliza patrones arquitectónicos para resolver problemas recurrentes
   - Emplea modelos y lenguajes específicos (ADL)

3. Patrones
   - Soluciones reutilizables a problemas comunes de diseño
   - Adaptadas a contextos específicos

Conceptos de Diseño

4. Separación de intereses (*separation of concerns*)
   - Divide un problema en partes independientes
   - Facilita la resolución de problemas
   - Reduce la complejidad durante la integración

5. Modularidad
   - Facilita el manejo intelectual el software
   - Evita diseños monolíticos (entramado de control, datos y procesos) que hacen que el software sea difícil de comprender
   - Reduce la complejidad durante la integración

6. Ocultación de la información
   - Diseña módulos con detalles internos privados
   - Limita el acceso a datos y algoritmos dentro de cada módulo

Conceptos de Diseño

7. Independencia funcional
   - Cada módulo realiza tareas específicas
   - Ofrece interfaces simples para facilitar su uso
   - Mejora la reusabilidad y facilita las pruebas

8. Refinamiento
   - Proceso iterativo de diseño de arriba hacia abajo
   - Descompone funciones abstractas en elementos concretos
   - Revela detalles progresivamente

9. Refactorización
   - Reorganiza el código sin cambiar su comportamiento
   - Mejora la compresión del código
   - Reduce los costos de modificación

Conceptos de Diseño

- La cohesión evalúa qué tan relacionadas están responsabilidades y las funciones dentro de un módulo
- El acoplamiento mide la interdependencia entre diferentes módulos del sistema

Alta cohesión + Bajo acoplamiento = Software fácil de mantener y escalar

Conceptos de Diseño

Cohesión

- Alta cohesión: Funciones relacionadas y enfocadas en una tarea
  - Fácil de mantener y reutilizar
  - Ejemplo: Clase ‘Factura’ que gestiona operaciones como calcular total, aplicar descuentos, generar recibo

- Baja cohesión: Funciones dispersas y sin relación clara
  - Difícil de entender y mantener
  - Ejemplo: Clase ‘Utilidades’ con funciones diversas como validar correos y generar reportes

Conceptos de Diseño

Acoplamiento

- Bajo acoplamiento: Mínima interdependencia entre módulos
  - Cambios en un módulo no afectan a otros
  - Ejemplo: Módulo ‘ProcesamientoDePago’ que interactúa con ‘Notificaciones’ mediante interfaces

- Alto acoplamiento: Módulos muy interconectados
  - Cambios con mucho riesgo y difícil mantenimiento
  - Ejemplo: Módulo ‘Inventario’ que modifica datos directamente en ‘Clientes’

Conceptos de Diseño

Relación entre Cohesión y Acoplamiento

Niveles de cohesión	Niveles de acoplamiento
Coincidental	**Sin acoplamiento directo**
Lógica	De datos
Temporal	Por estampado
Procedimental	De control
Comunicativa	Externo
Secuencial	Común
Funcional	Del contenido

Conceptos de Diseño

Relación entre Cohesión y Acoplamiento

- Lo ideal es:
  - Alta cohesión: Módulos enfocados
  - Bajo acoplamiento: Módulos independientes
- Pero podemos encontrarnos con varios problemas comunes:
  - Alta cohesión + Alto acoplamiento: Bien definidos pero muy interdependientes
  - Baja cohesión + Bajo acoplamiento: Independientes pero con funciones dispersas

Ejemplo: Sistema de Gestión de Pedidos

- Alta cohesión: Clase ‘Pedido’ maneja funciones específicas como agregar productos y confirmar el pedido
- Bajo acoplamiento: ‘Pedido’ interactúa con ‘Notificaciones’ mediante una interfaz, permitiendo cambios sin afectar el sistema.

Índice

Principios Básicos de Diseño

- DRY (Don't Repeat Yourself): Evitar duplicación de código
- YAGNI (You Ain't Gonna Need It): No hacerlo hasta que sea necesario
- KISS (Keep It Simple, Stupid): Mantenerlo simple
- SOLID: Conjunto de principios para un diseño de clases limpio y mantenible
    1. Responsabilidad Única (**S**RP): Cada clase debe tener una única responsabilidad
    2. Abierto-Cerrado (**O**CP): Las clases deben ser extendibles, no modificables
    3. Sustitución de Liskov (**L**SP): Las clases derivadas deben sustituir a sus clases base sin problemas
    4. Segregación de Interfaces (**I**SP): No obligues a una clase a implementar interfaces que no usa
    5. Inversión de Dependencias (**D**IP): Las clases de alto nivel deben depender de abstracciones, no de clases de bajo nivel

Índice

Niveles de Diseño

Diseño de Datos

- Objetivo: Organización y acceso eficiente a los datos
- Elementos clave:
  - Estructuras de datos (tablas, relaciones, claves)
  - Normalización (eliminación de redundancias)
  - Selección de estructuras eficientes (arrays, listas, árboles)
- Ejemplo: Sistema de gestión de biblioteca
  - Tablas: “Libros”, “Usuarios”, “Préstamos”
  - Relación: Préstamo asociado a usuario y libro

Niveles de Diseño

Diseño Arquitectónico

- Objetivo: Definir la estructura general del sistema
- Elementos clave:
  - Identificación de módulos principales (interfaz, lógica de negocio, datos)
  - Patrones arquitectónicos (MVC, microservicios, cliente-servidor)
  - Diagramas arquitectónicos (representación gráfica)
- Ejemplo: Sistema de comercio electrónico
  - Módulos: “Gestión de Productos”, “Carrito”, “Gestión de Pedidos”
  - Patrón: Arquitectura en capas

Niveles de Diseño

Diseño de Interfaz

- Objetivo: Definir cómo interactúan los módulos y usuarios
- Elementos clave:
  - Diseño de interfaces de usuario (UI)
  - Interfaces de programación (API)
  - Interacción con sistemas externos (REST, SOAP)
- Ejemplo: Sistema bancario
  - UI: Panel de usuario para consultar saldo
  - API: Acceso a datos en cajeros automáticos

Niveles de Diseño

Diseño Procedimental o de Funciones

- Objetivo: Definir algoritmos y procesos internos
- Elementos clave:
  - Diseño de algoritmos (pasos específicos)
  - Diagramas de flujo (representación lógica)
  - Pseudocódigo (descripción textual)
- Ejemplo: Cálculo del precio final de un pedido
  - Sumar precios, aplicar descuentos y añadir impuestos

Índice

Patrones de Diseño

- Los patrones de diseño son soluciones reutilizables para problemas comunes en el desarrollo de software.
- Características:
  - Efectividad: Resuelven problemas similares en diversas ocasiones
  - Reutilizables: Aplicables a diferentes problemas de diseño
- Un patrón es una plantilla, no una solución directa

- Categorías de Patrones de Diseño:
  1. Patrones Creacionales: Abstracción del proceso de creación de objetos, hace el sistema independiente de cómo se crean y componen los objetos.
  2. Patrones Estructurales: Se enfocan en cómo combinar clases y objetos para formar estructuras más grandes
  3. Patrones de Comportamiento: Están relacionados con algoritmos y asignación de responsabilidades, además, definen patrones de comunicación entre objetos.

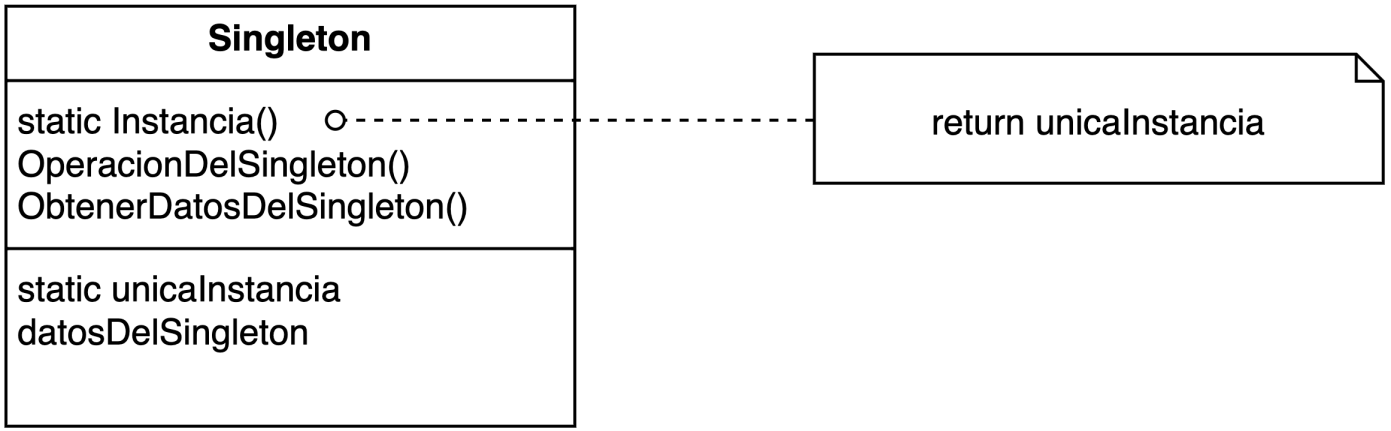
Patrones de Diseño

Creacionales	Estructurales	Comportamiento
Factory Method	Adapter	Chain of Responsability
Abstract Factory	Bridge	Command
Builder	Composite	Interpreter
Prototype	Decorator	Iterator
Singleton	Facade	Mediator
Model View Controller (MVC)	Flyweight	Memento
	Proxy	Observer
	Module	State
		Strategy
		Template Method
		Visitor

Índice

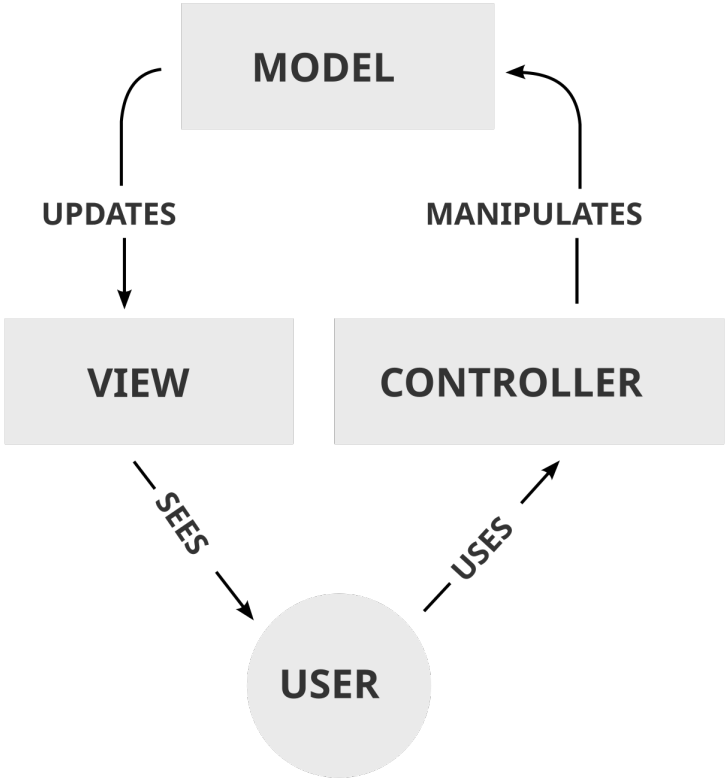
Patrón *Singleton*

- Garantiza que solo haya una instancia de una clase, controlando la creación de instancias, y proporciona un acceso global a ella
- Ejemplo:
  - Solo debe haber una cola de impresión en un sistema, aunque haya varias impresoras
  - La configuración del programa debe ser única, accesible por todos los componentes



Patrón Modelo-Vista-Controlador (MVC)

- Divide una aplicación en tres componentes:
  - Modelo: Gestiona los datos y la lógica de negocio
  - Vista: Presenta los datos de forma interactiva
  - Controlador: Intermedia entre el modelo y la vista, manejando eventos del usuario



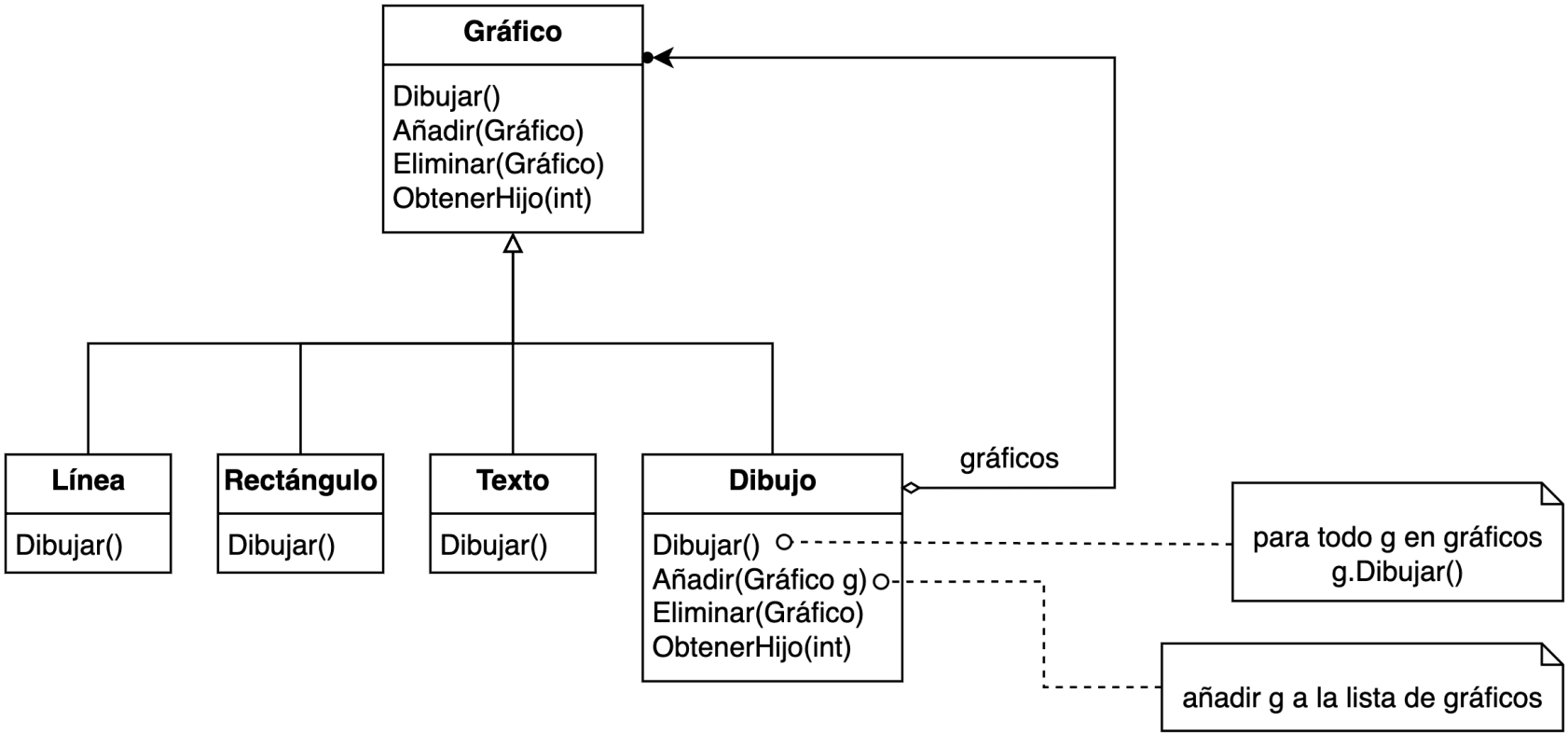
- Facilita la reutilización de código y el mantenimiento, separando la lógica de negocio de la interfaz de usuario.
- Ideal para aplicaciones con interfaces interactivas que requieren cambios frecuentes.

Índice

Patrón Compuesto (*Composite*)

- Permite tratar objetos individuales y composiciones de objetos de manera uniforme
- Organiza los objetos en una jerarquía en forma de árbol, donde los nodos pueden ser objetos simples o compuestos
- Ejemplo:
  - En un editor gráfico, los elementos simples como líneas y rectángulos se combinan para formar composiciones más complejas, como cuadros o diagramas
  - El patrón resuelve el problema de manejar estos elementos de manera uniforme mediante una interfaz común

Patrón Compuesto (*Composite*)



Índice

Patrón Iterador

- Permite acceder secuencialmente a los elementos de un objeto sin exponer su estructura interna
- Ejemplo:
  - En una lista, el patrón iterador permite recorrer sus elementos de diferentes formas sin modificar la estructura interna
  - El iterador es responsable de saber qué elementos se han recorrido y cuál es el actual

Patrón Observador

- Establece una relación de dependencia uno a muchos entre objetos. Cuando un objeto cambia de estado, todos los objetos dependientes se actualizan automáticamente
- Ejemplo:
  - En una hoja de cálculo, los gráficos y tablas se actualizan automáticamente cuando cambian los datos, sin que los gráficos dependan directamente de la hoja
  - Los objetos observadores monitorean los cambios y reflejan las actualizaciones en tiempo real