# Introducción a las Arquitecturas de los Sistemas de Información Empresarial

Pablo Sánchez

Dpto. Ingeniería Informática y Electrónica
Universidad de Cantabria
Santander (Cantabria, España)
p.sanchez@unican.es


# Advertencia

Todo el material contenido en este documento no constituye en modo alguno una obra de referencia o apuntes oficiales mediante el cual se puedan preparar las pruebas evaluables necesarias para superar la asignatura.

Este documento contiene exclusivamente una serie de diapositivas cuyo objetivo es servir de complemento visual a las actividades realizadas en el aula para la transmisión del contenido sobre el cual versarán las mencionadas pruebas evaluables.

Dicho de forma más clara, estas transparencias no son apuntes y su objetivo no es servir para que el alumno pueda preparar la asignatura.

# Índice

# Objetivos del Tema

1. Comprender qué es un *Sistema de Información Empresarial (SIE)*.
2. Comprender el concepto de *Arquitectura Software*.
3. Conocer los principales patrones arquitectónicos que aplican a los SIEs.
4. Comprender cómo se divide un SIE en capas.
5. Comprender la función de cada capa de un SIE.
6. Conocer las tecnologías que se utilizan para implementar las capas de un SIE.
7. Comprender cómo se pueden independizar módulos sw mediante *inyección de dependencias*.

# Bibliografía


Fowler, M. (2002).
*Patterns of Enterprise Application Architecture*.
Addison-Wesley Professional.

Esposito, D. and Saltarello, A. (2014).
*Microsoft .NET - Architecting Applications for the Enterprise*.
2 edition.

Taylor, R. N., Medvidovic, N., and Dashofy, E. (2009).
*Software Architecture: Foundations, Theory, and Practice*.
Wiley.

# Índice

# Sistema de Información Empresarial

## Sistema de Información Empresarial (SIE)

Un *Sistema de Información Empresarial* es un sistema sw que da soporte a diferentes procesos de negocio de una determinada organización.

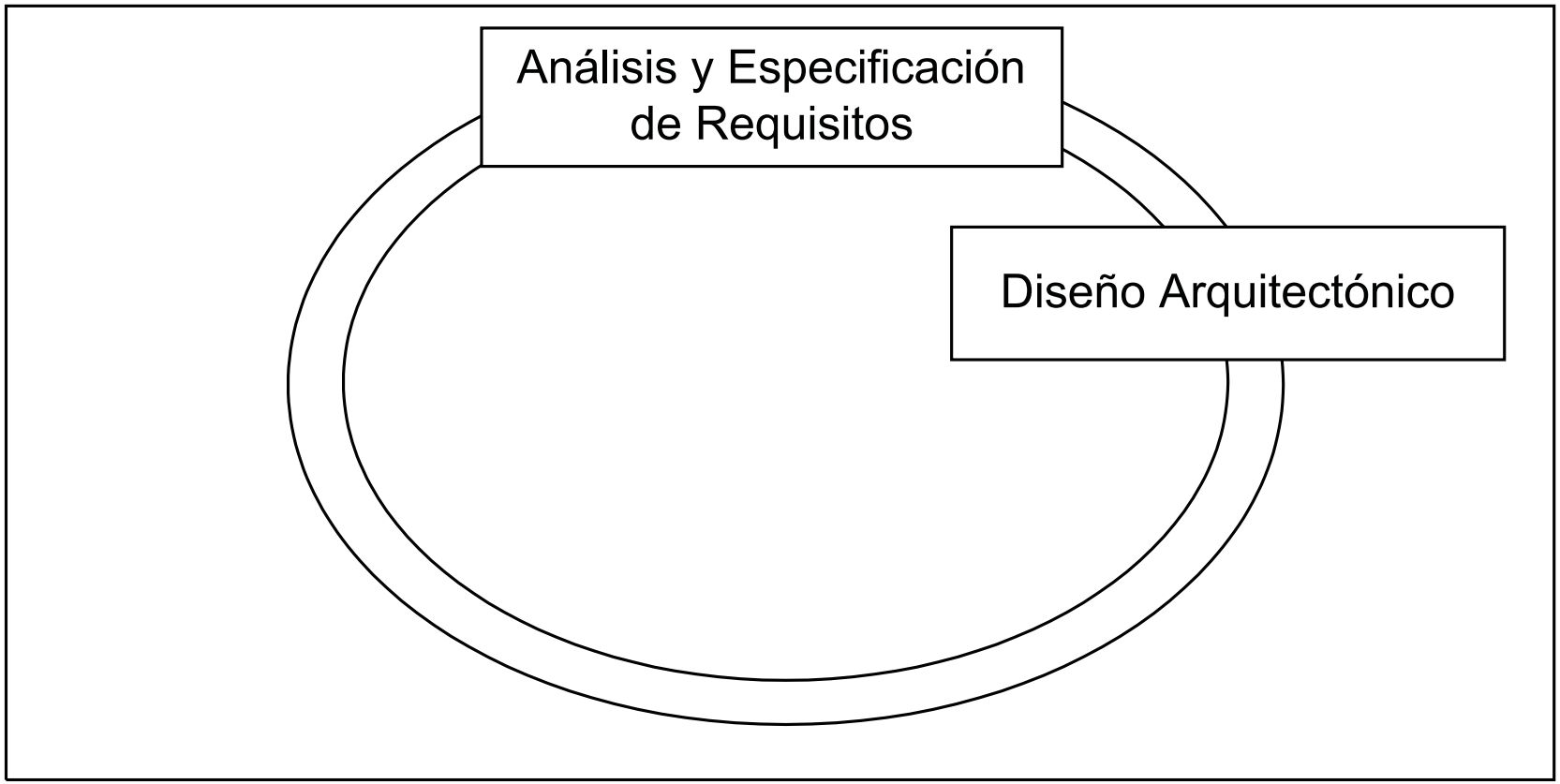
# Características de los SIEs

1. Necesita almacenar datos, y normalmente, en gran volumen.
2. Los datos que almacenan representan un activo importante y duradero en el tiempo.
3. Los gestión de los datos debe obedecer a ciertas *reglas de negocio*.
4. Las operaciones ejecutadas necesitan ser *transaccionales*.
5. Los datos pueden ser accedidos y manipulados de manera concurrente.
6. Utilizan un gran número de interfaces de usuario que pueden requerir de sistemas avanzados de visualización de datos.
7. Interoperan con otros Sistemas de Información Empresarial.

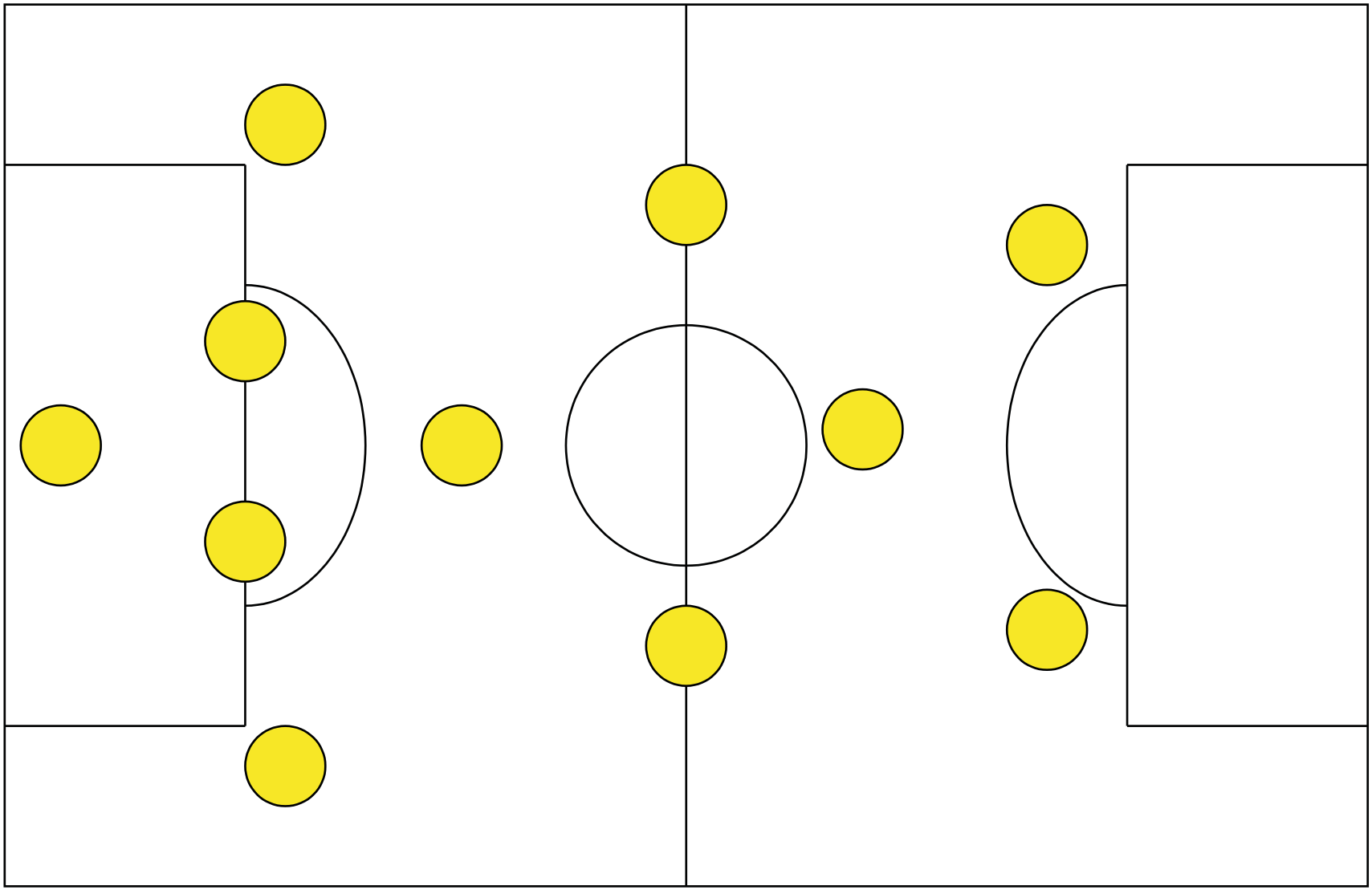
# Índice

# Índice

# Fases del Ciclo de Vida Sw



# ¿Por Qué Necesitamos Arquitecturas Sw?



# Índice

# Definiciones Arquitectura Sw

## Arquitectura Software [Taylor et al., 2009, Pohl et al., 2005]

La arquitectura de un sistema sw es el conjunto de las principales decisiones de diseño realizadas sobre el sistema. Toda arquitectura sw tiene una *estructura* y una *textura*.

## Textura Arquitectónica [Jazayeri et al., 2000, Pohl et al., 2005]

La *textura* de un arquitectura sw es el conjunto de reglas que gobiernan el desarrollo y diseño de un sistema software.

## Estructura Arquitectónica [Jazayeri et al., 2000, Pohl et al., 2005]

La *estructura* de una arquitectura sw es la descomposición de un sistema sw en sus partes principales y las relaciones entre dichas partes.

# Índice

# Patrones Arquitectónicos

## Patrón Arquitectónico

Un *patrón arquitectónico* es un conjunto con nombre de decisiones de diseño reutilizables Y aplicables a problemas de diseño arquitectónico recurrentes, pero que necesitan ser adaptadas a cada sistema concreto.

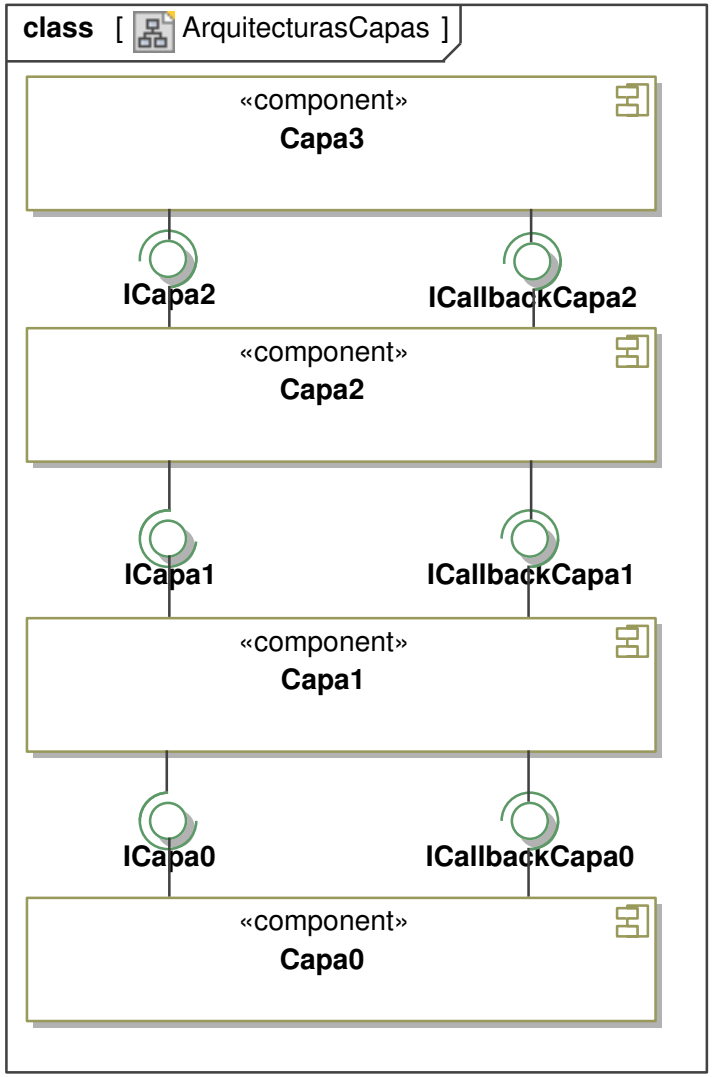
# Índice

# Arquitecturas en Capas - Principios Básicos

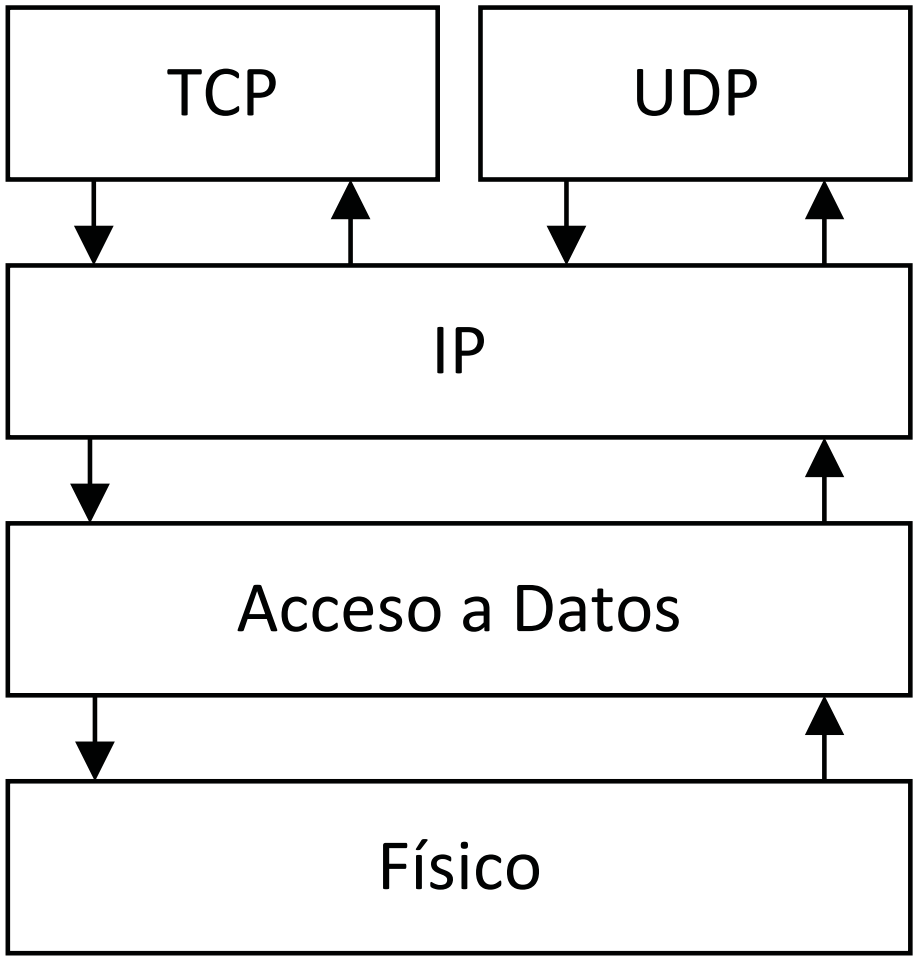
## Arquitecturas en Capas

- Un sistema se descompone en un conjunto de capas.
- Las capas se diseñan de abajo a arriba.
- Cada capa manipula un conjunto bien definido de elementos, ofreciendo un conjunto potente y estable de abstracciones para la manipulación de dichos elementos a las capas superiores.

# Arquitecturas en Capas



# Arquitecturas en Capas



# Arquitecturas en Capas - Beneficios

1. Permite controlar y gestionar la complejidad de un sistema software.
2. Independencia y capacidad de reemplazo de las capas.

# Arquitecturas en Capas - Problemas

1. (Rendimiento).
2. Esfuerzo requerido para el diseño.
3. Elementos no fácilmente aislables en capas.

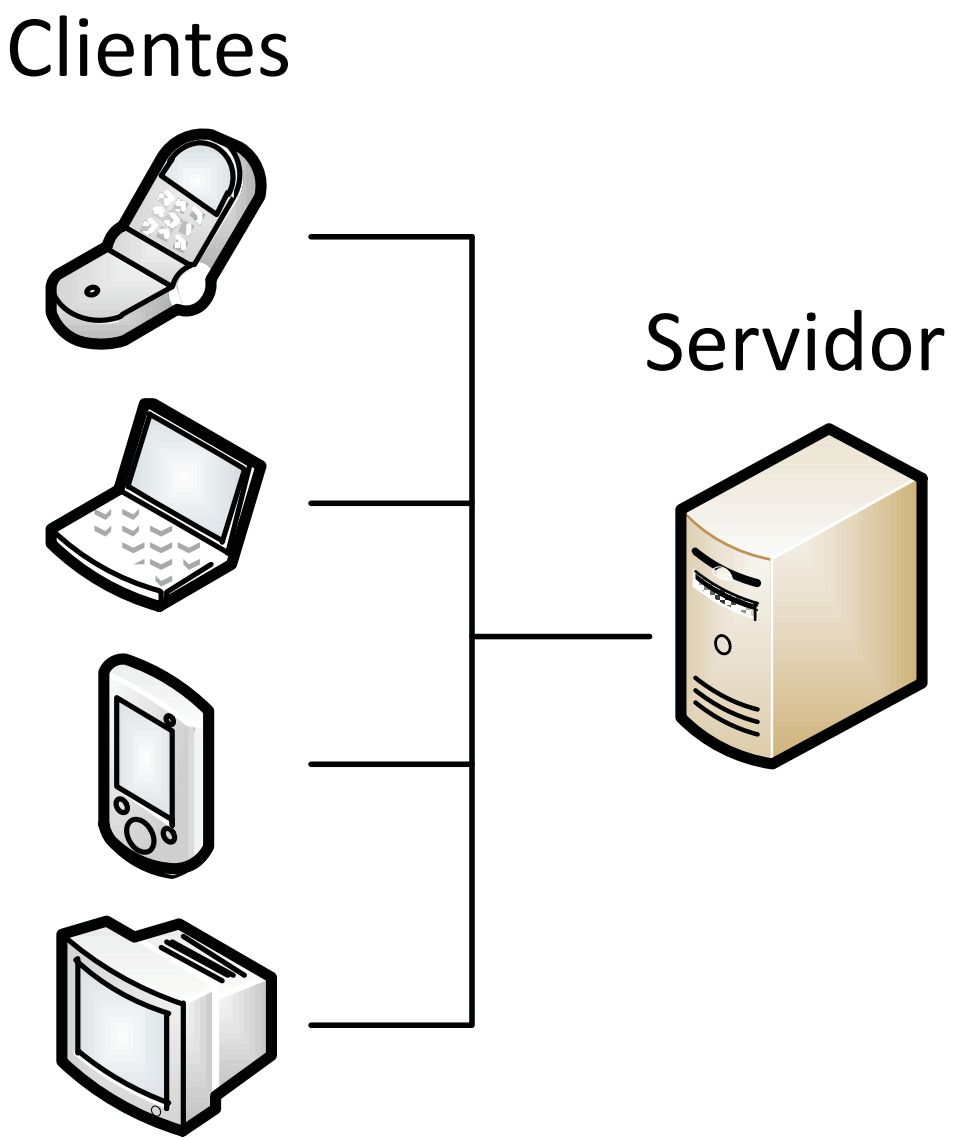
# Índice

# Arquitecturas Cliente/Servidor - Principio Básico

## Arquitecturas Cliente/Servidor

- Existe un nodo especializado denominado *servidor*.
- El servidor centraliza ciertos cómputos del sistema.
- Los clientes realizan peticiones al servidor, el cual las procesa y devuelve las respuesta a los clientes.

# Arquitecturas Cliente/Servidor



# Arquitecturas Client/-Servidor - Beneficios

1. Ubicuidad del sistema.
2. Facilidad de mantenimiento y actualización.
3. Eliminación de redundancias y posibles inconsistencias.
4. Permite aligerar los necesidades hardware de los clientes.
5. Permite trabajar con clientes heterogéneos.

# Arquitecturas Cliente-Servidor - Problemas

1. Efecto estrella de la muerte.
2. Necesidad de tener conexión con el servidor.
3. Problemas de escalabilidad y rendimiento del servidor.
4. Saturación de las redes de comunicación.

# Índice

# Código Móvil - Principio Básico

## Arquitecturas con Código Móvil

Un computador (genera y) almacena código y/o datos que se envían a otros computadores para su ejecución.

# Arquitecturas con Código Móvil - Beneficios

1. Permite distribuir y equilibrar la carga de trabajo.
2. Utilización y composición de recursos bajo demanda.
3. Permite aumentar la disponibilidad y tolerancia a fallos del sistema.
4. Facilitar el mantenimiento y la evolución del sistema.
5. Ubicuidad de ciertas funciones del sistema.

# Arquitecturas con Código Móvil - Problemas

1. Agujeros de seguridad.
2. Saturación de las redes de comunicación.

# Índice

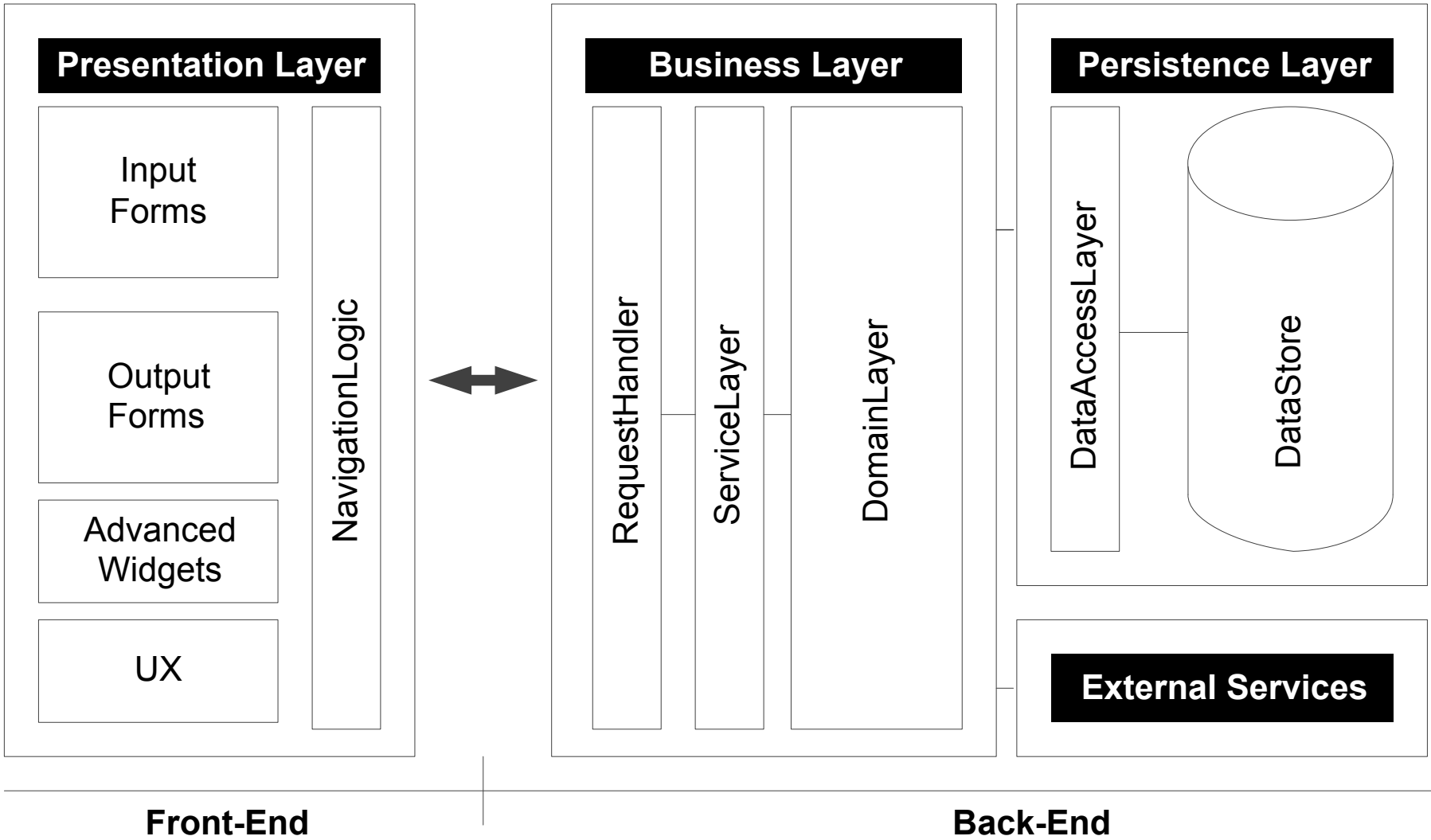
# Índice

# Sistema Monocapa

```
<% SQLtxt = "SELECT Producto, Cantidad, Precio
                FROM articulos"
   set rs = CreateObject("ADODB.Recordset")
   rs.Open SQLtxt,"DSN=Mibase" %>
<table>
  <% Do While NOT rs.EOF%>
      <tr>
        <td><%= rs("Producto")%></td>
        <td><%= rs("Cantidad")%></td>
        <td align="right">
          <%= FormatCurrency(rs("Precio"))%>
        </td>
      </tr>
  <% rs.MoveNext
      Loop
      rs.Close %>
</table>
```

# Índice

# Capas de un Sistema de Información Empresarial



# Responsabilidades de la Capa de Presentación

1. Permitir a los usuarios interactuar con el sistema.
2. Introducir datos en el sistema (validándolos previamente).
3. Visualizar los datos de salida de manera amigable al usuario.
4. Facilitar operaciones simples (filtros, ordenaciones y cambios de formato) sobre los datos.
5. Facilitar la navegación por el sistema.
6. Mejorar la experiencia de usuario (UX).
7. Gestionar la comunicación con el servidor.
8. Gestionar situaciones excepcionales.

# Responsabilidades de la Capa de Negocio

1. Atender las peticiones de los clientes.
2. Asegurar el cumplimiento de reglas de negocio existentes.
3. Asegurar la transaccionalidad de las operaciones de negocio.
4. Validar las peticiones de los clientes.
5. Recuperar y almacenar datos del almacén o almacenes persistentes.
6. Facilitar la eficiencia del sistema.
7. Controlar el acceso a los datos.
8. Gestionar la comunicación con los servicios externos.
9. Ejecutar operaciones (periódicas) del sistema.
10. Gestionar de manera adecuada casos excepcionales.
11. Ayudar a satisfacer los requisitos no funcionales.

# Responsabilidades de la Capa de Persistencia

1. Almacenar los datos de manera no volátil.
2. Recuperar datos del almacén persistente.
3. Asegurar la disponibilidad de los datos.
4. Controlar la integridad de los datos.
5. Asegurar un acceso eficiente a los datos.

# Índice

# Sistema Informático Web

## Sistema Informático Web

Un *Sistema Informático Web* es un sistema cliente servidor donde el cliente está codificado utilizando tecnologías web, como HTML, CSS y Javascript, siendo por tanto accesible desde un navegador web; y/o donde el cliente se comunica con el servidor por medio del protocolo HTTP.

# ¿Por Qué se Utilizan Tecnologías Web?

## Ventajas

1. Multiplataforma.
2. Onmipresencia de la web.
3. No precisan permisos especiales a nivel de red.
4. Facilidad de mantenimiento (código móvil).
5. Fuerte estandarización.
6. Favorecen la *recognizability*, reduciendo su curva de aprendizaje.
7. Madurez de las herramientas.

## Inconvenientes

1. Seguridad.
2. Tecnologías originalmente desarrolladas para *hipertexto*.
3. Tecnologías lentas.
4. Tecnologías dependientes de terceros.

# Índice

# Tecnologías de Implementación SIEs

## Presentation Layer

**Básico**
- HTML
- CSS
- Javascript
- AJAX

**Avanzado**
- jQuery
- Bootstrap
- Less, Sass

**Web Dinámica**
- JSP/ASP/PHP
- JSF

**MVC**

**RIA**
- GWT
- Vaadin

**SPA**
- AngularJS
- React

**Mobile**
- Android/Swift
- Xamarin

**Protocolos**
- HTTP
- HTTPS
- SOAP
- XML-RPC

**Formatos**
- HTML
- JSON
- XML
- PDF

## Business Layer

**Servidores de Aplicaciones / Frameworks**
- Spring
- JBoss, Glassfish, Weblogic
- Ruby on Rails
- NodeJS
- .NET

**Servicios**
- REST
- SOAP
- Websockects
- Servlets

**Entidades de Dominio**
- POJO's
- Enterprise Beans

## Persistence Layer

**Almacenamiento**
- Oracle, MySQL, SQL Server
- XML
- NoSQL
- In-memory databases
- Content Delivery Network

**Puentes de Persistencia**
- Hibernate
- My Ibatis
- Java Persistence API (JPA)
- LINQ

# Índice

# Despliegue de Aplicaciones Empresariales

1. Front-end en el cliente y back-end en uno o más servidores.
2. Dominio y persistencia pueden ir en el mismo servidor (*two tier*) o en servidores separados (three tiers).
3. Los clientes pueden ser pesados (PCs) o ligeros (Smartphones, tablets).
4. Trabajo sin conexión puede requerir parte de la capa de dominio (y persistencia) en el cliente.
5. La capa de presentación puede ser de código fijo (app, desktop) o móvil (HTML + Javascript).

# Índice

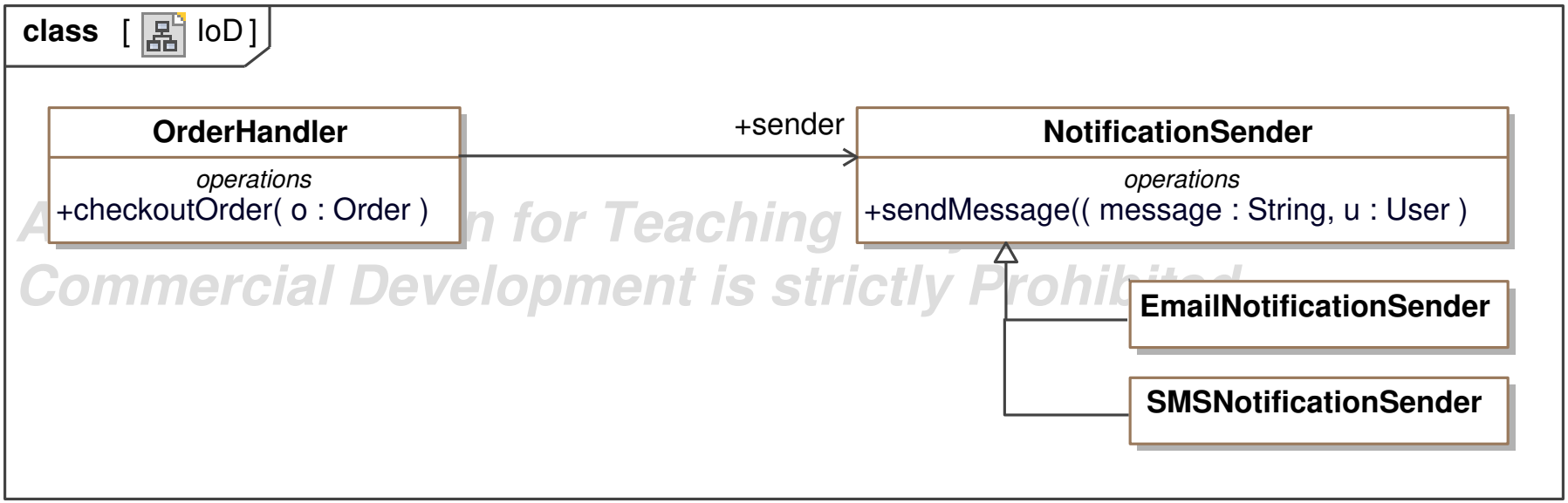
# Índice

# Principio de Inversión de Dependencias

## Principio de Inversión de Dependencias (IoD)

Las clases de un sistema software deberían depender de abstracciones estables y no de implementaciones concretas.

# Problema a Resolver I



# Problema a Resolver II

```
public class OrderHandler {

  // Depende de la clase abstracta
  protected NotificationSender sender;

  public OrderHandler() {
    //Al instanciarlo queda ligado a una clase concreta.
    this.sender = new SmsNotificationSender();
  } // Constructor
  ...
}
```

# Índice

# Inversión de Dependencias por Constructor

```
public class OrderHandler {

   protected NotificationSender sender;

   public OrderHandler(NotificationSender aSender) {
          this.sender =  aSender;
   } // Constructor
   ...
}
```

# Inversión de Dependencias por Constructor

```
public static void main(String[] args) {
   NotificationSender ns = new SmsNotificationSender();
   OrderHandler oh = new OrderHandler(ns);
}
```

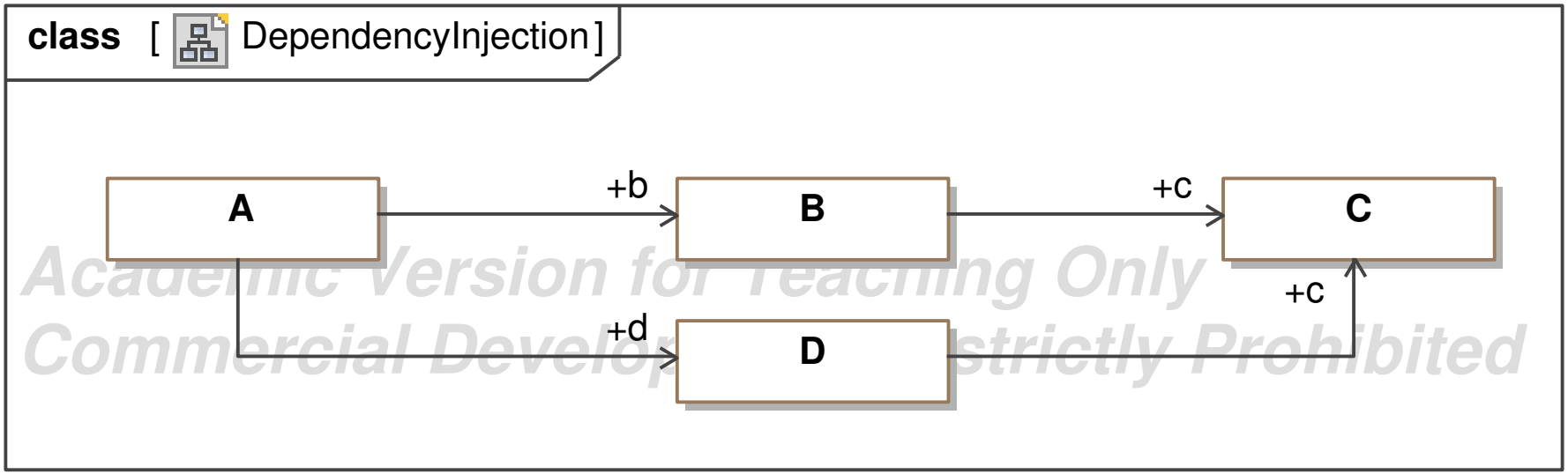
# Índice

# Inversión de Dependencias por *Setter*

```
public class OrderHandler {

  protected NotificationSender sender;

  public OrderHandler() {}

  public void setSender(NotificationSender aSender) {
    this.sender = aSender;
  }
  ...
}
```

# Índice

# Problema a Resolver



# Problema a Resolver

```
public static void main(String[] args) {
  // Para crear A me hace falta ...
  C c1 = new C_X();
  D d1 = new D_X(c1);
  C c2 = new C_X();
  B b1 = new B_X(c2);
  A a = new A_X(b1,d1);
}
```

# Solución - Inyector de Dependencias

- En un fichero de configuración, se especifican las subclases concretas a usar por cada clase abstracta.
- Se marcan de algún modo las dependencias a inyectar.
- Se crean objetos sin parámetros a través de una facilidad externa denominada *inyector de dependencias*.
- El inyector se encarga de crear los parámetros necesarios instanciando las subclases adecuadas conforme al fichero de configuración.

# Índice

# Inyección Simple - Definición

```
@Component
public class OrderHandler {

   @Autowired
   protected NotificationSender sender;

   public OrderHandler() {}

   public void processOrder(Order o) {
      this.sender.sendMessage("Order accepted",o.getUser());
   }
}
```

# Inyección Simple - Creación

```
public class Runner {

   @Autowired
   protected ApplicationContext context;

   @Override
   public void run(String... arg0) throws Exception {

      oh = context.getBean(OrderHandler.class);
      ...
           oh.processOrder(o);
   }
}
```

# Inyección Basada en Cualificadores I

```
@Component
@Qualifier("EmailNotifications")
public class EmailNotificationSender extends
    NotificationSender {

  @Override
  public void sendMessage(String message, User u) {}

}
```

# Inyección Basada en Cualificadores II

```
@Component
public class OrderHandler {

   @Autowired
   @Qualifier("EmailNotifications")
   protected NotificationSender sender;

   public OrderHandler() {}

   public void processOrder(Order o) {
      this.sender.sendMessage("Order accepted",o.getUser());
   }
}
```

# Inyección Basada en Anotaciones de Configuración I

```
@Component
public class OrderHandler {

   @Autowired
   protected NotificationSender sender;

   public OrderHandler() {}

   public void processOrder(Order o) {
      this.sender.sendMessage("Order accepted",o.getUser());
   }
}
```

# Inyección Basada en Anotaciones de Configuración II

```
@Configuration
public class IoD_DemoConfig {

   @Bean
   public NotificationSender sender() {
      return new SmsNotificationSender();
   }
}
```

# Inyección Basada en Configuración XML I

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    ...>
    <bean class="[...].iod.OrderHandler" autowire="byName">
    </bean>
    <bean id="sender"
          class="[...].iod.EmailNotificationSender">
    </bean>
</beans>
```

# Inyección Basada en Configuración XML II

```
@SpringBootApplication
@ImportResource("classpath:ApplicationContext.xml")
public class Runner implements CommandLineRunner {
```

# Índice

# ¿Qué Tengo que Saber de Todo Esto?

1. Comprender el concepto de *Sistema de Información Empresarial*.
2. Comprender el concepto de *Arquitectura Sw*.
3. Conocer los principios, ventajas e inconvenientes de las arquitecturas en capas, cliente-servidor y de código móvil.
4. Comprender qué es un sistema sw web, sus ventajas e inconvenientes.
5. Comprender por qué se divide un SIE en capas.
6. Conocer algunas de las tecnologías para la implementación de un SIE.
7. Comprender cómo se distribuyen las capas de un SIE.
8. Comprender qué es la *inversión de dependencias*.
9. Comprender qué es la *inyección de dependencias*.
10. Ser capaz de invertir dependencias *por constructor* y *por setter*.
11. Conocer cómo se inyectan dependencias en Spring.

# Referencias


Jazayeri, M., Linden, F. V. D., and Ran, A. (2000).
*Software Architecture for Product Families: Principles and Practice*.
Addison Wesley Professional.

Pohl, K., Böckle, G., and van der Linden, F. J. (2005).
*Software Product Line Engineering: Foundations, Principles and Techniques*.
Springer.

Taylor, R. N., Medvidovic, N., and Dashofy, E. (2009).
*Software Architecture: Foundations, Theory, and Practice*.
Wiley.