🎓 Resumen de Estudio: Arquitecturas de Software y SOLID

Resumen basado en el documento "APUNTES ARQUITECTURAS LIMPIAS".

🏛️ Unidad 4: Patrón de Arquitectura Monolítica

1. Concepto General

- **Definición:** Modelo tradicional donde **todas** las funcionalidades (interfaz, lógica, datos) están integradas en una **sola base de código** y se despliegan como una unidad indivisible.
- **Características Clave:**
  - **Simplicidad y Cohesión:** Todo en un mismo entorno.
  - **Autosuficiencia:** No depende de sistemas externos.
  - **Base de Datos Centralizada:** Una sola BD para todo.

2. Historia

- **60s-70s:** Mainframes (COBOL/Fortran).
- **80s:** Cliente-Servidor (pero monolíticos centralizados).
- **90s-2000:** Capas y MVC (organización interna).
- **2010+:** Nube y Microservicios (el monolito sigue vivo para casos simples).

3. Preguntas de Examen (IMP)

- **¿Con qué arquitecturas tiene ALTA cohesión el Monolito?**
  - R: Con **Capas** y **MVC**.
- **¿Con qué arquitectura tiene BAJA cohesión?**
  - R: Con **Microservicios**.

4. Ventajas vs. Desventajas

Ventajas	Desventajas
Simplicidad: Fácil de desarrollar y desplegar al inicio.	**Escalabilidad Limitada:** Solo escala verticalmente (más hardware).
Depuración: Todo el código está junto, fácil de rastrear.	**Mantenimiento:** Un cambio pequeño requiere redesplegar todo.
Rendimiento: Sin latencia de red (llamadas directas).	**Riesgo Global:** Un error en un módulo tumba todo el sistema.

Costo: Operación barata (infraestructura simple).	**Tecnología:** Difícil cambiar de lenguaje/framework (Stack acoplado).

🖥️ Unidad 5: Arquitectura Cliente-Servidor

1. Concepto

Modelo distribuido con dos roles claros:

- **Cliente:** Consume servicios, pide datos, interfaz ligera.
- **Servidor:** Provee servicios, procesa lógica pesada, centraliza datos.

2. Características (IMP)

- **Desarrollo Independiente:** Las tecnologías del cliente (ej. React) pueden ser totalmente distintas a las del servidor (ej. .NET).
- **Protocolos:** Comunicación bidireccional (TCP/IP).

3. Pros y Contras

- **Ventajas:** Centralización de datos (seguridad), clientes ligeros, portabilidad.
- **Desventajas:** El servidor es un **cuello de botella** (si falla, nadie trabaja), complejidad en actualizaciones de clientes distribuidos.

🍰 Unidad 6: Arquitectura en Capas (N-Tier)

1. Concepto

Organización del sistema en niveles horizontales con responsabilidades específicas.

- **Regla de Oro:** Cada capa solo habla con la inmediata inferior. (Presentación -> Negocio -> Datos).

2. Puntos Críticos (IMP)

- **Código Espagueti:** Si no se respeta el orden, se genera caos.
- **Pruebas Unitarias:** Permite probar cada capa por separado (Presentación, Negocio, Datos), aumentando la calidad.
- **Casos de Uso:** Ideal para aplicaciones empresariales tradicionales (ERP, CRM) o apps móviles con backend simple.

💠 Arquitectura Hexagonal (Puertos y Adaptadores)

1. Concepto (IMP)

Es la evolución de las capas. Su objetivo es **desacoplar la lógica de negocio** del mundo exterior.

- **Núcleo:** La lógica de negocio (centro).
- **Puertos:** Interfaces (contratos) que definen *qué* necesita el núcleo.
- **Adaptadores:** Implementaciones técnicas (BD, API, UI) que cumplen esos contratos.

2. Diferencia con Capas

- En Capas, la UI llama directo al Negocio.
- En Hexagonal, **no hay interacción directa**. Los adaptadores controlan las peticiones que entran y salen del núcleo.

3. Beneficios

- **Independencia:** Puedes cambiar la Base de Datos (Adaptador) sin tocar el Núcleo.
- **Testabilidad:** Fácil de probar simulando los adaptadores (Mocks).

🧼 Arquitectura Limpia (Clean Architecture)

1. Concepto

Organización en capas concéntricas basada en la **Regla de Dependencia**.

- **Regla (IMP):** Las dependencias **siempre** apuntan hacia adentro (hacia el núcleo). El núcleo no sabe nada de afuera.

2. Las Capas

1. **Dominio (Centro):** Entidades y reglas de negocio puras.
2. **Aplicación:** Casos de uso (orquestación).
3. **Adaptadores:** Controladores, Presentadores.
4. **Infraestructura (Borde):** Base de datos, Frameworks, UI.

3. Ventajas (IMP)

- **Independencia de Frameworks:** El framework es una herramienta, no tu sistema.
- **Testabilidad:** Se pueden hacer pruebas unitarias del núcleo sin base de datos ni servidor.

🧱 Principios SOLID

Son 5 normas para mejorar la calidad y mantenibilidad del software (POO).

1. **S - Responsabilidad Única (SRP):** Una clase debe tener una sola razón para cambiar (una sola tarea).
2. **O - Abierto/Cerrado (OCP):** Abierto para extensión, cerrado para modificación (usar herencia/polimorfismo en lugar de cambiar código viejo).
3. **L - Sustitución de Liskov (LSP):** Una subclase debe poder reemplazar a su clase padre sin romper el sistema.
4. **I - Segregación de Interfaces (ISP):** Mejor muchas interfaces pequeñas y específicas que una gigante.

5. **D - Inversión de Dependencia (DIP):** Depender de abstracciones (interfaces), no de implementaciones concretas. (Base de Clean/Hexagonal).

¡Mucho éxito en tu examen!