PATRONES DE DISEÑO

Mateo LLumigusin, David Asmal, Richard Gualotuña

Departamento de Ciencias de la Computación, Universidad de las Fuerzas Armadas ESPE

NRC: 23305

TUTORA Ing. Jenny Alexandra Ruíz Robalino

Fecha: 11/06/2025

1. **Definición del patrón de diseño**

   El modelo de capas es una forma común de organizar el software. Divide el sistema en partes horizontales, donde cada parte tiene un trabajo específico. Esta forma de trabajar ayuda a separar las tareas, hacer el sistema más modular y que cada parte trabaje de forma independiente. Así cada capa se concentra solo en una parte de la aplicación.

   Básicamente, cada capa funciona como un nivel que ofrece servicios a la capa superior y utiliza servicios de la capa inferior. Esto crea una estructura en la que la comunicación se produce de forma ordenada y controlada, lo que facilita el mantenimiento, el crecimiento y el cambio del sistema a lo largo del tiempo.

   Capa de presentación: Se ocupa de cómo el usuario interactúa y ve la información.

   Capa de lógica de negocio: Guarda las normas, los pasos y la lógica principal del sistema.

   Capa de acceso a datos: Administra cómo se guardan, se recuperan y se manejan los datos desde lugares como bases de datos.

   Capa de entidad o modelo: Establece las estructuras de datos o tipos de objetos que representan las cosas del sistema.

   El diseño por capas no solo crea una estructura organizada, sino que también separa las funciones, lo que ayuda a los equipos a trabajar juntos, probar partes individuales y agregar nuevas tecnologías a una capa sin cambiar las otras. Además, al usarse con otros diseños como la inyección de dependencias o el patrón repositorio, ayuda a seguir buenas prácticas de diseño, como la responsabilidad única y la inversión de dependencias.

2. Aplicación en la industria

El patrón de arquitectura en capas es uno de los más adoptados en la industria del software debido a su versatilidad, simplicidad y eficacia en la organización del código. Su aplicación se encuentra en múltiples sectores industriales, como el bancario, educativo, salud, comercio electrónico y administración pública.

- **Ventajas prácticas en entornos reales:**

  En aplicaciones empresariales, el patrón en capas promueve la separación de responsabilidades, lo que facilita el mantenimiento, escalabilidad y prueba del sistema. Cada capa puede desarrollarse, modificarse o reemplazarse independientemente, siempre que respete los contratos definidos entre capas. Esta separación es clave en entornos colaborativos y proyectos de gran escala.

  Por ejemplo, si un equipo detecta un error en la base de datos (capa de datos), puede corregirlo sin modificar la interfaz gráfica (capa de presentación). Asimismo, si se desea migrar el sistema a una nueva tecnología de frontend (como pasar de Angular a React), esto puede hacerse sin afectar la lógica del negocio, siempre que se mantenga la interfaz de comunicación con la capa intermedia.

- **Tecnologías que aplican arquitectura en capas:**

  Numerosos frameworks, lenguajes y plataformas modernas están diseñados para trabajar de forma natural con este patrón. Algunos ejemplos destacados:

  - **Java + Spring Boot:** Utiliza capas definidas para controladores (presentación), servicios (negocio) y repositorios (acceso a datos). Spring promueve el uso de anotaciones como @Controller, @Service y @Repository para delimitar cada capa.
  - **.NET con C# (ASP.NET Core):** Adopta el patrón en capas para manejar controladores (MVC), servicios, y acceso a datos con Entity Framework.

- **Node.js + Express + Angular/Vue/React:** Aunque más flexible, esta pila permite separar las capas con controladores en el backend, servicios intermedios y un frontend completamente independiente.
- **Python con Django:** Aunque Django sigue el patrón MTV (Model-Template-View), internamente puede organizarse en capas, separando vistas, lógica de negocio y modelos.

- **Casos de uso reales:**
  1. **Sistemas bancarios:** Donde la seguridad y la lógica del negocio son críticas, el patrón en capas permite encapsular procesos como validaciones, autorizaciones de transacciones, y conexión segura a bases de datos y servicios externos.
  2. **E-commerce (comercio electrónico):** Plataformas como Amazon y MercadoLibre utilizan arquitecturas en capas para diferenciar entre presentación (interfaz del usuario), lógica del carrito de compras o inventario, y el acceso a sistemas de bases de datos y microservicios.
  3. **Sistemas gubernamentales y ERP:** Aplicaciones como sistemas tributarios o de gestión de recursos empresariales requieren modularidad y capacidad de evolución. Separar las capas permite una mejor gestión de cambios, actualizaciones y auditorías de código.
  4. **Aplicaciones móviles y web modernas:** En el desarrollo con arquitecturas híbridas o PWA (Progressive Web Apps), el patrón en capas sigue siendo útil para organizar la lógica del cliente (frontend), la lógica de negocio (API RESTful) y el acceso a bases de datos o servicios en la nube.
- **Integración con otros patrones:**

  En muchas implementaciones industriales, el patrón en capas se combina con otros patrones arquitectónicos y de diseño, como MVC (Modelo-Vista-Controlador), MVVM, inyección de dependencias, repositorio y patrón fachada. Esto aporta aún más claridad estructural al sistema.

  Por ejemplo, una aplicación basada en Spring Boot puede seguir el patrón en capas y aplicar MVC en la capa de presentación, y patrones como DAO (Data

Access Object) en la capa de datos, fomentando así el diseño limpio y orientado a objetos.

3. **Ejemplo en código fuente**

El siguiente ejemplo demuestra cómo implementar el patrón de arquitectura por capas utilizando Java. Se simula un pequeño sistema de saludo a usuarios. El sistema valida si el usuario ingresado existe y si es mayor de edad, y en base a eso muestra un mensaje de bienvenida o un error.

Las capas que se aplican en el diseño son:

- Capa de presentación: Interactúa con el usuario (consola).
- Capa de negocio: Contiene la lógica para validar usuarios.
- Capa de acceso a datos: Simula la base de datos con una lista en memoria.
- Entidad: Representa la estructura del objeto Usuario.

Entidad

```java
package entidad;

public class Usuario {
    private String nombre;
    private int edad;

    public Usuario(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    public String getNombre() { return nombre; }
    public int getEdad() { return edad; }

    @Override
    public String toString() {
        return "Usuario{" + "nombre='" + nombre + '\'' + ", edad=" + edad + '}';
    }
}
```

Esta clase representa el modelo de datos. Define los atributos y métodos de la entidad Usuario. Es utilizada por todas las demás capas para representar la información del usuario.

Capa acceso a datos

```java
package datos;

import entidad.Usuario;

import java.util.ArrayList;
import java.util.List;

public class UsuarioDAO {
    private List<Usuario> usuarios;

    public UsuarioDAO() {
        usuarios = new ArrayList<>();
        usuarios.add(new Usuario(nombre:"Juan", edad: 25));
        usuarios.add(new Usuario(nombre:"Carlos", edad: 17));
        usuarios.add(new Usuario(nombre:"Andrea", edad: 30));
    }

    public List<Usuario> obtenerTodos() {
        return usuarios;
    }

    public Usuario buscarPorNombre(String nombre) {
        for (Usuario u : usuarios) {
            if (u.getNombre().equalsIgnoreCase(anotherString:nombre)) {
                return u;
            }
        }
        return null;
    }
}
```

Esta clase se encarga de acceder a los datos. En este caso, simula una base de datos mediante una lista de objetos Usuario. Proporciona métodos para obtener todos los usuarios y buscar por nombre.

Capa de lógica de negocio

```java
package negocio;

import datos.UsuarioDAO;
import entidad.Usuario;

import java.util.List;

public class UsuarioServicio {
    private UsuarioDAO usuarioDAO = new UsuarioDAO();

    public String generarSaludo(String nombre) throws ExcepcionUsuarioNoValido {
        Usuario usuario = usuarioDAO.buscarPorNombre(nombre);
        if (usuario == null) {
            throw new ExcepcionUsuarioNoValido("Usuario no encontrado: " + nombre);
        }

        if (usuario.getEdad() < 18) {
            throw new ExcepcionUsuarioNoValido("El usuario debe ser mayor de edad: " + nombre);
        }

        return "Bienvenido, " + usuario.getNombre() + ". Tienes " + usuario.getEdad() + " años.";
    }

    public List<Usuario> obtenerUsuarios() {
        return usuarioDAO.obtenerTodos();
    }
}
```

Aquí está la lógica del sistema. Esta clase usa UsuarioDAO para obtener datos y aplica reglas de negocio: verifica si el usuario existe y si es mayor de edad antes de devolver un saludo personalizado.

```java
package negocio;

public class ExcepcionUsuarioNoValido extends Exception {
    public ExcepcionUsuarioNoValido(String mensaje) {
        super(message: mensaje);
    }
}
```

Define una excepción que se lanza cuando un usuario no es válido. Se usa para manejar errores de manera controlada, evitando fallos inesperados en tiempo de ejecución.

Capa de presentación

```java
package presentacion;

import entidad.Usuario;
import negocio.UsuarioServicio;
import negocio.ExcepcionUsuarioNoValido;

import java.util.Scanner;

public class App {
    public static void main(String[] args) {
        UsuarioServicio servicio = new UsuarioServicio();
        Scanner scanner = new Scanner(source:System.in);

        System.out.println(x: "Usuarios disponibles:");
        for (Usuario u : servicio.obtenerUsuarios()) {
            System.out.println("- " + u.getNombre());
        }

        System.out.print(s: "\nIngresa tu nombre: ");
        String nombre = scanner.nextLine();

        try {
            String saludo = servicio.generarSaludo(nombre);
            System.out.println(x: saludo);
        } catch (ExcepcionUsuarioNoValido e) {
            System.out.println("❌ Error: " + e.getMessage());
        }

        scanner.close();
    }
}
```

Esta clase es el punto de entrada del programa. Muestra los usuarios disponibles, solicita al usuario que ingrese su nombre, y muestra un saludo o un mensaje de error según la lógica de la capa de negocio.

Corrida del Programa

```
Usuarios disponibles:
- Juan
- Carlos
- Andrea

Ingresa tu nombre: Juan
Bienvenido, Juan. Tienes 25 anios.
```

4. Bibliografía.

- Bass, L., Clements, P., & Kazman, R. (2012). Software architecture in practice (3rd ed.). Addison-Wesley.
- Microsoft. (n.d.). Layered architecture. Microsoft Docs. https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/layered
- Shaw, M., & Garlan, D. (1996). Software architecture: Perspectives on an emerging discipline. Prentice Hall.