Patrones arquitecturales
Arquitectura en capas

Diseño de Sistemas Software
Curso 2020/2021

Carlos Pérez Sancho

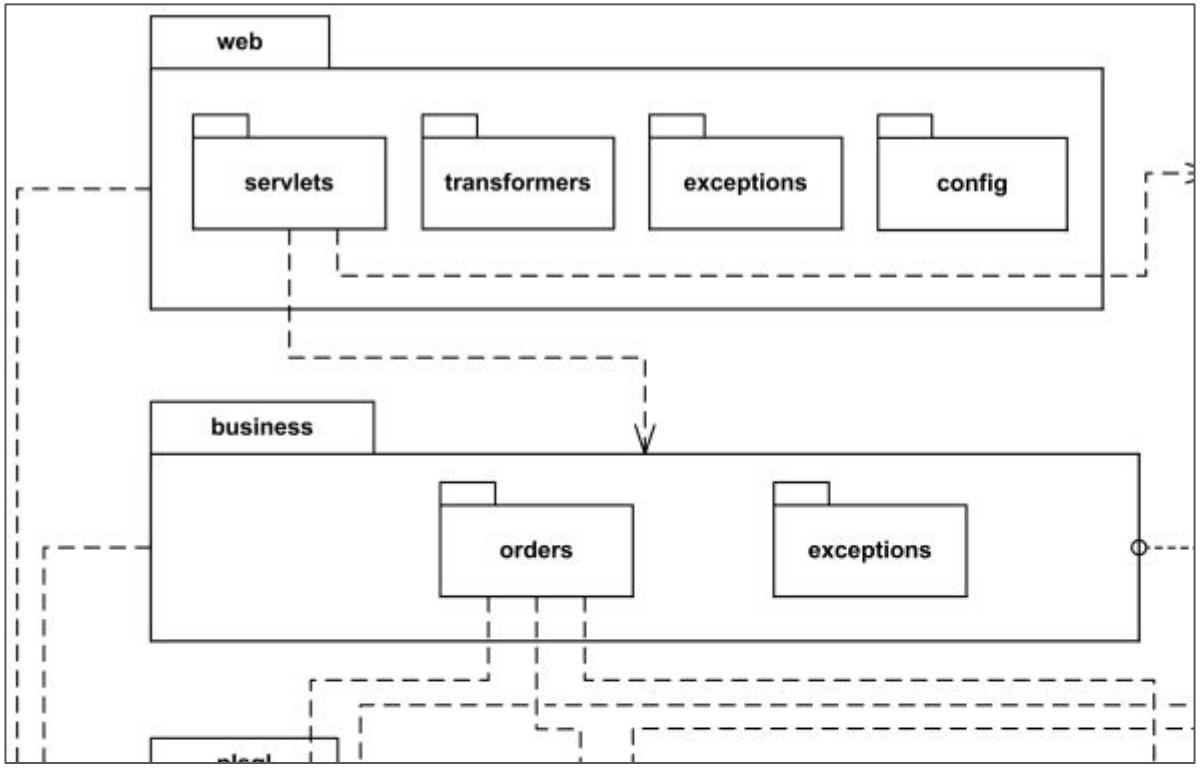
Universitat d'Alacant
Universidad de Alicante
Departament de Llenguatges i Sistemes Informàtics
Departamento de Lenguajes y Sistemas Informáticos

Arquitectura de software

- Distintas definiciones
  - “Descomposición de un sistema en componentes de alto nivel”
  - “Decisiones difíciles de cambiar”
  - “La arquitectura es el puente entre los objetivos de negocio (a menudo abstractos) y el sistema resultante (concreto)”

Arquitectura de software

- Un diseño arquitectural es una abstracción que muestra únicamente los detalles relevantes



Arquitectura de software

- La definición de una arquitectura debe incluir también una descripción del comportamiento de sus componentes
- La arquitectura prescribe la manera en la que se deben implementar los componentes y cómo deben interactuar, normalmente mediante restricciones
- Se deben tener en cuenta los requisitos no funcionales (p.ej. rendimiento esperado) y la distribución lógica y física de sus componentes

Arquitectura de software

- Las aplicaciones que no tienen una arquitectura formal, normalmente son
  - Altamente acopladas
  - Frágiles
  - Difíciles de cambiar
  - No tienen una visión o dirección claras
  - Es muy difícil determinar las características arquitecturales de una aplicación sin comprender el funcionamiento interno de cada componente y módulo en el sistema

Arquitectura de software

- Hay muchas formas de diseñar mal, pero sólo unas pocas de hacerlo bien
- El éxito en el diseño arquitectural es complejo y desafiante, por eso los diseñadores han encontrado maneras de capturar y reutilizar el conocimiento adquirido en otros sistemas
- Los patrones arquitecturales son formas de capturar estructuras de diseño de eficacia probada, de manera que puedan ser reutilizadas

¿Qué es un patrón de diseño?

A hombros de gigantes

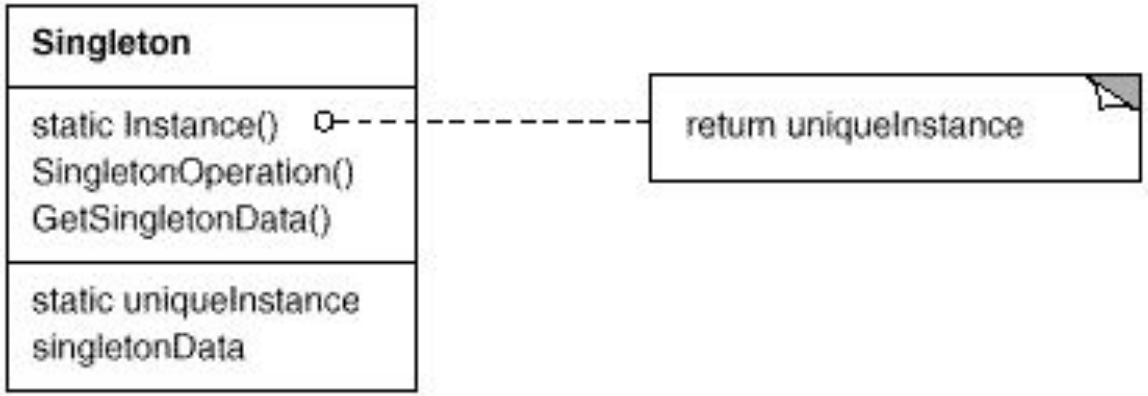
- La mayoría de problemas de diseño no son nuevos, sólo cambia el dominio de la aplicación
- No sólo se reutiliza el código, también podemos reutilizar soluciones anteriores
- Podemos reutilizar la experiencia de los expertos documentada a través de **patrones de diseño**

Patrones de diseño

- Son soluciones documentadas a problemas frecuentes en el diseño de software
  - Nombre del patrón
  - Problema y su contexto
  - Solución
  - Consecuencias
- Conocer los patrones de diseño más habituales ayuda a comprender mejor los sistemas, incluso observando directamente el código

Ejemplo

- Patrón **Singleton**
- Motivación: asegurar que una clase sólo tiene una única instancia, y proporcionar un punto de acceso global
- Solución:



- Consecuencias: acceso controlado a la única instancia, mejor que el uso de variables globales, ... (más en la bibliografía)

De vuelta a la arquitectura de software

Patrones arquitecturales

- Los patrones arquitecturales expresan esquemas de organización estructural fundamentales
- Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y principios para organizar sus relaciones
- Para los patrones más habituales es frecuente encontrar ***frameworks*** que permiten implementar una aplicación sin tener que escribirla desde cero

Frameworks

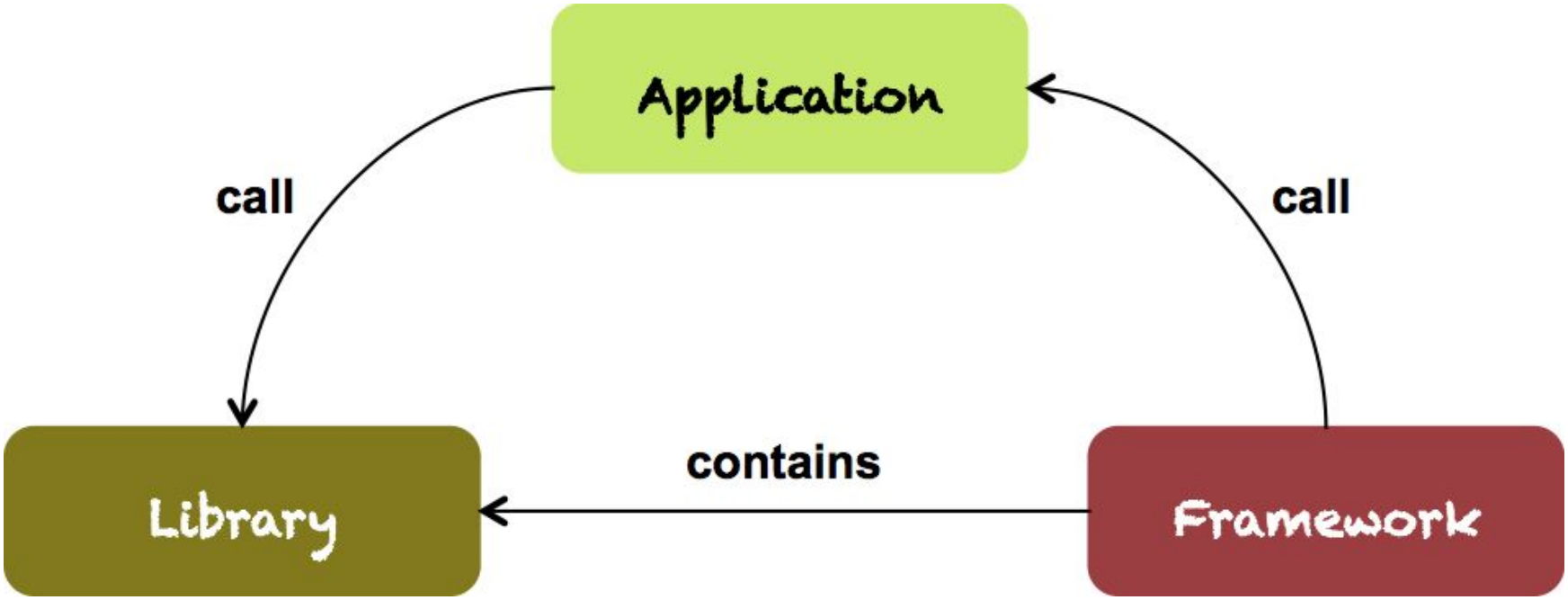
Frameworks

- Un framework es una aplicación reutilizable, “semi-completa” que se puede especializar para producir aplicaciones personalizadas
- Incorporan numerosos patrones de diseño en su implementación
- Muchos frameworks siguen el principio ***Convention over configuration***: no es necesario crear archivos de configuración si se respetan las convenciones de nombres (p.ej. mapeado de rutas en ASP.NET MVC)

Frameworks: principales características

- **Modularidad**: encapsulando los detalles de implementación detrás de interfaces estables
- **Reusabilidad**: definiendo componentes genéricos que se pueden reutilizar para crear nuevas aplicaciones
- **Extensibilidad**: proporcionando métodos “gancho” que permiten a las aplicaciones extender sus interfaces estables
- **Inversión de control** (Principio Hollywood): permite al framework (en lugar de a cada aplicación) determinar qué métodos de la aplicación serán invocados como respuesta a eventos externos

Frameworks: inversión de control



Fuente

Frameworks

- Beneficios de usar frameworks
  - Desarrollo más rápido
  - Código más estable y robusto
  - Mayor seguridad
  - Menor coste de desarrollo
  - Soporte de la comunidad

Frameworks

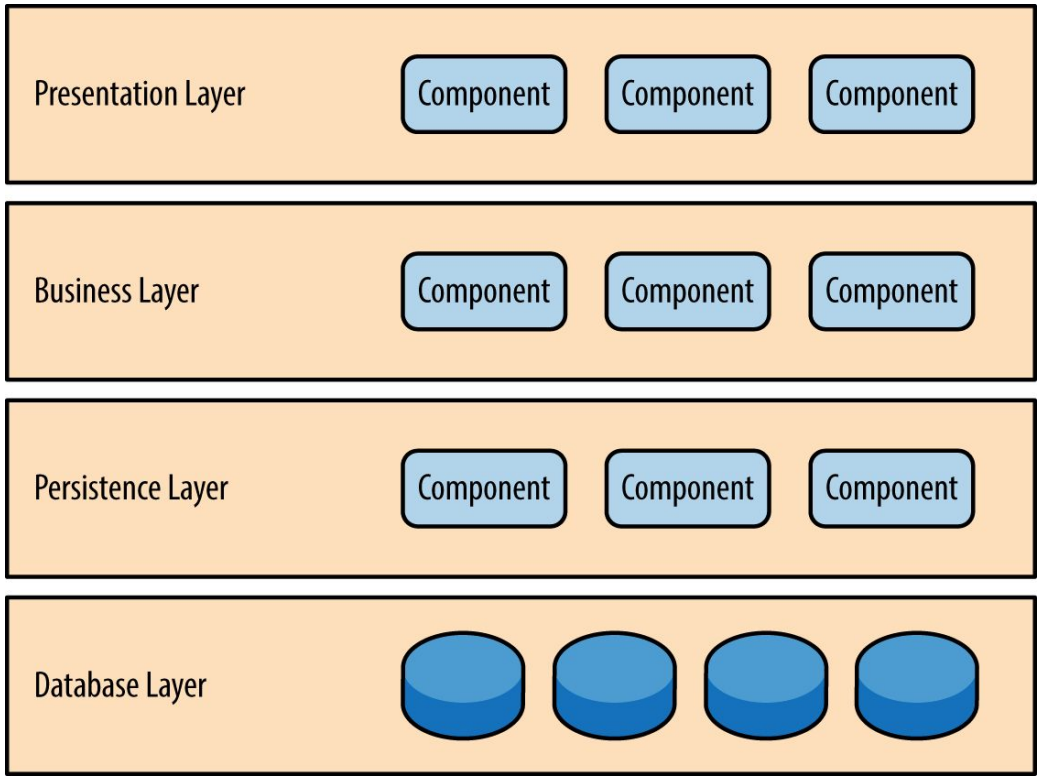
- Desventajas del uso de frameworks
  - Curva de aprendizaje
  - Problemas de integración con otros frameworks o librerías
  - Limitaciones
  - Es más difícil depurar el código
  - El código es público https://www.theregister.co.uk/2018/05/08/equifax_breach_may_2018/

Arquitectura en capas

Patrones arquitecturales

Arquitectura en capas

- Descompone una aplicación en componentes situados en distintos niveles horizontales llamados capas



Arquitectura en capas

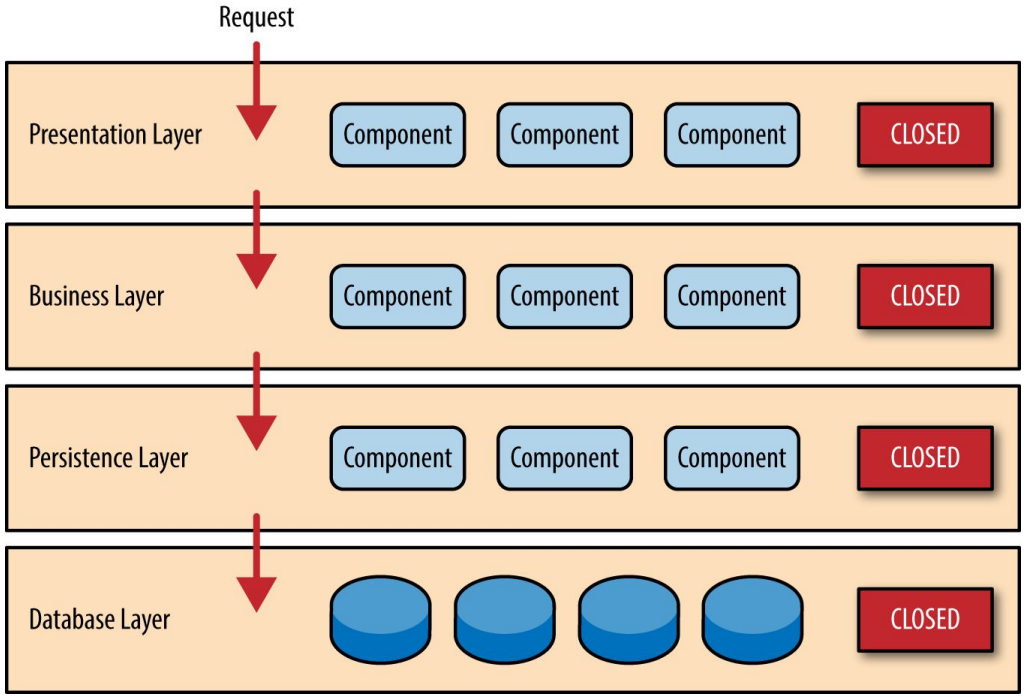
- Cada capa se encarga de una tarea específica, abstrayendo los detalles a las demás capas → separación de intereses
- Capas típicas (aunque el patrón no prescribe ninguna):
  - **Presentación:** interacción con el usuario
  - **Servicios:** funcionalidades de alto nivel
  - **Lógica de negocio:** ejecución de las reglas de negocio
  - **Persistencia (acceso a datos):** comunicación con la BBDD
  - **Base de datos:** almacenamiento de información

Arquitectura en capas

- La capa de lógica de negocio y la capa de persistencia no deben depender nunca de la capa de presentación
- La capa de lógica de negocio y la capa de persistencia pueden juntarse en una única capa en algunas circunstancias
- La capa de servicios ofrece funcionalidades compuestas a partir de las clases de la capa de lógica de negocio, ocultando su complejidad

Arquitectura en capas

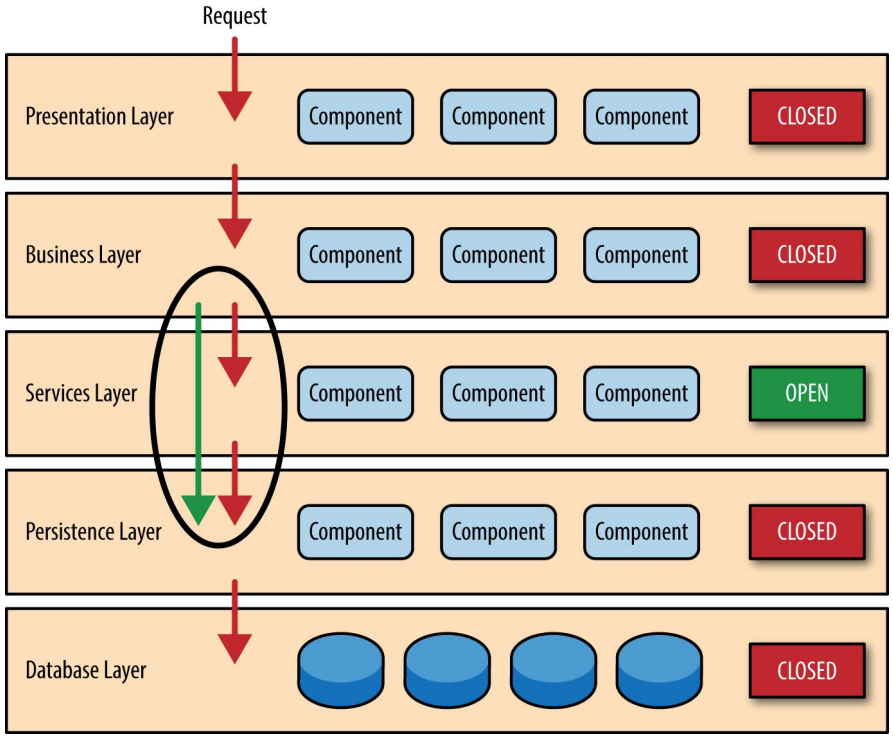
- **Arquitectura cerrada:** la comunicación va de una capa a la inmediatamente inferior
- Disminuye el impacto de los cambios y la complejidad del sistema



Arquitectura en capas

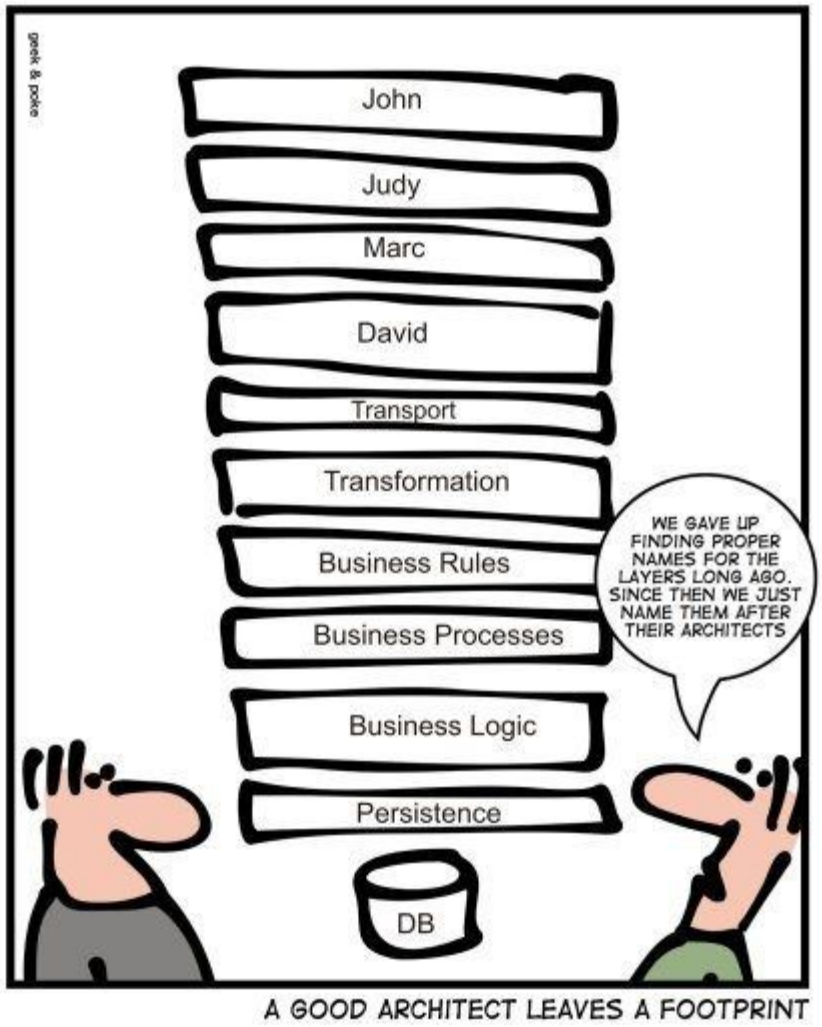
- **Arquitectura abierta:** las capas superiores se pueden comunicar con todas o algunas de las inferiores

- Necesario cuando la muchas peticiones se limitan a pasar de una capa a otra sin ninguna lógica de negocio asociada

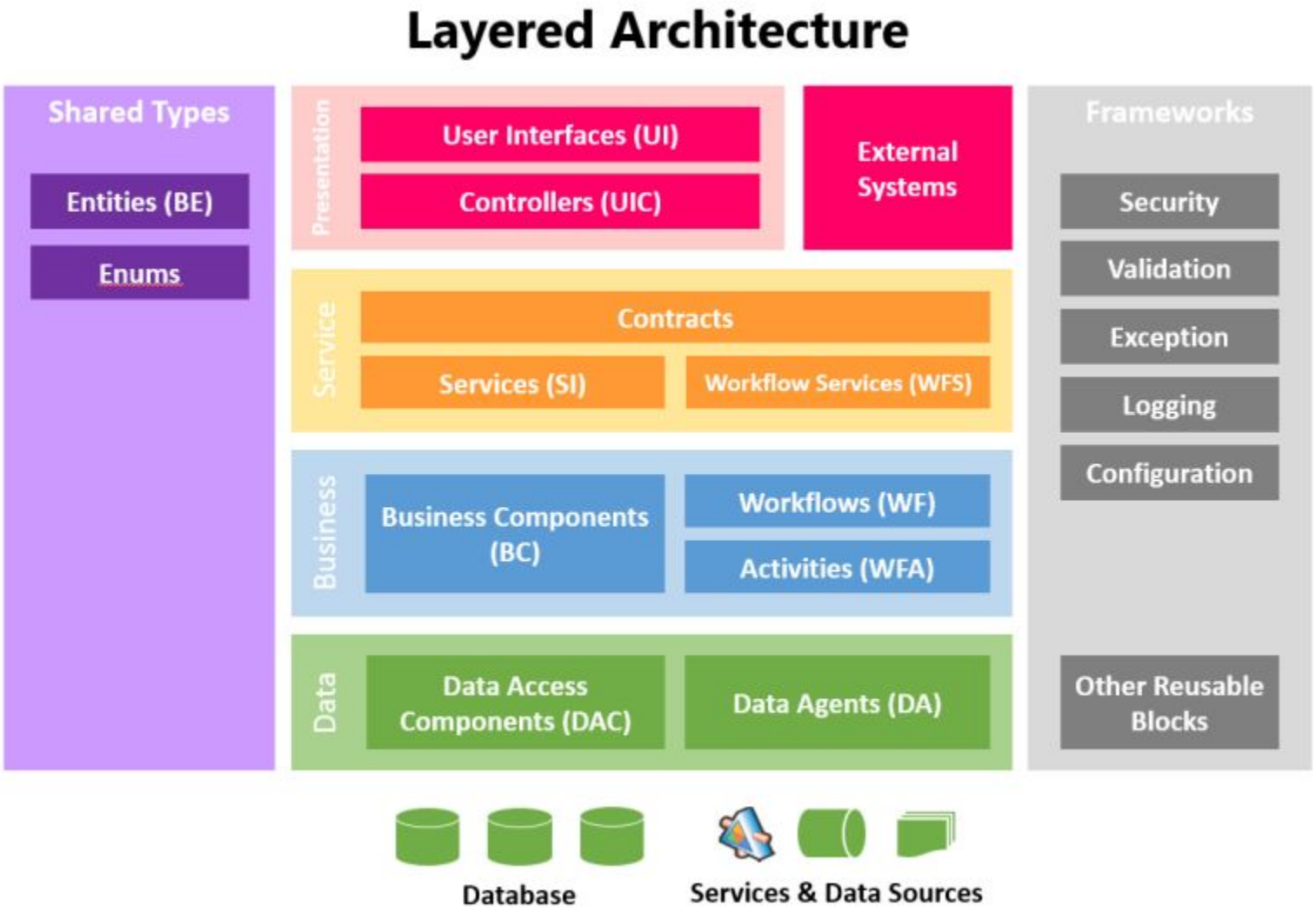


Arquitectura en capas

Se pueden crear tantas capas como sea necesario, siempre que cada capa tenga un propósito claro y separado de las demás



Ejemplo: arquitectura .NET



Fuente

Consideraciones

- Es la arquitectura más adecuada para la mayoría de aplicaciones
- No es necesaria cuando todas las peticiones pasan de una capa a otra sin procesamiento adicional (antipatrón sumidero)
- Cuando las capas no están distribuidas físicamente puede terminar convirtiéndose en una aplicación monolítica, lo que puede ser un riesgo para la escalabilidad

Análisis del patrón

- **Agilidad:** baja, los cambios normalmente afectan a varias capas y son lentos
- **Despliegue:** lento, un cambio en un componente puede requerir el despliegue de toda la aplicación
- **Pruebas:** el aislamiento entre capas facilita las pruebas
- **Rendimiento:** generalmente bajo, por el sobrecoste de la comunicación entre capas
- **Escalabilidad:** baja, si las capas no se distribuyen entre varios nodos
- **Desarrollo:** fácil, es un patrón muy conocido y extendido. Permite repartir el trabajo de cada capa a distintos expertos (BBDD, diseño de GUI, etc.)

¿Preguntas?

Bibliografía

- Bass, L., Clements, P., Kazman, R. (2012). ***Software Architecture in Practice, Third Edition***. Addison-Wesley Professional.
  Leer en Safari Books Online
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. (1996). ***Pattern-Oriented Software Architecture, Volume 1, A System of Patterns***. John Wiley & Sons.
  Leer en Safari Books Online
- Richards, M. (2015). ***Software Architecture Patterns***. O'Reilly Media, Inc.
  Leer en Safari Books Online
- Fayad, M., Schmidt, D.C. (1997). ***Object-Oriented Application Frameworks***. In Communications of the ACM, Special Issue on Object-Oriented Application Frameworks, Vol. 40, No. 10, October 1997.
  Enlace (febrero 2017)