10 patrones comunes en la arquitectura de software

Como te decíamos al principio, el desarrollo de software sigue una serie de pasos en su construcción, pero ¿sabes cuáles son? Aunque existen distintos modelos de arquitectura y procesos de desarrollo, podemos encontrar los siguientes arquetipos comunes, que describen los elementos y la relación entre ellos:

1. **Programación por capas.**

Puede utilizarse para estructurar programas que pueden descomponerse en subtareas. Cada una de ellas proporciona servicios a la capa siguiente y podemos encontrar los 4 comunes: capa de presentación, de aplicación, de lógica de negocios y de acceso a datos.

2. **Arquitectura de microservicios.**

Basa la construcción de las aplicaciones en un conjunto de pequeños servicios que se ejecutan en su propio proceso y se comunican con mecanismos ligeros. Por ejemplo: una API con recursos HTTP. Cada uno de estos servicios independientes se encargará de implementar una funcionalidad.

3. **Patrón cliente-servidor.**

El primero, se encarga de proporcionar servicios a múltiples componentes del cliente, mientras que este solicita servicios del servidor. Se trata de una especie de 'escucha' constante de las solicitudes del cliente.

4. **Patrón maestro-esclavo.**

Suele utilizarse para replicaciones en la base de datos (la maestra es la fuente autorizada y las esclavas se sincronizan con ella). Estas dos partes distribuyen el trabajo y calculan el resultado final de toda la actividad que realizan dichos esclavos. Este patrón es una arquitectura fundamental que los desarrolladores utilizan cuando tienen dos o más procesos que necesitan ejecutarse de forma simultánea.

5. **Filtro de tubería.**

Se utiliza, sobre todo, para la estructura de sistemas que producen y procesan una secuencia de datos. En la ingeniería de software, el filtro de tubería se aplica cuando los datos de entrada deben transformarse en datos de salida a través de componentes para el cálculo. Los componentes reciben el nombre de 'filtros' conectados entre sí por 'tuberías' que transmiten los datos.

6. **Patrón intermediario.**

Es usado para estructurar sistemas distribuidos con componentes desacoplados (pueden interactuar entre sí). El responsable de coordinar la comunicación entre los componentes es

el intermediario. Podemos encontrarlo en software de Message Broker, como la plataforma de software Apache ActiveMQ, por ejemplo.

7. **Modelo-vista-controlador.**

Es el conocido MVC, que divide una aplicación interactiva en tres partes (modelo, vista, controlador) encargadas de contener la funcionalidad, mostrar la información al usuario y manejar su entrada. Este patrón de arquitectura de software separa los datos y la lógica de negocio de una aplicación de su representación.

8. **Patrón de intérprete.**

Es usado para el diseño de un componente que interpreta programas escritos en un lenguaje y define cómo hacer la evaluación de las líneas de programas. "La idea básica es tener una clase para cada símbolo del idioma".

9. **Patrón de pizarra.**

Sus principales elementos son: la pizarra (memoria global estructurada), fuente de conocimiento (módulos especializados) y componente de control (encargado de seleccionar y ejecutar los módulos). Suele utilizarse para el reconocimiento de voz, identificaciones, seguimientos, etc.

10. **Patrón de igual a igual.**

Todos los elementos individuales se les denomina 'pares', que pueden funcionar tanto como 'cliente', como 'servidor'. Además, pueden ir cambiando su rol con el paso del tiempo.

Conclusiones acerca de los patrones arquitectónicos

Aunque todos los patrones que hemos definido en este post comunican la imagen de un sistema, no son arquitecturas en sí. El mundo de los desarrolladores es largo y complejo, pero la arquitectura de los sistemas y aplicaciones es vital para su funcionamiento. La arquitectura es quien diseña todas las entrañas del software, desde la estructura de sus datos hasta los algoritmos, y permite:

- **Crear una base sólida para el proyecto.**
- **Definir una plataforma escalable.**
- **Garantizar un código propio mucho más visible.**
- **Reducir los costes y evitar las posibles duplicaciones.**
- **Mejorar el rendimiento y la calidad.**