¿Qué es Arquitectura de Software?

los inicios de la informática, la programación se consideraba un arte y se desarrollaba como tal, debido a la dificultad que entrañaba para la mayoría de las personas, pero con el tiempo se han ido descubriendo y desarrollando formas y guías generales, con base a las cuales se puedan resolver los problemas. A estas, se les ha denominado Arquitectura de Software, porque, a semejanza de los planos de un edificio o construcción, estas indican la estructura, funcionamiento e interacción entre las partes del software. En el libro "An introduction to Software Architecture", David Garlan y Mary Shaw definen que la Arquitectura es un nivel de diseño que hace foco en aspectos "más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema".

**La programación por capas** es una arquitectura [cliente-servidor](https://es.wikipedia.org/wiki/Cliente-servidor) en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario.

[](https://commons.wikimedia.org/wiki/File:Tres_capas.PNG)

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, solo se ataca al nivel requerido sin tener que revisar entre código mezclado. Un buen ejemplo de este método de programación sería el [modelo de interconexión de sistemas abiertos](https://es.wikipedia.org/wiki/Modelo_de_interconexi%C3%B3n_de_sistemas_abiertos).

Además, permite distribuir el trabajo de creación de una aplicación por niveles; de este modo, cada grupo de trabajo está totalmente [abstraído](https://es.wikipedia.org/wiki/Abstracci%C3%B3n_(programaci%C3%B3n_orientada_a_objetos)) del resto de niveles, de forma que basta con conocer la [API](https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones) que existe entre niveles.

En el [diseño](https://es.wikipedia.org/wiki/Dise%C3%B1o) de [sistemas informáticos](https://es.wikipedia.org/wiki/Sistema_inform%C3%A1tico) actual se suelen usar las [arquitecturas multinivel](https://es.wikipedia.org/wiki/Arquitectura_software) o Programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

El más utilizado actualmente es el diseño en tres niveles

La **Arquitectura dirigida por eventos**, *Event-driven architecture* o EDA, es un patrón de [arquitectura software](https://es.wikipedia.org/wiki/Arquitectura_software) que promueve la producción, detección, consumo de, y reacción a [eventos](https://es.wikipedia.org/wiki/Evento_estad%C3%ADstico).

Un evento puede ser definido como "un cambio significativo en un estado". Por ejemplo, cuando un consumidor compra un coche, el estado del coche pasa de "se vende" a "vendido". La arquitectura del sistema del vendedor de coches debe tratar este cambio de estado como un evento, cuyo suceso puede ser conocido en otras aplicaciones en la arquitectura. Desde una perspectiva formal, lo que es producido, publicado, propagado, detectado o consumido es un mensaje (típicamente asíncrono) llamado notificación del evento, y no el evento en si mismo, el cuál es el cambio de estado que disparó la emisión del evento. Los eventos no viajan, solamente ocurren. Por otro lado, el término evento es frecuentemente usado para denotar el mensaje de notificación en sí mismo, lo cual puede llevar a algún tipo de confusión.

Este patrón arquitectónico puede ser aplicado por el diseño e implementación de aplicaciones y sistemas que transmitan eventos entre componentes software que estén emparejados libremente y servicios. Un sistema dirigido por eventos está compuesto típicamente de emisores de eventos (o agentes) y consumidores de eventos (o "sink" en inglés). Los consumidores tienen la responsabilidad de llevar a cabo una reacción tan pronto como el evento esté presente. La reacción puede o no puede ser completamente proporcionada por el consumidor en sí mismo. Por ejemplo, el consumidor debe tener solamente la responsabilidad de filtrar, transformar y reenviar el evento a otro componente o debe proporcionar una reacción propia a algún evento.

Construir aplicaciones y sistemas alrededor de una arquitectura dirigida por eventos permite a estas aplicaciones y sistemas ser construidos de una manera que facilita un mayor grado de reacción, debido a que los sistemas dirigidos por eventos están, por el diseño, más normalizados para entornos no predecibles y asíncronos.

**El patrón de arquitectura Microkernel** se aplica a sistemas de software que deben estar habilitados para adaptarse a requerimientos cambiantes del sistema. Separa un núcleo de funcionalidad mínima de la funcionalidad extendida y de partes específicas al cliente. También sirve como un socket para conectores en estas extensiones y coordinar su colaboración.

Las funciones centrales de un SO son controladas por el núcleo (kernel) mientras que la interfaz del usuario es controlada por el entorno (shell). Por ejemplo, la parte más importante del DOS es un programa con el nombre "COMMAND.COM" Este programa tiene dos partes. El kernel, que se mantiene en memoria en todo momento, contiene el código máquina de bajo nivel para manejar la administración de hardware para otros programas que necesitan estos servicios, y para la segunda parte del COMMAND.COM el shell, el cual es el interprete de comandos.

**La arquitectura de Microservicios** supone por definición un sistema altamente distribuido, que es considerablemente más complejo que uno monolítico. El sistema es más complejo, más flexible, más dinámico y sus costes (económicos y de operación) también son más altos. Cada microservicio vive en un entorno en el que colabora con otro conjunto de microservicios, es un entorno complejo, por lo que la automatización de todos los procesos ligados a la entrega del software (desde el checkin hasta el despliegue en producción) es especialmente importante.

Elegir una arquitectura basada en Microservicios o una monolítica depende de las necesidades que tenga el negocio al que de soporte el sistema en cada momento, y de la fase en la que se encuentre. En el corto plazo la aplicación monolítica va a arrancar más rápido (lo-necesitamos-en-3-semanas) y va a necesitar menos recursos/infraestructura. En el medio/largo plazo, si el negocio prospera, tiene éxito y crece, el sistema necesitará atender de forma independiente las distintas necesidades de cada una de sus “features”, y eso lo hará mejor con un conjunto de Microservicios centrados en cada una de ellas.