**Patrones de Arquitectura de Software**

Son aquellos que expresan un esquema organizativo estructural fundamental para sistemas de software.

Que objetivos tienen:

Además, también es importante reseñar el concepto de anti-patrón de diseño, que con forma semejante a la de un patrón, intenta prevenir contra errores comunes de diseño en el software. La idea de los anti-patrones es dar a conocer los problemas que acarrean ciertos diseños muy frecuentes, para intentar evitar que diferentes sistemas acaben una y otra vez en el mismo callejón sin salida por haber cometido los mismos errores.

ofrecen soluciones a problemas de arquitectura de software en ingeniería de software. Dan una descripción de los elementos y el tipo de relación que tienen junto con un conjunto de restricciones sobre cómo pueden ser usados.

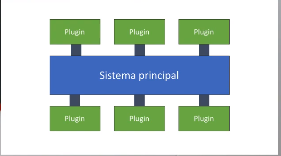
Un patrón arquitectónico es más un concepto que captura elementos esenciales de una arquitectura de software. Muchas arquitecturas diferentes pueden implementar el mismo patrón y por lo tanto compartir las mismas características.

Uno de los aspectos más importantes de los patrones arquitectónicos es que encarnan diferentes atributos de calidad. Por ejemplo, algunos patrones representan soluciones a problemas de rendimiento y otros pueden ser utilizados con éxito en sistemas de alta disponibilidad.

**Ejemplos:**

**Arquitectura microkernel o plugins:**

Hay básicamente 2 componentes el primero el sistema principal que es donde este concentrado la gran parte de la aplicación o el corazón de esta y por el otro lado están los plugins que se conectan a ese sistema principal para aumentar la funcionalidad.



Esto se asemeja mucho al principio de abierto y cerrado, ósea que el sistema principal siempre va a estar cerrado a modificaciones o tenga muy pocas, y este abierto para ser extendido a través de los plugins.

**Sistema principal:**  es el que contiene la funcionalidad básica de la aplicación lo que significa que su lógica es independiente de los plugins y de cada usuario.

Tendrá el acceso a los recursos generales del sistema que en algún momento un plugin requiera.

**Plugins:** contengan ciertas cosas especiales o especificas, pueden también tener procesamiento complejo ósea que si una tarea es muy pesada para el sistema principal el usuario puede instalar este plugin para que sea este el que tenga esa lógica.

Estos también deberían incluir integraciones que no están soportadas o contempladas por el sistema principal.

Todos estos plugins su regla es que deberían ser independientes tanto del sistema principal como de otros pugins

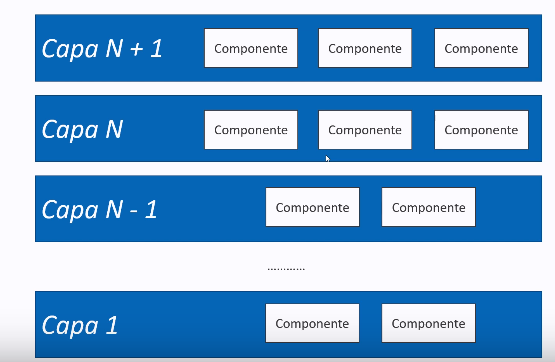
Hay algunos aspectos que se deberían cumplir:

**Registro:** como el sistema va a saber que pluggins a cargar o cuales no, un consejo es ponerlos en una misma ubicación y que de ahí el sistema principal los tome.

**Interacción:** como van a interactuar los plugins con el sistema principal, si por eventos, o por algún estadar como REST o SOAP.

**Arquitecturas por Capas en la nube:**

Como funciona la arquitectura por capas:



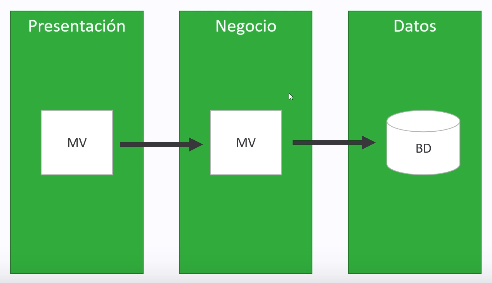
Una capa superior puede acceder a una capa inferior y estas capas no se pueden saltar para acceder a una capa inferior debería pasar por la intermedia

¿Como llevarla a la nube?



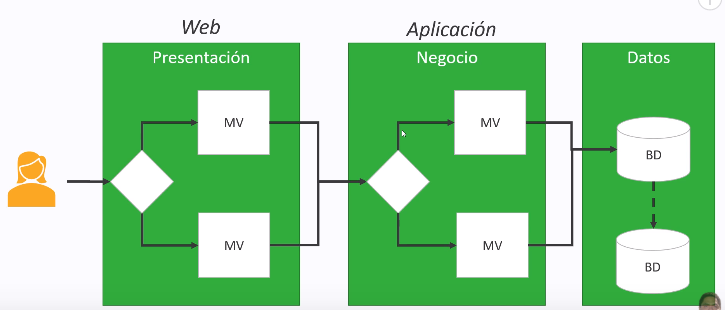
Con las mismas tres capas, se podrían tener una maquina virtual en la capa de presentación y otra en negocio y además la base de datos.

Esto nos ayuda a que si hay un cambio en la parte de negocio no se tengan que hacer cambios en la capa de presentación



Además de esto se deberían hacer otras cosas como en la capa de presentación y negocio se podría tener una segunda máquina virtual, además de una segunda BDD, y a esto anexarle un balanceador de carga esto nos ayudaría a enrutar el tráfico según algunas condiciones cada una de estas máquinas virtuales deberían ser sin estado, ya que si se empiezan a guardar por ejemplo sesiones o estados en cada una de estas el usuario solo quedaría amarrado a una maquina virtual y en el momento en que se quiera enrutar a un servicio diferente ahí habrá un problema.

En lo que se refiere a la base de datos lo que se hace una replica donde se puedan recuperar o reemplazar la base de datos principal.



**Arquitectura Orientada a Servicios:**

La Arquitectura SOA consiste en una serie de estándares empleados en la integración de sistemas y aplicaciones heterogéneos en plataformas y protocolos de comunicación con metodologías bien definidas.

El propósito de SOA no es solo crear una infraestructura integrada sino que esta permita cambios posteriores en respuesta a los cambios en las necesidades de la empresa.

La habilitación de servicios y su entrega parecerían ser el núcleo de la SOA pero el valor real de esta arquitectura es la automatización del negocio. Así, el enfoque en este tipo de arquitectura no debe estar tanto en los servicios, sino en los procesos y cómo mejorarlos.