PROCESO DIRECCIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: ANALISIS Y DESARROLLO DE SISTEMAS DE INFORMACION.
- Código del Programa de Formación: 228106
- Nombre del Proyecto:
- Fase del Proyecto: PLANEACIÓN
- Actividad de Proyecto: DISEÑAR LA ESTRUCTURA TECNOLÓGICA DEL SISTEMA INTEGRAL
- Competencia: PARTICIPAR EN EL PROCESO DE NEGOCIACIÓN DE TECNOLOGÍA INFORMÁTICA PARA PERMITIR LA IMPLEMENTACIÓN DEL SISTEMA DE INFORMACIÓN.
- Resultados de Aprendizaje: DEFINIR ESTRATEGIAS PARA LA ELABORACIÓN DE TÉRMINOS DE REFERENCIA Y PROCESOS DE EVALUACIÓN DE PROVEEDORES, EN LA ADQUISICIÓN DE TECNOLOGÍA, SEGÚN PROTOCOLOS ESTABLECIDOS.
- Duración de la Guía

2. PRESENTACIÓN

Para muchos, dominar la arquitectura de software es uno de los objetivos que han buscado durante algún tiempo, sin embargo, no siempre es claro el camino para dominarla, ya que la arquitectura de software es un concepto abstracto que cada persona lo interpreta de una forma diferente, dificultando con ello comprender y diseñar con éxito una arquitectura de software.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

- Materiales:
  Git, Visual code
  Portatil ó Computador de Escritorio
- Ambiente Requerido
- Descripción de la(s) Actividad(es)

EVOLUCIÓN DEL DESARROLLO DE SOFTWARE

En la imagen 1 se visualizan los grandes hitos de la evolución del desarrollo de software, las necesidades emergentes de cada uno de esos momentos y los avances tecnológicos que surgieron como solución. Se pone énfasis en la necesidad presente desde el inicio de contar con estrategias y técnicas que permitieran el desarrollo de productos confiables en el marco de proyectos predecibles. Esta necesidad fue la que motivó la creación de la Ingeniería de Software

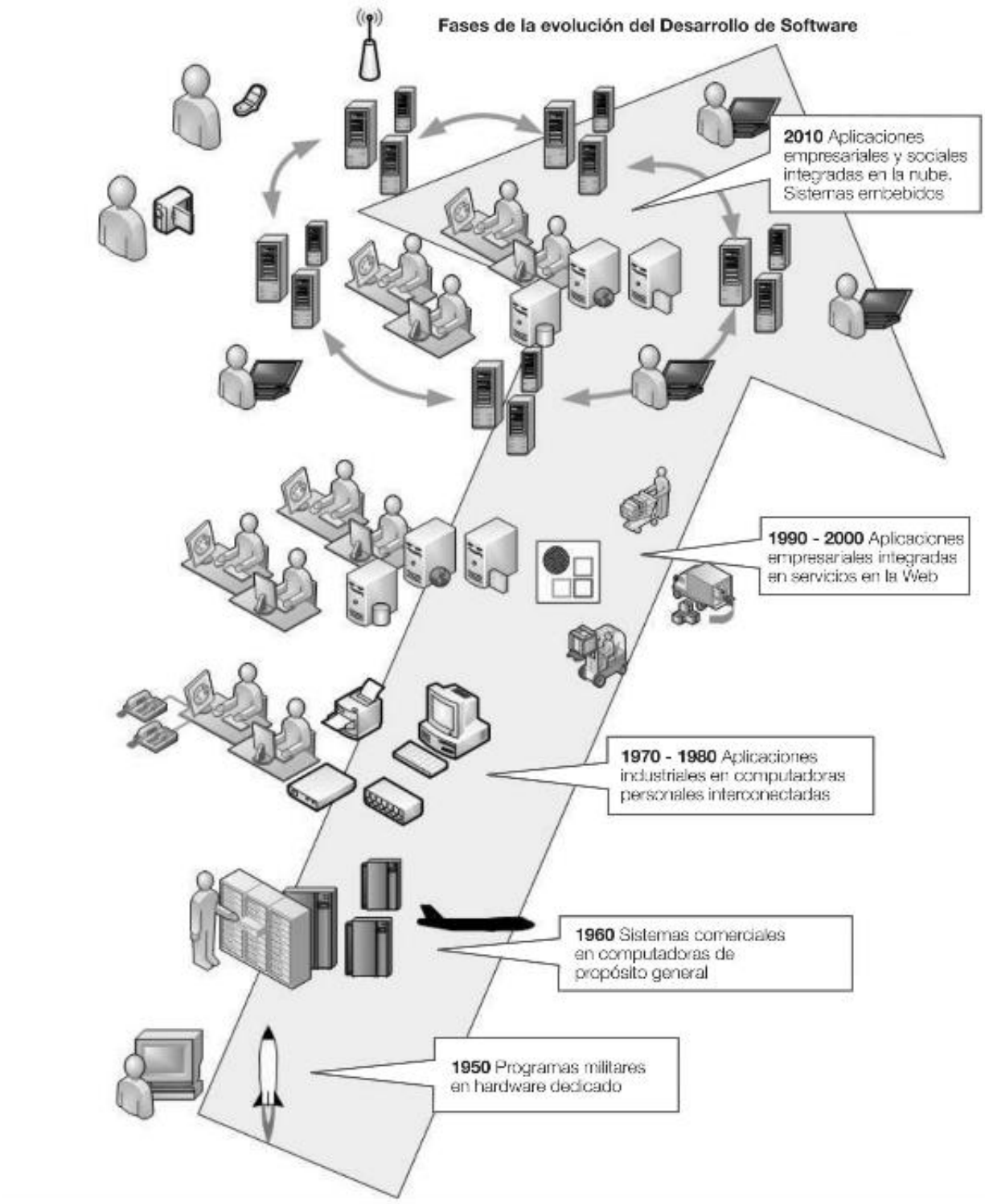

Imagen 1 – Evolución del desarrollo de software

ARQUITECTURA DE SOFTWARE

Antes de comenzar revisemos algunas definiciones de arquitectura de software:

“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 “

— "An introduction to Software Architecture" de David Garlan y Mary Shaw

“La Arquitectura de Software se refiere a las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos. “

— Software Engineering Institute (SEI)

“El conjunto de estructuras necesarias para razonar sobre el sistema, que comprende elementos de software, relaciones entre ellos, y las propiedades de ambos. “

— Documenting Software Architectures: Views and Beyond (2nd Edition), Clements et al, AddisonWesley, 2010

“La arquitectura de software de un programa o sistema informático es la estructura o estructuras del sistema, que comprenden elementos de software, las propiedades visibles externamente de esos elementos y las relaciones entre ellos. “

— Software Architecture in Practice (2nd edition), Bass, Clements, Kazman; AddisonWesley 2003

¿Que tienen en común las anteriores definiciones?

Todas coinciden en que la arquitectura **se centra en la estructura del sistema, los componentes que lo conforman y la relación que existe entre ellos**.

PATRONES DE DISEÑO

Los patrones de diseño tienen un impacto relativo con respecto a un componente, esto quiere decir que tiene un impacto menor sobre todo el componente. Dicho de otra forma, si quisiéramos quitar o remplazar el patrón de diseño, solo afectaría a las clases que están directamente relacionadas con él, y un impacto imperceptible para el resto de componentes que conforman la arquitectura.

A principios de la década de 1990 fue cuando los patrones de diseño tuvieron su gran debut en el mundo de la informática a partir de la publicación del libro Design Patterns, escrito por el grupo Gang of Four (GoF) compuesto por Erich Gamma, Richard Helm, Ralph Johnson y John Vlisides, en el que se recogían 23 patrones de diseño comunes que ya se utilizaban sin ser reconocidos como patrones de diseño.

Es importante mencionar que la utilización de patrones de diseño demuestra la madurez de un programador de software ya que utiliza soluciones probadas para problemas concretos que ya han sido probados en el pasado. Toma en cuenta que el dominio de los patrones de diseño es una práctica que se tiene que

perfeccionar y practicar, es necesario conocer las ventajas y desventajas que ofrece cada uno de ellos, pero sobre todo requiere de experiencia para identificar dónde se deben de utilizar.

Lo más importante de utilizar los patrones de diseño es que evita tener que reinventar la rueda, ya que son escenarios identificados y su solución está documentada y probada por lo que no es necesario comprobar su efectividad. Los patrones de diseño se basan en las mejores prácticas de programación.

Los patrones de diseño pretenden:

 Proporcionar un catálogo de soluciones probadas de diseño para problemas comunes conocidos.

 Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.

 Crear un lenguaje estándar entre los desarrolladores. Facilitar el aprendizaje a nuevas generaciones de programadores.

Asimismo, no pretenden:

 Imponer ciertas alternativas de diseño frente a otras.

 Imponer la solución definitiva a un problema de diseño.

 Eliminar la creatividad inherente al proceso de diseño.

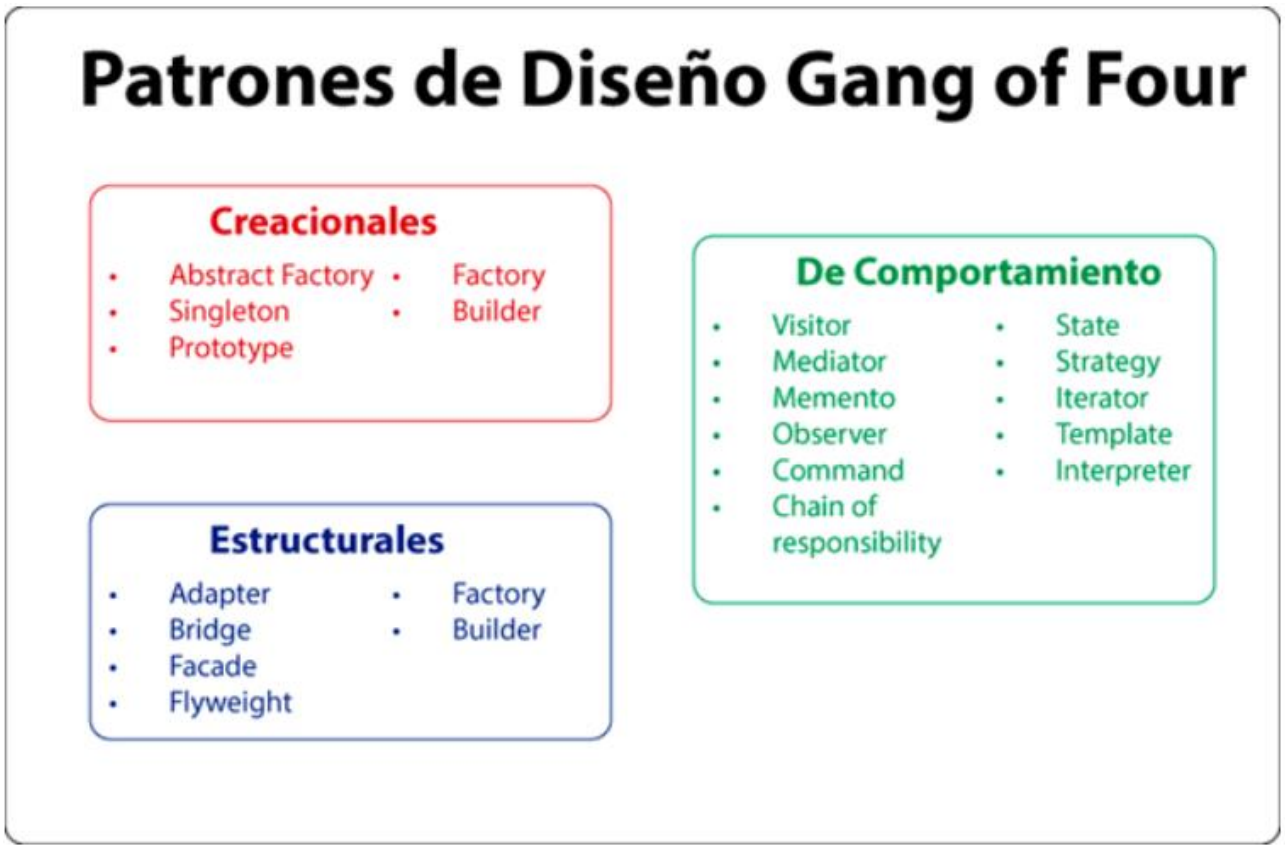

Imagen 2 – Lista de los patrones de diseño

PATRONES ARQUITECTONICOS

Los patrones arquitectónicos tienen un gran impacto sobre el componente, lo que quiere decir que cualquier cambio que se realice una vez construido el componente podría tener un impacto mayor.

A continuación, veremos las principales definiciones de los patrones arquitectónicos.

“Expresa una organización estructural fundamental o esquema para sistemas de software. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para organizar las relaciones entre ellos. “

— TOGAF

“Los patrones de una arquitectura expresan una organización estructural fundamental o esquema para sistemas complejos. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades únicas e incluye las reglas y pautas que permiten la toma de decisiones para organizar las relaciones entre ellos. El patrón de arquitectura para un sistema de software ilustra la estructura de nivel macro para toda la solución de software. Un patrón arquitectónico es un conjunto de principios y un patrón de grano grueso que proporciona un marco abstracto para una familia de sistemas. Un patrón arquitectónico mejora la partición y promueve la reutilización del diseño al proporcionar soluciones a problemas recurrentes con frecuencia. Precisamente hablando, un patrón arquitectónico comprende un conjunto de principios que dan forma a una aplicación.”

— Achitectural Patterns - Pethuru Raj, Anupama Raman, Harihara Subramanian

“Los patrones de arquitectura ayudan a definir las características básicas y el comportamiento de una aplicación.”

— Software Architecture Patterns - Mark Richards

Tanto los patrones de diseño cómo los patrones arquitectónicos pueden resultar similares ante un arquitecto inexperto, pero por ninguna razón debemos confundirlos, ya son cosas diferentes.

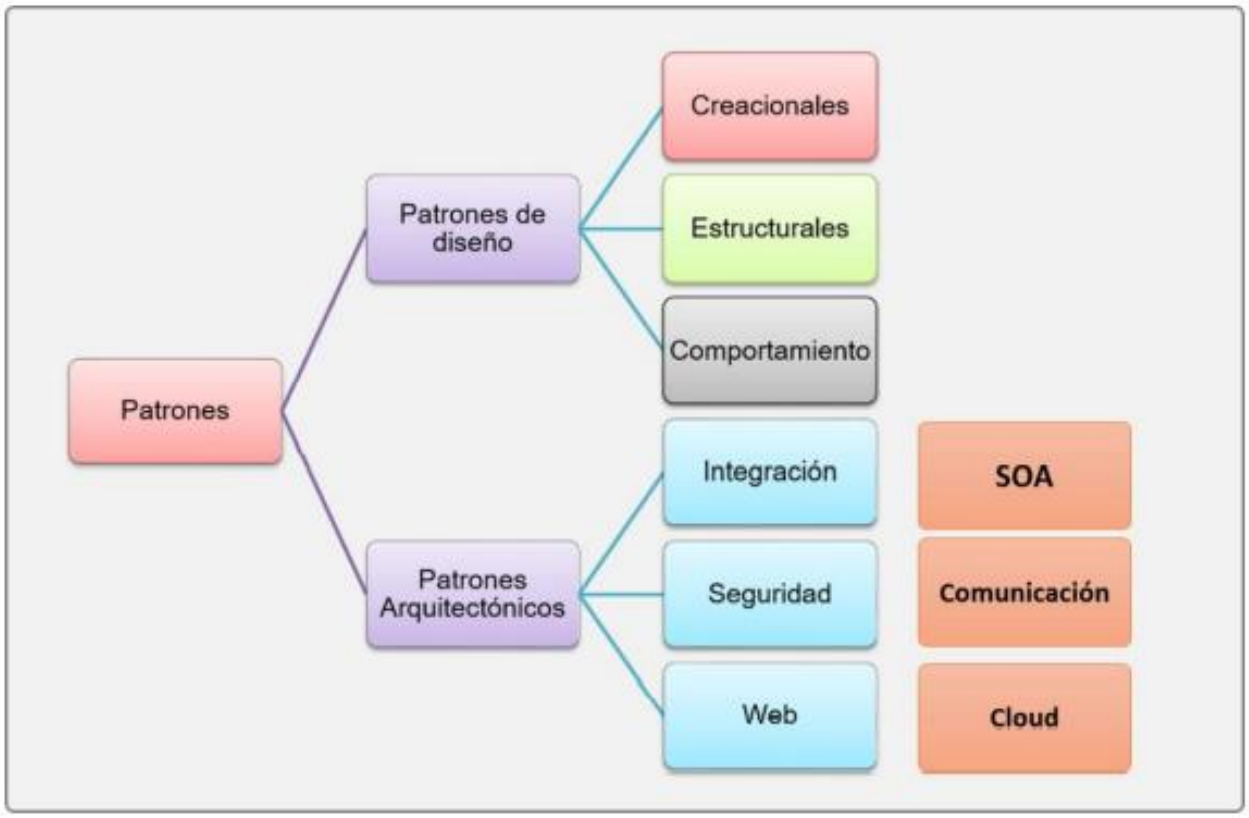

Imagen 3 – Consolidado de patrones

¿CÓMO DIFERENCIAR UN PATRÓN ARQUITECTÓNICO?

Los patrones arquitectónicos son fáciles de reconocer debido a que tiene un impacto global sobre la aplicación, e incluso, el patrón rige la forma de trabajar o comunicarse con otros componentes, es por ello que a cualquier cambio que se realice sobre ellos tendrá un impacto directo sobre el componente, e incluso, podría tener afectaciones con los componentes relacionados.

Un ejemplo típico de un patrón arquitectónico es el denominado “Arquitectura en 3 capas”, el cual consiste en separar la aplicación 3 capas diferentes, las cuales corresponden a la capa de presentación, capa de negocio y capa de datos:

Imagen 4 – Arquitectura de capas

ESTILOS ARQUITECTÓNICOS

Un estilo arquitectónico, es necesario regresarnos un poco a la arquitectura tradicional (construcción), para ellos, un estilo arquitectónico es un método específico de construcción, caracterizado por las características que lo hacen notable y se distingue por las características que hacen que un edificio u otra estructura sea notable o históricamente identificable.

En el software aplica exactamente igual, pues un estilo arquitectónico determina las características que debe tener un componente que utilice ese estilo, lo cual hace que sea fácilmente reconocible. De la misma forma que podemos determinar a qué periodo de la historia pertenece una construcción al observar sus características físicas, materiales o método de construcción, en el software podemos determinar que estilo de arquitectura sigue un componente al observar sus características.

Algunas definiciones de estilos arquitectónicos:

"Un estilo arquitectónico define una familia de sistemas en términos de un patrón de organización estructural; Un vocabulario de componentes y conectores, con restricciones sobre cómo se pueden combinar. "

— M. Shaw and D. Garlan, Software architecture: perspectives on an emerging discipline. Prentice Hall, 1996.

"Un estilo de arquitectura es una asignación de elementos y relaciones entre ellos, junto con un conjunto de reglas y restricciones sobre la forma de usarlos"

—Clements et al., 2003

"Un estilo arquitectónico es una colección con nombre de decisiones de diseño arquitectónico que son aplicables en un contexto de desarrollo dado, restringen decisiones de diseño arquitectónico que son específicas de un sistema particular dentro de ese contexto, y obtienen cualidades beneficiosas en cada uno sistema resultante."

—R. N. Taylor, N. Medvidović and E. M. Dashofy, Software architecture: Foundations, Theory and Practice. Wiley, 2009

LA RELACIÓN ENTRE PATRONES DE DISEÑO, ARQUITECTÓNICOS Y ESTILOS ARQUITECTÓNICOS

Para una gran mayoría de los arquitectos, incluso experimentados, les es complicado diferenciar con exactitud que es un patrón de diseño, un patrón arquitectónico y un estilo arquitectónico, debido principalmente a que, como ya vimos, no existe definiciones concretas de cada uno, además, no existe una línea muy delgada que separa a estos tres conceptos.

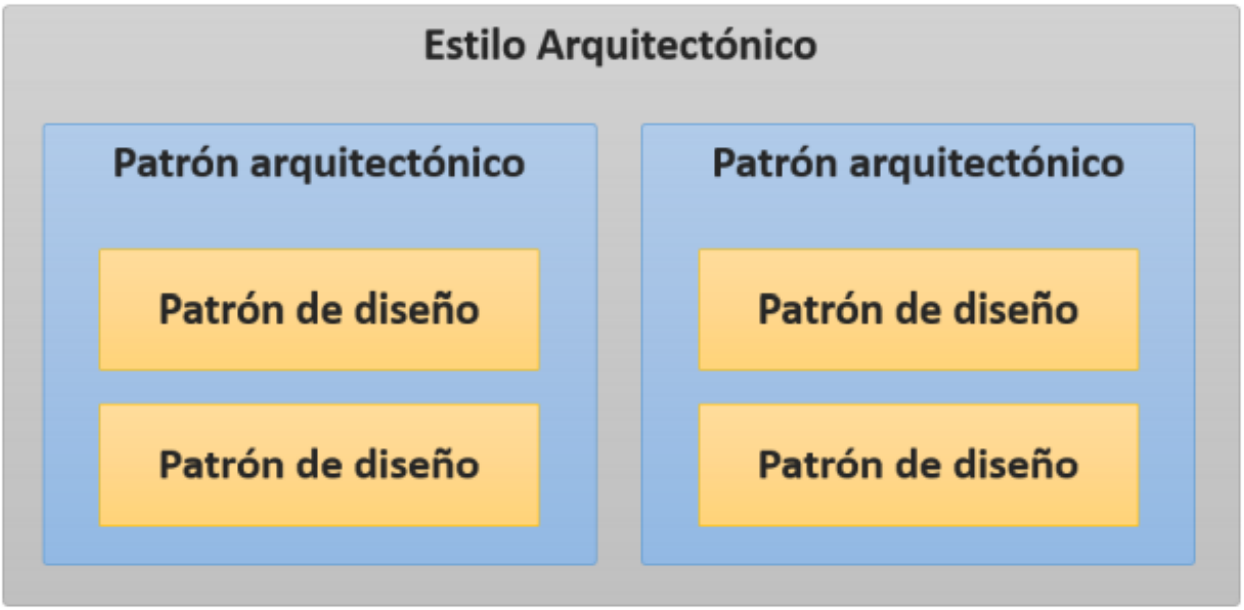

Imagen 5 – Relación entre estilo arquitectónico, patrón arquitectónico y patrón de diseño

ESTILOS ARQUITECTONICOS

Un estilo arquitectónico establece un marco de referencia a partir del cual es posible construir aplicaciones que comparten un conjunto de atributos y características mediante el cual es posible identificarlos y clasificarlos.

ARQUITECTURA EN CAPAS

La arquitectura en capas es una de las más utilizadas, no solo por su simplicidad, sino porque también es utilizada por defecto cuando no estamos seguros que arquitectura debemos de utilizar para nuestra aplicación. La arquitectura en capas consta en dividir la aplicación en capas, con la intención de que cada capa tenga un rol muy definido, como podría ser, una capa de presentación (UI), una capa de reglas de negocio (servicios) y una capa de acceso a datos (DAO), sin embargo, este estilo arquitectónico no define cuantas capas debe de tener la aplicación, sino más bien, se centra en la separación de la aplicación en capas (Aplica el principio Separación de preocupaciones (SoC))

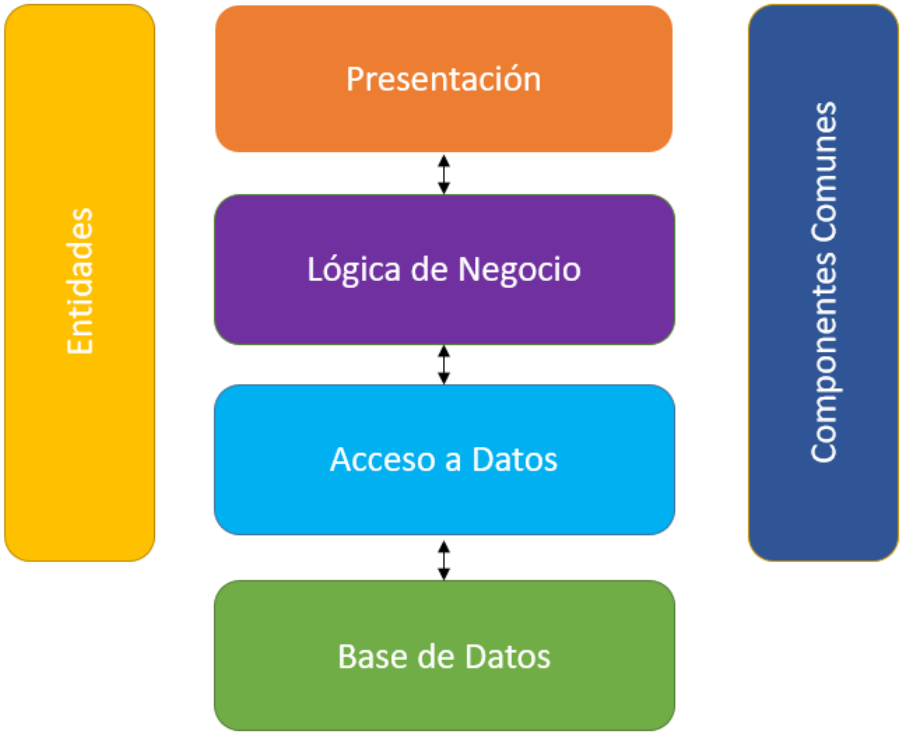

Imagen 6 – Arquitectura por capas

En la práctica, la mayoría de las veces este estilo arquitectónico es implementado en 4 capas, presentación, negocio, persistencia y base de datos, sin embargo, es habitual ver que la capa de negocio y persistencia se combinan en una solo capa, sobre todo cuando la lógica de persistencia está incrustada dentro de la capa de negocio.

COMO SE ESTRUCTURA UNA ARQUITECTURA EN CAPAS

En una arquitectura en capas, todas las capas se colocan de forma horizontal, de tal forma que cada capa solo puede comunicarse con la capa que está inmediatamente por debajo, por lo que, si una capa quiere comunicarse con otras que están mucho más abajo, tendrán que hacerlo mediante la capa que está inmediatamente por debajo. Por ejemplo, si la capa de presentación requiere consultar la base de datos, tendrá que solicitar la información a la capa de negocio, la cual, a su vez, la solicitará a la capa de persistencia, la que a su vez, la consultará a la base de datos, finalmente, la respuesta retornará en el sentido inverso hasta llegar la capa de presentación.

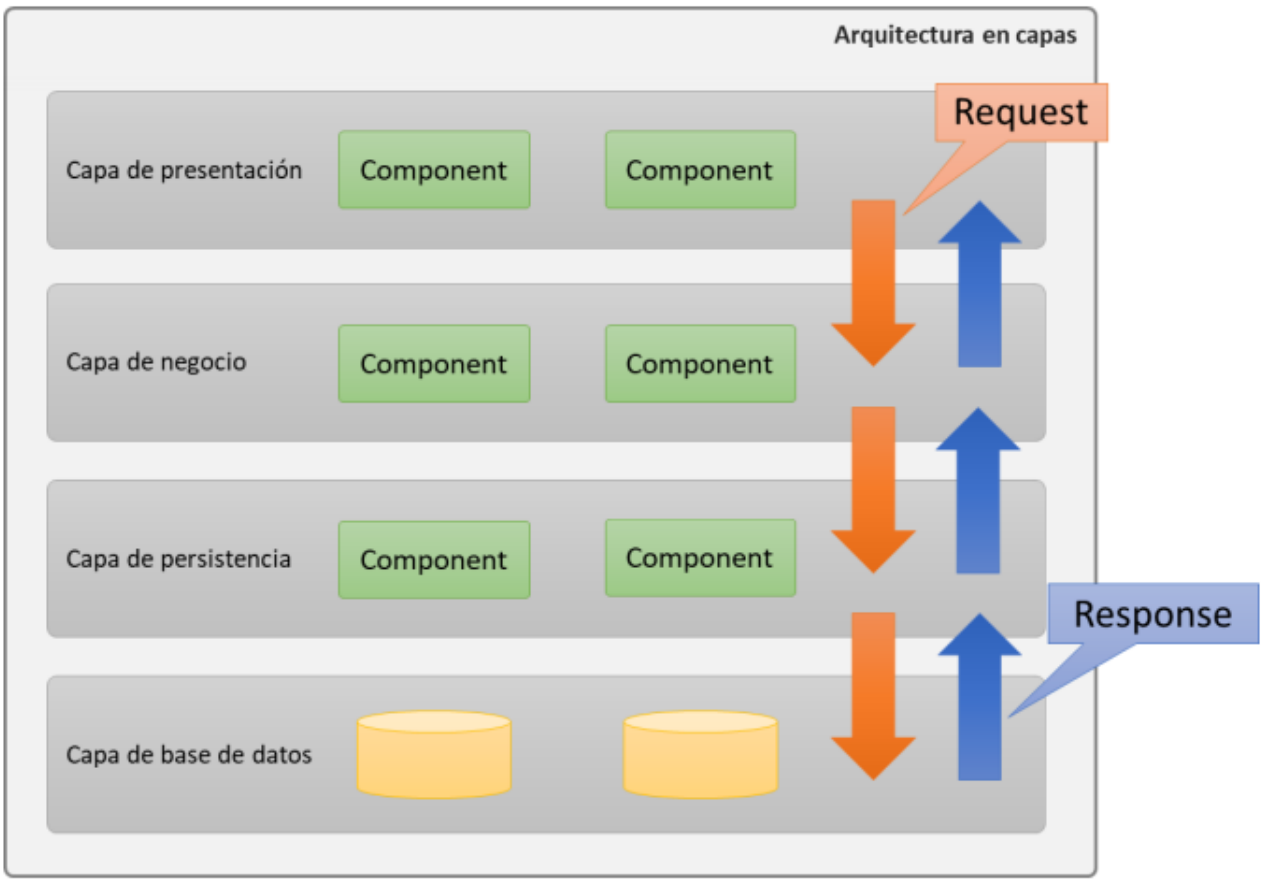

Imagen 7 – Comunicación entre capas

Un detalle a tener en cuenta en esta arquitectura, es que cada capa debe de ser un componente independiente, de tal forma que se puedan desplegar por separado, incluso, es habitual que estos componentes residan en servidores separados pero que se comunican entre sí.

La separación de la aplicación en capas busca cumplir con el principio de separación de preocupaciones, de tal forma que cada capa se encargue una tarea muy definida, por ejemplo, la capa de presentación solo se preocupa por presentar la información de forma agradable al usuario, pero no le interesa de donde viene la información ni la lógica de negocio que hay detrás, en su lugar, solo sabe que existe una capa de negocio que le proporcionará la información. Por otra parte, la capa de negocio solo se encarga de aplicar todas las reglas de negocio y validaciones, pero no le interesa como recuperar los datos, guardarlos o borrarlos, ya que para eso tiene una capa de persistencia que se encarga de eso. Por otro lado, la capa de persistencia es la encargada de comunicarse a la base de datos, crear las instrucciones SQL para consultar, insertar, actualizar o borrar registros y retornarlos en un formato independiente a la base de datos. De esta forma, cada capa se preocupa por una cosa y no le interesa como le haga la capa de abajo para servirle los datos que requiere.

Respetar el orden de las capas es muy importante, ya que brincarnos una capa para irnos sobre una más abajo suele ser un grave error, ya que bien es posible hacerlo, empezamos a crear un desorden sobre el flujo de comunicación, lo que nos puede llevar al típico anti patrón conocido como código espagueti, el cual consiste en que empezamos a realizar llamadas desde cualquier capa a otra capa, haciendo que el mantenimiento se vuelva un verdadero infierno.

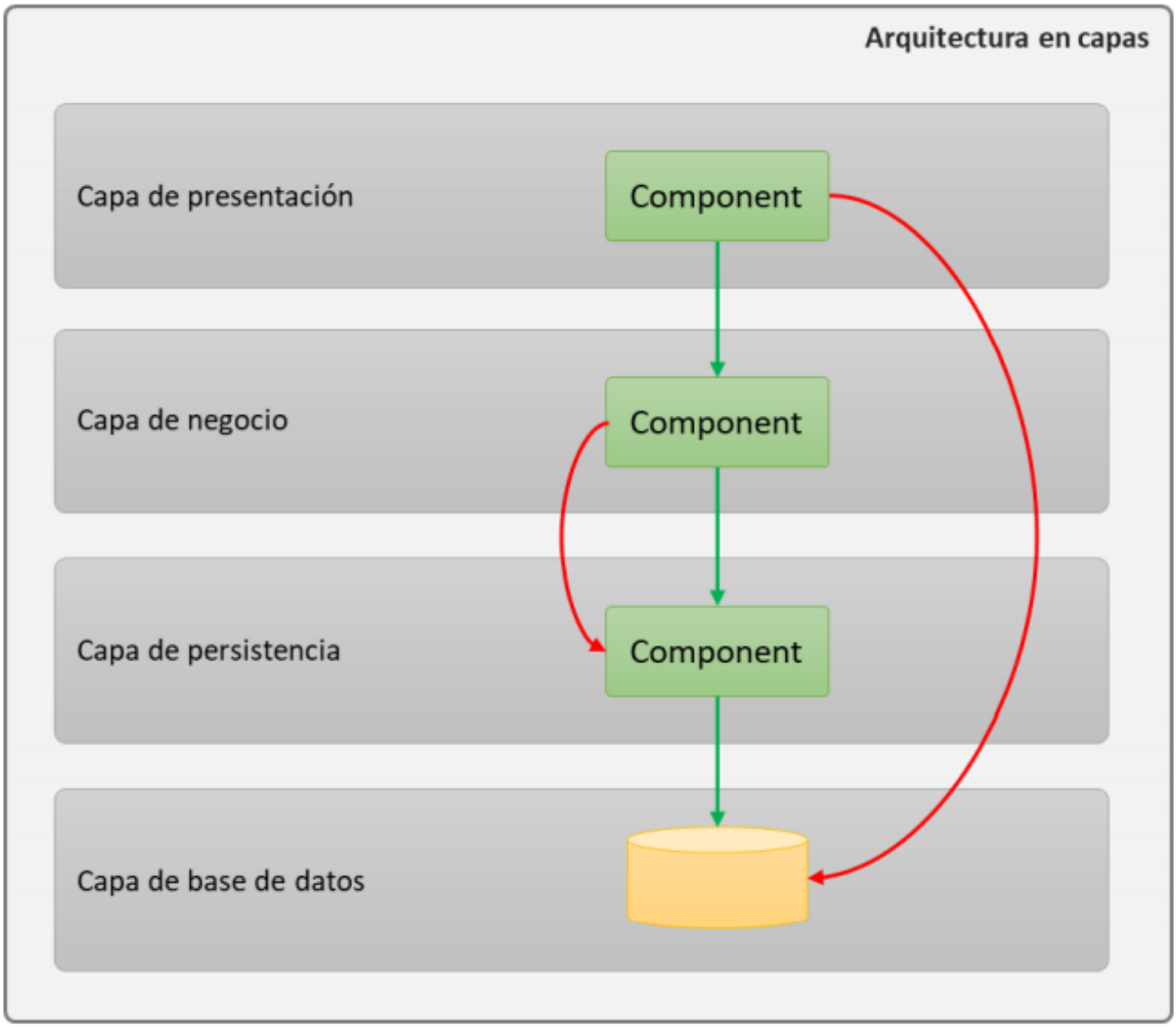

Imagen 8 – Salto indebido entre capas

Respetar el orden de las capas es importante por un término llamado Aislamiento de capas, el cual consiste en que cualquier cambio realizado en una capa no debería de afectar a otras capas. Permitir que una capa se comunique con otras no debidas, puede provocar que el cambio en una capa tenga un impacto sobre esta. Por ejemplo, si permitimos que la capa de presentación se comunique con la capa de persistencia y esta cambia su modelo de datos, tendríamos un impacto directo sobre la capa de presentación, pues está ya esperaba una estructura, la cual acabamos de cambiar. Ahora te preguntarás, si cambia el modelo de datos, seguramente también afecta a la capa de negocio, entonces cual es el caso de usar capas, pues bien, la ventaja es que mediante las capas podemos controlar hasta donde se propaga el impacto de un cambio, es decir, si cambia el modelo de datos, solo impactamos a la capa de negocio, pero esta podrá controlar el cambio para respetar el forma con el que la presentación recibe los datos, por lo que controlamos el impacto solo hasta la capa superior inmediata. En otro caso, si permitimos que una capa acceda a cualquier capa, los cambios se propagarían por todas las capas. Es importante resaltar que las capas funcionan con contratos

bien definidos, donde sabemos exactamente que reciben y que responde, esto hace que, si hacemos un cambio, pero no afectamos el contrato, aislamos el cambio y la capa superior ni se enterará del cambio.

Capas abiertas y cerradas

Hace un momento dejamos muy en claro que una capa solo se puede comunicar con la capa que está inmediatamente por debajo, sin embargo, hay algunas excepciones que en esta sección abordaremos. Si bien una arquitectura en capas se construye para satisfacer las necesidades de una aplicación, estas en ocasiones hacen uso de servicios genéricos o de utilidad, que por lo general son compartidos por varias aplicaciones, estos servicios podrían representar una nueva capa de servicios, lo que nos trae un problema, la capa de presentación debe de pasar por la capa de servicios o la capa de negocio, en este punto creo que todos coincidimos en que debería de pasar por la capa de negocio, lo que automáticamente nos indica que entonces la capa de negocio debería de llamar a la capa de servicios para consultar la capa de persistencia:

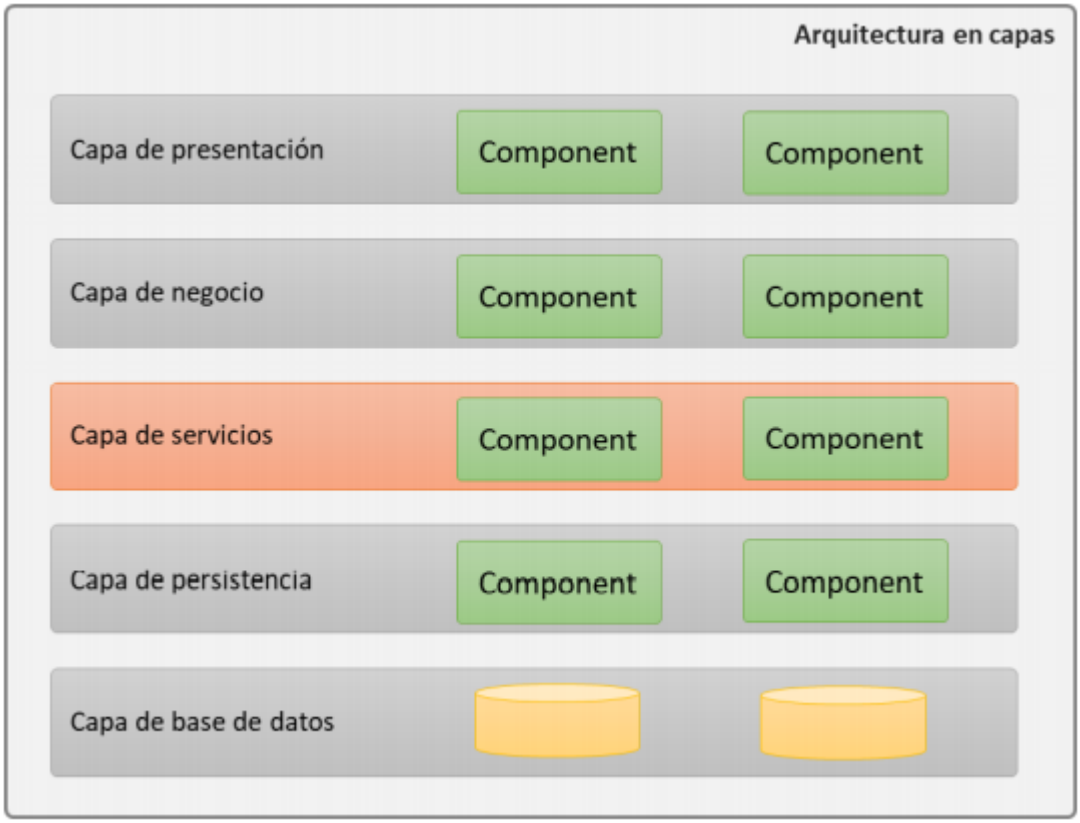

Imagen 10 – Capas genéricas

Si prestas un poco de atención, verás que esta arquitectura es totalmente ilógica, pues la capa de servicios por ningún motivo tomará en cuenta la capa de nuestra aplicación, ya que fueron diseñados para usos generales y no para cumplir con las reglas de nuestro proyecto. Esta nueva capa ha venido a complicar todo ¿cierto? Por suerte tenemos el concepto de capas abiertas y cerradas. Las capas cerradas es lo que hemos estado analizando todo este tiempo, es decir, son capas que no podemos brincar por ningún motivo, mientras que las capas abiertas son aquellas que está justificado brincarnos.

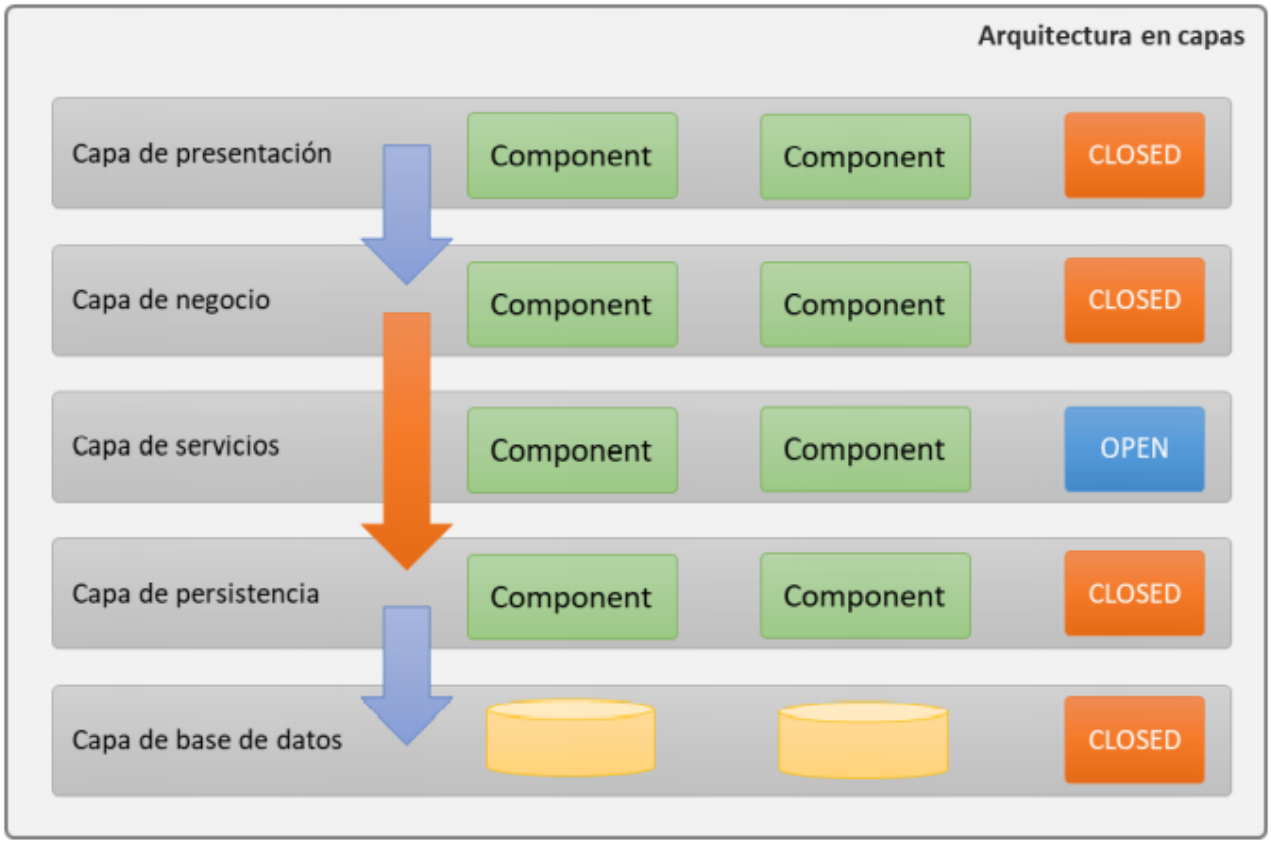

Imagen 11 – Capas abiertas y capas cerradas

En este nuevo diagrama hemos dejado en claro que la capa de servicios es una capa opcional, la cual podemos saltarnos para pasar de la capa de negocio a la capa de persistencia sin necesidad de pasar por la capa de servicios. Ahora bien, el hecho de que tengamos este nuevo termino de capas abiertas y cerradas significa que tenemos luz verde para saltarnos todas las capas con la justificación de que es una capa abierta, ya que todas las capas que sean clasificadas como abiertas deberán estar perfectamente justificadas, de lo contrario entraríamos incumpliendo el aislamiento de las cajas.

ARQUITECTURA EN 3 CAPAS

Acabamos de mencionar que la arquitectura en capas solo propone la construcción de capas, pero no define cuales deben de ser el número de capas o lo que tiene que hacer cada una, lo que nos deja un sabor de boca medio extraño, pues sabemos que es bueno crear capas, pero al mismo tiempo, no sabemos con certeza cuantas capas crear y como dividir de responsabilidad de la aplicación entre estas capas.

La arquitectura de 3 capas viene a aterrizar mejor el concepto del estilo arquitectónico en capas, definiendo cuales son las capas y que responsabilidad deberá tener cada una. Cabe mencionar que la arquitectura de 3 capas es solo una definición más concreta del patrón arquitectónico en capas. Uno de los principales problemas al construir aplicaciones, es que vamos construyendo todo el software como una enorme masa sin forma, donde un solo componente se encarga de la presentación, la lógica de negocio y el acceso a datos, lo que hace que las aplicaciones sean- sumamente difíciles de mantener y escalar, al mismo tiempo es

complicado para los desarrolladores dividir el trabajo, ya que muchas clases están relacionadas con otras y modificar una clase puede tener un gran impacto sobre muchas más.

Esta problemática era muy común sobre todos con tecnologías web antiguas, como PHP, ASP o JSP, donde era común encontrar la lógica para crea la vista, las reglas de negocio e incluso la conexión a la base de datos sobre el mismo archivo, por ejemplo:

```php
<?php
// -- Declaración de variables PHP y de acceso a BBDD
$base_datos = "coches";
$tabla = "niscoches";
$direccion_bd = "localhost";
$usu_bd  = "root";
$pass_bd = "55555";
 ?>

<html>
<head>
  <title>Test-MySQL</title>
</head>
<body bgcolor="#e0e0e0">

<!-- ---- Conexion a la BBDD ---- -->
<?php
// Conectamos con BBDD
$conexion = mysql_connect ($direccion_bd,$usu_bd,$pass_bd) or die ("No es posible acceder a la BBDD");
?>

<h4 align=center> <FONT COLOR=#3333dd>Consulta de datos (Precio < 15000)</h4>

<!-- ---- Inicio consulta (1) PHP --- -->
<table width="300" border="1" align="center" cellpadding="0" cellspacing"0">
<tr>
<td>MARCA</td>
<td>MODELO</td>
<td>COLOR</td>
<td>PRECIO</td>
</tr>
<?php
   $consulta="SELECT * FROM $tabla WHERE  (precio < 15000) " ;
   $resultado=mysql_db_query ($base_datos,$consulta,$conexion);
   while ($registro = mysql_fetch_array ($resultado))
   {
      print ("<tr>");
      print ("<td>$registro[marca]</td>\n");
      print ("<td>$registro[modelo]</td>\n");
      print ("<td>$registro[color]</td>\n");
      print ("<td>$registro[precio]</td>\n");
      print ("</tr>");
   }
  mysql_free_result($resultado);
?>
</table>
<!-- ---- Fin de consulta PHP ---- -->

<!-- --- Desconexion de la BBDD --- -->
<?php
mysql_close($conexion);
?>
</body>
</html>
```

Imagen 12 – Vista que conecta a la base de datos

El código en PHP, el cual permite definir como se vería la página e incrustas fragmentos de código en las cuales podíamos hacer casi cualquier cosa.

Cuando tecnologías como JSP, PHP, ASP llegaron al mercado, era muy común ver este tipo de páginas, las cuales eran sumamente difícil de mantener y prácticamente no se podía reutilizar nada de código, por lo general, lo que muchos desarrolladores hacían era copiar el código de una página y pegarlo en otra, lo cual era sumamente ineficiente, y al detectarse un bug en ese código se tenía que arreglar todas las páginas donde se había copiado el código (Infringían el principio Don't Repeat Yourself (DRY)) . No es que JSP, PHP o ASP fueran malos lenguajes, o que no hubiera forma de solucionar esto, solo se muestra lo que hacían muchos de los programadores inexpertos.

Pero a pesar de lo que muchos puedan creer, este problema no es exclusivo de aplicaciones web, también se presenta en aplicaciones de escritorio o móviles, ya que al final, todas estas cuentan con presentación, lógica de negocios y acceso a datos, es por ello que la arquitectura de 3 capas se presenta como una solución general para resolver este tipo de problemas.

Como hemos analizando, separar la aplicación en unidades de código cohesivas es necesario para una mejor organización del proyecto, pero las ventajas que ofrecen no son solo las de organizar código, si no que ayuda mucho a la reutilización del mismo.

La arquitectura en 3 capas está basada en el estilo de arquitectura en capas, donde cada capa se encarga de una parte del programa, y el orden de ejecución siempre va de la capa de más arriba a la de más abajo, sin embargo, la arquitectura de 3 capas define exactamente las capas que debe de tener la aplicación y la responsabilidad de cada una.

En resumen, la arquitectura define que una aplicación debe de estar diseñada en 3 capas, las cuales son:

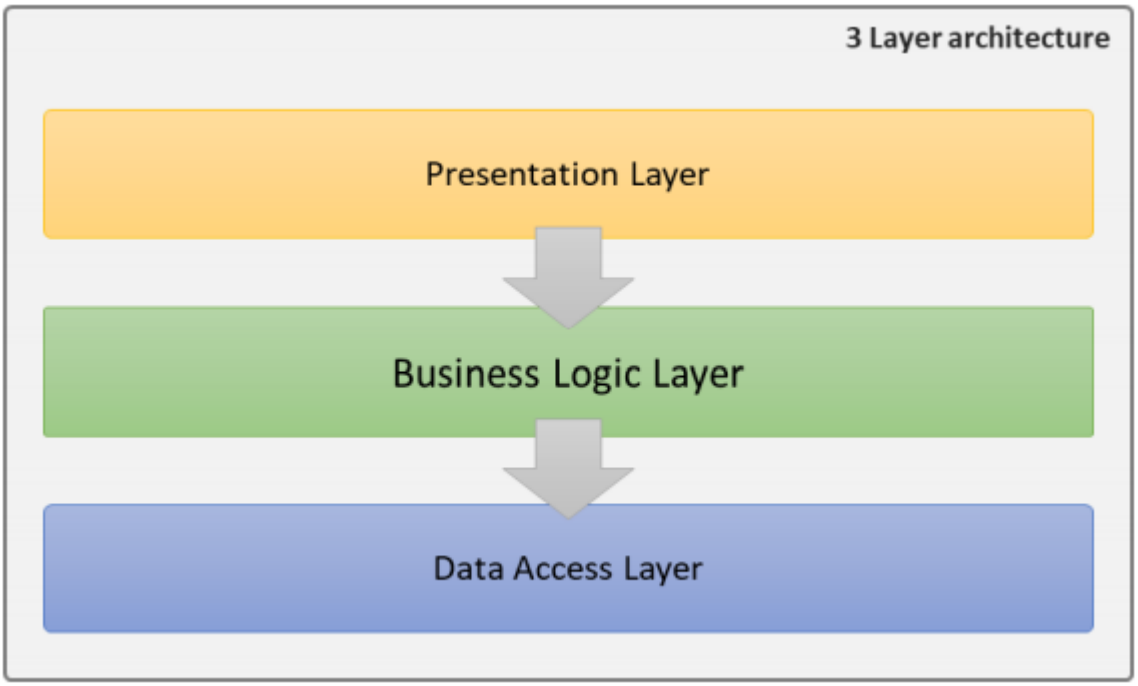

Imagen 13 – Arquitectura de 3 capas

1. Capa de presentación: Es la encargada de crear la vista o interface gráfica de usuario, puede ser una aplicación web, una app de escritorio o una app móvil. Generalmente escrito en HTML, Javascript, CSS, Android/Swift, etc.

2. Capa de lógica de negocio: Esta capa contiene la lógica de negocio y las operaciones de alto nivel que la capa de presentación puede utilizar. Generalmente escrito en Java, Python, .NET, NodeJS, etc.

3. Capa de acceso a datos: Esta capa corresponde a la capa donde están los datos, por ejemplo, MySQL, Oracle, PostgresSQL, MongoDB, etc.

En una arquitectura en capas, es necesario que la ejecución inicie de las capas superiores a las inferiores, de tal forma que el usuario siempre tendrá que ejecutar la aplicación por la capa de presentación, la cual deberá hacer llamadas a la capa de negocio para recuperar o actualizar los datos, la cual, a su vez, tendrá que comunicarse a la capa de datos. Como regla general, en una arquitectura de 3 capas, no se utiliza el concepto de capa abierta, ya que no tiene sentido que la vista se comunique directamente con la capa de datos.

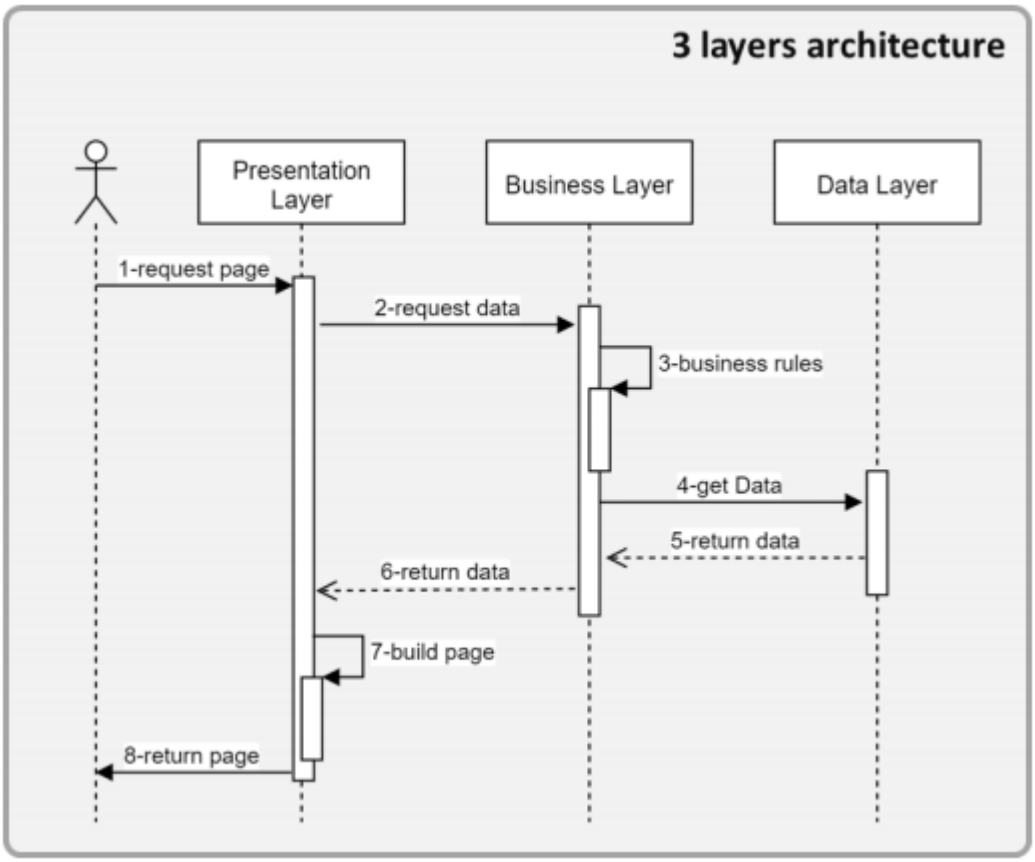

Imagen 14 – Diagrama de secuencia de una arquitectura de 3 capas

El diagrama anterior ilustra mejor como se da la ejecución en una arquitectura de 3 capas, el cual simula consulta de una nueva página:

1. Un usuario solicita a la aplicación una nueva página

2. La capa de presentación determina cual es la página que solicita el usuario y los datos que requiere para renderizar la página, por lo que solicita a la capa de negocios los datos.

3. La capa de negocio realiza algunas reglas de negocio, como validar el usuario, o filtrar la información según el usuario.

4. La capa de negocio realiza la consulta a la capa de datos.

5. La base de datos retorna los datos solicitados por la capa de negocio.

6. La capa de negocio convierte los datos a un formato amigable y la regresa a la capa de presentación.

7. La capa de presentación recibe los datos y construye la vista a partir de estos.

8. La capa de presentación retorna la nueva página al cliente.

CARACTERÍSTICAS DE UNA ARQUITECTURA EN CAPAS

En esta sección analizaremos las características que distinguen a una arquitectura en capas del resto. Las características no son ventajas o desventajas, si no que más bien, nos sirven para identificar cuando una aplicación sigue un determinado estilo arquitectónico.

Las características se pueden convertir en ventajas o desventajas solo cuando analizamos la problemática que vamos a resolver, mientras tanto, son solo características:

• La aplicación se compone de capas, en la cual cada capa tiene una sola responsabilidad dentro de la aplicación.

• Las capas de la aplicación son totalmente independientes de las demás, con la excepción de la capa que está inmediatamente debajo.

• Deben de existir al menos 3 capas para considerar una aplicación por capas.

• Toda la comunicación se hace siempre de forma descendente, pasando desde las capas superiores a las más inferiores.

VENTAJAS

• Separación de responsabilidades: Permite la separación de preocupaciones (SoC), ya que cada capa tiene una sola responsabilidad.

• Fácil de desarrollar: Este estilo arquitectónico es especialmente fácil de implementar, además de que es muy conocido y una gran mayoría de las aplicaciones la utilizan.

• Fácil de probar: Debido a que la aplicación construida por capas, es posible ir probando de forma individual cada capa, lo que permite probar por separada cada capa.

• Fácil de mantener: Debido a que cada capa hace una tarea muy específica, es fácil detectar el origen de un bug para corregirlo, o simplemente se puede identificar donde se debe aplicar un cambio.

• Seguridad: La separación de capas permite el aislamiento de los servidores en subredes diferentes, lo que hace más difícil realizar ataques.

DESVENTAJAS

• Performance: La comunicación por la red o internet es una de las tareas más tardadas de un sistema, incluso, en muchas ocasiones más tardado que el mismo procesamiento de los datos, por lo que el hecho de tener que comunicarnos de capa en capa genera un degrado significativo en el performance.

• Escalabilidad: Las aplicaciones que implementan este patrón por lo general tienda a ser monolíticas, lo que hace que sean difíciles de escalar, aunque es posible replicar la aplicación completa en varios nodos, lo que provoca un escalado monolítico.

• Complejidad de despliegue: En este tipo de arquitecturas es necesarios desplegar los componentes de abajo arriba, lo que crea una dependencia en el despliegue, además, si la aplicación tiende a lo monolítico, un pequeño cambio puede requerir el despliegue completo de la aplicación.

• Anclado a un Stack tecnológico: Si bien no es la regla, la realidad es que por lo general todas las capas de la aplicación son construidas con la misma tecnología, lo que hace amarremos la comunicación con tecnologías propietarias de la plataforma utilizada.

• Tolerancia a los fallos: Si una capa falla, todas las capas superiores comienzan a fallar en cascada.

CUANDO DEBO DE UTILIZAR UNA ARQUITECTURA EN CAPAS

La arquitectura por capas es una de las arquitecturas más versátiles, incluso es considerada por mucho como el estilo arquitectónico por defecto, pues es tan versátil que se puede aplicar a muchos de las aplicaciones que construimos día a día. Es común ver que los arquitectos utilizan esta arquitectura cuando no están seguros de cual utilizar, lo que la hace una de las arquitecturas más comunes.

ENTORNO EMPRESARIAL

Sin lugar a duda, la arquitectura en capas es muy utilizada en aplicaciones empresariales, donde desarrollamos los típicos sistemas de ventas, inventario, pedidos o incluso sitios web empresariales que tienden a ser aplicaciones monolíticas, pues permite crear aplicaciones con cierto orden que se adapta muy bien a este tipo de empresas muy ordenadas, donde todo tiene que tener un orden muy definido y justificado.

En este tipo de entorno hay poca innovación y mucha burocracia, lo que impide proponer arquitecturas mucho más elaboradas que se salga de los esquemas típicos, además, no requiere de mucho mantenimiento o perfiles muy especializados, lo que favorece la proliferación de este tipo de arquitecturas. Por otra parte, las aplicaciones empresariales son por lo general de uso interno, lo que implica que no tiene una proyección de crecimiento exponencial, lo que ayuda a compensar el hecho de que este estilo arquitectónico tiene un escalamiento limitado.

APLICACIONES WEB

Las aplicaciones por lo general siguen el patrón arquitectónico conocido como MVC (Modelo-Vista-Controlador), el cual consiste en separar la aplicación en tres partes, la vista (páginas web), los datos (Modelo) y el controlador (lógica de negocio) en partes separadas, lo cual se adapta muy bien con la arquitectura en capas. En común ver que las aplicaciones de hoy en día se desarrollan en capas, donde por una parte se construye la interface gráfica con tecnologías 100% del FrontEnd como Angular, React o Vue, la cual corresponde a la capa de la vista, luego, llaman a un API REST el cual contiene la capa de negocio y persistencia y finalmente tenemos la base de datos, haciendo que nuestra aplicación tenga 3 o 4 capas, según como se construya la aplicación

ARQUITECTURA DE USO GENERAL

Como mencionamos al inicio, esta arquitectura es una buena opción cuando no estamos muy seguros que de arquitectura utilizar, incluso, es una buena arquitectura para arquitectos principiantes, pues tiene una estructura muy clara y que no tiene muchos problemas de implementación. Sin embargo, es importante analizar las desventajas y características de esta arquitectura para no implementarla en una aplicación para la cual no dará el ancho, al final, es nuestra responsabilidad como arquitectos que el sistema funcione y garantizar su funcionamiento durante un tiempo considerable.

EJERCICIO

1. Como podría realizar un escalamiento vertical en una arquitectura por capas y monolítica
2. Como podría realizar un escalamiento horizontal en una arquitectura por capas y monolítica.
3. Como podría implementar un load balancer, estructure un diseño que permita la implementación en el sistema propuesto.

CASO DE ESTUDIO

Realizar el caso de estudio anexo donde se evidencie del diagrama de componentes y diagrama de despliegue creado para solucionar la problemática planteada en una arquitectura por capas y que permita gestionar el proceso de negocio. Adicional cree la estructura del proyecto en github.

Suponga las indicaciones dadas por el instructor.

4. ACTIVIDADES DE EVALUACIÓN

Evidencia de Conocimiento: Realizar una infografía digital sobre la arquitectura por capas, realice una presentación sobre las 3 preguntas y el caso de estudio.

Evidencia de Desempeño: Realiza la sustentación de cada evidencia de conocimiento.

Evidencia de Producto: Entrega de los documentos requeridos en las evidencias de conocimiento.

Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Evidencias de Conocimiento:	**Reconoce las principales características de la arquitectura por capas**	Redacción
Evidencias de Desempeño:	**Interpreta el uso de una aplicación arquitectura por capas en un entorno empresarial**	Presentación
Evidencias de Producto	**Realiza el entregable con la documentación completa**	Entrega de la guía con todas las evidencias requeridas.

1. **GLOSARIO DE TÉRMINOS**
   De acuerdo a la práctica realizar su propio glosario de términos.

6. REFERENTES BIBLIOGRÁFICOS

Construya o cite documentos de apoyo para el desarrollo de la guía, según lo establecido en la guía de desarrollo curricular

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	**Néstor Rodríguez**	**Instructor**	**Teleinformática**	**NOVIEMBRE-2020**

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					