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

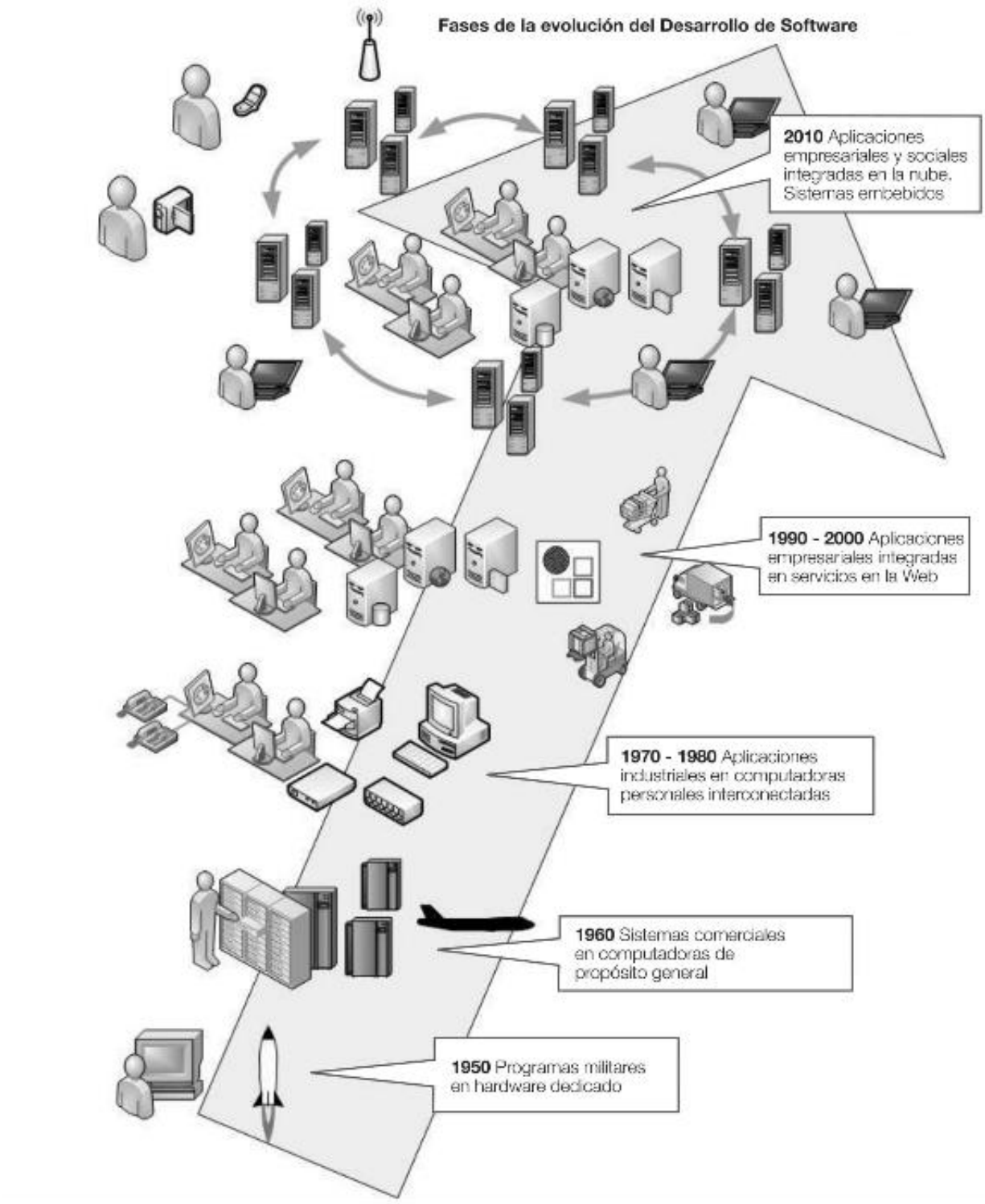

Imagen 1 – Evolución del desarrollo de software

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.

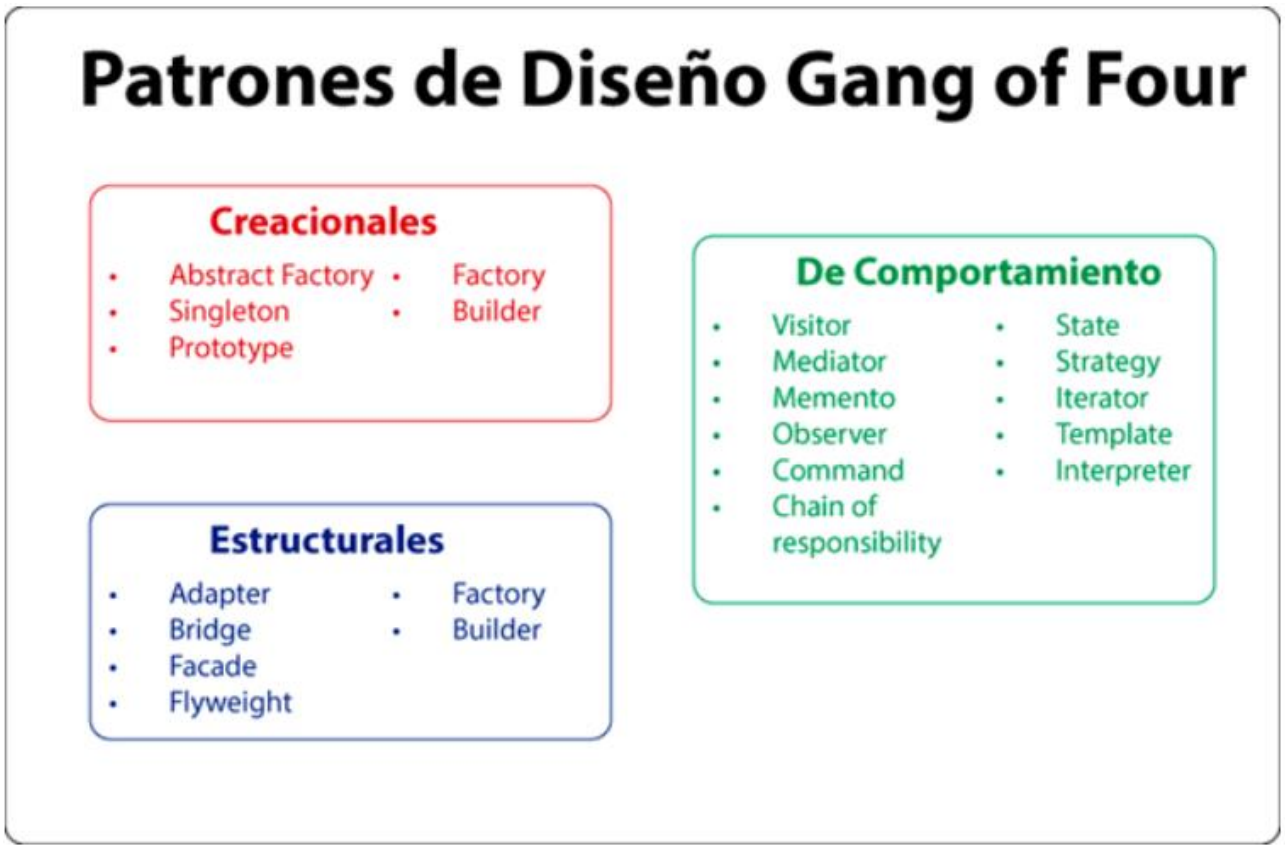

Imagen 2 – Lista de los patrones de diseño

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.

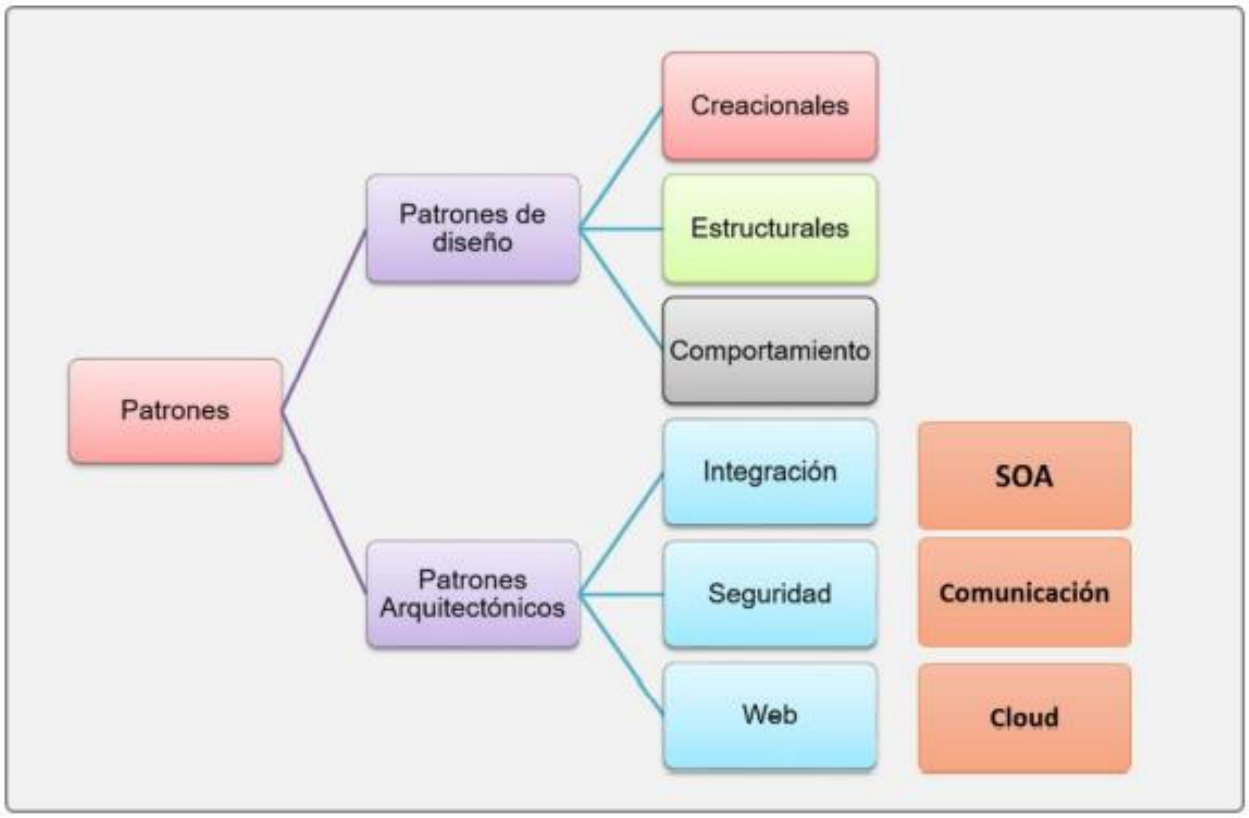

Imagen 3 – Consolidado de patrones

¿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.

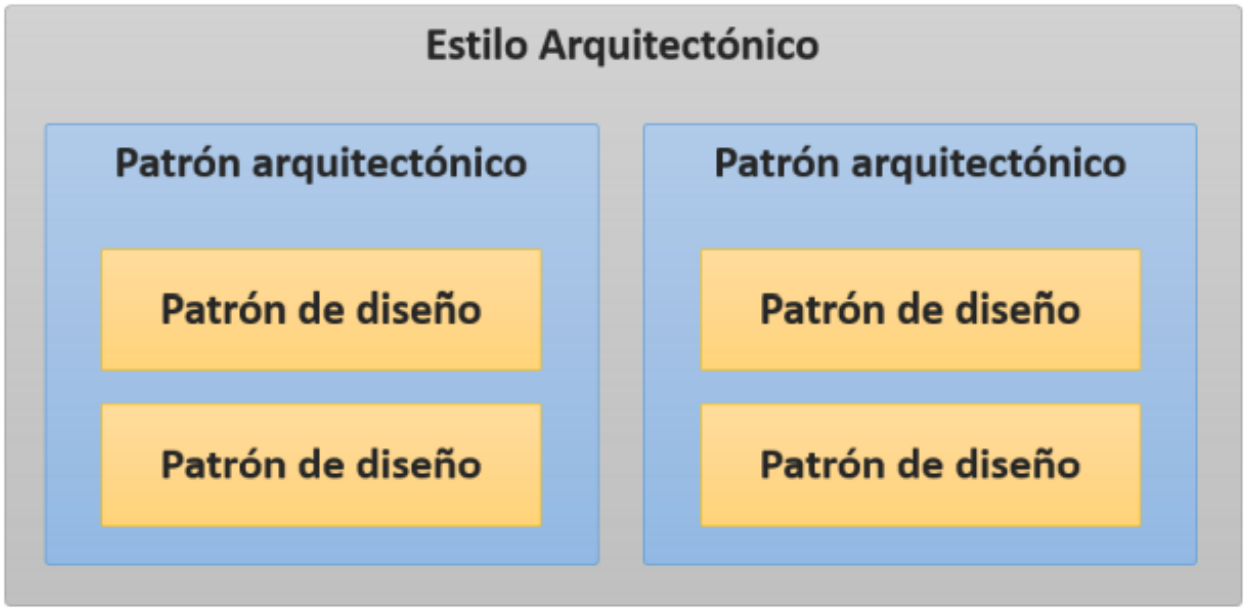

Imagen 5 – Relación entre estilo arquitectónico, patrón arquitectónico y patrón de diseño

ESTILOS ARQUITETONICOS

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.

Monolítico

El estilo arquitectónico monolítico consiste en crear una aplicación autosuficiente que contenga absolutamente toda la funcionalidad necesaria para realizar la tarea para la cual fue diseñada, sin contar con dependencias externas que complementen su funcionalidad. En este sentido, sus componentes trabajan juntos, compartiendo los mismos recursos y memoria. En pocas palabras, una aplicación monolítica es una unidad cohesiva de código.

Un monolítico podrías estar construido como una sola unidad de software o creada a partir de varios módulo o librerías, pero lo que la distingue es que al momento de compilarse se empaqueta como una solo pieza, de tal forma que todos los módulos y librerías se empaquetarán junto con la aplicación principal.

El estilo monolítico no es algo que haya sido planeado o ideado por alguien en particular, si no que todas las aplicaciones al inicio de la computación nacían con este estilo arquitectónico. Solo hace falta recordar los sistemas antiguos, donde todo funcionaba en una súper computadora, la cual realizaba todas las tareas. Recordemos que al inicio no existían el internet, por lo que no había forma de consumir servicios externos para realizar determinadas tareas, en su lugar, el sistema

monolítico tenía que implementar absolutamente toda la funcionalidad necesaria para funcionar, y de esta forma ser auto suficiente.

Con el tiempo, llego el internet y con ello la posibilidad de consumir servicios externos, llegaron arquitecturas modulares que permitían separar el código en unidades de software más manejables, cohesivas y fácil de administrar, sin embargo, con todos estos avances, siguen existiendo las aplicaciones Monolíticas, las cuales son vistas por los inexpertos como algo malo o incluso como un Anti patrón, pero la realidad es que esto está muy alejado de la realidad.

A pesar de todo el estigma que tienen las aplicaciones monolíticas, la realidad es que, hasta el día de hoy, las aplicaciones monolíticas siguen teniendo un protagonismo muy importante y siguen existiendo caso donde son totalmente indispensables para mantener la operación de las empresas.

Puede parecer un poco tonto el solo hecho de pensar en hacer una aplicación monolítica hoy en día, sin embargo, todavía hay escenarios donde son totalmente necesaria, solo imagina el sistema venta y facturación de una pequeña empresa, el software de los equipos médicos, programas de escritorio, como procesadores de texto o incluso sistemas más completos como los clásicos CRM o ERP.

Todos estos sistemas muchas veces funcionan de forma independiente, sin acceso a internet y necesitan una autonomía total, solo imagina que un equipo médico no funcione si no se puede conectar a internet o que necesite de servicios externos para operar, eso podría costar vidas, o que el cajero de una tienda no pueda vender o administrar su inventario porque una dependencia no está disponible, eso podría costar la perdida de ventas. Lo cierto es que las aplicaciones monolíticas son cada vez menos atractivas, pero hasta el día de hoy, tiene aplicaciones donde difícilmente podrán ser remplazadas.

Una falsa creencia es que, una aplicación monolítica es un caos por dentro, donde todo el código está amontonado, no hay una estructura clara y que por lo general tiene miles de clases u objetos, sin embargo, esto es solo una mala fama que se le ha dado, si bien es verdad que se podía dar el caso, recordemos que eso también se podría dar en cualquier estilo de arquitectura, pues eso dependen más bien del programador y no del estilo arquitectónico.

Otra falsa creencia es creer que las aplicaciones Monolíticas son solo las aplicaciones grandísimas que hacen un montón de cosas, pero lo cierto es que un monolítico puede ser de una sola clase, o de miles, lo que define un estilo monolítico no es el número de clases, archivos o líneas de código, lo que lo define es que es autosuficiente, es decir, que tiene toda la funcionalidad para operar por sí mismo y sin depender de nadie más.

Como se estructura un Monolítico

En los Monolítico podemos tener una serie de paquetes bien organizados y un código muy claro, donde cada paquete puede tener cierta parte de la funcionalidad y están desacoplados uno de otro (recordemos que eso es independiente del estilo arquitectónico). Sin embargo, al momento de

compilarse el código, todo se empaqueta como un solo software. Cabe mencionar que cualquier librería que sea requerida, será exportada como parte del archivo compilado de salida.

Algo a tomar en cuenta es que cuando los módulos son compilados por separado y son instalados como complementos ya no se trata de un Monolítico y pasar a ser un estilo arquitectónico de Microkernel.

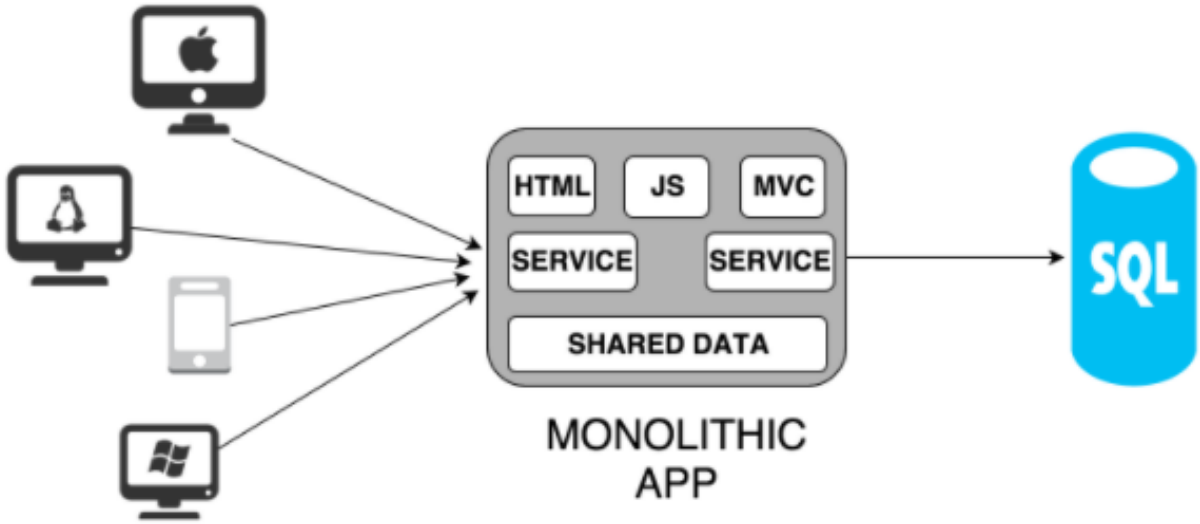

Imagen 6- Aplicación monolítica

Características de un Monolítico

Las características que distinguen al estilo Monolítico del resto pueden ser ventajas o desventajas, ya que, 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:

1. Son aplicaciones autosuficientes (no requieren de nada para funcionar).
2. Realizan de punta a punta todas las operaciones para terminar una tarea.
3. Son por lo general aplicaciones grandes, un que no es un requisito.
4. Son por lo general silos de datos privados, es decir, cada instalación administra si propia base de datos.
5. Todo el sistema corre sobre una solo plataforma.

Ventajas

• Fácil de desarrollar: Debido a que solo existe un componente, es muy fácil para un equipo pequeño de desarrollo iniciar un nuevo proyecto y ponerlo en producción rápidamente.

• Fácil de escalar: Solo es necesario instalar la aplicación en varios servidores y ponerlo detrás de un balanceador de cargar.

• Pocos puntos de fallo: El hecho de no depender de nadie más, mitiga gran parte de los errores de comunicación, red, integraciones, etc. Prácticamente los errores que pueden salir son por algún bug del programador, pero no por factores ajenos.

• Autónomo: Las aplicaciones Monolíticas se caracterizan por ser totalmente autónomas (auto suficientes), lo que les permite funcionar independientemente del resto de aplicaciones.

• Performance: Las aplicaciones Monolíticas son significativamente más rápidas debido que todo el procesamiento lo realizan localmente y no requieren consumir procesos distribuidos para completar una tarea.

• Fácil de probar: Debido a que es una sola unidad de código, toda la funcionalidad está disponible desde el inicio de la aplicación, por lo que es posible realizar todas las pruebas necesarias sin depender de nada más.

Desventajas

• Anclado a un Stack tecnológico: Debido a que todo el software es una sola pieza, implica que utilicemos el mismo Stack tecnológico para absolutamente todo, lo que impide que aprovechemos todas las tecnologías disponibles.

• Escalado Monolítico: Escalar una aplicación Monolítica implica escalar absolutamente toda la aplicación, gastando recursos para funcionalidad que quizás no necesita ser escalada (en el estilo de Microservicios analizaremos como solucionar esto).

• El tamaño sí importa: sin albur, las aplicaciones Monolíticas son fáciles de operar con equipo pequeños, pero a medida que la aplicación crece y con ello el equipo de desarrollo, se vuelve cada vez más complicado dividir el trabajo sin afectar funcionalidad que otro miembro del equipo también está moviendo.

• Versión tras versión: Cualquier mínimo cambio en la aplicación implicará realizar una compilación del todo el artefacto y con ello una nueva versión que tendrá que ser administrada.

• Si falla, falla todo: A menos que tengamos alta disponibilidad, si la aplicación Monolítica falla, falla todo el sistema, quedando totalmente inoperable.

• Es fácil perder el rumbo: Por la naturaleza de tener todo en un mismo módulo es fácil caer en malas prácticas de programación, separación de responsabilidades y organización de las clases del código.

• Puede ser abrumador: En proyectos muy grandes, puede ser abrumador para un nuevo programador hacer un cambio en el sistema.

EJERCICIO

1. Como realizo escalamiento vertical de una aplicación monolítica
2. Como realizo escalamiento horizontal de una aplicación monolítica
3. Como realizo un load balancer en una aplicación monolítica.
4. Como puedo escalar una aplicación monolítica basándose en el libro “Art of Scalability” de Michael Fisher

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 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 monolítica, realice una presentación sobre las 4 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 monolítica**	Redacción
Evidencias de Desempeño:	**Interpreta el uso de una aplicación monolítica 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)					