**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

| PROGRAMA DE FORMACIÓN | Tecnólogo en aseguramiento de la calidad de software |
| --- | --- |

| COMPETENCIA | 220501095 - Diseñar la solución de software de acuerdo con procedimientos y requisitos técnicos | RESULTADOS DE APRENDIZAJE | 220501095-01  Describir la funcionalidad de los componentes. |
| --- | --- | --- | --- |

| NÚMERO DEL COMPONENTE FORMATIVO | 06 |
| --- | --- |
| NOMBRE DEL COMPONENTE FORMATIVO | Patrones de diseño de *software* |
| BREVE DESCRIPCIÓN | Los patrones de diseño de *software* surgen a partir de experiencias desarrolladas en la industria y representan un conjunto de formas estandarizadas, probadas y repetibles, que permiten resolver uno o varios problemas particulares que se presentan en el diseño del *software*. |
| PALABRAS CLAVE | Arquitectura de *software*, GOF, patrones de diseño |

| ÁREA OCUPACIONAL | 2 - Ciencias naturales, aplicadas y relacionadas |
| --- | --- |
| IDIOMA | Español |

1. **Tabla de contenidos**

**Introducción**

1. **Conceptos GOF**
2. **Patrones de arquitectura**
   1. Patrón multicapa
   2. Patrón Modelo Vista Controlador
   3. Arquitectura monolítica
   4. Microservicios
   5. Diseño de la arquitectura
   6. Estilos arquitectónicos
3. **Patrones comportamentales**
   1. Estrategia
   2. Comando
   3. *Iterator*
4. **Patrones creacionales**
   1. *Singleton*
   2. Fábrica abstracta
5. **Patrones estructurales**
   1. Fachada
   2. *Delegate*
6. **Vistas estáticas**
7. **Diagrama de despliegue**
8. **Diagrama de componentes**
9. **Desarrollo de contenidos**

**Introducción**

Le damos la bienvenida al componente formativo “Patrones de diseño de *software*”. Recordemos que los patrones de diseño surgen como un concepto, inicialmente, para el área de arquitectura e ingeniería civil, pues era en estas disciplinas donde se presentaban casos particulares que requerían de soluciones a problemáticas particulares.

El concepto de patrón de diseño se empezó a asociar al diseño orientado a objetos y, en lugar de referirse a la forma de colocar paredes, puertas o ventanas, se refería a la forma en que se construyen clases, objetos, interfaces, y a la forma en cómo estos deben interactuar

.



A continuación, se abordan varios de los patrones de diseño más representativos en la industria del software, los cuales nos permitirán, a futuro, implementar soluciones más robustas, de acuerdo con altos estándares de calidad y recomendaciones ampliamente conocidas en la industria, lo que adicionalmente nos dará un plus como desarrolladores de software en una industria cada vez más competitiva globalmente.

El uso de patrones de diseño permite acelerar el proceso de desarrollo, al proporcionar paradigmas probados y comprobados en el diseño de software. Estandarizar las soluciones a problemas comunes permite tambien mejorar la comunicación entre desarrolladores, ya que se emplean nombres bien conocidos y entendidos a la hora de describir un problema de diseño y cómo abordar su solución.

1. **Conceptos GOF**

| Vector gratuito divertidos hombres de negocios | Los patrones GoF se presentan como una forma indispensable de abordar la programación y surgen debido a: **E**r**ich Gamma, Richard Helm, Ralph Jonson y John Vlissides**, quienes tratan el tema en su libro “*Design Patterns—Elements of Reusable Software*”; por esta razón, a estos patrones se les conoce con el nombre de **la pandilla de los cuatro** (*GoF, gang of four*). Según estos autores, los patrones de diseño se basan principalmente en dos principios de diseño orientado a objetos:   * Programa una interfaz, no una implementación. * Favorece la composición de objetos sobre la herencia. |
| --- | --- |

Para EcuRed (s. f.), los patrones de diseño tienen las siguientes características:



Los patrones que describen GOF suelen clasificarse en tres grandes categorías según su finalidad: **patrones comportamentales, patrones creacionales y patrones estructurales**. Una breve explicación de estos podrá consultarse a continuación, pues en el transcurso de este componente serán explicados con mayor detalle:



1. **Patrones de arquitectura**

| Vector gratuito fondo de construcción hecha de piezas de plástico | Para hablar de patrones, primero es necesario hablar de [arquitectura de *software*](https://apiumhub.com/es/arquitectura-software-barcelona/)*.* Esta consiste en una disciplina dentro del desarrollo de *software*. Como ya se sabe, este desarrollo es algo intangible y puede ser tan complejo como las necesidades que se quieren resolver, por lo tanto, es importante tener en cuenta que, cuando se piensa en la escalabilidad del *software* y su usabilidad, utilizando los recursos disponibles y sacándole el máximo provecho a esta arquitectura, posibilitando un ciclo de vida mucho más largo, se debe pensar que una aplicación de *software* es una especie de lego, donde todas sus piezas se encajan, para poder cumplir con diversas funciones. |
| --- | --- |

Dentro de estos patrones de arquitectura, se encuentran diferentes tipos, que permiten generar *software* con mayor rendimiento. Algunos de estos son:

* 1. **Patrón multicapa**

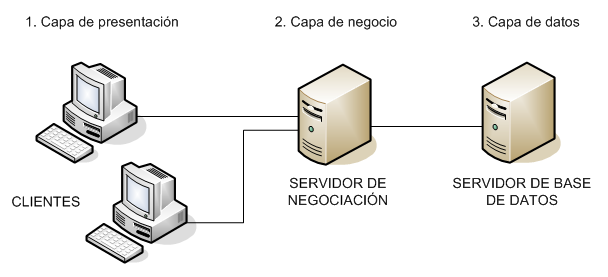
Son patrones de X niveles y se organizan en capas horizontales. Muchas de las aplicaciones informáticas han tomado este patrón como arquitectura, funciona conectando los componentes, pero estos no dependen uno del otro. Cada una de estas capas cumple con una función específica en el desarrollo de la aplicación. Su objetivo primordial es la separación de la lógica de negocios de la lógica de diseño, es decir, separar la capa de datos de la capa de presentación al usuario.

La ventaja que ofrece consiste en que el desarrollo se puede llevar a cabo en varios niveles. En caso de realizar algún cambio, sólo se afecta el nivel requerido, sin tener que revisar todo el código.

| Ejemplo  El modelo de interconexión de sistemas abiertos, donde la capa de presentación se encarga de todo el diseño y desarrollo visual de parte del usuario, mientras que la capa de negocio se encarga de ejecutar las reglas y la lógica del negocio. Cada capa tiene funciones específicas, una capa informa cómo debe comportarse la venta de un producto, teniendo en cuenta la lógica y reglas establecidas para registrar esa venta, de esta forma, el patrón permite delegar funcionalidades propias de cada capa. |
| --- |

En la siguiente figura, se puede observar la arquitectura en tres capas. Fue creada en *Microsoft Visio* 2003, para ayudar a entender gráficamente y de una forma más fácil este tema.

**Figura 1**



Este patrón puede dividirse entre capas tantas veces se decida según su arquitectura, solo se debe tener en cuenta que, aunque es una buena forma de desarrollar, si existen muchas capas, también podría ser contraproducente, así que depende del juicio del arquitecto de *software* la distribución de capas que contemple.

**Ventajas**

* Facilidad al momento de realizar pruebas.
* Facilidad para el mantenimiento de la aplicación.
* Se puede desarrollar por separado, delegando capas por desarrollador.
* Permite actualizar y mejorar la aplicación por separado.
  1. **Patrón Modelo Vista Controlador**

Este modelo, también identificado como **MVC,** es una arquitectura que también se separa por capas, solo que en este caso son tres capas las que se dedican o son responsables de delegar funcionalidades específicas, como se puede observar a continuación:



Es así como se pueden observar algunas de las responsabilidades de cada capa, para lo que hay que seguir unos pasos específicos al momento de iniciar sesión en una aplicación, observe:



* 1. **Arquitectura monolítica**

La arquitectura monolítica es aquella en la que el software está estructurado de forma tal que todos los aspectos funcionales del mismo quedan acoplados y sujetos en un mismo programa. En esta arquitectura, cada proceso o microservicio es un elemento independiente. Es utilizada en aplicaciones en las que todo se desarrolla en una misma parte, aunque también se utilizan módulos para desarrollar.

**Figura 2**

*Monolith*

|  |  |
| --- | --- |

En el momento de compilarse, lo hace de forma completa, como una sola, y se comprende en una interfaz del lado del cliente, una aplicación del lado del servidor y una base de datos. Al igual que las otras arquitecturas, tiene sus ventajas y desventajas:

**Tabla 1**

*Ventajas y desventajas de la arquitectura monolítica*

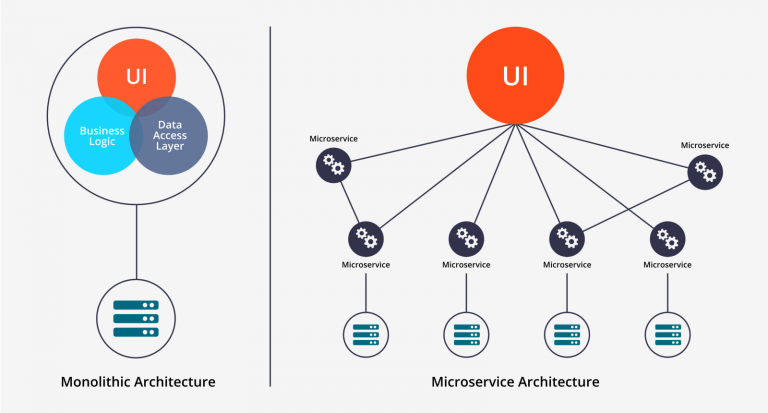
| **Ventajas** | **Desventajas** |
| --- | --- |
| Fácil de probar y *debuggear.* | Alto acoplamiento: si se hace un cambio en alguna parte del código, puede cambiar el comportamiento en otra parte del mismo. |
| Fácil de desplegar. | Código muy grande. |
| Fácil de desarrollar. | Costos altos al momento de su escalabilidad. |
|  | Difícil de mantener. |
|  | Dificultad en las mejoras y actualizaciones. |

* 1. **Microservicios**

Esta arquitectura se enfoca en crear pequeños programas API, que permiten que los servicios se comuniquen, posibilitando que las aplicaciones con estas arquitecturas puedan evolucionar y adaptarse muchísimo mejor. Además, se pueden incluir servicios de otras plataformas u otras aplicaciones, ya que pueden ser consumidos para la manipulación de la información.

**Figura 3**

*Microservicios*



*Nota.* Imagen tomada de Hiberus. (2021)

Dentro de sus características está que cada servicio que se desarrolla se puede hacer de forma independiente, sin que afecte el funcionamiento del resto de la aplicación; además, se crean para solucionar problemas específicos, haciéndolos especializados, y, en caso de que un servicio de estos se vuelva muy grande, puede seguirse dividiendo en más microservicios.

Entre los beneficios de los microservicios, se tienen:



* 1. **Diseño de la arquitectura.**

Una arquitectura de *software* describe los componentes básicos de un sistema de *software* y su combinación interna. En el marco del desarrollo de software, esta arquitectura es la decisión más temprana del diseño de una aplicación. Determina los criterios de calidad, como la mantenibilidad, modificabilidad, seguridad y rendimiento, además, le permite analizar de forma más clara cómo debe comportarse la aplicación que se va a desarrollar, teniendo en cuenta el acceso a ella y su capacidad.

**Figura 4**

Diseño de la arquitectura.



* 1. **Estilos arquitectónicos**

Los estilos son un complemento de los patrones de diseño, como los arquitectónicos de *software,* y lo importante aquí es que se preocupan por los *stakeholders*. Por lo tanto, los estilos se preocupan por el tratamiento estructural del *software*.

Cuando se aplican estilos arquitectónicos en el desarrollo de *software*, se le da un valor distintivo a la calidad del *software*. Los estilos son un conjunto de componentes, como bases de datos o módulos de cómputo, que son utilizados para realizar una función y, por medio de conectores, permiten la comunicación, coordinación y cooperación entre ellos, **“el todo es la suma de sus partes”.**



Hay diferentes tipos de estilos arquitectónicos, algunos de los más conocidos son:



1. **Patrones comportamentales**

Como se había mencionado en los conceptos GOF, los patrones de diseño comportamentales se centran en definir la forma en cómo los objetos interactúan entre ellos por medio de mensajes.

****

* 1. **Estrategia**

El patrón estrategia permite encapsular un conjunto de algoritmos de forma que puedan ser seleccionados dinámicamente durante el tiempo de ejecución de acuerdo con las acciones del cliente. Este patrón es una de las formas en las que se ven reflejadas fácilmente las características de la programación orientada a objetos, particularmente, lo referente a encapsulamiento y polimorfismo (Landa, 2018c).

**Figura 5**

*Ejemplo de Patrón estrategia*

|  |  |
| --- | --- |

Este patrón de diseño es útil cuando una misma funcionalidad puede ser provista usando diferentes mecanismos, algoritmos o estrategias, que serán seleccionadas dependiendo de las acciones realizadas por el cliente en el momento que está ejecutando el programa.

Para un mejor entendimiento del patrón, suponga que quiere implementar una calculadora, la cual provee un conjunto de operaciones (suma, resta, multiplicación y división) que serán usadas por el cliente según su deseo. Cada una de estas operaciones representa una estrategia diferente y será el cliente quien invocará la ejecución de cada una de ellas dependiendo de su deseo, por medio de una interfaz que, usando las propiedades del polimorfismo, se transformará para poder responder a cada solicitud.

| Ejemplo  Consideremos el ejemplo de implementar una calculadora que ofrece un conjunto de operaciones (suma, resta, multiplicación y división) para que el cliente las utilice según necesite. Cada una de estas operaciones constituye una estrategia distinta. Será el cliente quien decida cuál de ellas ejecutar, basándose en sus requerimientos, a través de una interfaz. Esta interfaz, aprovechando las propiedades del polimorfismo, se adapta para responder adecuadamente a cada tipo de solicitud. |
| --- |

* 1. **Comando**

El patrón comando permite aislar los objetos que realizan una petición de los objetos concretos encargados de recibir y realizar dicha acción. Esto permite, entre otras cosas, que las peticiones puedan ser enviadas a varios receptores y, si se maneja el estado de las solicitudes, controla acciones de tipo *Undo* y *Redo*.

El patrón comando necesita la implementación de varios elementos (Landa, 2018c), así:



En la siguiente figura, se puede ver cada uno de los componentes del patrón y cómo interactúan entre ellos.

**Figura 6**

*Diagrama patrón comando*



| **Ejemplo**  Para entender mejor este patrón, imagine el siguiente contexto: una persona (cliente) quiere hacer uso del televisor y, para poder realizar esto, hace todas las solicitudes de servicios por medio del control remoto (*Invoker*). El control remoto se comunica con una interfaz que se encarga de responder a las solicitudes de cada uno de los comandos que el usuario puede hacer, como, por ejemplo, prender el televisor, apagar el televisor, subir el volumen, etc. Cada comando realiza una acción particular sobre el televisor (Receptor). |  |
| --- | --- |

* 1. ***Iterator***

Este patrón de diseño está orientado al trabajo con colecciones y facilita el acceso a todos los elementos de la colección sin tener la necesidad de conocer su estructura.

En este patrón, se reconocen dos elementos clave: los enumeradores y los *iteradores*:



El enumerador, se encarga de implementar un conjunto de métodos estándar para poder establecer la secuencia con la que se debe recorrer la estructura. Entre los métodos más comunes, se encuentra, por ejemplo, el método *moveNext*(), el cual indica si existe o no un próximo elemento por recorrer; el método *Current*(), que devuelve el valor actual de la colección según la posición actual en la secuencia; y el método *Reset*(), que permite iniciar nuevamente la secuencia desde su punto de partida. El *iterador* necesita del enumerador para poder hacer el proceso de recorrido (Landa, 2018c).

En la siguiente figura, se puede ver cada uno de los componentes del patrón y cómo interactúan entre ellos.

**Figura 7**

*Diagrama patrón iterador*



Este tipo de operaciones son tan comunes en los sistemas actuales que los lenguajes de programación ya poseen una implementación propia del patrón *iterador*.

1. **Patrones creacionales**

Una de las tareas habituales en el proceso de construcción de software es distribuir responsabilidades entre un conjunto de módulos o clases, siguiendo los principios establecidos por los paradigmas de programación, como la programación orientada a objetos. No obstante, al abordar un requerimiento específico, será necesario instanciar objetos de distintos tipos que, asumiendo sus responsabilidades, ejecutan cierta lógica.

****

* 1. ***Singleton***

Este patrón de diseño creacional se encarga de definir la forma en que se puede garantizar que exista una única instancia de una clase particular en el contexto de la aplicación. Esto es útil para casos en los que, por cuestiones de manejo de memoria o de la lógica del negocio, se requiere que sea el mismo objeto quien responda todos los mensajes, independientemente del contexto actual de la aplicación.

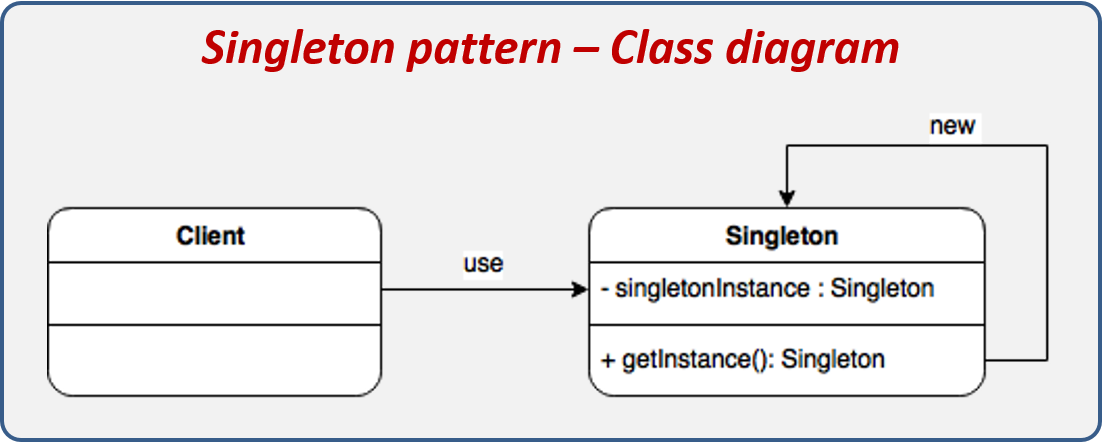
Este patrón se ve reflejado en el cuerpo de la misma clase que se requiere sea instancia una sola vez.

| **EJEMPLO**  Puede ser el manejo de conexiones a bases de datos. En algunos casos, no sería conveniente instanciar nuevos objetos de conexión cada vez que se genere un evento.  Vector gratuito ilustración de alojamiento de sitio web degradado |
| --- |

A continuación, se puede ver cada uno de los componentes del patrón relacionados entre el cliente y los requerimientos.

**Figura 8**

*Diagrama patrón Singleton*



* 1. **Fábrica abstracta**

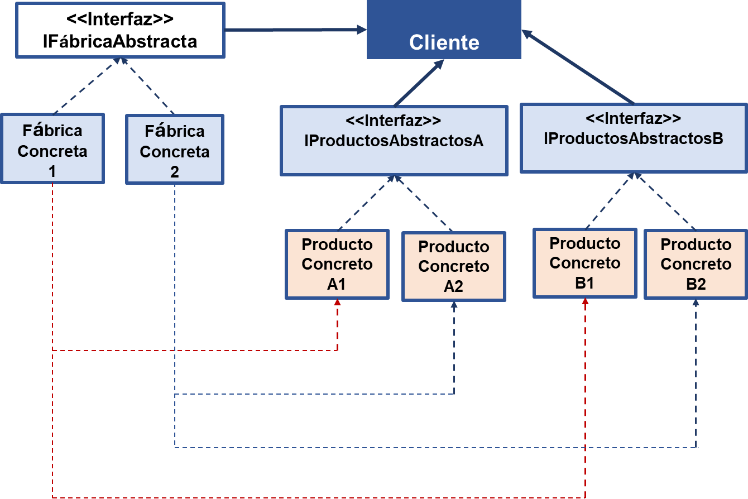
Es un patrón de diseño creacional que permite producir familias de objetos relacionados sin especificar sus clases concretas. En el siguiente recurso, podrá apreciar en detalle sus características.



En la siguiente figura se puede ver cada uno de los componentes.

**Figura 9**

Componentes de un patrón de fábrica abstracta

****

1. **Patrones estructurales**

Los patrones estructurales proveen una orientación relacionada con la forma de definir los componentes de los objetos.

* 1. **Fachada**

El patrón fachada se utiliza cuando el sistema está compuesto por varios subsistemas y se hace complejo gestionar los mensajes que debe realizar el cliente en cada uno de estos subsistemas. Este patrón permite generar al cliente una vista de alto nivel que simplifica el control y el envío de mensajes a los subsistemas, ocultando los detalles relacionados con la gestión de las clases e instancias, como se detalla en el siguiente recurso educativo.

Existen diferentes variaciones del patrón fachada (Landa, 2018c):

****

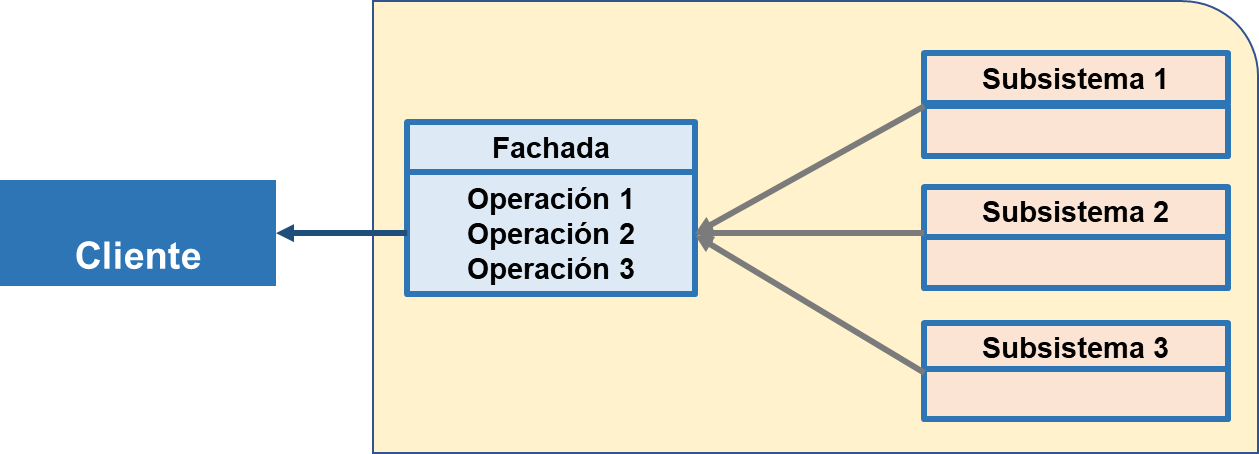
En este patrón se reconocen tres partes fundamentales (Landa, 2018c):



En la siguiente figura, se puede ver cada uno de los componentes del patrón.

**Figura 10**

Componentes del patrón fachada



* 1. ***Delegate***

| Vector gratuito ilustración del concepto de colaboración | El patrón *delegate* se usa cuando se quieren reutilizar y extender funcionalidades de una clase sin hacer uso de la herencia. Este patrón permite, de cierta forma, implementar algo similar a la herencia múltiple, que no es admitida por algunos lenguajes de programación, pero, adicionalmente, permite tener un control más detallado sobre este proceso, ya que se puede ocultar parte de los elementos heredados o, incluso, compartir elementos que no son posibles de heredar bajo el mecanismo de herencia tradicional. |
| --- | --- |

Este patrón lo que busca es evitar asumir todas las responsabilidades en una sola instancia y delegar las actividades en otras instancias que son especializadas en resolver dichas tareas.

1. **Vistas estáticas**

La vista estática está encargada de modelar los conceptos significativos del dominio de la aplicación desde sus propiedades internas y las relaciones existentes. Se denomina vista estática porque no modela el comportamiento del sistema ni muestra las variaciones que se puedan presentar por efecto del tiempo.

Los elementos fundamentales de la vista estática son las clases, que describen los conceptos del dominio del problema, y las relaciones, que pueden ser de tipo asociación, generalización y de dependencia. Entre los diagramas de UML que se utilizan para representar la vista estática del sistema, se encuentran (ITCA-FEPADE, s. f.):



A continuación, se detallan algunos diagramas de UML que permiten tener una visión más generalizada de los sistemas de información en desarrollo o desarrollados por el equipo de trabajo.

1. **Diagrama de despliegue**

Los diagramas de despliegue hacen parte de los tipos de diagrama propuestos por UML y su objetivo es la representación de la arquitectura del sistema en términos de *hardware* y *software* físico, y los medios por los cuales se conectan. Este tipo de diagrama es muy útil para el proceso de despliegue del sistema.

Los diagramas de despliegue utilizan un conjunto de elementos gráficos que tienen una representación y significado estandarizados. A continuación, se detalla cada uno de ellos (Cinergix Pty. Ltd., 2021):



1. **Diagrama de componentes**

El diagrama de componentes es uno de los diagramas propuestos en UML que representa una vista estática del sistema de información y hace parte de los diagramas estructurales. Este diagrama proporciona una vista de alto nivel de los componentes dentro del sistema y generalmente se construye posterior a la construcción del diagrama de clases.



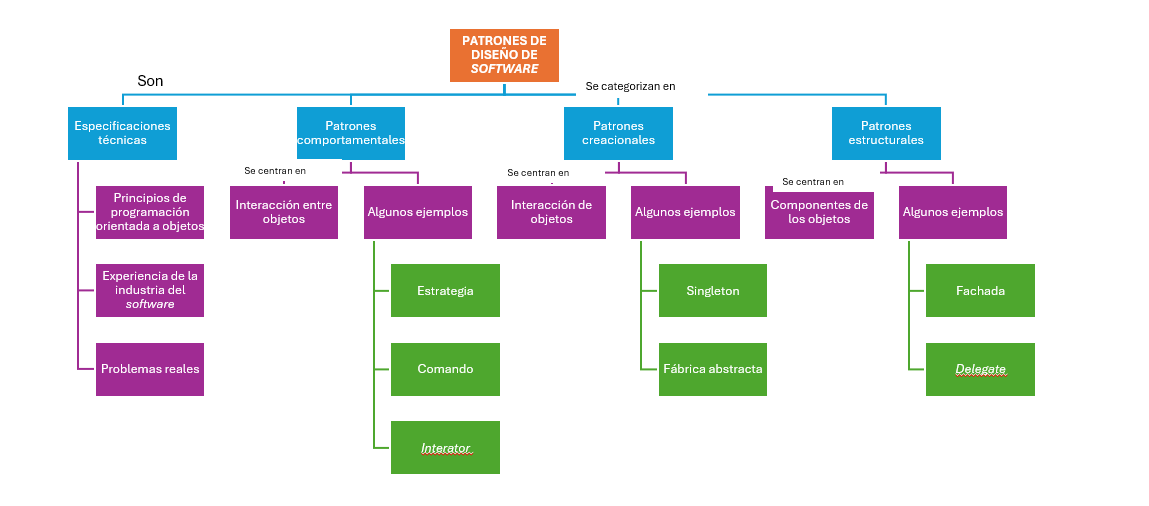
Este tipo de diagrama es muy útil para arquitecturas orientadas a servicios, permite mostrar la estructura general del código, por lo que puede ser usado para mostrar las funciones del sistema que se construye a cualquier parte interesada.

Los elementos que conforman un diagrama de componente son los siguientes (Diagramas UML, 2019):



1. **Síntesis**

A continuación, se muestra un mapa conceptual de los elementos más importantes desarrollados en este componente:



1. **Actividades didácticas (opcionales si son sugeridas)**

| **DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA** | |
| --- | --- |
| **Nombre de la Actividad** | **Características representativas en patrones de diseño.** |
| **Objetivo de la actividad** | **Afianzar las características más importantes de algunos patrones de diseño de *software*.** |
| **Tipo de actividad sugerida** | **Arrastrar y soltar el nombre del patrón de diseño con la característica que lo identifica.** |
| **Archivo de la actividad**  **(Anexo donde se describe la actividad propuesta)** | **CF01\_Actividad didactica** |

1. **Material complementario**

| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| --- | --- | --- | --- |
| Patrones de diseño | Leiva, A. [DevExperto]. (2020). *Patrones de diseño software: Repaso completo en 10 minutos* <https://www.youtube.com/watch?v=6BHOeDL8vls&feature=youtu.be> | Video YouTube | <https://youtu.be/6BHOeDL8vls> |
| Diagrama de despliegue | Landa, N. [nicosiored]. (2018b). *Diagrama de Despliegue - 22 - Tutorial UML en español*. <https://www.youtube.com/watch?v=NSB0ATJUavA&feature=youtu.be> | Video YouTube | <https://youtu.be/NSB0ATJUavA> |
| Diagrama de componentes | Landa, N. [nicosiored]. (2018a). *Diagrama de Componentes I - 20- Tutorial UML en español* <https://www.youtube.com/watch?v=oOycG_n1ARs&feature=youtu.be> | Video YouTube | <https://youtu.be/oOycG_n1ARs> |
| Arquitectura Multicapa | Garrido, A. [Universitat Politècnica de València – UPV]. (2017). *Arquitectura del software multicapa*.  <https://www.youtube.com/embed/kHvxX1E9vIU> | Video YouTube | <https://www.youtube.com/embed/kHvxX1E9vIU> |
| Patrón Modelo Vista Controlador | Fazt. (2017). *¿What is the MVC Pattern?, Simple explanation*.  <https://www.youtube.com/embed/ANQDmqBYwns> | Video YouTube | <https://www.youtube.com/embed/ANQDmqBYwns> |
| Arquitectura Monolítica | Lazy Loading. (2021). *Arquitectura monolítica vs microservicios*. <https://www.youtube.com/embed/99YMeCBk3jw> | Video YouTube | <https://www.youtube.com/embed/99YMeCBk3jw> |
| Estilos de Arquitectura | Mercury 7w7. (2020). *Estilos de Arquitectura de Software*.  <https://www.youtube.com/watch?v=PK9TTcTosTw> | Video YouTube | <https://www.youtube.com/watch?v=PK9TTcTosTw> |
| Arquitectura Microservicios | MegaPractical. (2016). *Arquitectura Orientada a Servicios (SOA), Enterprise Service Bus TIBCO en español*.  <https://www.youtube.com/embed/o_Br2vZ4uQY> | Video YouTube | <https://www.youtube.com/embed/o_Br2vZ4uQY> |

1. **Glosario**

| **TÉRMINO** | **SIGNIFICADO** |
| --- | --- |
| Acoplamiento: | es la forma y nivel de interdependencia entre módulos de **software**; una medida de qué tan cercanamente conectados están dos rutinas o módulos de **software.** |
| API: | es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el *software* de las aplicaciones. **API** significa interfaz de programación de aplicaciones. Las **API** permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. |
| Aplicación: | es un programa **informático** diseñado como una herramienta para realizar operaciones o funciones específicas. Generalmente, son diseñadas para facilitar ciertas tareas complejas y hacer más sencilla la experiencia **informática** de las personas. |
| Base de datos: | es una recopilación organizada de información o **datos** estructurados, que normalmente se almacena de forma electrónica en un sistema informático. |
| *Browser:* | es el término inglés que se utiliza para identificar a un navegador web o navegador de Internet. Consiste en un *software*, programa o incluso aplicación que ofrece al usuario el acceso a la red. |
| Delegar: | dar [una persona o un organismo] un poder, una función o una responsabilidad a alguien para que los ejerza en su lugar o para obrar en representación suya. |
| Escalabilidad: | capacidad de adaptación y respuesta de un sistema con respecto al rendimiento del mismo a medida que aumentan de forma significativa el número de usuarios del mismo. |
| Ícono: | signo que representa un objeto o una idea con los que guarda una relación de identidad o semejanza formal. |
| Implementación: | poner en funcionamiento o aplicar métodos, medidas, etc., para llevar algo a cabo. |
| Interfaz: | la conexión física y funcional que se establece entre dos aparatos, dispositivos o sistemas que funcionan independientemente uno del otro. En este sentido, la comunicación entre un ser humano y una computadora se realiza por medio de una **interfaz.** |
| Patrón: | quitar de una cosa lo malo, lo que es extraño o lo que no sirve, para dejarla pura. Los *Patrones de diseño* (*Design Patterns*) son una solución general, reutilizable y aplicable a diferentes problemas de diseño de *software*. |
| Servidor: | un **servidor** es un conjunto de computadoras capaz de atender las peticiones de un cliente y devolverle una respuesta en concordancia. |
| Sintaxis: | es la “parte de la gramática que estudia el modo en que se combinan las palabras y los grupos que éstas forman para expresar significados, así como las relaciones que se establecen entre todas esas unidades”. |
| Sitio web: | es un conjunto de páginas **web** accesibles a través de **Internet**, convenientemente enlazadas y con una finalidad concreta. |
| *Stakeholders:* | se relacionan con las entidades clave de los proyectos de **desarrollo**: requerimientos, actividades y resultados. |
| UML: | lenguaje unificado de modelado. Relaciona un conjunto de diagramas estandarizados para la representación de sistemas de información desde diferentes tipos de vista. |
| Web: | conjunto de información que se encuentra en una dirección determinada de Internet. |

1. **Referencias bibliográficas**

Amazon Web Services [AWS]. (s. f.). *Microservicios*. <https://aws.amazon.com/es/microservices/>

Ayala, J. (s.f.). Modelo de Aplicación de Java Empresarial. <https://jmaw.blogspot.com/2012/09/modelo-de-aplicacion-de-java-empresarial.html>

Bahit, E. (2011). POO y MVC en PHP. <https://www.slideshare.net/eugeniabahit/poo-y-mvc-en-php-por-eugenia-bahit>

Blancarte, O. (2020). *Arquitectura Monolítica*. Reactive Programming. <https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/monolitico>

Cinergix Pty. Ltd. (2021). *La Guía Fácil de los Diagramas de Despliegue UML*. Blog de Creately. <https://creately.com/blog/es/diagramas/tutorial-de-diagrama-de-despliegue/>

Cuéllar, J. (2010). Estilos y patrones básicos en arquitectura de software. <https://josecuellar.net/estilos-patrones-basicos-arquitectura-software/>

Diagramas UML. (2019). *Diagrama de componentes*. <https://diagramasuml.com/componentes/>

EcuRed. (s. f.). *Patrones Gof*. <https://www.ecured.cu/Patrones_Gof>

Fusap. (2021). INTEGRACIÓN SOA-OSB: TELECOM. <https://www.fusap.com.ar/desarrollos.html>

Gamma, E., Helm, R., Johnson, R., Vlissides, J. y Booch, G. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

Hernández, R. (2021). El patrón modelo-vista-controlador: Arquitectura y *frameworks* explicados. <https://www.freecodecamp.org/espanol/news/el-modelo-de-arquitectura-view-controller-pattern/>

Hiberus. (2021, marzo 4). *De una arquitectura tradicional a una arquitectura microservicios*. Hiberus Tecnología. <https://www.hiberus.com/crecemos-contigo/de-una-arquitectura-tradicional-a-microservicios/>

Landa, N. [nicosiored]. (2018c). *Patrones de Diseño de Software* [Video]. YouTube. <https://www.youtube.com/playlist?list=PLM-p96nOrGcbqbL_A29b0z3KUXdq2_fpn>

Landeta\_P. (2013). *2 2 estilos arquitectónicos*. Slideshare. <https://es.slideshare.net/landeta_p/2-2-estilos-arquitectonicos>

Marrero, D. (2016). *Patrón MVC.* <https://daniellamikaelson.files.wordpress.com/2016/07/20160327_56f72c4319db8.jpg?w=840>

Novoseltseva, E. (2020). *Los 5 principales patrones de Arquitectura de Software*. Apiumhub. <https://apiumhub.com/es/tech-blog-barcelona/principales-patrones-arquitectura-software/>

Rojas, Y. (2021). Base de Datos. <https://guiadelempresario.com/wp-content/uploads/2021/05/Base-de-datos-orientada-a-objetos.jpeg>

Ruelas, U. (2017). Poa-aop-programación-orientada-a-aspectos-aspect-oriented-programming-ejemplo-qué-es-paradigma. <https://codingornot.com/que-es-la-programacion-orientada-a-aspectos-aop/poa-aop-programacion-orientada-a-aspectos-aspect-oriented-programming-ejemplo-que-es-paradigma>

Zambrano, L. (2017). Diseño de la arquitectura estratificada de la solución. <https://www.researchgate.net/figure/Figura-1-Diseno-de-la-arquitectura-estratificada-de-la-solucion_fig1_318509442>

1. **Control del documento**

|  | Nombre | Cargo | Dependencia  *(Para el SENA indicar Regional y Centro de Formación)* | Fecha |
| --- | --- | --- | --- | --- |
| Autor(es) | Jonathan Guerrero Astaiza | Experto temático | Regional Cauca – Centro de teleinformática y producción industrial | Septiembre de 2021 |
| David Eduardo Lozada Cerón | Experto Temático | Regional Cauca, Centro de Teleinformática y Producción Industrial | Octubre del 2021 |
| María Inés Machado López | Diseñadora Instruccional | Regional Distrito Capital. Centro de Diseño y Metrología | Octubre de 2021 |
| Carolina Coca Salazar | Revisora Metodológica y Pedagógica | Regional Distrito Capital. Centro de Diseño y Metrología | Octubre de 2021 |
| Darío González | Corrección de estilo | Regional Tolima – Centro Agropecuario La Granja | Noviembre 2021 |

1. **Control de cambios**

|  | Nombre | Cargo | Dependencia | Fecha | Razón del cambio |
| --- | --- | --- | --- | --- | --- |
| Autor(es) | Paola Alexandra Moya | Evaluadora instruccional | Centro de Servicios de Salud | Febrero 2024 | Actualización |
|  | Olga Constanza Bermúdez Jaimes | Responsable Línea de Producción Antioquia | Centro de Servicios de Salud | Febrero 2024 | Actualización |