**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”. Para comenzar el recorrido, visite el recurso didáctico que se muestra a continuación:



Para la elaboración de este componente, se abordaron varios autores conocidos en **patrones de diseño de *software***, de quienes se han citado y referenciado conceptos y ejemplos para los fines educativos de esta materia, en el entendido de que el conocimiento es social y, por lo tanto, es para ser usado por quienes necesitan adquirirlo. Se espera que este documento sea útil para todos aquellos, aprendices y lectores en general, que estén interesados en acercarse a asuntos básicos del **aseguramiento de la calidad de *software***.

1. **Conceptos GOF**

| Puzzle vector gratis en AI, SVG, EPS o PSD | Los patrones GOF son presentados como una forma indispensable de enfrentarse a la programación y surgen a raíz de: Erich 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:

* Constituyen soluciones concretas que, aunque se representan de forma genérica, son aplicados para resolver problemas reales.
* Son especificaciones técnicas basadas en los principios de la programación orientada a objetos y, dependiendo del lenguaje de programación, la forma de implementarlos puede variar.
* Son frecuentemente utilizados, ya que se construyen a partir de la experiencia acumulada en la industria del desarrollo de *software*.
* Favorecen la implementación de las características de la programación orientada a objetos, como la encapsulación, las jerarquías y el polimorfismo.
* No necesariamente el uso de un patrón implica el uso de palabras claves, reservadas o de librerías especializadas.
* Los patrones generalmente hacen referencia al uso de interfaces, clases y objetos que deben ser ajustados de acuerdo con la solución concreta a desarrollar.

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**

| Concepto Empresarial De Coworking. Compañeros De Trabajo Armando  Rompecabezas. Metáfora De Creación De Equipos Ilustración del Vector -  Ilustración de meta, estrategia: 186354521 | 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**

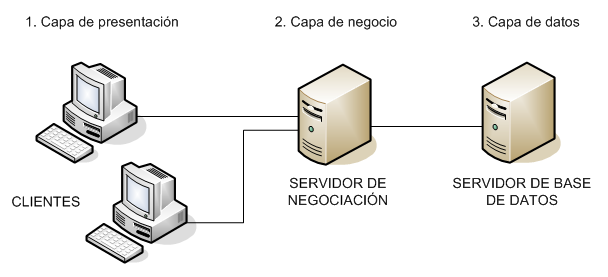
Son patrones de X niveles y se organizan en capas horizontales. Muchas de las aplicaciones informáticas han tomado este patrón como arquitectura, funciona conectando los componentes, pero estos no dependen uno del otro. Cada una de estas capas cumple con una función específica en el desarrollo de la 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. Un ejemplo de esto es 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**

*Arquitectura de tres capas*



**Nota:** [**https://upload.wikimedia.org/wikipedia/commons/e/ea/Tres\_capas.PNG**](https://upload.wikimedia.org/wikipedia/commons/e/ea/Tres_capas.PNG)

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. 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**

| Microservicios: ventajas y contras de la arquitectura descentralizada | by  Good Rebels | Medium | 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. |
| --- | --- |

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.**

| DISEÑO DE LA ARQUITECTURA DEL SOFTWARE” | Deahesy22&#39;s Blog | 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. |
| --- | --- |

* 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**

**Figura 2**

*Ejemplo de Patrón 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). |
| --- | --- |

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. | Calculadora en Windows 10: Más científica que nunca en todo |
| --- | --- |

* 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 Figura 3, se puede ver cada uno de los componentes del patrón y cómo interactúan entre ellos.

**Figura 3**

*Diagrama patrón comando*



| 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). | Dibujo De Televisor Pantalla Plana Con Control Remoto Para Pintar Y  Colorear Partido De Futbol En La Television | COLOREAR DIBUJOS VARIOS |  Dibujo De Televisor Pantalla Plana Con Control Remoto Para |
| --- | --- |

* 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, por lo tanto, 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 4**

*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 labores comunes en el proceso de construcción de *software* es precisamente distribuir responsabilidades en un conjunto de módulos o clases siguiendo los principios definidos en los paradigmas de programación, como, por ejemplo, la programación orientada a objetos. Sin embargo, al momento de dar solución a un requerimiento particular, se requerirá instanciar objetos de diferentes tipos, los cuales, bajo sus responsabilidades, implementan algún tipo de lógica.





* 1. ***Singleton***

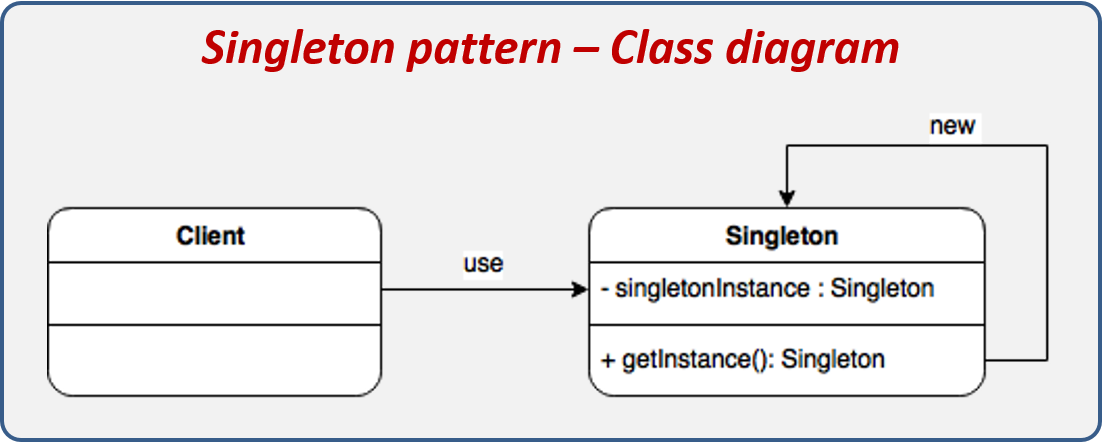
| Este patrón de diseño creacional se encarga de definir la forma en que se puede garantizar que exista una única instancia de una clase particular en el contexto de la aplicación. Esto es útil para casos en los que, por cuestiones de manejo de memoria o de la lógica del negocio, se requiere que sea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

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

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

**Figura 5**

*Diagrama patrón Singleton*



Nota: <https://reactiveprogramming.io/books/patterns/img/patterns-articles/singleton-diagram.png>

* 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.



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.



* 1. ***Delegate***

| Ilustración de Concepto De Delegación De Autoridad Sobrecargue Las Tareas  Dedeleinadas De La Mujer De Negocios A Los Empleados Office People Comparte  Trabajo Crear Una Estructura Estable De Gestión Profesional De Dibujos | 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. |
| --- | --- |





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.):

* Diagrama de clases
* Diagrama de objetos
* Diagrama de componentes

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.

| Un componente puede ser *software,* como, por ejemplo, las bases de datos o una interfaz de usuario; y también puede ser *hardware,* como un dispositivo o incluso una unidad de negocio, como, por ejemplo, la nómina, el inventario, proveedores, etc.  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. | Definición Software - Hardware-Software-Condorí-Escudero |
| --- | --- |

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



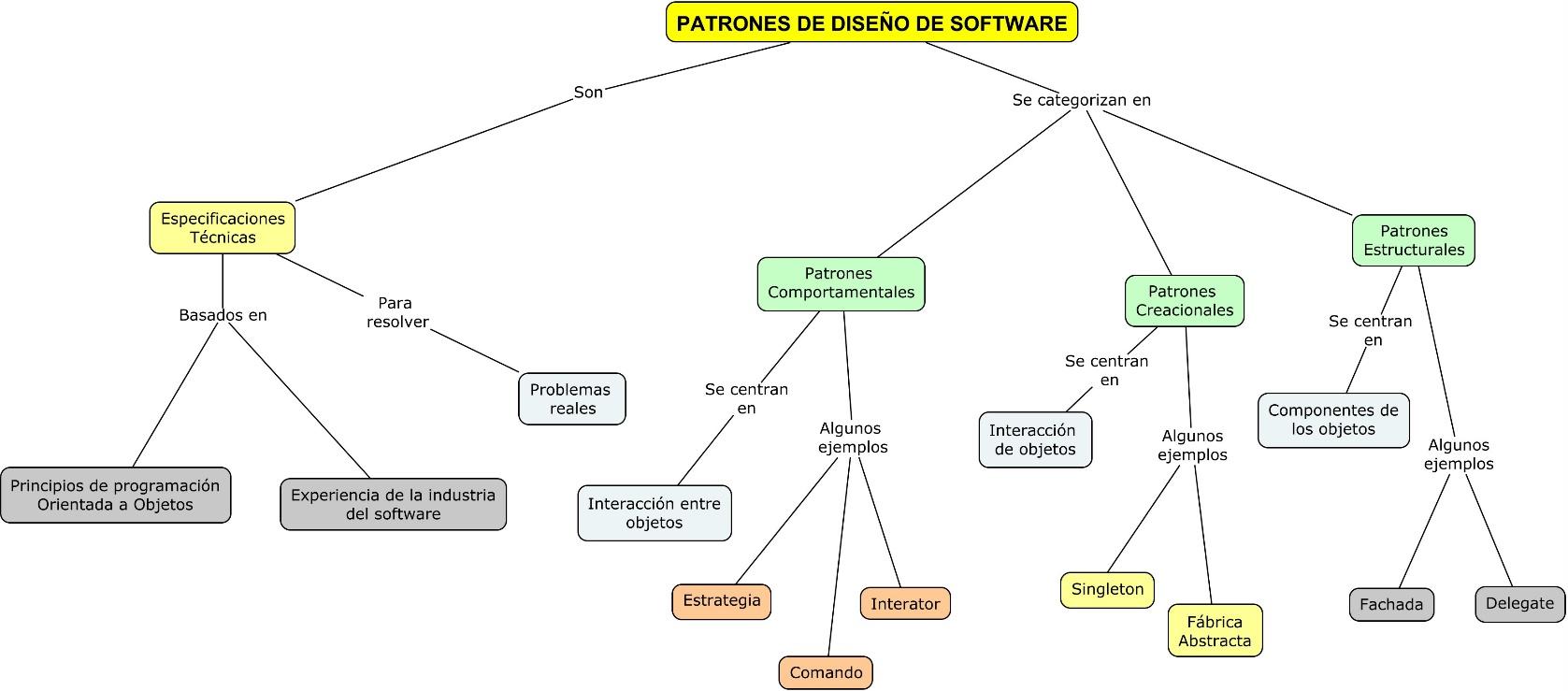
1. **Síntesis**

Los patrones de diseño de *software* son de gran importancia para los desarrolladores, ya que estos recogen un conjunto de buenas prácticas que fueron abordadas por gran cantidad de personas en el contexto de problemas reales muy bien definidos. Por lo anterior, ya no es necesario iniciar desde cero para plantear una solución.

Los patrones de diseño, por sus características, están categorizados en tres grandes grupos: comportamentales, creacionales o estructurales. Es decir, dependiendo de la forma como se quiere que se comporte una funcionalidad, o como se quiere que se creen los objetos, o como se quiere definir su estructura, se van a tener que implementar varios patrones de diseño al mismo tiempo. De hecho, el desarrollo de un proyecto profesional requerirá siempre la implementación de varios patrones de diseño.

Existen muchos más patrones de diseño que los desarrollados en este componente formativo, y a medida que pase el tiempo, seguramente aparecerán muchos más. Se le invita a observar los materiales complementarios para profundizar en la temática.

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** | **Repaso de 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)** | **Anexo Actividad didáctica 1** |

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* [Video]. YouTube. <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* [Video]. YouTube. <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* [Video]. YouTube. <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* [Video]. YouTube.  <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* [Video]. YouTube.  <https://www.youtube.com/embed/ANQDmqBYwns> | Video YouTube | <https://www.youtube.com/embed/ANQDmqBYwns> |
| Arquitectura Monolítica | Lazy Loading. (2021). *Arquitectura monolítica vs microservicios* [Video]. YouTube. <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* [Video]. YouTube.  <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* [Video]. YouTube.  <https://www.youtube.com/embed/o_Br2vZ4uQY> | Video YouTube | <https://www.youtube.com/embed/o_Br2vZ4uQY> |

1. **Glosario**

| **TÉRMINO** | **SIGNIFICADO** |
| --- | --- |
| Acoplamiento | El **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 | Una **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 | Una **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 | Una **base de datos** es una recopilación organizada de información o **datos** estructurados, que normalmente se almacena de forma electrónica en un sistema informático. |
| *Browser* | ***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 | Como **interfaz** se designa, en informática, 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 | Según su **definición,** la **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 | Un **sitio web** es un conjunto de páginas **web** accesibles a través de **Internet**, convenientemente enlazadas y con una finalidad concreta. |
| *Stakeholders* | Los ***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/>

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/>

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

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

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

ITCA-FEPADE. (s. f.). *Análisis y diseño de sistemas. 2.2.2 Diagramas UML estáticos*. <https://virtual.itca.edu.sv/Mediadores/ads/222_diagramas_uml_estticos.html>

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>

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

Universidad de Alicante. (s. f.). *Modelo Vista Controlador (MVC)*. Servicio de Informática ASP.NET MVC 3 Framework. <https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html>

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) |  |  |  |  |  |