Los patrones de diseño brindan una solución ya aprobada y documentada a problemas de software, es importante tener en cuenta los siguientes elementos: el nombre del patrón, tener claro cuando podemos utilizar el patrón, la descripción abstracta del problema y las consecuencias o costos beneficios.

Los patrones de diseño hacen más fácil reutilizar diseños y arquitecturas, expresando técnicas probadas como diseño de patrones lo hacen más accesible para los desarrolladores de nuevos sistemas, los patrones de diseño te ayudan a escoger alternativas de diseño que hacen un sistema reutilizable y evitan otras alternativas que comprometan la reutilidad.

Los patrones de diseño incluso pueden mejorar la documentación y el mantenimiento de sistemas ya existentes.

El diseño de patrones ayuda al desarrollador a obtener un diseño correcto y más rápido.

Cada patron de diseño describe un problema diferente el cual ocurre una y otra vez en nuestro entorno y después describen el núcleo de la solución a este problema.

A pesar de que el diseño de patrones describe diseños orientados a objetos, estos están basados en soluciones practicas que han sido implementadas en los principales lenguajes de programación orientados a objetos.

La elección del lenguaje de programación es muy importante ya que influencia en algunos puntos de vista

La descripción de los patrones de diseño es a través de un formato consistente, cada patrón es dividido en secciones conforme al siguiente modelo. El modelo nos da una estructura uniforme de información lo cual nos hace más fácil aprender, comparar y usar los patrones de diseño

Algunas de las maneras en que los patrones de diseño ayudan a resolver problemas son:

\*Encontrando objetos apropiados

\* Determinación de la granularidad del objeto

\*Especificando interfaces de objetos

\*Especificando implementaciones de objetos

\*Clase contra interfaz de herencia

\*Programando una interfaz, no una implementación

\*Poner a trabajar mecanismos de reutilización

\*Herencia contra composición

Existen varios tipos de patrones de diseño, tales como:

\*Patrones Creacionales: Inicialización y configuración de objetos.

\*Delegacion

\*Herencia contra tipos parametrizados

\*Relacionando las estructuras de tiempo de ejecución y tiempo de compilación

\*Diseñando para cambiar

\*Programas de aplicación

\*Conjuntos de herramientas

\*Frameworks

El problema a solucionar por este patrón es el de crear diferentes familias de objetos, como por ejemplo la creación de interfaces gráficas de distintos tipos.

Parte del principio de que las subclases determinan la clase a implementar.

Es prototipado ya que se basa en la clonación de ejemplares copiándolos de un prototipo.

Restringe la instanciación de una clase o valor de un tipo a un solo objeto.

Uno de los más conocidos y utilizados de este tipo es el MVC (Model View Controler )

Este patrón plantea la separación del problema en tres capas: la capa model, que representa la realidad; la capa controler, que conoce los métodos y atributos del modelo, recibe y realiza lo que el usuario quiere hacer; y la capa vista, que muestra un aspecto del modelo y es utilizada por la capa anterior para interaccionar con el usuario.

Especifica que una aplicación consta de un modelo de datos, de información de presentación y de información de control. El patrón requiere que cada uno de estos elementos esté separado en distintos objetos.

El modelo contiene únicamente los datos puros de aplicación; no contiene lógica que describe cómo pueden presentarse los datos a un usuario.

La vista presenta al usuario los datos del modelo. La vista sabe cómo acceder a los datos del modelo, pero no sabe el significado de estos datos ni lo que el usuario puede hacer para manipularlos.

Y el controlador está entre la vista y el modelo. Escucha los sucesos desencadenados por la vista y ejecuta la reacción apropiada a estos sucesos. En la mayoría de los casos, la reacción es llamar a un método del modelo. Puesto que la vista y el modelo están conectados a través de un mecanismo de notificación, el resultado de esta acción se reflejará automáticamente en la vista.

\*Patrones Estructurales: Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes.

Algunos ejemplos de estos patrones son:

Adaptador (Adapter): Convierte una interfaz en otra.

Puente (Bridge): Desacopla una abstracción de su implementación permitiendo modificarlas independientemente.

Objeto Compuesto (Composite): Utilizado para construir objetos complejos a partir de otros más simples, utilizando para ello la composición recursiva y una estructura de árbol.

Envoltorio (Decorator): Permite añadir dinámicamente funcionalidad a una clase existente, evitando heredar sucesivas clases para incorporar la nueva funcionalidad.

Fachada (Facade): Permite simplificar la interfaz para un subsistema.

Peso Ligero (Flyweight): Elimina la redundancia o la reduce cuando tenemos gran cantidad de objetos con información idéntica.

Apoderado (Proxy): Un objeto se aproxima a otro.

\*Patrones de Comportamiento: Más que describir objetos o clases, describen la comunicación entre ellos.

Algunos ejemplos de este tipo de patrón son:

Cadena de responsabilidad (Chain of responsibility): La base es permitir que más de un objeto tenga la posibilidad de atender una petición.

Orden (Command): Encapsula una petición como un objeto dando la posibilidad de “deshacer” la petición.

Intérprete (Interpreter): Intérprete de lenguaje para una gramática simple y sencilla.

Iterador (Iterator): Define una interfaz que declara los métodos necesarios para acceder secuencialmente a una colección de objetos sin exponer su estructura interna.

Mediador (Mediator): Coordina las relaciones entre sus asociados. Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones.

Recuerdo (Memento): Almacena el estado de un objeto y lo restaura posteriormente.

Observador (Observer): Notificaciones de cambios de estado de un objeto.

Estado (Server): Se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo.

Estrategia (Strategy): Utilizado para manejar la selección de un algoritmo.

Método plantilla (Template Method): Algoritmo con varios pasos suministrados por una clase derivada.

Visitante (Visitor): Operaciones aplicadas a elementos de una estructura de objetos heterogénea.