PATRONES: es una pareja de Problema/Solución con un nombre y que es aplicable a otros contextos con una sugerencia sobre la manera de usarlo en situaciones nuevas.

Características:

- Es un lenguaje compartido para comunicar la experiencia adquirida al tratar con estos problemas recurrentes y sus soluciones.
- Un patrón de diseño se puede ver como una encapsulación de una solución reutilizable que se ha aplicado con éxito para resolver un problema de diseño común.
- Aunque los patrones de diseño se refieren a las formas más conocidas de resolver problemas, no todas las mejores prácticas en la resolución de problemas se consideran patrones.
- Los patrones de diseño no brindan soluciones a todos los problemas que se encuentran en el diseño y desarrollo de software del mundo real. Esto se debe a que dan soluciones a problemas en un contexto particular que puede no producir una solución efectiva en otro contexto.

Ventajas:

- Estructuras de diseño probadas
- Solución alternativa
- Combinación de patrones
- Completitud a soluciones caseras
- Facilidad para separar los aspectos que cambian

PATRONES GRASP: describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Es un acrónimo que significa General Resposability Asignment Software Patterns y se eligió para indicar la importancia de captar estos principios, si se quiere diseñar eficazmente el software orientado a objetos.

PATRON	PROBLEMA	SOLUCION	BENEFICIOS
Experto	Definir las interacciones entre los objetos, es decir, la asignación de responsabilidades a las clases.	Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.	- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece el hecho de tener sistemas más robustos y de fácil mantenimiento. - El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clase “sencillas” y más cohesivas que son más fáciles de aprender y de mantener.
Creador	Determinar quién será responsable de la creación de nuevas instancias de alguna clase.	Asignarle a la clase B la responsabilidad de crear una instancia de clase A en uno de los siguientes casos: - B agrega los objetos A (agregación). - B contiene los objetos A (composición). - B registra las instancias de los objetos A. - B utiliza específicamente los objetos A. - B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado.	Se brinda soporte a un bajo acoplamiento, lo cual supone menos dependencias respeto al mantenimiento y mejores oportunidades de reutilización.

Bajo acoplamiento	Determinar que las clases tengan la menor dependencia posible.	Asignar una responsabilidad para mantener bajo acoplamiento.	- No se afectan por cambios de otros componentes. - Fáciles de entender por separado. - Fáciles de reutilizar.
Alta cohesión	Determinar que las clases tengan las responsabilidades muy relacionadas.	Asignar una responsabilidad de modo que la cohesión siga siendo alta.	- Mejoran la claridad y facilidad con que se entiende el diseño. - Se simplifican el mantenimiento y las mejoras en funcionalidad. - A menudo se genera un bajo acoplamiento. - La ventaja de una gran funcionalidad soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.
Controlador	Determinar que clases deben encargarse de atender un evento del sistema.	Asignar la responsabilidad del manejo de un mensaje de los eventos del sistema a una clase que represente las siguientes opciones: - El "sistema" global (controlador de fachada). - La empresa u organización global (controlador de fachada). - Algo en el mundo real que es activo y que pueda participar en la tarea (controlador de tareas). - Un manejador artificial de todos los eventos del sistema de un caso de uso (controlador de casos de uso).	- Mayor potencial de los componentes reutilizables: garantiza que la empresa o los procesos de dominio sean manejados por la capa de los objetos del dominio y no por la de la interfaz. - Reflexionar sobre el estado del caso de uso: a veces es necesario asegurarse de que las operaciones del sistema sigan una secuencia legal o poder razonar sobre el estado actual de la actividad y las operaciones en el caso de uso subyacente.
Polimorfismo	¿Cómo manejar las alternativas basadas en el tipo? ¿De qué manera crear componentes de software conectables?	Cuando por el tipo varían las alternativas o comportamientos afines, las responsabilidades del comportamiento se asignarán - mediante operaciones polimórficas - a los tipos en que el comportamiento presenta variantes.	Es fácil agregar las futuras extensiones que requieren las variaciones imprevistas.
Fabricación pura	Determinar que clases tendrán la responsabilidad de crear clases que no tengan que ver con lo conceptual y no afecten los patrones de Alta Cohesión y Bajo Acoplamiento.	Asignar un conjunto altamente cohesivo de responsabilidades a una clase artificial que no representa nada en el dominio del problema: una cosa inventada para dar soporte a una alta cohesión, un bajo acoplamiento y reutilización.	- Se brinda soporte a una Alta Cohesión porque las responsabilidades se dividen en una clase de granularidad fina que se centra exclusivamente en un conjunto muy específico de tareas afines. - Puede aumentar el potencial de reutilización debido a la presencia de las clases de fabricación pura de granularidad fina, cuyas responsabilidades pueden utilizarse en otras aplicaciones.
No Hables con Extraños	Determinar que clases tendrán ciertas responsabilidades para evitar conocer la estructura de objetos indirectos.	Se asigna la responsabilidad a un objeto directo del cliente para que colabore con un objeto indirecto, de modo que el cliente no necesite saber nada del objeto indirecto.	Bajo acoplamiento.
Indirección	Determinar que clases tendrán ciertas responsabilidades a fin	Se asigna la responsabilidad a un objeto intermedio para que medie entre otros	Bajo acoplamiento.

	de evitar el acoplamiento directo.	componentes o servicios, y éstos no terminen directamente acoplados. El intermediario crea una indirección entre el resto de los componentes o servicios.	

INTERFAZ: Una interfaz específica es un conjunto de características públicas. La idea clave es separar la especificación de funcionalidad (interfaz) de su implementación por un clasificador como una clase o subsistema. Una interfaz no se puede instanciar, solo declara un contrato que se puede realizar por cero o más clasificadores.

- Interfaces proporcionadas: es el conjunto de interfaces realizadas por un clasificador.
- Interfaces requeridas: conjunto de interfaces que son requeridas por un clasificador para su operación.

COMPONENTE: Representa una parte modular de un sistema que encapsula sus contenidos y cuya manifestación se reemplaza dentro de su entorno. Actúa como una caja negra cuyo comportamiento externo está completamente definido por sus interfaces proporcionadas y requeridas. Debido a esto, un componente se puede reemplazar por otro que soporte el mismo protocolo. Pueden tener atributos y operaciones y pueden participar en relaciones de asociación y generalización.

DISEÑAR CON INTERFACES: Un uso potente de las interfaces es proporcionar la posibilidad de conectar elementos a los sistemas. Una de las formas de hacer que los sistemas sean flexibles al cambio es diseñar el sistema de modo que las extensiones se puedan conectar fácilmente. Las interfaces son la clave para esto. Si puede diseñar sistemas en torno a interfaces, entonces las asociaciones y envíos de mensaje ya no están unidos a objetos de una clase determinada sino que están unidos a una interfaz determinada.

VENTAJAS Y DESVENTAJAS DE LAS INTERFACES

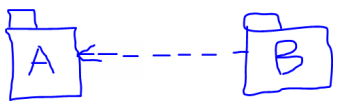
- Libera el modelo de dependencias de implementación y por lo tanto aumenta su flexibilidad y extensibilidad.
- Permite reducir el número de dependencias entre clases, subsistemas y componentes y empieza a proporcionar control sobre la cantidad de acoplamiento en un modelo.
- Se busca un equilibrio entre flexibilidad y complejidad.
- Existe un coste de rendimiento a la flexibilidad.
- Necesita flexibilidad para tratar con los aspectos fluidos pero puede simplificar los sistemas al prescindir de cierta cantidad de flexibilidad para las partes más estables.

ARQUITECTURA DE SOFTWARE

Arquitectura: son formas de descomponer, conectar y relacionar partes de un sistema. Tiene que ver con el diseño macro, con la subdivisión de un sistema en subsistemas o paquetes, y con especial cuidado con el acoplamiento entre las partes.

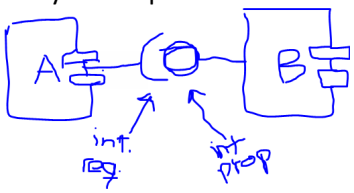
Dentro de los diagramas que se pueden usar para mostrar la Arquitectura de Software se encuentran:

- Diagrama de paquetes: se usa para mostrar agrupamientos de distintos tipos y dependencias entre esos agrupamientos. Dentro de B hay clases que refieren a B depende de A. Hay acoplamiento y cuanta menos dependencia se tenga, mejor.



A.

- Diagrama de componentes: permite visualizar la estructura de alto nivel del sistema y el comportamiento del servicio que estos componentes proporcionan y usan a través de interfaces.
B proporciona una interfaz requerida por A para que A pueda funcionar.



- Diagrama de despliegue: es definir la vista física. Se tienen diferentes partes de software y se tiene que ver en qué nodos van a correr y qué tipos de conexión necesito entre equipos.

Propuestas para el diseño Macro:

Separación de intereses: apunta a separar la aplicación por zonas de cambio. Así, si se espera que la interfaz de usuario cambie por razones diferentes, o más menudo, que la lógica de la aplicación, los componentes que implementen la interfaz de usuario deberían estar separados de los que implementen la lógica de negocio. Algunas de las razones por las cuales se separa son:

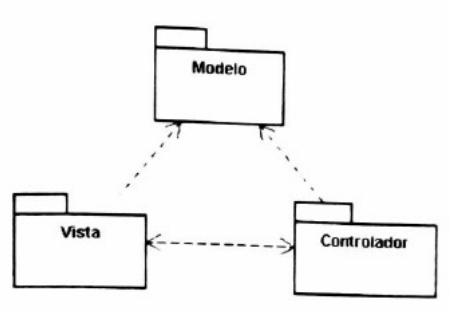
- Tecnología de software: en distintas partes puede ser diferentes.
- Equipos de desarrollo: las personas que desarrollen se especialicen en esas partes por separado.
- Forma de implementación: las partes en cuestión pueden tener más de una forma de implementación.
- Hardware: las partes deben correr ya sea en el presente o en el futuro en computadoras diferentes.

Siguiendo estas premisas, una aplicación puede tener los siguientes módulos:

- Lógica de negocio de la aplicación: involucra las reglas de negocio y requerimientos.
- Presentación o interfaz de usuario: se ocupa de mostrar a usuarios humanos lo que está ocurriendo en una aplicación y tomar las acciones de los usuarios, validar los datos de entrada y enviarlos a otras partes de la aplicación.
- Persistencia y acceso a datos: es habitual que se necesite que los datos almacenados en memoria persistan en el tiempo.
- Acceso de nuestra aplicación a servicios externos.
- Acceso de otras aplicaciones a nuestros servicios.
- Administración de seguridad: involucra los perfiles de usuario

PATRONES DE DISEÑO ARQUITECTONICO: expresa un esquema de organización estructural fundamental para sistemas de software. Provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y guias para organizar las relaciones entre los mismos.

MVC: Es un patrón macro que consiste en separar la aplicación en tres partes: modelo, vista y controlador. Muy adecuado en aplicaciones con alta interacción con el usuario.



- Modelo: es el conjunto de componentes correspondientes a la lógica de la aplicación. La idea de este modelo es que tenga bajo acoplamiento con vistas y controladores. Toda la interacción se debe hacer mediante métodos de consulta, que informen el estado del modelo, comandos que permitan modificar dicho estado y mecanismos de notificación para informar a los observadores o vistas.
- Vista: es la parte del sistema que administra la visualización y presentación de la información. Su principal tarea es observar al modelo para actualizar las variaciones, de modo que represente el estado del modelo y sus cambios de estado. Es un conjunto de componentes altamente dependiente del dispositivo y la tecnología de visualización asi como también del modelo.
- Controlador: es el responsable de manejar el comportamiento global de la aplicación. Su principal tarea consiste en recibir eventos del usuario y decidir qué es lo que se debe hacer, mapeándolos en comandos hacia el modelo, que eventualmente lo modifican. El controlador también es ampliamente dependiente de los dispositivos y mecanismos de interacción del usuario y también del modelo.

Ventajas:

- Permite la implementación sencilla de aplicaciones cliente-servidor.
- Aísla claramente la lógica del negocio (modelo) de la interfaz de usuario (vista y controlador).
- Desarrollo simultáneo: permite múltiples desarrolladores trabajando en el modelo, el controlador y las vistas.
- Alta cohesión.
- Bajo acoplamiento.
- Facilidad de modificación: esto se debe a la separación de responsabilidades.
- Múltiples vistas para un modelo.

CAPAS: La idea de los patrones basados en capas es que cada modulo depende de una única capa inferior y brinda servicios a una o más capas superiores. Por lo tanto, cada capa conoce la inmediata inferior pero no las capas superiores que utilizan sus servicios, ni la inferiores adyacentes.

Ventajas:

- Abstracción y encapsulamiento: cada capa se refiere a una abstracción diferente y puede ser analizada y comprendida sin saber cómo están implementadas el resto de las capas.
- Bajo impacto de cambios: se puede cambiar la implementacion de una capa sin afectar a las demás.
- Bajo acoplamiento: la dependencia entre capas se mantiene al minimo. Cada capa depende una sola capa inferior.
- Una misma capa puede dar servicios a varias capas superiores colocadas en paralelo.

Desventajas:

- A veces un cambio en una capa inferior lleva a cambiar capas superiores que dependen de ella, con un efecto cadena que puede resultar amplio.
- Muchas capas pueden dañar el desempeño por la necesidad de llamadas sucesivas de métodos.

Dos capas: nació con las aplicaciones cliente-servidor. Hay un modulo de interfaz de usuario (cliente) y otro que tiene algo de lógica mezclada con acceso a datos persistentes. Es bastante bueno como solución cuando lo único que se quiere es mostrar un conjunto de tablas de datos y dar acceso a un usuario que las gestione.

Tres capas: implica dividir el sistema en: capa de interfaz de ususario, lógica de la aplicación y capa de acceso a datos. Las capas se comunican mediante interfaces expuestas por cada una. En este modelo, la interfaz de usuario es la capa superior y muestra lo que ocurre en la capa de la lógica de la aplicación. Asimismo, esta última capa depende de la capa de acceso de datos, en la medida que se comunica con ella cada vez que necesita algo y lo obtiene de ella.

Ventajas:

- Posibilita el cambio de modelo sin modificar la interfaz de usuario o el acceso a datos.
- Posibilitar la evolución de la interfaz de usuario sin cambiar el modelo o la base de datos.
- Desarrollar múltiples interfaces de usuario o módulos de acceso a datos para un mismo modelo.
- Posibilidad de utilizar diferentes herramientas y plataformas en cada capa.
- Posibilidad de utilizar diferentes paradigmas en capas distintas.
- Cada capa puede ser desarrollada por un equipo distinto, con cronogramas relativamente independientes.

Más capas: el uso de cada arquitectura particular dependerá del tipo de aplicación que se vaya a desarrollar.

INTERFACES GRÁFICAS DE USUARIO (GUI): Es un elemento cuyo objetivo es suministrar un medio amigable mediante el cual el usuario pueda interactuar con la aplicación para procesar entradas y obtener salidas.

Características:

- Establece la comunicación entre el usuario y el programa. Esta comunicación es crucial, ya que determina todo lo que el usuario puede realizar con el programa. Es también llamada "Interacción".
- Incluye la presentación de las entradas y salidas. Aquí están también incluidos los reportes de salida y formularios de entrada.
- Incluye las transiciones entre pantallas.
- En general surgen a partir del diseño de prototipos, los cuales se construyen rápidamente, y en algunos casos, utilizando herramientas software para su construcción.

Factores:

1. Tipos de Usuarios: son los actores de diverso nivel para los cuales se destina el diseño de la interfaz de usuario. Son quienes deben probar y evaluar dicho diseño. Es importante analizar el nivel de experiencia técnica de los usuarios respecto de la informática, y respecto del proceso de negocio al cual apunta la aplicación.
2. Factores que afectan al Humano: consiste en analizar una serie de cuestiones relacionadas con las dificultades que puedan encontrar diferentes personas al utilizar aplicaciones:
   - × Uso excesivo de tecnicismos y acrónimos informáticos.
   - × Diseño que no es obvio o menos que intuitivo.
   - × Diseño Recargado: lo importante de una aplicación es el servicio y la información que la misma ofrece al usuario. Con esto en mente, la creatividad solo tiene sentido si simplifica el uso, si realmente añade algo valioso.
   - × Incapacidad para distinguir entre acciones alternativas.
   - × Enfoques inconsistentes de la resolución de problemas.
   - × Excesivos Tiempos de respuesta: problema fundamental cuando se trata de aplicaciones web. Deben analizarse las siguientes cuestiones:
     - Según norma ISO 9241 el límite de tiempo para que una persona mantenga su atención en una pantalla mientras espera es de aproximadamente 10 segundos.
     - El tiempo de espera para sentir una interacción instantánea es de una décima de segundo.
     - Ambos valores anteriores deben analizarse en función de la velocidad de acceso que esperamos para los clientes.

   Estas dificultades dan como resultado pánico, frustración, confusión, uso indebido, abandono y otras consecuencias indeseables. Para intentar solucionar estos problemas, se tienen que:
   - ✓ Entender a los usuarios y sus tareas.
   - ✓ Involucrar al usuario en el diseño de la interfaz.
   - ✓ Probar el funcionamiento del sistema en usuarios reales, y en situaciones de trabajo real.
   - ✓ Practicar el diseño interactivo, iterativo e incremental.
3. Ergonomía: es la utilización del conocimiento científico que se tiene sobre los seres humanos (psicología, fisiología, medicina) con el propósito de mejorar el ámbito laboral. Las dos características principales son:
   - La comodidad durante el uso, que consiste en reducir el cansancio físico y mental lo más que se pueda.
   - La seguridad, que consiste en elegir apropiadamente soluciones para proteger al usuario.

   La ergonomía de sitios web puede definirse dentro del contexto de Internet como la capacidad de un sitio web de responder a las necesidades de los usuarios de manera eficaz y proporcionarles comodidad mientras navegan.
   - ❖ El usuario siempre deberá darse cuenta de qué hacer después. La interfaz debe en todo momento mantener informado al usuario acerca del estado de ejecución de la aplicación.
   - ❖ Las pantallas deberán formatearse de modo que los diversos tipos de información, instrucciones y mensajes siempre aparezcan en la misma área de exhibición general. Posición estándar para mensajes de dialogo, etc.

- ❖ Utilizar atributos de exhibición con moderación.
- ❖ Especificar valores por emisión y respuestas que el usuario debería ingresar.
- ❖ Un usuario no debería continuar en la pantalla si antes no corrigió un error sobre un dato ingresado.
- ❖ Anticipar los errores que los usuarios podrían cometer.

4. Terminología y Tono del Dialogo: El flujo total entre pantallas y mensajes se denomina dialogo.
    - Tono: Utilice oraciones simples, siempre es mejor el lenguaje conversacional en vez de lenguaje escrito formal. No apele a cuestiones que afecten el humor y la confianza de la persona.
    - Terminología: evite tecnicismos informáticos, la mayoría de abreviaturas, sea consistente en el uso de la terminología.

Como Elaborar y Diseñar una Interfaz de usuario: Prototipos: Método iterativo incremental que consta de los siguientes pasos:

1. Realizar un Mapa del Dialogo de la Interfaz de Usuario: aquí se abarca la coordinación de pantallas en función de los requerimientos de usuario. Necesitamos una herramienta que nos permita coordinar las pantallas que pueden generarse. En este caso nos puede asistir el Diagrama de Transición de Estados, el cual se usa para esquematizar la secuencia y variaciones de las pantallas que pueden generarse cuando el usuario utiliza el sistema.
2. Realizar un prototipo del diálogo y de la interfaz del usuario: Consiste en realizar prototipos de todas las nuevas pantallas que no se hayan identificado en la etapa de diseño. Luego se realiza un prototipo del dialogo para verificar condiciones y caminos de ejecución, y observar el funcionamiento de una determinada parte del código.
3. Obtener la retroalimentación del usuario: Consiste en ofrecer a grupos de usuarios de distinto nivel la ejecución de los prototipos creados en los pasos anteriores. Los usuarios prueban y experimentan el diseño de la interfaz antes de realizar la programación extensiva y la implementación real.

Norma ISO 9241: es una norma internacional que abarca los Requisitos Ergonómicos para trabajos de oficinas con pantallas de visualización de datos. En su apartado 10, trata de los principios ergonómicos generales aplicables a cualquiera de las técnicas específicas de diálogo. Teniendo en cuenta el diseño de interfaces gráficas, la norma define siete principios generales aplicables a cualquiera de las técnicas específicas de diálogo:

1. Capacidad de adecuación a la tarea: Un diálogo es susceptible de adecuarse a la tarea en la medida en que asiste al usuario para lograr un acabado de la misma con eficiencia y eficacia.
2. Autodescriptividad: Un diálogo es auto descriptivo en la medida en que cada una de sus etapas es directamente comprensible a través de una retro-alimentación con el sistema o es explicada al usuario con arreglo a su necesidad de información.
3. Controlabilidad: Un diálogo es controlable en la medida en que permite al usuario conducir la totalidad del curso de la interacción hasta lograr el objetivo.
4. Conformidad con las expectivas del usuario: Un diálogo es conforme con las expectativas del usuario en la medida en que se corresponde con el conocimiento que éste tiene de la tarea, así como con su formación, experiencia y las convenciones comúnmente aceptadas.
5. Tolerancias de errores: Un diálogo es tolerante a los errores en la medida en que, a pesar de los errores que se cometan en la entrada, se puede lograr el resultado que se pretende sin realizar correcciones o cuando éstas son mínimas.
6. Adaptabilidad individual: Un diálogo es susceptible de adaptarse al individuo en la medida en que el sistema de diálogo puede modificarse de acuerdo a las habilidades y necesidades de cada usuario en particular, en relación con una tarea determinada.
7. Fácil de aprender: Un sistema de diálogo facilita su aprendizaje en la medida en que proporciona medios, guías y estímulos al usuario durante la etapa de aprendizaje.