Universidad Técnica Federico Santa María – Sede Viña del Mar
Programa de Ingeniería de Ejecución en Software

6 Diseño de Software

Fundamentos de Ingeniería de Software

Eduardo Fiol Mujica <eduardo.fiol@usm.cl>

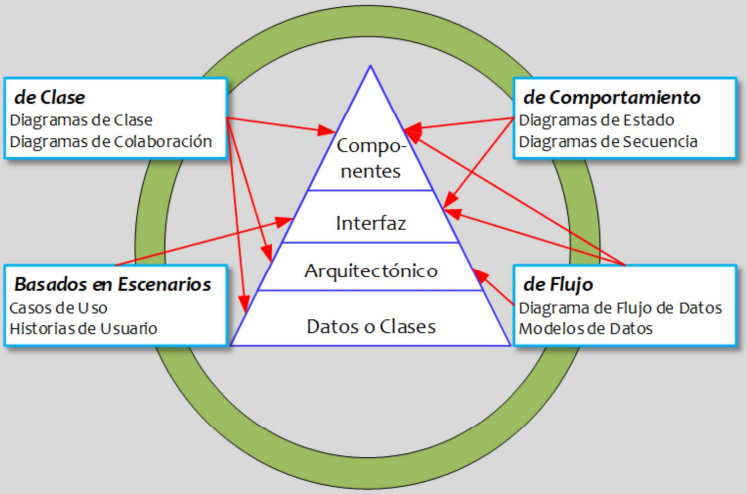
2022

6 Diseño de Software

- A partir del documento de **Especificación de Requisitos** acordado con el Cliente, el Diseñador debe **traducir** todos los deseos en soluciones concretas
- La solución es un **Diseño**, el cual satisfará las necesidades del usuario

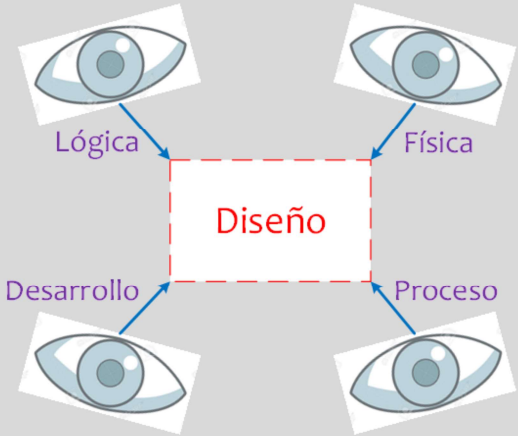
6 Diseño de Software

Traducción desde el Modelo de Análisis hacia el Modelo de Diseño



6 Diseño de Software

Diferentes Visiones



6 Diseño de Software

1. Qué es Diseño?
2. Descomposición y Modularidad
3. Estilos y Estrategias Arquitectónicos
4. Tópicos en la creación del Diseño
5. Características de un buen Diseño
6. Técnicas de mejoramiento del Diseño
7. Evaluación y Validación del Diseño
8. Documentación del Diseño
9. Diseño Orientado a Objetos

6.1 ¿Qué es Diseño?

- **Proceso creativo** de transformación
- También se le llama así a la descripción de la solución
- Especificación de Requisitos describe el problema
- Solución satisface requisitos de la especificación
- Número de soluciones es ilimitado
- Naturaleza de la solución puede cambiar
- Se puede deber a cambio de percepción del usuario o de sus necesidades
- Cambio al diseño mismo o a los requisitos

Es el proceso creativo de transformar el problema en una solución.
También se llama Diseño a la descripción de la solución.
Se usa la Especificación de Requisitos para describir el problema.
Luego, se establece que algo es solución al problema si satisface los requisitos de la especificación.
Generalmente, el número de soluciones es ilimitado.
La naturaleza de la solución puede cambiar, al tiempo que va siendo descrita o implementada.
Esto no es inusual ni poco razonable.
Simplemente, se puede deber a un cambio de percepción del usuario o de sus necesidades.
El cambio puede ser al diseño mismo como a los requisitos (y por tanto al diseño).

6.1 ¿Qué es Diseño?

- Transformar requisitos en un sistema que funciona ⇒ diseñador debe satisfacer al cliente y al equipo de desarrollo
- Punto de partida: Especificación de requisitos
- Diseño es un proceso iterativo de dos pasos:
  a. **Diseño Conceptual**, expresa funcionalidad del sistema (qué hará), lógico, alto nivel, preliminar
  b. **Diseño Técnico**, expresa forma del sistema (cómo lo hará), físico, bajo nivel, detallado
- Debería describirse en dos documentos equivalentes, mapeables en ambos sentidos
- Se debe iterar: diseño conceptual → diseño técnico → refinar diseño conceptual → refinar diseño técnico → etc →

a. Diseño Conceptual

- Describe aspectos fundamentales del sistema
  - Ámbito (límites)
  - Entidades
  - Atributos
  - Relaciones
- Responde preguntas como:
  - De dónde vienen los datos?
  - Qué les sucede a los datos en el sistema?
  - Cómo se verá el sistema para los usuarios?
  - Qué opciones se les ofrecerán a los usuarios?
- Debe tener las siguientes características:
  - Estar escrito en el lenguaje del usuario (sin jerga)
  - Describir las funciones del sistema
  - Ser independiente de la implantación
  - Ser consistente con los requisitos (trazabilidad)

Diseño Conceptual

Describir aspectos fundamentales del sistema

- ámbito (límites)
- entidades
- atributos
- relaciones

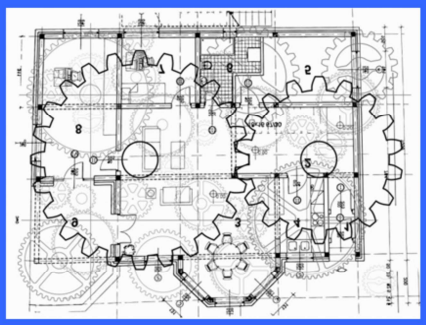
Responde preguntas como:

- De dónde vienen los datos?
- Qué les sucede a los datos en el sistema?
- Cómo se verá el sistema para los usuarios?
- Qué opciones se les ofrecerán a los usuarios?

Debe tener las siguientes características:

- Estar escrito en el lenguaje del usuario (sin jerga).
- Describir las funciones del sistema.
- Ser independiente de la implementación.
- Ser consistente con el documento de requisitos (trazabilidad)

b. Diseño Técnico



- Descripción técnica de la especificación de diseño conceptual
- Describe:
  - Configuración de hardware
  - Jerarquía y función de componentes de software
  - Necesidades del software
  - Estructuras y flujos de datos
  - Arquitectura de la red
- Y cualquier cosa que traduzca los requisitos en soluciones

Diseño Técnico

Es una descripción técnica de la especificación de diseño conceptual

Describe:

Configuración de hardware.

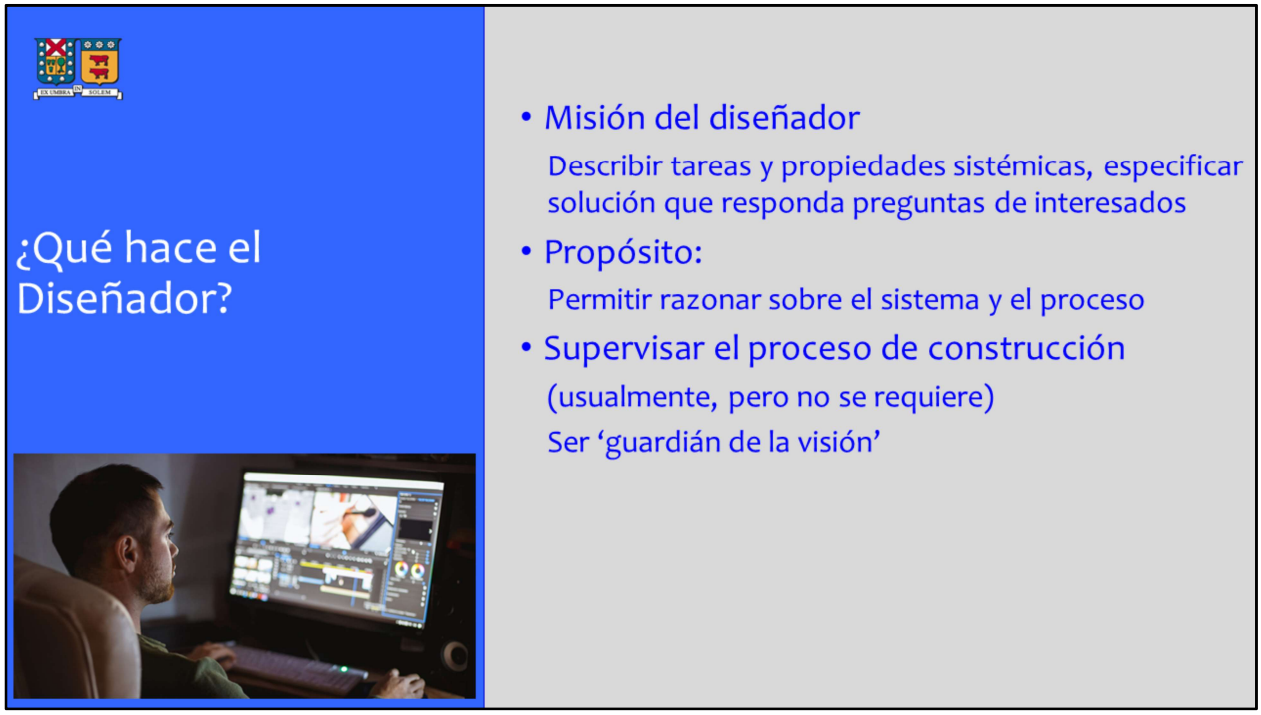
Jerarquía y función de los componentes de software

Necesidades del software.

Estructuras y flujos de datos.

Arquitectura de la red.

Y cualquier cosa que traduzca los requisitos en soluciones.



Misión del diseñador

Con una descripción de tareas y propiedades sistémicas, especificar una solución en términos de componentes que responda las preguntas de los interesados

...permitir evaluar a priori el sistema a ser construido

...ser 'guía de acción' a los constructores

...permitir elaborar un plan de construcción

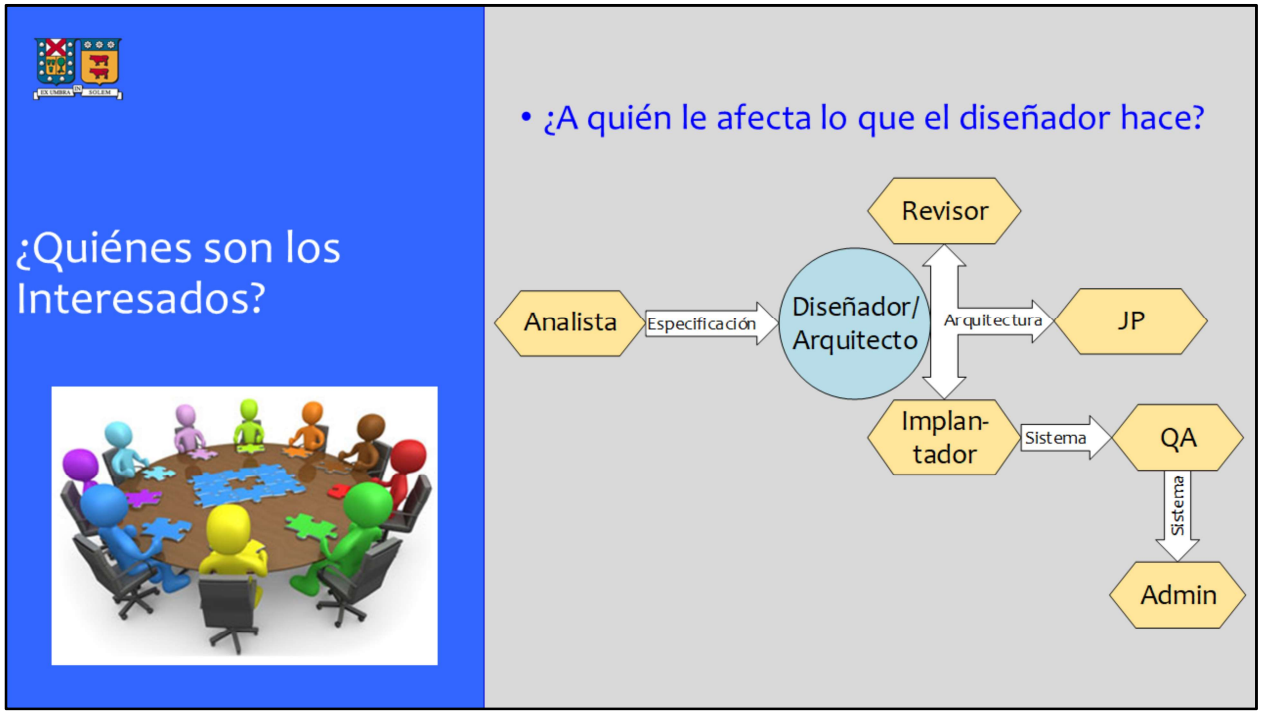
Propósito:

Permitir razonar sobre el sistema y el proceso

Supervisar el proceso de construcción

(usualmente, pero no se requiere)

Ser 'guardián de la visión'



¿A quién le afecta lo que el diseñador hace?

Los afectados son:

- Analista (representa al cliente) (*)
- Implantador (desarrollador) (*)
- Jefe de proyecto
- QA y/o integradores
- Administrador del sistema
- Revisor

¿Por qué es importante el Diseño?

- Planificación/asignación de trabajo. Facilita asignación de módulos y componentes
- Reducción del acoplamiento
- Alta cohesión
- Calidad
- Mantenibilidad

Planificación/asignación de trabajo. Facilita asignación de módulos y componentes a desarrolladores

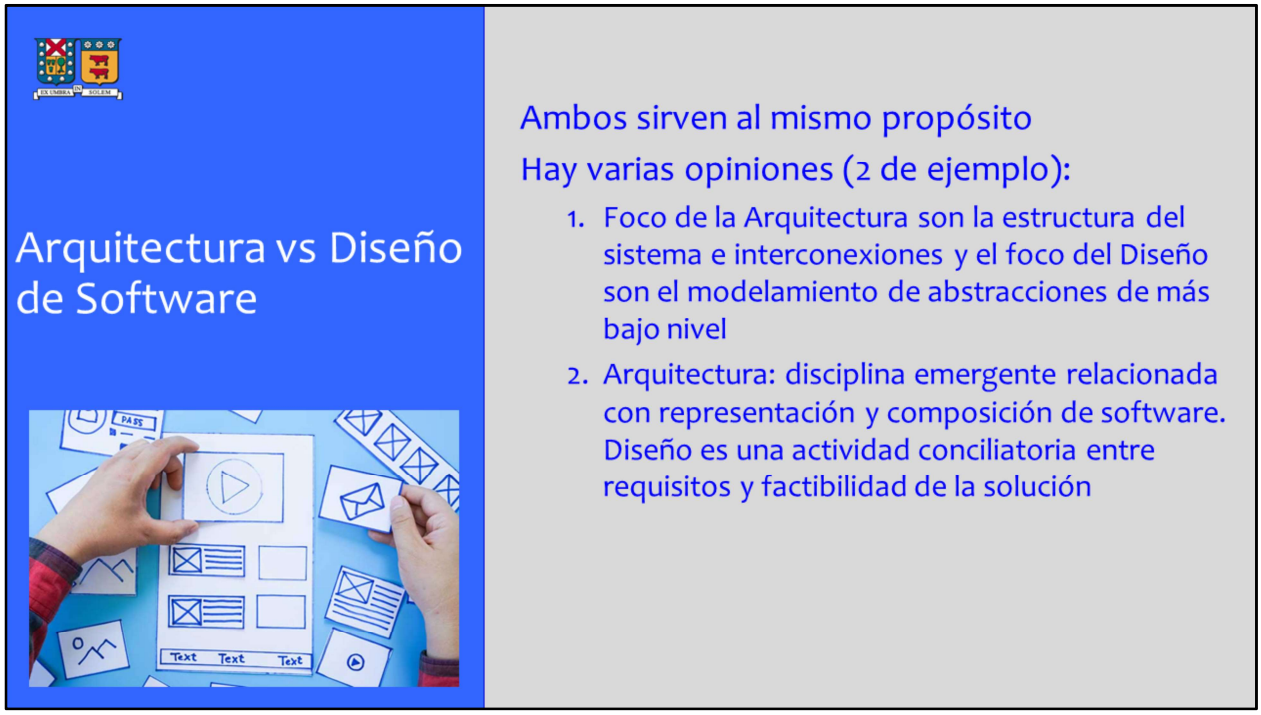
Reducción del acoplamiento:

> Eliminación de dependencias innecesarias
>
> Efecto en la ejecución (sistemas muy acoplados se caen completos si falla un componente)

Alta cohesión. Menos componentes con responsabilidades mal definidas

Calidad. Software más fácil de probar y revisar

Mantenibilidad. Sistemas bien diseñados son fáciles de describir y de abordar



Ambos sirven al mismo propósito

Hay varias opiniones (2 de ejemplo):

1. Foco de la Arquitectura son la estructura del sistema e interconexiones y el foco del Diseño son el modelamiento de abstracciones de más bajo nivel. Al refinarse la Arquitectura, sus conectores se distribuyen en elementos de bajo nivel, resultando en el Diseño.
2. Arquitectura: disciplina emergente del Diseño relacionada con representación y composición de software. Así, el Diseño es una actividad conciliatoria entre los requisitos (función) y la factibilidad de una solución.

6.2 Descomposición y Modularidad

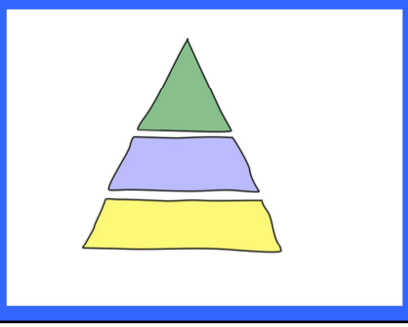
- Diseño determina conjunto de componentes e interfaces que satisfacen requisitos
- Hay varias formas de crear un buen diseño, todas tienen conceptos y métodos comunes
- La **descomposición** es uno de ellos
- Wasserman sugiere 5 maneras:
  a. Descomposición Modular
  b. Descomposición Orientada a los Datos
  c. Descomposición Orientada a los Eventos
  d. Diseño de Afuera hacia Adentro
  e. Diseño Orientado a Objetos
- Diseño se deriva de trabajar una combinación de descripciones en una **jerarquía**

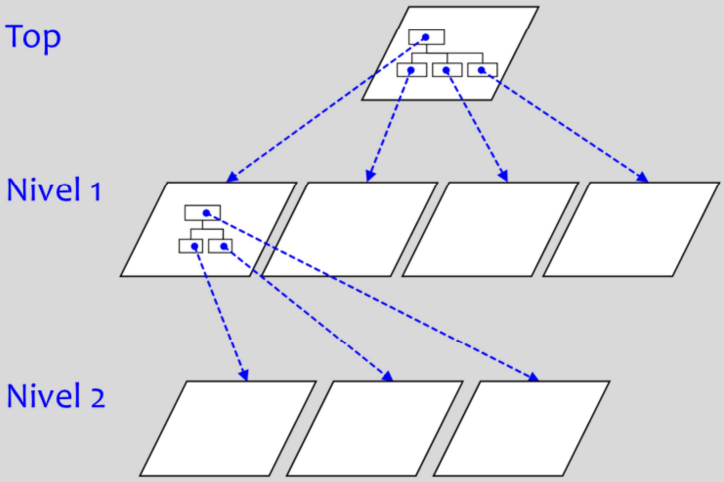
Diseñar un sistema es determinar un conjunto de componentes e interfaces entre componentes que satisfacen un conjunto especificado de requisitos.
Hay varias formas (estrategias) de crear un buen diseño, aunque todas comparten conceptos y métodos comunes.
La descomposición es uno de ellos.
Wasserman sugiere 5 maneras:

- Descomposición Modular.
- Descomposición Orientada a los Datos.
- Descomposición Orientada a los Eventos.
- Diseño de Afuera hacia Adentro.
- Diseño Orientado a Objetos.

Un diseño se deriva trabajando sobre las descripciones de datos, eventos, entradas del usuario, descripciones funcionales, o una combinación de éstos, creando una jerarquía con detalle creciente.

Jerarquía

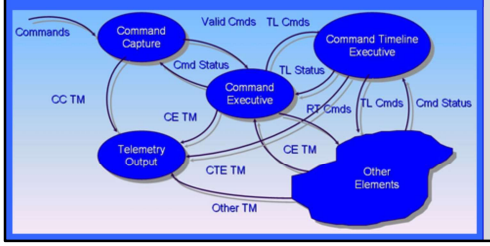




a. Descomposición Modular

- Basada en asignar funciones a componentes
- Diseñador comienza con una descripción de alto nivel de las funciones a implementar
- Luego, construye explicaciones de bajo nivel de cómo se organiza cada componente y cómo se relaciona con los otros componentes

b. Descomposición Orientada a los Datos



- Basado en estructuras de datos externas
- Descripción de alto nivel, muestra estructuras de datos generales
- Descripciones de bajo nivel proporcionan detalle de qué elementos de datos estarán involucrados y cómo se relacionan

c. Descomposición Orientada a los Eventos

- Basado en eventos que el sistema debe manejar y utiliza información de cómo dichos eventos modifican el estado del sistema
- La descripción de alto nivel cataloga los diferentes estados del sistema
- Las descripciones de bajo nivel muestran cómo ocurren los cambios de estado

d. Diseño de Afuera hacia Adentro

- Enfoque de caja negra, basado en las entradas del usuario al sistema
- Descripción de alto nivel lista todas las entradas posibles que un usuario puede hacer
- Descripciones de bajo nivel muestran qué hace el sistema con cada entrada, incluyendo las salidas generadas

Descomposición y Modularidad

- Sin importar el enfoque usado, cada uno separa al diseño en módulos
- Sistema es modular si cada módulo realiza una sola función y sus entradas y salidas están bien definidas
- Componente de diseño es una entidad con entradas y salidas bien definidas
- Componente bien diseñado tiene entradas esenciales a su función y todas las salidas son producidas por sus acciones

Sin importar el enfoque de diseño usado, cada uno separa al diseño en sus partes, llamadas módulos o componentes.

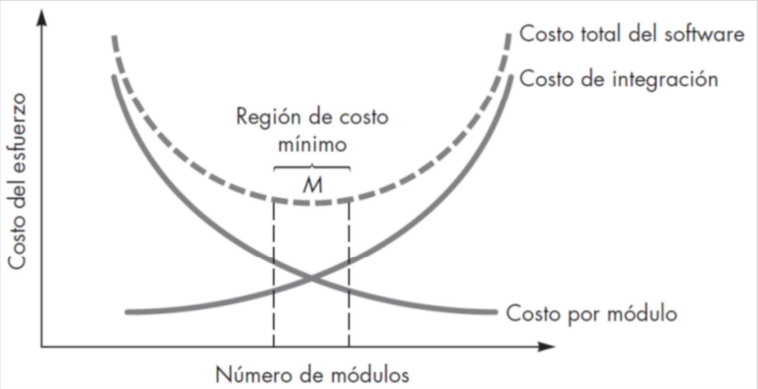
Se dice que el sistema es modular si cada módulo desempeña una sola función y sus entradas y salidas están bien definidas.

Un componente de diseño es una entidad con entradas y salidas bien definidas.

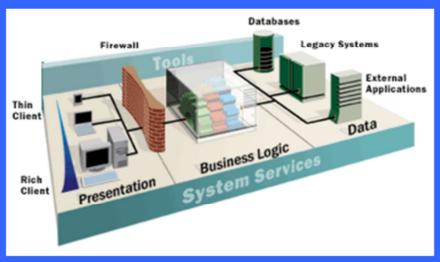
En un componente bien diseñado todas las entradas son esenciales a su función y todas las salidas son producidas por sus acciones.

Costo de Modularizar

No se puede modularizar infinitamente



6.3 Estilos y Estrategias Arquitectónicos



- Arquitectura primer paso en el diseño
- Se distinguen 3 niveles de diseño:
  - **Arquitectura**, asocia capacidades con módulos
  - **Diseño de Código**, algoritmos + datos
  - **Diseño Ejecutable**, lleva diseño a memoria
- Se trabaja en forma iterativa

Arquitectura de software primer paso en el diseño de software

Se distinguen 3 niveles de diseño:

Arquitectura, asocia capacidades del sistema con componentes (módulos)

Diseño de Código, involucra algoritmos y estructuras de datos

Diseño Ejecutable, lleva el diseño de código a nivel de memoria, formatos de datos

Se debe trabajar en forma iterativa

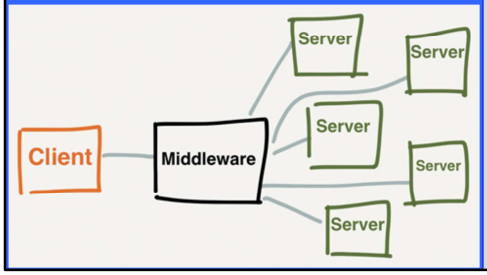
6.3 Estilos y Estrategias Arquitectónicos

a. Capas
b. Cliente/Servidor
c. Maestro/Esclavo
d. Tuberías y Filtros
e. Agente
f. Igual a Igual (Pares)
g. Bus de eventos
h. Modelo Vista-Controlador
i. Pizarra
j. Intérpretes

a. Capas

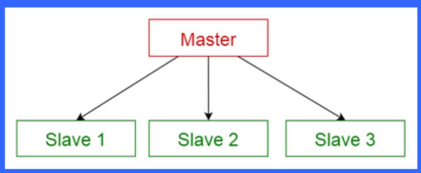
- Cada capa provee servicios a la superior
- Diseño incluye protocolo de interacción
- Toma ventaja de la noción de abstracción
- Permite la reutilización fácilmente
- No es fácil estructurar un sistema en capas
- Relación requisitos – nivel de abstracción de las capas no es evidente
- Interacción entre capas agrega overhead
- Usos típicos:
  - Aplicaciones de escritorio generales
  - Aplicaciones web de comercio electrónico

b. Cliente/Servidor



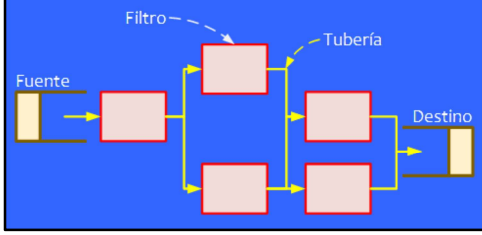
- Los Sistemas Distribuidos son un tipo de arquitectura prevaleciente
- En Cliente/Servidor la funcionalidad de la aplicación se particiona en dos o más:
  - **Cliente**, interactúa con usuario y genera requerimiento de servicio
  - **Servidor**, responde a requerimientos
- Arquitectura extensible
- Alta heterogeneidad de plataformas
- Usos:
  - Aplicaciones en línea como correo electrónico, uso compartido de documentos y banca

c. Maestro/Esclavo



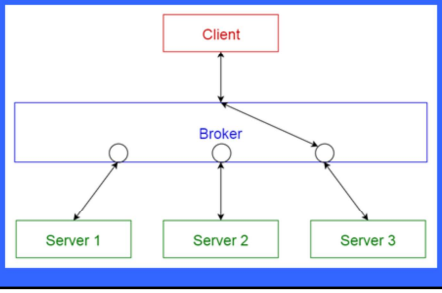
- Consiste en dos partes: maestro y esclavos
- Componente maestro distribuye el trabajo entre componentes esclavos idénticos y calcula el resultado final de los resultados que éstos devuelven
- Usos:
  - En la replicación de la base de datos, la BD maestra se considera como la fuente autorizada y las esclavas se sincronizan con ella
  - Periféricos conectados a un bus (unidades maestra y esclava)

d. Tuberías y Filtros



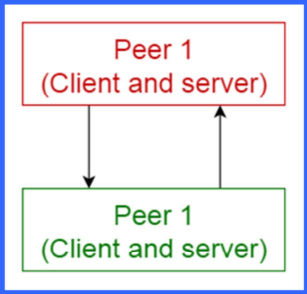
- La transformación de entradas en salidas se realiza en el componente (filtro), que tiene asociados flujos de datos (tuberías) a las E/S
- Filtros independientes, no importa el orden
- Se pueden agrupar filtros (reusabilidad), lo que facilita la evolución de un sistema
- Usos:
  - Compiladores. Filtros consecutivos realizan análisis léxico y sintáctico y generan el código
  - Flujos de trabajo en bioinformática
  - Mallas de Sistemas Batch

e. Agente



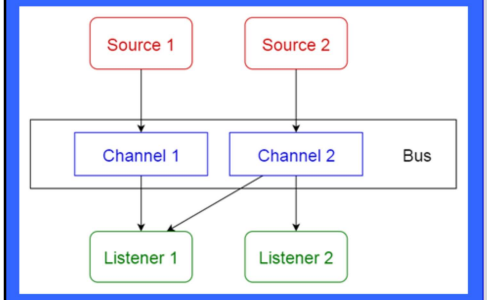
- Estructura sistemas distribuidos con componentes desacoplados
- Componentes pueden interactuar entre sí mediante invocación de servicios remotos
- Intermediario es responsable de coordinar la comunicación entre componentes
- Servidores publican sus capacidades a un intermediario
- Clientes solicitan servicio del intermediario y éste redirecciona el cliente a un servicio
- Usos
  - Message Broker como Apache ActiveMQ, Apache Kafka, RabbitMQ y JBoss Messaging

f. Igual a Igual (Pares)



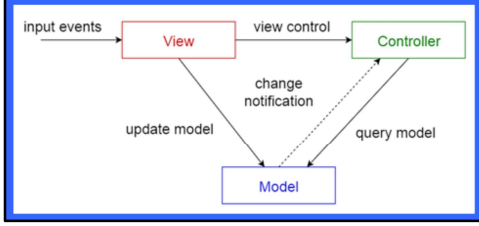
- Componentes actúan como **pares**
- Pueden funcionar como **cliente**, solicitando servicios de otros pares, y **servidor**, proporcionando servicios a otros pares
- Un par puede actuar como un cliente o como un servidor o como ambos y puede cambiar su rol dinámicamente en el tiempo
- Usos:
  - Redes de intercambio de archivos (Gnutella, G2)
  - Protocolos multimedia (P2PTV, PDTP)

g. Bus de eventos



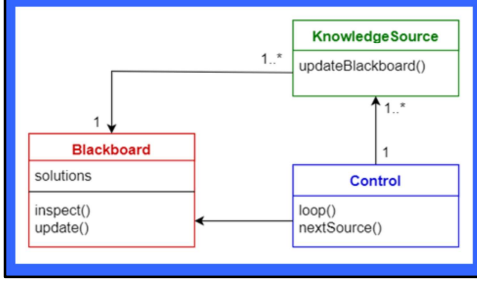
- Trata principalmente con eventos y tiene 4 componentes principales:
  - Fuente de evento
  - Escucha de evento
  - Canal
  - Bus de evento
- Fuentes publican mensajes en canales particulares en un bus de eventos
- Escucha se suscriben a canales y son notificados de los mensajes que se publican
- Usos:
  - Desarrollo de Android
  - Servicios de notificación

h. Modelo Vista-Controlador (MVC)



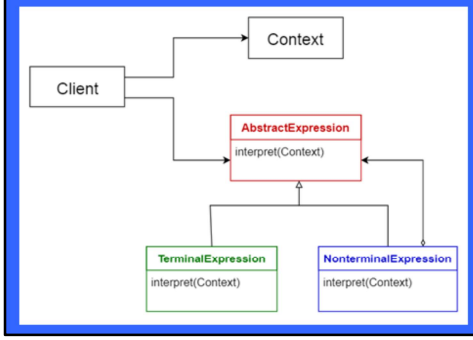
- Divide aplicación interactiva en 3 partes:
  - **Modelo** contiene funcionalidad y datos básicos
  - **Vista** muestra información al usuario
  - **Controlador** maneja la entrada del usuario
- Separa representaciones internas de información de cómo se presenta y acepta la información del usuario
- Desacopla componentes y permite reutilización eficiente del código
- Usos:
  - Arquitectura para aplicaciones WWW en los principales lenguajes de programación.
  - Marcos web como Django y Rails

i. Pizarra



- Útil cuando no hay estrategias de solución
- Consta de 3 componentes principales:
  - **Pizarra**: memoria global con objetos de solución
  - **Fuente** de conocimiento: módulos especializados
  - **Control** : selecciona, configura y ejecuta módulos
- Componentes acceden a pizarra y producen nuevos objetos que se agregan a ésta
- Componentes buscan tipos de datos en la pizarra por calce de patrones con la fuente
- Usos:
  - Reconocimiento de voz
  - Identificación y seguimiento de vehículos
  - Identificación de la estructura proteica

j. Intérpretes



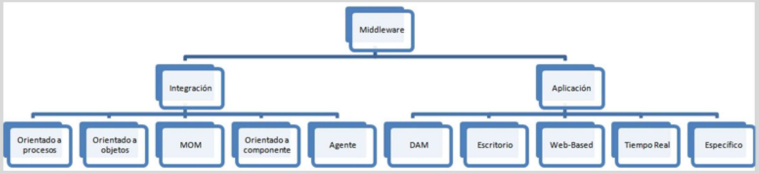
- Usado para diseñar componente que interpreta programa escritos en un lenguaje
- Especifica cómo evaluar sentencias
- Idea: tener una clase para cada símbolo
- Intérprete acepta string en pseudocódigo, lo convierte en código real y lo ejecuta
- Estilo restringido a ciertos tipos de problema
- Usos:
  - Lenguajes de consulta de base de datos (SQL)
  - Idiomas utilizados para describir protocolos de comunicación

6.4 Tópicos en la creación del Diseño

- Al crear el diseño, hay varias preguntas que responder, entre ellas:
  - ¿Qué es lo mejor para la aplicación?
  - ¿Qué es cómodo para el diseñador?
  - ¿Qué tiene sentido en la arquitectura global?
- No hay un "mejor" estilo o método absoluto
- Cada día emergen nuevos
- Hay varios temas que abordar y problemas que enfrentar
  a. Modularidad y niveles de abstracción
  b. Diseño Colaborativo
  c. Diseño de Interfaz de Usuario
  d. Concurrencia
  e. Patrones de Diseño y Reutilización

a. Modularidad y Niveles de Abstracción

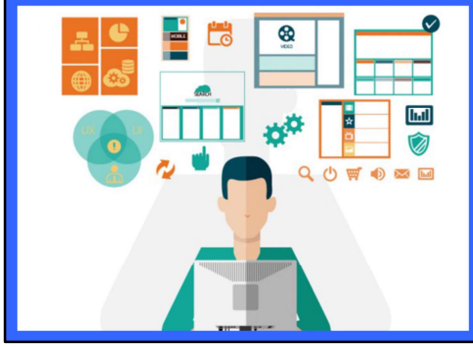
- **Modularidad**: característica del buen diseño, donde los componentes tienen un propósito, entradas y salidas bien definidos
- En la descomposición, los componentes de un nuevo nivel refinan los del nivel superior
- El primer nivel es el más abstracto y el último el más concreto
- Cada nivel le oculta al superior los detalles
- Cada nivel da una visión diferente del sistema



b. Diseño Colaborativo

- Diseño es una actividad en equipo
- Personas trabajan en modo colaborativo
- Cada persona se encarga de una parte
- Comunicación es fundamental
- Diferencias personales generan problemas
- Retroalimentación es fundamental
- El proceso, además, es iterativo

c. Diseño de Interfaz de Usuario



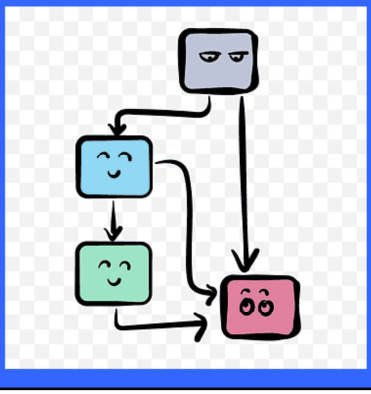
- Interfaz de usuario compleja, las personas tienen diferentes estilos de percepción, comprensión y trabajo
- Interfaz aborda 5 elementos clave: metáfora, modelo mental, modelo de navegación, apariencia y forma de interacción
- Interfaz debe considerar 2 problemas clave para ser efectiva y cómoda: cultura y preferencias del usuario
- Para determinar características de la interfaz, hay varios otros aspectos a considerar

c. Diseño de Interfaz de Usuario

Estilos de interacción

Estilo	Ventajas	Desventajas	Ejemplos
Manipulación directa	•Interacción intuitiva •Fácil de aprender	•Difícil de implementar •Sólo útil cuando hay una metáfora visual	•Juegos de video •Sistemas CAD
Selección de menú	•Evita errores •Requiere poco tipeo	•Lento para expertos •Complejo si hay muchas opciones	•Mayoría de sistemas de propósito general
Llenado de formularios	•Entrada de datos simple •Fácil de aprender •Verificable	•Ocupa mucha pantalla •Problema si opciones no coinciden con los campos a llenar	•Control de inventario •Procesamiento de préstamos
Lenguaje de comandos	•Poderoso y flexible	•Difícil de aprender •Mal manejo de errores	•Sistemas operativos •Sistemas de control
Lenguaje natural	•Accesible a usuarios casuales •Extensible	•Requiere mucho tipeo •Sistemas poco confiables	•Sistemas de recuperación de información

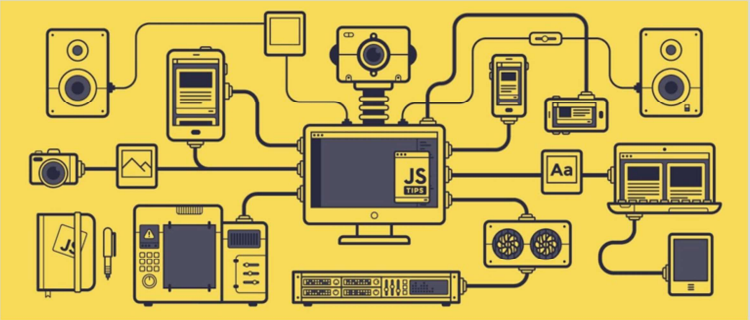
d. Concurrencia



- En muchos sistemas, las acciones son concurrentes, en vez de secuenciales
- Principal problema: potencial pérdida de consistencia de los datos, producto de las acciones concurrentes
- Necesidad de sincronizar procesos para permitir que dos o más actividades concurrentes no se estorben mutuamente
- Uso de exclusión mutua, monitores y guardianes

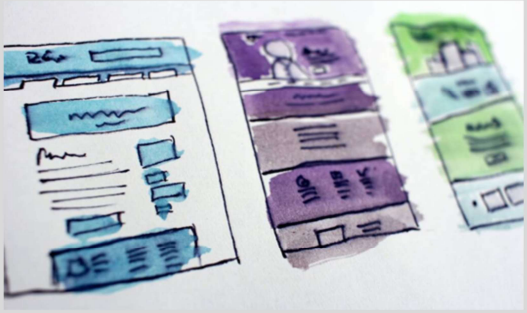
e. Patrones de Diseño y Reutilización

- Al diseñar un nuevo sistema, éste se puede parecer, en algún grado, a otros ya diseñados
- Patrones de diseño abordan este tema
- Un patrón es una **estructura de diseño común** que permite que ésta sea reusable

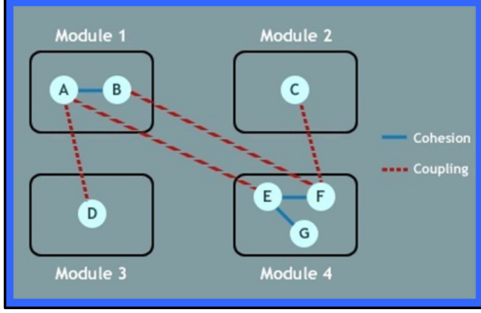


6.5 Características de un buen Diseño

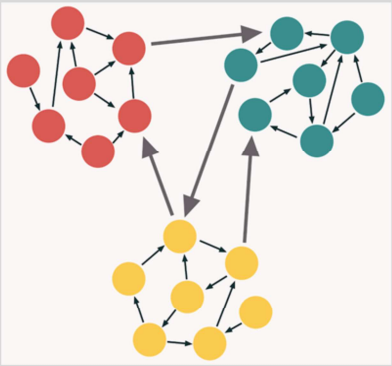
- Un buen diseño debería conducir a un producto de **calidad**
- Un buen diseño debe tener:
  a. Independencia de Componentes
  b. Identificación y manejo de excepciones
  c. Prevención y Tolerancia a fallas

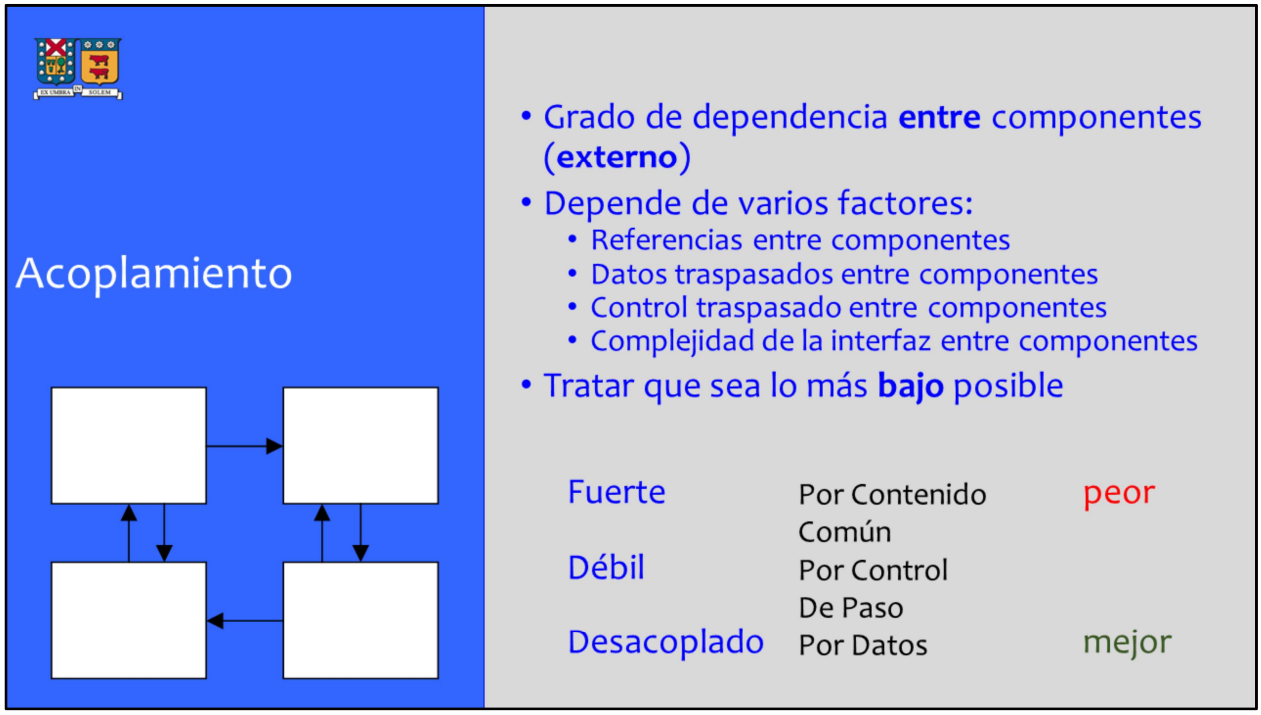


a. Independencia de Componentes



- **Abstracción** y **Ocultamiento de Información** permiten examinar componentes y sus relaciones
- Independencia de componentes se reconoce y mide a través de:
  - **Acoplamiento**
  - **Cohesión**





Por Contenido
Un componente modifica un dato de otro componente.
Puede ser modificando datos directamente o "saltando" a ese componente mediante GO TO.
Es el menos deseable de todos.

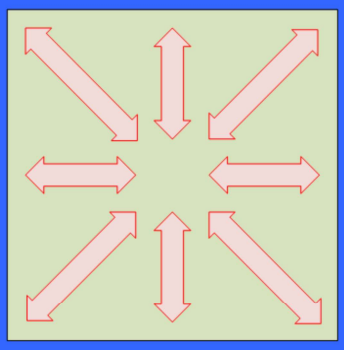
Común
Dos o más componentes usan un dato común, inicializado en otro.
Esto sucede típicamente con el uso de variables globales.
También con el uso de archivos sin resguardo.

Por Control
Un componente traspasa datos a otro para controlarlo (variables de control).
El componente que recibe las variables de control se comporta según el valor que éstas tienen
Es decir, el subordinado realiza más de una función, esto es controlado por el que lo invoca.

De Paso
Un componente traspasa los datos a otro, a través de una estructura.
Aquí no sólo se traspasan datos, sino también estructura, lo cual debe ser conocido por el invocado
Esto hace que la interfaz entre ambos sea poco clara.

Por Datos
El traspaso de datos es normal, usando el traspaso de parámetros explícito.
El número de parámetros debería ser el mínimo.
Es el más deseable de todos.

Cohesión



- Grado de integración **dentro** de cada componente (**interno**)
- Un módulo debe hacer una y sólo una función
- Extensión natural del ocultamiento de la información
- Tratar que sea lo más **alta** posible

Alta	Funcional	mejor
	Secuencial	
	Comunicacional	
	Procedural	
	Temporal	
	Lógica	
Baja	Coincidental	peor

Funcional
Cada elemento de procesamiento es esencial para realizar la función.
Es el grado más alto de cohesión.

Secuencial
Cuando la salida de una parte de un componente es la entrada de otra parte del mismo.

Comunicacional
Un conjunto de funciones que opera sobre el mismo conjunto de datos.
Los datos se acceden todos juntos, para acelerar la E/S y luego son procesados por funciones sin relación.

Procedural
A menudo, ciertas funciones deben llevarse a cabo en un orden determinado.
Ejemplo, todo dato debe ser ingresado, validado y manipulado, siempre en ese orden.

Temporal
Es un conjunto de funciones que ocurren en instantes similares.
Caso típico: inicialización de variables. Pero una variable debe inicializarse lo más tarde posible.

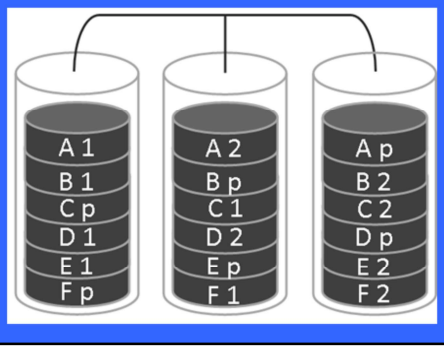
Lógica
Es sólo un conjunto de funciones similares.
Ejemplo: validaciones o impresión de líneas de detalle, sin importar en ambos casos a quién afecta.

Coincidental
Un componente contiene funciones no relacionadas, es sólo por conveniencia de alguien.
Es la peor de todas.

b. Identificación y manejo de excepciones

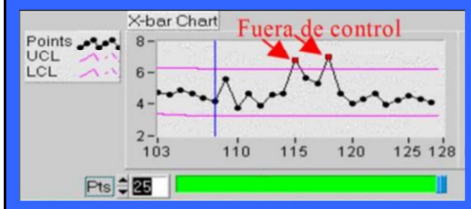
- Especificación de requisitos dice qué hace el sistema, pero no lo que **NO** hace
- Ejemplo clásico: aspectos de seguridad
- Para darle cabida, es necesario primero identificar todas las excepciones conocidas
- Luego, diseñar el sistema de manera que éste maneje adecuadamente dichas excepciones, sin degradar el sistema

c. Prevención y Tolerancia a fallas



- Un error de desarrollo se manifiesta en una falla, la cual puede propagarse, alterando la ejecución de código correcto
- Un buen diseño debe tratar de anticiparse a las fallas (no esperar que el programa se caiga) y manejarlas minimizando las perturbaciones y maximizando la seguridad
- Comprende:
  i. **Detección Activa**
  ii. **Corrección**
  iii. **Tolerancia**

i. Detección Activa de Fallas



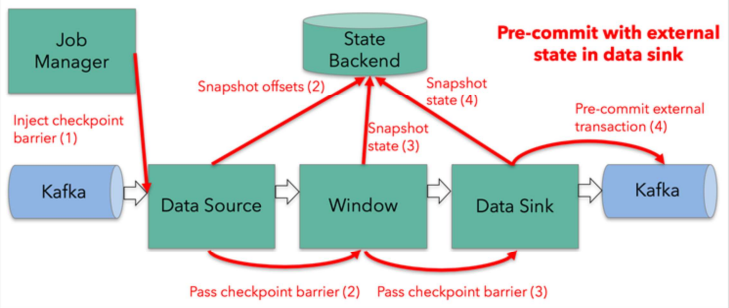
- Se realiza **antes**
- Hay varias técnicas
- Una es la sospecha mutua, donde las **validaciones** son una forma típica
- Otra es la utilización de **redundancia** de código y/o datos, como la cuadratura contable o la suma de chequeo

ii. Corrección de Fallas

- Se realiza **después**
- Es la compensación del sistema frente a la presencia de una falla
- La corrección puede hacerse de inmediato o después de detectada la falla
- La criticidad del sistema determina qué estrategia utilizar
- Mantenimiento del sistema es otro factor a considerar para decidir la estrategia a usar

iii. Tolerancia a Fallas

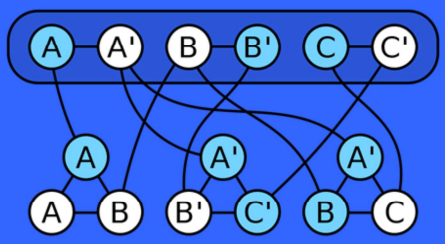
- Muchas veces, corregir una falla puede ser demasiado caro, riesgoso o inconveniente
- En vez de ello, el sistema continuará operando sin perturbar a los usuarios
- Tolerancia a fallas: aislar del daño provocado



6.6 Técnicas de mejoramiento del Diseño

- Para especificar el Diseño, se usan técnicas similares a la Especificación de Requisitos
- Se utilizan varias notaciones adaptadas para expresar el cómo del Diseño
- Hay varias formas de mejorar el Diseño:
  a. **Reducción de Complejidad**
  b. **Diseño por Contrato**
  c. **Prototipación del Diseño**
  d. **Análisis de Árbol de Fallas**

a. Reducción de Complejidad

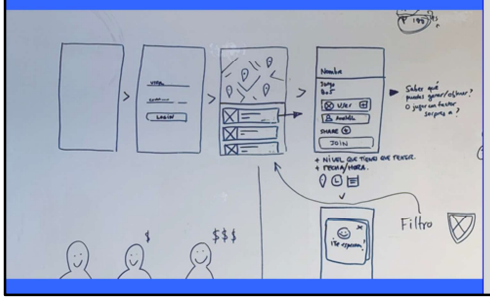


- Al crear un nuevo diseño, se desea mejorar su estructura, sin cambiar su naturaleza
- Esto permite comprender más fácilmente la solución
- Más que el número de elementos o de relaciones, lo que se reduce son los entrecruzamientos entre conectores
- Es equivalente a la factorización en Álgebra

b. Diseño por Contrato

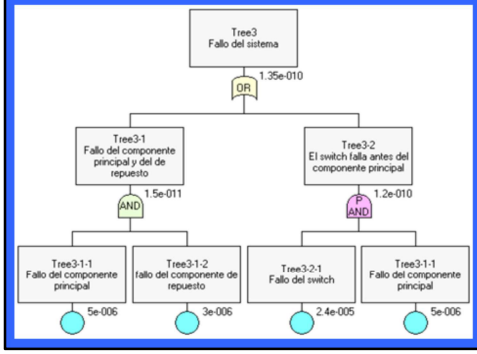
- Software: conjunto de componentes que se comunican, cuya interacción está basada en contratos
- Contrato gobierna interacción de un componente con los demás
- Componente cliente adopta estrategia para ejecutar un conjunto de tareas, invocando a otros componentes, los proveedores
- **Contrato** tiene características, que comprenden obligaciones, beneficios y restricciones de consistencia

c. Prototipación del Diseño



- A veces, cliente requiere subsistema o característica no fácil de obtener
- La prototipación permite evaluar el diseño, de manera similar al análisis de requisitos, para ver si realmente resolverá el problema
- Prototipo omite detalles de funcionalidad y rendimiento del sistema real y permite enfocarse en los aspectos a evaluar
- Es un modelo exploratorio, no un producto

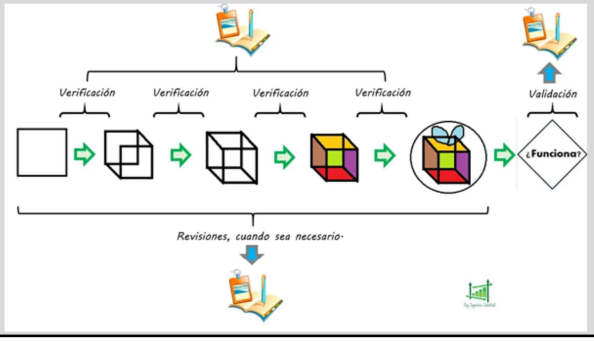
d. Análisis de Árbol de Fallas



- Importante considerar la identificación, corrección y tolerancia a fallas
- Análisis de árbol de fallas permite buscar y encontrar causas potenciales de fallas
- En este árbol, los nodos representan las fallas y los enlaces las relaciones entre fallas
- Al recorrer el árbol, se puede establecer la relación entre una falla y su(s) causa(s)
- También se pueden usar para calcular la probabilidad de ocurrencia de una falla

6.7 Evaluación y Validación del Diseño

- Luego de diseñar un sistema, éste se chequea
  - que satisfaga todos los requisitos
  - que incorpore las características de buen diseño
- Hay algunas técnicas que ayudan:
  a. Medición de la Calidad del Diseño
  b. Comparación de Diseños
  c. Revisiones del Diseño



a. Medición de la Calidad del Diseño

- Hay propuestas de medir ciertos aspectos clave de la calidad de un diseño; por ejemplo: cohesión y acoplamiento
- La mayoría está definida en el contexto de la orientación a objetos

b. Comparación de Diseños

PRICING PLANS	STARTER	BUSINESS	PREMIUM
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean ligula eget dolor. Aenean massa. Cum sociis natoque penatibus	✓	✓	✓
Magnis dis parturient montes, nascetur ridiculus mus. Donec quam. ultricies nec, pellentesque eu, sem.	✓	✓	✓
Nulla consequat massa quis enim. Donec pede justo, fringilla vel.	✗	✓	✓
Vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet, venenatis, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt.	✗	✓	✓
Vivamus elementum semper nisi, vulputate eleifend	✗	✓	✓
Etiam rhoncus, sem quam semper libero, sit amet • aenean lectus ligula porttitor eu, consequat vitae eleifend	✗	✗	✓
• Iquam lorem ante, dapibus in, viverra quis, feugiat a, tellus	✗	✗	✓
Nulla consequat massa quis enim. Donec pede justo, fringilla velat, aliquet nec Vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet.	✗	✗	✓

- Medir atributos de diseño para probar que una nueva versión es mejor que la anterior
- Una misma especificación puede conducir a muchos diseños, pero ¿cuál es mejor?
- Una forma es usar tablas de comparación:
  - Cada fila es un atributo y cada columna es una estrategia de diseño
  - Cada celda es el puntaje de cómo se logra un atributo a través de un diseño particular
  - Cada fila tiene una ponderación, basada en la importancia del atributo medido
  - Finalmente, se suman las columnas en forma ponderada

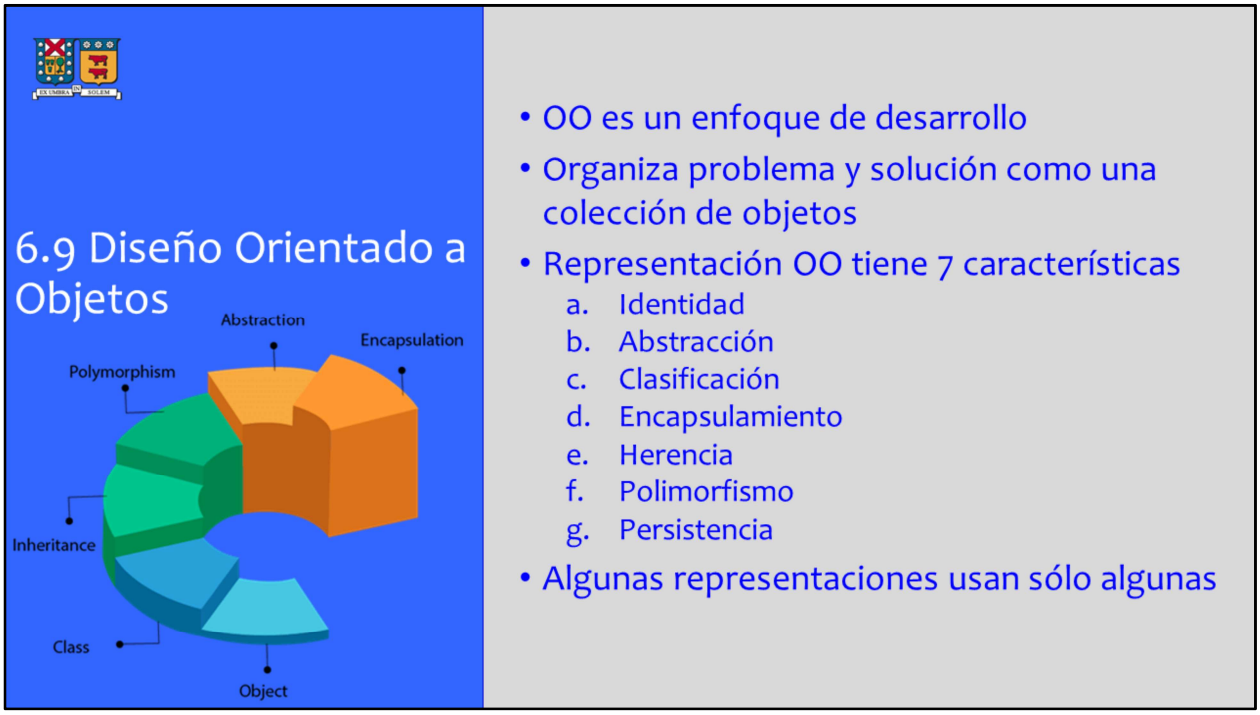
c. Revisiones del Diseño

- Es un proceso de tres pasos:
  - **Revisión Preliminar**. Se examina el Diseño Conceptual, junto con los clientes
  - **Revisión Crítica**. Se presenta el Diseño Técnico a otros desarrolladores
  - **Revisión de Programas**. Se analiza con los programadores y se retroalimentan
- En cada revisión, responder preguntas:
  - ¿Es una solución al problema?
  - ¿Es modular, bien estructurado y entendible?
  - ¿Se puede mejorar?
  - ¿Es portable?
  - ¿Reutiliza y es reusable a la vez?
  - ¿Es fácil de modificar/expandir?
  - ¿Está bien documentado?

6.8 Documentación del Diseño

- Importancia de documentos que describen el sistema a ser construido
- Uno para clientes y otro para desarrolladores
- Ambas partes pueden superponerse
- Uso de diagramas mostrando abstracción
- Si es distribuido, incluir topología de red
- Si cliente pide, incluir medidas de rendimiento
- Diseño y Análisis de requisitos se deben referenciar mutuamente, para consistencia

Un producto importante del proceso de diseño es un conjunto de documentos que describen el sistema a ser construido.
Una parte le debe indicar a clientes y usuarios (en lenguaje sencillo) qué hará el sistema.
La otra le debe indicar a los desarrolladores (en lenguaje técnico) la estructura, los datos y las funciones del sistema.
Ambas partes pueden superponerse.
La utilización de diagramas es muy frecuente, mostrando todos los niveles de abstracción.
Si el sistema es distribuido, se debe incluir también la topología de red, la relación entre nodos y la ubicación de éstos.
Si el cliente lo solicita, se deben incluir medidas de rendimiento.
El diseño y el análisis de requisitos se deben referenciar mutuamente (referencia cruzada), para chequear consistencia.



La *orientación a objetos* (OO) es un enfoque de desarrollo de software

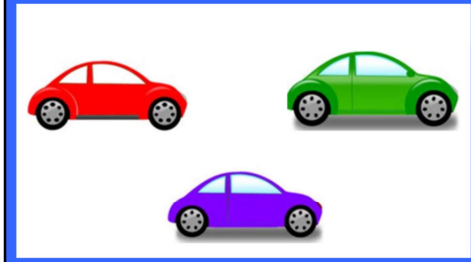
Organiza el problema y su solución como una colección de objetos discretos. Tanto la estructura de datos como su comportamiento están incorporados en la representación.

Una representación OO tiene 7 características:

1. Identidad
2. Abstracción
3. Clasificación
4. Encapsulamiento
5. Herencia
6. Polimorfismo
7. Persistencia

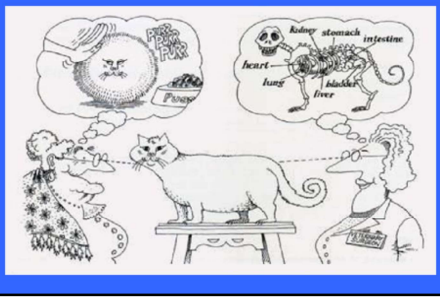
Algunas representaciones usan sólo algunas de estas características. Por ello, son denominadas “basadas en objetos”.

a. Identidad



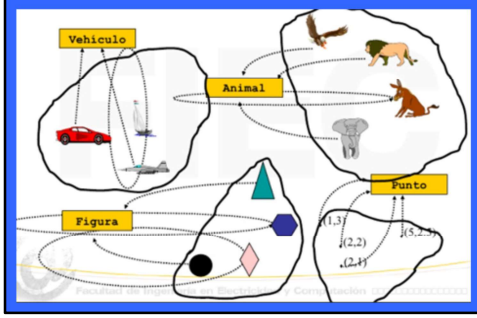
- Se refiere a que los datos se organizan en entidades discretas y distinguibles (objetos)
- Un objeto tiene estados y comportamientos asociados a éste
- Cada objeto tiene un nombre o referencia, para distinguirse de los demás

b. Abstracción



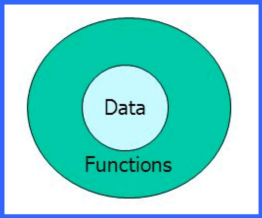
- La abstracción es esencial para construir cualquier sistema, sea OO o no
- La abstracción en un sistema OO ayuda a representar los diferentes puntos de vista incorporados en el sistema en desarrollo
- Juntas, las abstracciones forman una jerarquía que muestra cómo las diferentes perspectivas se relacionan unas con otras

c. Clasificación



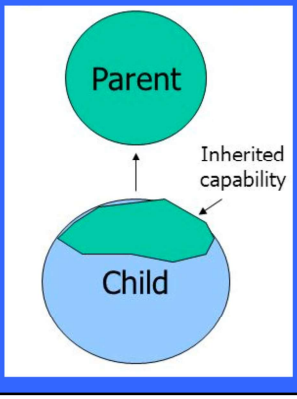
- Un grupo de objetos con características comunes se puede considerar una clase
- La OO utiliza la clasificación para agrupar objetos y asociar atributos y operaciones (métodos) a cada clase
- Cada objeto es una instancia de una clase
- Cada instancia tiene sus propios valores asociados a los atributos, pero comparte los nombres de los atributos y métodos con las demás instancias de la clase

d. Encapsulamiento



- La OO utiliza el encapsulamiento para indicar que, tanto la estructura (los atributos) como el comportamiento (los métodos) están ocultos del exterior
- La cápsula protectora puede ser transparente u opaca

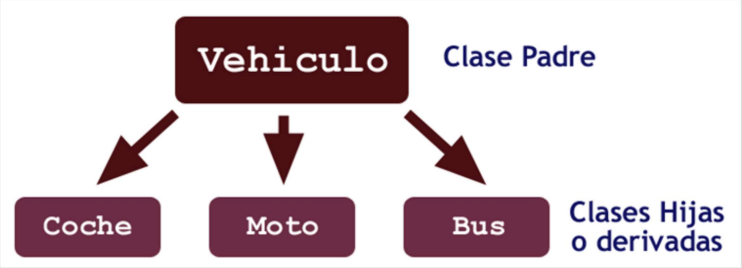
e. Herencia



- En la jerarquía de clases, toda clase hereda su estructura y comportamiento a sus respectivas subclases
- En esta jerarquía, es posible apreciar las similitudes y diferencias entre subclases

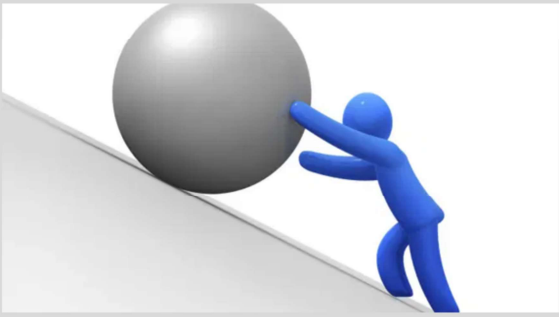
f. Polimorfismo

- Comportamiento: acción de un objeto
- Es gatillado por la recepción de un mensaje
- A veces, un mismo comportamiento puede verse diferente por las clases y subclases
- Puede implementarse por más de un método

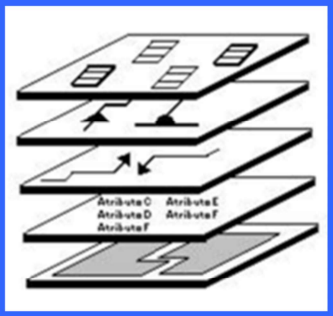


g. Persistencia

- Es la capacidad del nombre, estado y comportamiento de un objeto de trascender en tiempo y espacio
- Estas características se mantienen, a medida que el objeto va siendo transformado

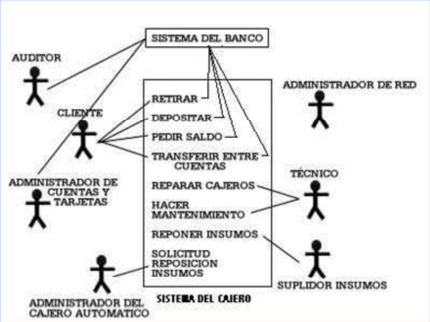


Proceso de Desarrollo OO



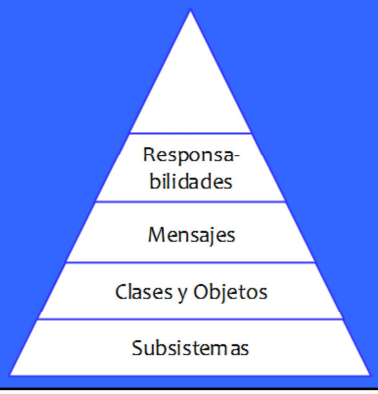
- Ventaja: consistencia del lenguaje, describe el problema y la solución en iguales términos
- Descripción de clases desde 3 perspectivas:
  - **Estática** Objetos (atributos, comportamiento, …)
  - **Dinámica** Comunicación, control, estados, etc
  - **Restrictiva** Cardinalidad, comportamiento dinámico
- OO es una filosofía de representación, no un ciclo de vida en sí, puede ser usada en éstos

i. Requisitos



- El Análisis de Requisitos OO se hace en el lenguaje del usuario y discute los conceptos y escenarios más probables en el dominio de la aplicación
- Los conceptos incluyen: Información, Servicios y Responsabilidades
- El conocimiento del dominio permite a los desarrolladores entender el contexto en que será usado el sistema y describir los requisitos de manera que el usuario los entienda

ii. Diseño



- Normalmente, la Especificación de Requisitos se representa en OO para que los diseñadores generen la solución
- Descripción muy similar a la elaboración primaria de la solución
- Hay dos guías para esta representación:
  - Identificar y representar clases y objetos
  - Identificar interacciones y relaciones objeto-clase
- Diseño del Sistema es una abstracción de alto nivel de lo que será el diseño de programa

iii. Codificación y Testing

- Completado el diseño de programas, se describe el sistema mediante modelos de objetos, atributos y comportamientos
- La codificación traduce estos modelos a través de un lenguaje de programación OO, lo cual requiere un refinamiento previo de los modelos
- El testing no es muy diferente al usado en otros enfoques
- Se lleva a cabo el testing unitario y luego el integrado

¿Preguntas?