Arquitectura de SW

→ Define la organización de un sistema en base a sus componentes, sus relaciones, evolución y diseño.

→ Requerimientos funcionales: lo que debe hacer el sistema en sí.
→ Requerimientos no funcionales: no son obligatorios ni necesarios.

Requerimientos no funcionales:

(R) **Performance**: desempeño de la cantidad de recursos utilizados bajo condiciones
(R) **Escalabilidad**: capacidad del sw de tratar con multiples usuarios.
(A) **Mantenibilidad**: capacidad del sw para ser modificado.
(R) **seguridad**: el sw debe manejar un cierto nivel de seguridad.
- seguridad del usuario: autentificacion (login) / autorizacion (funciones usuario).
- seguridad de los datos: encriptacion (cifrado) / integridad (datos =) / no repudio.

(R) **confiabilidad**: sw disponible la mayor parte del tiempo, regla de los 5 nueves
(A) **Integrabilidad**: capacidad para intercambiar info con otro, permitir agregar componentes
(R) **Portabilidad**: capacidad del sw de ser transpuesto de forma eficiente
(R) **Verificabilidad**: el sistema debe ser capaz de autochequearse.
(A) **Soportabilidad**: diagnosticar los errores, ser capaz de identificar errores y corregirlos del sistema.

Arquitecturas

- modelos de estructuración:

1 modelo de repositorio: grandes cantidades de datos que deben ser compartidos

Dos formas de almacenaje:
centralizado: los datos en un mismo lugar.
distribuido: los datos en distintos lugares.

Dos formas de actuar:
modelo pasivo: responde a los requerimientos de datos cuando llegan y los recibe.
modelo activo: se realizan cosas constantemente, revisa datos y avisar que hacer

ventajas:
eficiente de compartir datos, escalable y facil de mantener.
desventajas:
punto unico de falla, dificil cambio de un modelo de datos.

2 modelo cliente-servidor: se basa en servidores, ofrecen servicios específicos en donde los clientes requieren servicios, se comunican por redes.

ventajas:
procesamiento distribuido, datos distribuidos, escalable horizontalmente
desventajas:
datos no compartidos, cd de datos en cada server, no hay centralización

3. modelo de capas: conjunto de capas que ofrecen servicios específicos, cada capa tiene una interfaz bien definida

ventajas

desarrollo incremental, flexible y mantenible

desventajas

difícil de estructurar, baja performance, dependencia cruzada.

4. Objetos distribuidos: cada componente puede ser cliente o servidor cada objeto provee y recibe servicios, objetos se comunican por ORB.

ventajas:

diseño flexible, escalable, fácil de agregar objetos

desventajas

complejo, difícil construcción, baja performance.

5. Arquitectura Orientada a Servicios (SOA):

- evolución del modelo cliente servidor.
- servicios administrados centralizadamente, conecta aplicaciones como servicios, flujo de datos controlados.
- Tres componentes: cliente, bus, servicio
- clientes y servicios hablan con el bus esperando que pase algo.
- El bus puede transportar un pedazo de requerimiento en algo que una aplicación específica entienda.
- Integra sistemas existentes.

ventajas:

reutiliza sistemas existentes, bajo nivel de acoplamiento

desventajas:

alta de dependencia de estándares, baja performance, punto único de falla

6. Arquitectura Cloud: se externalizan los servicios computacionales. infraestructura (IaaS), Plataforma (PaaS), Aplicación (Saas)

ventajas:

reducción de costos, alta disponibilidad, servicios ubicuos.

desventajas:

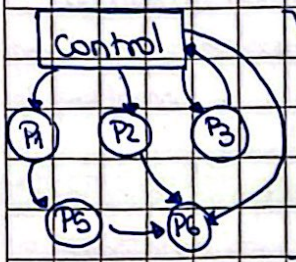
dependencia de entes externos (seguridad, conectividad, confidencialidad)

Modelos de Control

1. modelo call return: invoco al proceso y espero la respuesta.

- es simple, predecible, rígido, testeable, bloqueante.
- se tiene que esperar a que el proceso termine para que empiece el otro.

2. modelo administrador: el componente de control es mas inteligente y puede ejecutarse en simultanea, por ende, es ahora una malla de procesos.
• proceso no bloqueante, coordinacion de procesos, logica centralizada.



• malla de manejo de procesos:
→ coordina la ejecucion de cada proceso de acuerdo a la malla.
→ Para solucionar el cuello de botella o punto unico de falla, se duplica el control.

Modelo de Control Basado en Eventos
• evento: alarma que se activa en el sistema ante la ocurrencia de un fenomeno predecido o no.
• control descentralizado: todos controlan o nadie controla.

1. modelo de transmisión multiple:
• es modelo publisher-suscriber.
• componentes distribuidos: gabilla eventos, suscribe eventos, publica eventos.
• cuando el evento se gabilla, no se sabe si sera procesado.

ventaja:
activacion descentralizada, evolucion simple, distribucion de componentes.

desventaja:
respuesta incierta, varios manejadores para el mismo evento.

2. modelo manejo de interrupciones
• sistema en tiempo real
• un manejador para cada tipo de interrupcion.
• respuesta inmediata, procesamiento paralelo, no bloqueante.

Descomposicion modular
• componentes se dividen en modulos
1. modelo de objetos: Clases debilmente acopladas, interfaces bien definidas, objetos creados a partir de las clases, operaciones se coordinan entre objetos.
2. modelo de flujo de datos: se descomponen en procesos funcionales, transforman entradas en salidas, procesos dependientes, flujo predeterminado.

Arquitecturas de dominio especifico:
1. Arquitectura genericas: generalizacion de sistemas reales, analisis de sistemas existentes.
2. Arquitectura de referencias: estudios del dominio de una aplicacion.

Patrones de Arquitectura

- Son esquemas genericos.
- dan solución de diseño a un problema
- Para los patrones se determina:

- nombre del patron, contexto: situacion en la que se origina el problema, problema: descripcion generica del problema, solucion: estructura que soluciona el problema.

* Patrones Simples

a. Capas: se estructura aplicaciones descomponendolas en tareas con diferentes niveles de abstraccion.

- el contexto es que el sistema esta estructurado con diversos niveles de accion.
- problema: organizacion inadecuada
- solucion: estructurar en un esquema multicapas. ; se usa una capa base, luego se avanza capa a capa utilizando los servicios ofrecidos en la capa anterior; los componentes estan estructurados en modulos relacionados
- caracteristicas: capa solo habla con la capa anterio, no hay dependencias entre capas, componentes interactuan entresi pero queda acoplado.
- implementacion: determinar numero de capas, asignar responsabilidades a cada capa, especificar servicios y establecer metodo de comunicacion entre capas.
- ventajas: cambios afectan solo a nivel local, reutilizacion de capas
- desventajas: ineficiente, cambios afectan a la cascada de proceso.

b. Tubos y filtros: se usa para procesar flujos de datos, donde cada actividad es un filtro que esta unido a un tubo que se conecta a un filtro posterior

- contexto: procesar flujo de datos
- problema: descomponer el procesamiento en una serie de filtros, que transformen datos de entrada entregados por un tubo, transformar los datos de manera independiente y sin estados y entrega los datos de salida de un tubo
- solucion: los tubos conectan el origen de los datos con un filtro o filtros con filtros con la salida de datos (FIFO). Los filtros aplican una transformacion de datos de entrada en datos de salida.
- implementacion: dividir el sistema en una secuencia de pipes y filters, donde el proceso sea ordenado e independiente. Hay que definir un formato de datos y el esquema para el manejo de errores
- ventajas: no requiere archivos intermedios, filtros reutilizables.
- desventajas: Conversion de datos, errores pueden afectar el flujo de

c. Pizarron: útil cuando no hay una solución completa para el problema, es decir, participan varios sistemas que aportan a la solución.
- contexto: no hay solución completa.
- Problema: conocimiento parcial de la solución y cada solución requiere diferentes áreas y especialidades.
- solución: conjunto de sistemas independientes que trabajan de forma colectiva, comparten los datos, para llegar a una solución en conjunto. se tiene un control centralizado que se encarga de coordinar la ejecución del sistema, no existe comunicación directa entre ellos.

* Sistemas Interactivos

1. modelo vista controlador (MVC): sistema se divide en 3:
i: modelo: contiene los datos y las funcionalidades esenciales
ii: vista: comunicación con el usuario.
iii: controlador: controla los cambios que se le efectuan al modelo.
→ interfaz de usuario: V + C
→ Lógica del negocio: C + M
- contexto: sistemas interactivos con interfaz flexible.
- Problema: interfaz con diferentes representaciones. posee una interfaz cambiante, facil de modificar y si se quiere agregar una nueva funcionalidad, implica modificar la interfaz
- solución: la solución consta de 3 componentes
• comunicación (vista): envia datos y despliega al usuario los datos en respuesta del sistema
• administración (controlador): define el comportamiento del sistema y recibe eventos de la interfaz y solicita los servicios respectivos al modelo.
• procesamiento (modelo): provee funcionalidades requeridas por la vista. encapsula el manejo de datos.
- implementación: separar la funcionalidad del sistema de la interacción del usuario; se tiene que diseñar: modelo, vista, controlador y las relaciones entre ellos.
- ventajas: soporta múltiples vistas, flexible, mantenible
- desventajas: complejidad, en las vista no hay acceso a datos

2. Presentación Abstracción Control (PAC): patron que tiene una estructura jerarquica de agentes que cooperan entre ellos para dar una solución. Los agentes proveen la funcionalidad de la aplicación.

Agente → Presentación: Interfaz
Abstracción: Funcionalidad
Control: Comunicación con agentes

- contexto: sistema interactivo desarrollados utilizando agentes
- problema: Tener un sistema interactivo mediante agente funcionando en el sistema de manera integrada. Los agentes deberán tener interfaces propias de comunicación, estados y datos privados
- solución: definir la estructura jerarquica del sistema con tres niveles de agentes: alto nivel (funcionalidad central del sistema), intermedio (relaciona agentes de bajo nivel), bajo nivel (manejo interfaz especifica de usuario). Cada agente es responsable de una parte de la funcionalidad del sistema y esta compuesto por presentacion (aspecto visible del agente), abstraccion (modelo de datos internos y operaciones sobre ello) y control ((conexion entre presentacion y abstraccion, comunica a los agentes),
- Implementacion: definir funcionalidad central del sistema. Luego se tiene que estructurar la jerarquia de agentes que se van a utilizar. Luego se define cada agente, su funcionalidad, interfaz, etc.
- Ventaja: soporta multitarea, asignacion de responsabilidades especificas
- Desventaja: baja eficiencia, complejo control.

* Patrones Adaptables

1. Micro kernel: sistemas que deben adaptarse a cambios en los requerimientos, separa la funcionalidad en: minima, común para todos. Y extendida: especifica de un cliente.
- contexto: sistema con interfaz de programacion parecidas.
- problema: sistemas que perduran en el tiempo. Estos deben adaptarse a cambios de infraestructura, ya sea hw o sw.
- solucion: establecer un componente central, que tenga las funcionalidades basicas del sistema, que administre recursos y provee servicios de comunicacion entre componentes.
- ventajas: portabilidad, escalabilidad, adaptabilidad.
- desventajas: sistema complejo, baja eficiencia

2. Reflection:
- sistemas que deben adaptarse a cambios en los requerimientos
- cambio dinamico de la estructura de un sistema.
- Se para funcionalidad en:
  - nivel base: logica de la aplicación.
  - nivel meta: propiedades del sistema.

Resumen Arquitectura de Software:

Edificio del parlamento Rumano.

Descomposición modular:

Modelo de objeto:

- Se tiene una estructura rígida y bien definida de objetos.
- Se definen atributos y métodos.
- Son débilmente acopladas, es decir, máquina como usuario son independientes.
- Se coordinan operaciones entre objetos.

Modelo de flujo de datos:

- Se descomponen por funcionalidades.
- Se transforman inputs como output según se requiera.
- Tiene una dependencia de otros procesos.
- El flujo es estandarizado.
- Idealmente, para procesos automatizados (batch).

Arquitectura de dominio específico:

Arquitectura genérica:

- Análisis de sistemas similares y reales en funcionamiento.
- Busca no reinventar la rueda, sino encontrar en el mercado algo similar y funcional para lograr nuestro objetivo.

Arquitectura de referencia:

- Idea y estudio de una arquitectura en concreto.
- Se busca generar una "plantilla".
- Se debe estudiar el comportamiento del negocio o aplicación para generar.

Patrones de arquitectura:

Es una solución que alguien propone o se propone, con el objetivo de resolver un problema o una situación.
Lo que se va a ver es recomendaciones de como se va a diseñar el sistema que se quiere construir, de tal manera de ahorrar trabajo.
Posee características como lo es:

- Esquema genérico.
- Probado: Solución probada, una característica o sistema en particular o una problemática en particular.
- Recurrente: Se vuelve a usar n veces, debido a lo mismo es posible que una persona lo utilice y esta última tenga la confianza de que es la mejor forma de resolver o diseñar y/o construir un sistema que interese.

Un patrón de arquitectura, ¿Qué implica?
El patrón implica un conjunto de especificaciones; en donde se tiene que ver si es que la problemática se adecua o se cumple con ese conjunto de especificaciones, que me están dando. Define entonces:

- Componentes: Los componentes que se van a usar, qué es lo que cada componente va a hacer, cómo van a estar esos componentes relacionados entre sí.
- Responsabilidades.
- Relaciones.

El patrón entrega todo lo anterior, entrega una maqueta diciendo, de una manera, una maqueta, la cual uno puede visualizar, todo lo que se necesita de forma que se relaciona, interactúa con el resto.

Entonces, ¿cómo se describe un patrón?:

- Nombre del patrón (identificarlo por nombre).
- Contexto (¿Cuál es la situación, cuál es la problemática que queremos resolver o la que este patrón resuelve?).
    - Situación que origina el problema.
- Problema (el problema no hace referencia a lo catastrófico, la connotación correcta es: ¿qué es lo que uno busca resolver?, ¿Qué es lo tú necesitas?).
    - Fuerzas presentes en el contexto.
- Solución.
    - Estructura (componentes): cómo los distintos componentes se relacionan entre sí, qué es lo recomendado hacer, qué es lo que hace cada uno de los componentes
    - Comportamiento.

Nota: en realidad son 3 pilares, el nombre no se considera un pilar
Entonces, cada vez que queramos describir un patrón, tenemos que guiarnos por estos 4 títulos o subtítulos y/o características del patrón.

Motivación:

- A la hora de implementar soluciones, generalmente ya existen, por lo que se puede hacer es buscar "soluciones" ya prediseñadas con el fin de ahorrar trabajo inventando.

- Generalmente, estos patrones son de uso recurrente y han sido adaptados y probados de forma continua por la continuidad, por lo que se sabe sus pro y contras.

¿Qué posee un patrón?:

- Nombre.
- Contexto del problema.
- Problema como tal.
- Solución.

Contexto:

- ¿Qué origina el problema?, es decir, como llegamos a este.
- ¿En qué área está?, ya sea el modelo de datos, la infraestructura u otro campo.
- Detalles: Acá se describe todo del mismo, desde las problemáticas generadas (siempre desde el contexto, ej: negocio), hasta posibles formas a tratar el mismo.

En el contexto se explora cuál es la situación, cuál es la problemática que queremos resolver, en qué situación o fundamento es más adecuado. En el fondo, lo que estamos diciendo con el contexto es todo lo que el ambiente debe de tener, para utilizar este patrón.

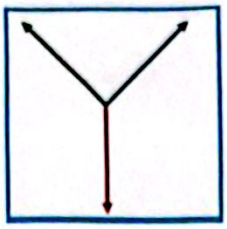
- Situación que origina el problema.
- Ámbito del problema.
- Descripción de las situaciones que lo originan.
    - Descripción general.
    - Descripción detallada.
- Difícil de precisar completamente.

Problema (lo que queremos resolver):

Aquí aparecen las fuerzas, bueno... ¿Qué son las fuerzas? Son los distintos requerimientos y restricciones y características, que puedo tener, que debo considerar al momento de usarlo.

¿Qué es lo que queremos hacer? ¿Qué es lo que queremos implementar?

- Descripción: Se busca ver los detalles del mismo ya un nivel más "informativo".
- Se definen posibles candidatos u componentes a modificar, para ver qué se debe resolver.
- Importante tener en cuenta el contexto, ya que los requerimientos del negocio no se deben pasar a llevar o restricciones del sistema.
- Fuerzas presentes en el contexto: Se llaman fuerzas, porque hay tres extremos, y lo que se debe de encontrar es el punto medio entre las fuerzas presentes en el problema. El problema es que el ambiente puede llegar a estar funcional, pero no estaría por ejemplo cumpliendo varias propiedades de los requerimientos funcionales, es decir, tengo seguridad y escalabilidad, nomás puedo hacer ímpetu a una de las anteriores, pero no descartar la otra, sino por ejemplo dejarla en un punto medio con las demás; pero si llego a tener más seguridad disminuye mi incremento de escalabilidad.



- Propiedades: Es lo que hace, lo que se va a resolver.
- Requisitos: Qué se necesita tener para el funcionamiento adecuado de este patrón, por ejemplo: el patrón apunta a funcionar en sistemas de alta disponibilidad o apunto a sistemas mantenibles o ambos. Entonces define las características de los requerimientos funcionales y no funcionales que este patrón cubre.
- Restricciones: Por ejemplo, no se puede usar en un ambiente interactivo o este patrón solo funciona o tiene un comportamiento adecuado cuando se usa en tal sistema (Sistema operativo, por ejemplo). El patrón define cuáles son el marco de acción en el cual el patrón se mueve.

Solución: La solución entrega...

- Esquema de solución del problema (un esquema, para el problema dado un contexto).
- Balance de fuerzas.
- Estructura.
  - Componentes.
  - Relaciones.
- Comportamiento.
  - Organización de componentes.
- Priorización de fuerzas.
  - Mecanismo de control definido que se debe de seguir, que se debe de utilizar.
  - Se priorizan cuáles son las características (requerimientos no funcionales) que se van a mantener y cuáles se van a desechar.

En resumen de solución: aquí se pasa la receta paso a paso (según el viejo glo)

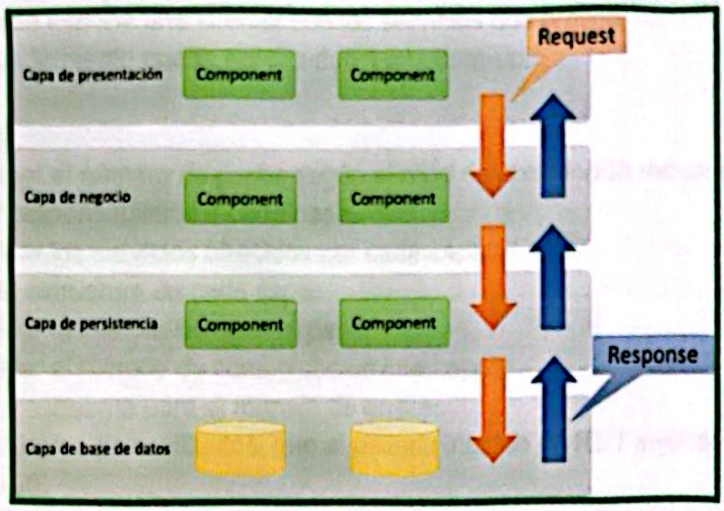
- Primero se analiza la solución de forma conceptual o esquemática.
- Se hace una comparativa entre los pro y contras.
- Se analiza el impacto que tendrá con otros componentes.
- Se ve el comportamiento que tendrán luego de su implementación.

Clasificación:

- Patrones simples (más simples de aplicar o implementar y/o manejar).
- Sistemas interactivos.
- Patrones adaptables (van adecuándose a la situación, es decir, aprendiendo según sea la situación).
- Sistemas distribuidos (patrones adaptados para sistemas distribuidos).

Patrones Simples:

Capas:



Este patrón consiste en la descomposición del sistema en distintos niveles, por ejemplo si uno quiere hacer un sistema de ventas hay que dividirlo en distintos niveles (niveles de abstracción), visión usuario, visión transportista de como llevar los productos a distintos lugares, donde y cuándo y cuál es la ruta óptima, visión de producción (que es lo que el sistema puede entregar a esa área o gerencia) y así sucesivamente. Según el ámbito de acción que se va a mover.

Capa de más arriba selección del producto, luego abajo está la del medio de pago, luego más abajo producción (ambiente y bienes para controlar el proceso productivo), más abajo se tiene componentes que permiten la asignación del producto a los medios de despacho que ya tenga sea transportista, lo que sea.

- Sistemas de se descomponen a diferentes tareas.
- Contexto: Sistemas que contienen una jerarquía de distintos niveles con acciones.
- Problema: Mala organización de componentes, con problemas de escalabilidad y mantenibilidad.
- Solución: Se generan capas con distintos niveles de funcionalidades, donde la más baja está el "core" y avanza utilizando servicios de capas contiguas.
    - Capa base con nivel de abstracción más bajo.
    - Avanzar capa a capa utilizando los servicios de la capa inmediatamente anterior.
    - Componentes estructurados en módulo relacionados.
- Se debe definir la cantidad de capas y separar los servicios y funcionalidades, entre estas se debe de considerar las dependencias.
- Ventajas: Estandarización, Reutilización.
- Desventajas: Cambios provocan efecto cascada, complejo, ineficiencias o "burocrático", nivel más alto requiere hablar con todo.

Solución (Características):

Cada capa tiene un ámbito de acción, lo que le permite relacionarse con su o sus capas vecinas. Y en cada capa que en el fondo es una agrupación teórica lógica están todos los componentes, módulo y/o procesos.

- La capa K se relaciona solamente con la capa K-1.
- No hay dependencias entre capas-
- Cada capa puede estar integrada por distintos componentes.
- Los componentes pueden interactuar entre sí, pero quedan acoplados.

- Cada capa expone una interfaz con los servicios que provee.
- El comportamiento puede ser top-down o buttom-up.

Implementación:

- Determinar el número de capas según el nivel de abstracción requerido.
- Asignar responsabilidad a cada capa.
- Especificar los servicios ofrecidos por cada capa.
- Definir la estructura de cada capa.
- Especificar la interfaz de cada capa.
- Especificar el método de comunicación intercapas.
- Definir el esquema para el manejo de errores.
    - Errores premeditados: que el usuario ingrese un RUT inválido (validación de rut).
    - Errores aleatorios: Digamos que un proceso en la capa 4, le entregaba a el área de producción, le entregaba materiales para hacer la producción; ahí falla algo, incertezas con la base de datos por ejemplo.

Ventajas:

- Componentes estandarizados.
- Cambios que afectan el nivel local.
- Reutilización de capas / componentes.

Desventajas:

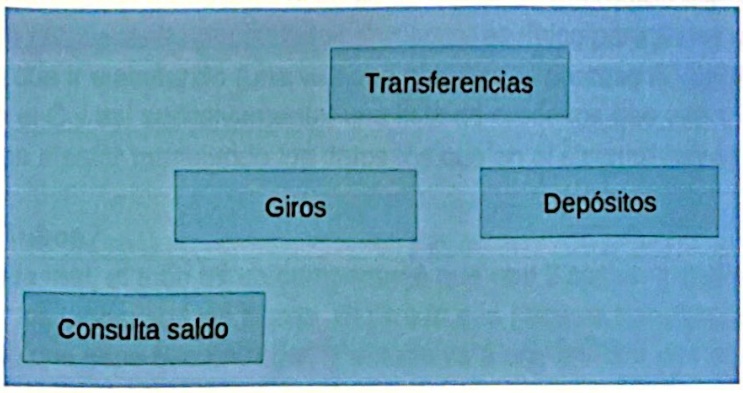
- Cambios afectan en cascada.
- Ineficiencia.
- Complejo de definir.

Ejercicio:
Un banco requiere un sistema para manejar las cuentas corrientes de sus clientes.
Específicamente, requiere tener las siguientes operaciones:

- Consulta del saldo de una cuenta.
- Depósito en una cuenta.
- Giro desde una cuenta.
- Transferencia entre cuentas.

Se pide este sistema utilizando la arquitectura de Capas.
R: ¿Qué es lo que decía el patrón?, y que es lo que hará cada una de estas capas, cuál es su responsabilidad y como se van a acceder a los servicios ofrecidos por esa capa y comunicación entre los distintos módulos.

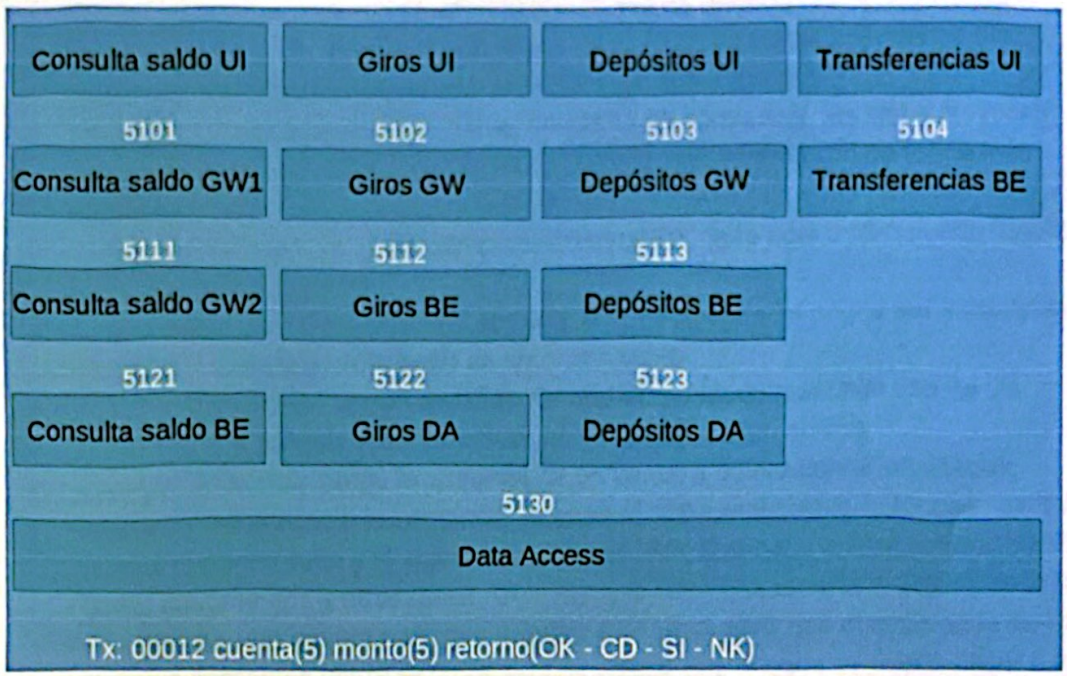
- Ahora se debe definir una jerarquía de las operaciones.



Consulta de saldo más especifico y transferencia más genérico.
Se tienen tres niveles:

1. Transferencias.
2. Giros y Depósitos.
3. Consulta Saldo.

Aunque se necesitan 2 niveles más, ya que se necesita de una interfaz de usuario (como los usuarios acceden a estos servicios), también se necesita interfaz para cada una de las funcionalidades que ofrece el sistema (transferencia, giros, depósitos y consulta saldo). Se puede agregar una capa de acceso a los datos del usuario



GW: Gateway, programas que pasan la información de una capa a la otra.
BE: Backend
DA: Data Access.

Nota: Generalmente, se evita el patrón por capas por el tema de la interdependencia y la dificulta de aplicar cambios en distintos niveles encadenados unos con otros, para el contexto del ejercicio anterior no es conveniente. Pero en el caso de OSI o LINUX sí es lo ideal y lo que debe de utilizarse.

Tubos y filtros (cadena de procesos):

Este patrón es el adecuado de cuando el contexto de donde queremos implementarlo es un flujo de datos que van pasando por distintos procesos, es típico para sistemas de cadenas de procesos hay que ir ejecutando (una vez que empieza el proceso A, luego empieza el proceso B, luego el C y así sucesivamente; una lista de procesos que van a ir ejecutándose uno tras otro y van a estar manejando los datos los que en el sistema hayamos definido).

¿Qué hacen los **Tubos**?
Tal como en la vida real, el tubo es un componente que une 2 cosas. Y cuáles son esas dos cosas unidas por los tubos, son los **filtros**. El filtro lo que hace es transformar los datos de lo que le llego a lo que tiene que entregar. Y el tubo va a unir un filtro con otro filtro,

entonces los datos que están siendo entregados por el filtro 1 van a llegar al tubo 1 y así estará conectado con el filtro 2 que va a recibir los datos del filtro 1 y los va a procesar y va a generar un resultado y ese resultado le hará llegar al tubo 2, el cual va a estar conectado al filtro 3, etc.

- Estructura aplicaciones en actividades para procesar flujo de datos en que cada actividad (o transformación) es un filtro que se está usando por un tubo de los filtros contiguos.
- Los tubos es un componente que une los filtros y los filtros transforman los datos según lo que le llego y lo que debe de entregar.
- Procesamiento de datos en base a actividades y transformaciones.
- Contexto: Procesamiento de información (flujos de datos).
- Problema: Se debe descomponer el procesamiento en varias actividades (filtros), cambiado los datos según se requiera para enviarlos (tubos).
- Solución: Los tubos conectan filtros, con el fin de comunicar, los filtros provocan cambios en el input/output con el fin de trabajar con información de forma más cómoda.
- Ventajas: Reutilizable, procesamiento en paralelo, cada tubo o filtro puede funcionar de forma independiente.
- Desventajas: La información no se comparte, la conversión podría ser ineficiente.
- Ejemplo implementación: bash de linux con pipes.
- Otro ejemplo puede ser un workflow de una empresa en que cada una de las personas que tengan que interactuar con ese flujo serían el filtro.
- Ejemplo de banco: tengo la sucursal de un banco y quiero cobrar un cheque, entonces me acerco a la ventanilla, le paso el cheque al cajero, entonces nosotros somos el primer tubo y el cajero es el primer filtro, que hace el filtro (cajero) válida la firma, documentos y otros datos. Una vez que hace eso toma el cheque y se lo pasa o lo deja en una gaveta para que vengan a retirarlo, para que el encargado de las cuentas corrientes (filtro) se encargue de validar ese cheque se puede pagar o no. Entonces viene un tubo (otro encargado) que toma el cheque de la bandeja de salida en el cual lo dejo el cajero, se transporta al encargado de cuentas corrientes (filtro) se revisan los datos y así sucesivamente. Si es que hay un desperfecto con el cheque, viene otro encargado a ver el caso pasando de un tubo a un nuevo filtro.

Problema:

- Descomponer el procesamiento en una serie de actividades (filtros) que transforman datos de entrada (recibidos desde un tubo) en datos de salida (entregados en un tubo).
- Transformaciones independientes y sin estado.
- Cada actividad es un filtro.
- Los filtros se comunican entre sí mediante tubos.
- Por ejemplo, el proceso de venta de una tienda online, hay que asimilarlo a un proceso de flujo de datos, en que el usuario ingresa, quiero comprar tal producto, se hace al programa, el cual se va a verificar si es que el producto y la cantidad actual existe; entonces se tendría otro punto de partida, el filtro que se tendría que es el de validador de entrada de datos. Para luego aplicar una transacción los filtros son los programas y los tubos es el medio que comunica los filtros y fluyen en ellos los datos.

Solución:

- Tubos (pipe):
  - conecta origen de datos con un filtro.
  - conecta filtro con filtro.
  - conecta filtro con salida de datos.
  - esquema de procesamiento FIFO.
- Filtros (filters):
  - Aplica procesos de transformación de datos de entrada en datos de salida.
  - Implementa un flujo que solicita datos (al filtro anterior).
  - Implementa un flujo que entrega datos (al filtro siguiente).
  - Filtros independientes, no tienen idea de lo que hay antes o después. Tampoco saben quien o como se creó lo que se le entrega, solo hace su tarea y la entrega.
    a) Estado no compartido.
    b) Desconocimientos de otros filtro.
    c) Filtros activos o pasivos: Un filtro activo está constantemente preguntando al tubo (oye traeme datos, el filtro pide que le traigan datos por el tubo) si es que tiene algo para él. En el caso del pasivo (el que espera hasta que el filtro traiga un dato para procesar), este solo será activado cuando llegue algo para él.

Implementación:

- Dividir el sistema en una secuencia de procesos ordenados e independientes.
- Definir el formato de los datos transmitidos por los tubos (rendimiento o flexibilidad).
- Especificar el procesamiento de cada filtro.
- Construir los filtros.
- Definir el esquema para el manejo de errores.

Ventajas:

- Arquitectura flexible: uno puede construir lo que quiera y poder distribuir el sistema de tal manera que cada parte encargada de un filtro sepa lo que debe recibir, como lo tiene que procesar y como debe ser su salida. Se pueden construir tubos en paralelos de tal manera que siempre el punto de intersección reciba el input que necesita.
- No requiere de archivos intermedios.
- Filtros reutilizables: se pueden usar los filtros las veces que uno quiera.
- Procesamiento paralelo (eficiencia).
- Construcción independiente.

Desventajas:

Patrón solo adecuado para procesar flujos de datos.

- Información no compartida: cada filtro maneja su propia información, la única comunicación que se tiene con filtros anteriores o siguientes son por el tubo, el cual solo entrega set de datos. Pueden ocurrir los típicos problemas que pasan cuando no se comparte información, como redundancia, replicas, etc.
- Conversión de datos (ineficiencia): convertir los datos de un formato a otro dependiendo de la construcción de los siguientes filtros.

- Errores pueden afectar el flujo de procesamiento: se necesita un modelo de control que permita manejar el que hacer cuando hay un error. Se podría aplicar el Call-Return.

Pizarrón:

Este patrón sirve cuando necesitamos hacer que varios sistemas coexistan, convivan y se comuniquen entre sí para obtener algún resultado, pero ese resultado es incierto, no sabemos si llegaremos a él o no. Ejemplo: reconocimiento de imágenes.

- Sistemas no completamente definidos donde comparten diversos sistemas.
- Contexto: No existe solución completa que abarque todo.
- Problema: La solución puede requerir diversos paradigmas, con lo cual no se tendría una solución global o completa.
- Solución: Trabajo colaborativo, compartir datos, integrar sistemas, generar soluciones parciales (divide and conquer).
- Patrón útil cuando:
  - No hay una solución completa y específica para un problema.
  - Participan varios sistemas que aportan su conocimiento.
  - Ejemplos: inteligencia artificial, reconocimiento de imágenes, toma de decisiones.

Contexto:
Dominio en el que no hay una solución completa.

Problema:
- Conocimiento parcial de la solución.
- Cada solución requiere diferentes paradigmas.
- El problema abarca muchas especialidades.
- No es factible una solución completa.
- Hay soluciones parciales que cubren parte del problema.
- Módulos aportan parcialmente a la solución.
- Cada sistema contribuyente usa sus datos en diferentes representaciones.

Solución:
- Conjunto de sistemas independientes.
- Trabajo colaborativo.
- Datos compartidos, como un git (un repositorio).
- Cada sistema se especializa en parte del problema.
- Control centralizado que coordina la ejecución de los sistemas.
- Genera soluciones parciales que pueden ser desechadas.
- No hay comunicación directa entre los sistemas.

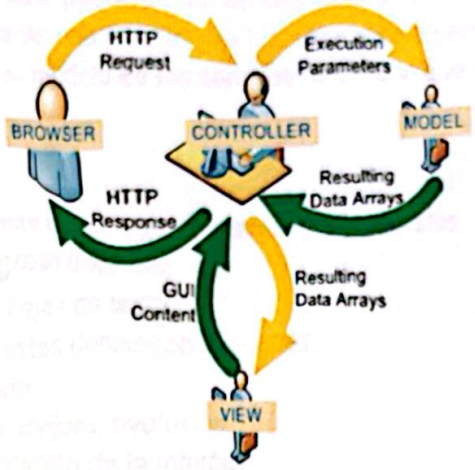
Patrón orientado al área de investigación más que comercial.

(Repositorio: El viejo nunca paso esta wea como patrón qlo)

Patrones Interactivos:

Modelo vista controlador (MVC):

El sistema está dividido en 3 partes como el modelo vista controlador, Lo que normalmente no le asigna a cada uno de estos tres componentes no es lo que corresponde.



¿Qué es el modelo?

El modelo de datos, pero el modelo de datos + la funcionalidad que maneja ese modelo de datos que es lo básico del sistema; entonces parte del modelo es un conjunto de funciones o métodos que acceden a los datos y que van a recibir requerimientos a partir de otros niveles.

- Modelo: datos y funcionalidad principal. Modelo de datos, más la funcionalidad que maneja ese modelo de datos, es un conjunto de funciones y métodos que acceden a los datos y que reciben peticiones de los otros niveles para acceder a esos datos.
- Vista: Comunicación con usuario, es lo que yo le voy a mostrar al usuario.
- Controlador: Verifica cambios al modelo. Controla cambios al modelo, componente que se preocupa de manejar a la vista y al modelo.
- Características: interfaz de usuario es: vista + controlador, lógica del negocio es: controlador + modelo.
- Contexto: Sistemas muy interactivos con interfaz flexible.
- Problema: Gran variedad de formularios y vistas con ingreso de datos, es decir, interacción con usuario o acciones que provocan una acción.
- Solución: Separar funcionalidades en los tres componentes modelo, vista y controlador.
- Controlador desacopla la vista del modelo. La vista no sabe del modelo y viceversa, de eso se encarga el controlador, este entrega todo lo que necesita la vista de parte del modelo.
- Flexibilidad, mantenibilidad, adaptabilidad. Flexible, porque la vista se encarga de pedir datos, recibir resultados
- y mostrárselo al usuario; se pueden añadir más vistas y no se van a estorbar la vista con el modelo.
- Mantenibilidad es porque se pueden agregar nuevos requerimientos sin estorbar lo que ya tengo, ya que cada vista es independiente de las otras. Adaptabilidad, el sistema se va adecuando a los requerimientos.
- La vista no está atada al modelo y el modelo no está atado a la vista. La vista corre por su cuenta, el modelo procesa por su cuenta, entrega los resultados que le están pidiendo y el controlador es el orquestador entre estos dos elementos.

- Ventajas: Soporta múltiples vistas e interfaces, flexible, existe una gran cantidad de framework que implementan este.
- Desventajas: Complejidad, el manejo de las interacciones debe ser intuitivo.

Con respecto al Contexto, este patrón se puede usar cuando se tenga que hacer un sistema interactivo, el cual necesite de una interfaz flexible. Un ejemplo podría ser un juego online, vista lo que ve el usuario, el modelo es los cambios de estado que ocurren en el juego que se registran.

Problema:

- Interfaz con diferente representación: texto, gráficos, listas, iconos.
- Paradigmas de ingreso diversos:
  - Digitación: cajas de texto.
  - Selección: listas desplegables, iconos.
  - Ingreso mixto.
- Interfaz cambiante: mejora, evolución.
- Facilidad de modificación de la interfaz.
- Funcionalidad nueva implica modificar interfaz.
- Presentación de la información en multi-formato.

Solución:

Tres componentes:

- Conmutación (vista):
  - Envía requerimientos del usuario.
  - Recibe datos (del modelo) y los despliega al usuario.
- Administración (controlador):
  - Define el comportamiento del sistema.
  - Recibe eventos (desde la interfaz de usuario) y solicita servicios al modelo.
- Procesamiento (modelo):
  - Provee la funcionalidad requerida por las vistas.
  - Encapsula el manejo de los datos.

Implementación:

- Separa la funcionalidad de la interacción del usuario.
- Diseñar e implementar.
  - El modelo.
  - Las vistas.
  - Los controladores.
  - La relación entre vistas y controladores.

Ventajas:

- Modelo soporta múltiples vistas.
- Flexible, mantenible, adaptable.
- Frameworks implementan MVC.

Desventajas:

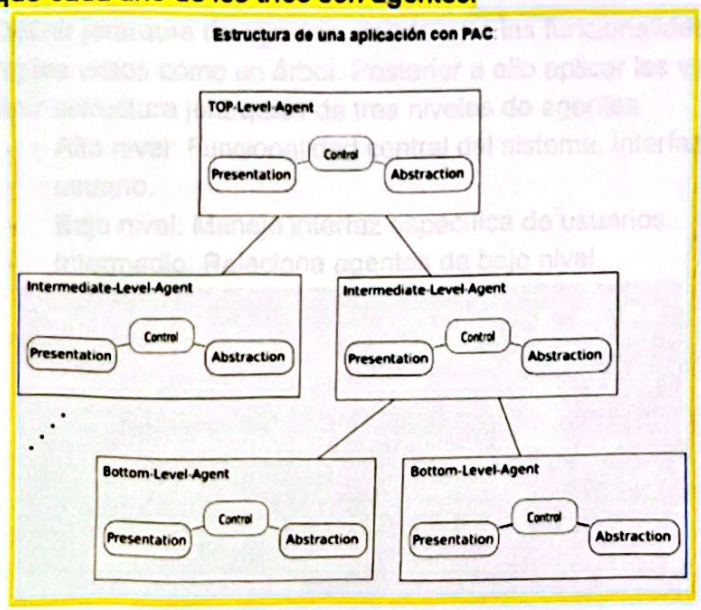
- Modelo acoplado con vistas y controladores: la implementación es de forma desacoplada, pero no hay que olvidar que uno no funciona sin el otro.

- Vistas sin acceso a los datos (ineficiente): la vista no tiene idea de los datos, para obtener datos debe de generar eventos (ej: click en botón), se debe de pasar por un intermediario para tener un elemento que se quiere.
- Complejidad: sería más fácil si directamente desde la página se acceden a los datos, pero eso ya sería hablar de otro tipo de sistema, MVC está atado a este patrón.

Presentación Abstracción Control (PAC):

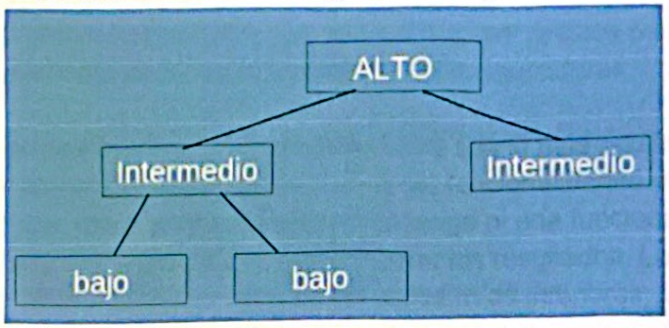
Existe una estructura es "agentes" jerárquica, que provee las funcionalidades de la aplicación.

Otro patrón para sistemas interactivos, la gran diferencia de MVC es que cada uno de los componentes se asocia en tríos y cada trío es independiente y cumplen cierta función en el sistema. Se dice que cada uno de los tríos son agentes.



- Compuesto por tres componentes.
  - Presentación: Interfaz.
  - Abstracción: Funcionalidad y modelo de datos.
  - Control: Comunicación con agentes y presentación.
- Agente es responsable de una parte de la funcionalidad.
- Provee la funcionalidad de la aplicación.
- Estructura jerárquica de agentes cooperativos.
- Segmentación del sistema en subsistemas especializados. Dividir el sistema en funcionalidades y asignar esas funcionalidades a tríos.
- La estructura es por niveles, los agentes que están arriba controlan lo que pase abajo y los que están abajo solo los que hacen el trabajo. Cada agente van a encargarse de resolver algún requerimiento. Ejemplo: autentificación de usuario:
  - Presentación: Ventana o página donde se deberá poner todo el input necesario para lograr el requerimiento.
  - Abstracción: Recibe los datos y verifica la valides de los datos puestos.
  - Control: Responde al que lo mando a hacer el trabajo, el resultado.
- Contexto: Sistemas interactivos separados por funcionalidades (agentes).

- Problema: Separar funcionalidades para los agentes, generar interfaces flexibles con bajo acoplamiento. Trabajo cooperativo entre agentes.
    - Estructurar un sistema interactivo mediante agentes funcionando en forma integrada.
    - Generar interfaces flexibles de usuarios.
    - Separar la presentación de la funcionalidad.
    - Bajo acoplamiento.
    - Agentes con interfaces propias de comunicación.
    - Agentes con estado y datos privado. Cada agente es maneja sus datos y su estado y eso es lo único que necesita
    - saber el resto. La gran gracia de este patrón es la colaboración entre agentes, pese a que cada uno funcione de manera autónoma ante su función
    - Trabajo cooperativo de los agentes.
- Solución: Definir jerarquía de agentes, donde este las funcionalidades del "core" y las más simples vistos como un árbol. Posterior a ello aplicar las vistas de PAC.
    - Definir estructura jerárquica de tres niveles de agentes.
        - Alto nivel: Funcionalidad central del sistema, interfaz global de usuario.
        - Bajo nivel: Manejo interfaz específica de usuarios.
        - Intermedio: Relaciona agentes de bajo nivel.



- Ventajas: Multitasking, Independientes entre sí, responsabilidades separadas entre agentes.
    - Asigna responsabilidades específicas: cada agente se encarga de una funcionalidad específica y tiene todo lo necesario para resolver ese requerimiento.
    - Funcionamiento independiente: Funcionan sin la intervención de otros. Bueno para la Mantenibilidad debido a que no se van a afectar los agentes si es que integro nuevos agentes al sistema.
    - Soporta multitarea: múltiples procesos en paralelo debido a la independencia de los agentes.
- Desventajas: Sistema complejo, difícil de controlar y mantener.
    - Sistema complejo: La estructuración es compleja.
    - Baja eficiencia: Comparado con sistema monolítico por la coordinación de los elementos. Las operaciones necesitan coordinación y comunicación entre distintos agentes.
    - Complejo mecanismo de control.

<u>Requerimientos no funcionales:</u>

Si queremos tener ciertos requerimientos no funcionales, debemos de poner de nuestra parte.

- Performance:
  - Sistema debe tener poco componentes, la comunicación entre componentes toma tiempo, por lo tanto, mientras más componentes haya, pero performance tendrá.
  - Operaciones críticas concentradas en uno o dos componentes.
  - Disminuir la comunicación entre componentes. Los componentes deben ser lo más autosuficiente dentro de lo posible.
  - Bajo uso de la red y operaciones de entrada/salida.
- Escalabilidad:
  - Componentes auto-contenidos. Los componentes deben ser capaces de procesar independientemente, no deben de necesitar ayuda de otro u otros componentes.
  - Componentes con bajo acoplamiento.
  - Crecimiento horizontal. Distribuir creando más instancias de componente.
- Mantenibilidad:
  Con Mantenibilidad se refiere a prever errores o NO correcciones.
  - Componentes auto-contenidos.
  - Especificación detallada.
  - Evitar datos compartidos. Los datos deben ser propios para evitar errores.
  - Generadores de datos separados de los consumidores.
- Seguridad:
  Seguridad del procesamiento. El procesamiento sea lo más oculto posible.
  - Usar arquitecturas por capas. Dividir las funcionalidades de tal forma que la capa que tenga acceso al usuario no tenga ni una funcionalidad del sistema, que solo capte requerimientos y mostrar los resultados. Los procesos importantes deben de estar en las capas m´as inferiores, lo m´as lejos del usuario dentro de lo posible.
  - Mecanismos de validación en cada capa.
  - Recursos críticos en capas internas.
  - Consumidores separadas de los productores.
- Confiabilidad:
  - Componentes redundantes. Varias instancias de un componente por si acaso alguno falla. Más trabajadores de los que necesitas.
  - Fácil migración de componentes.
  - Bajo acoplamiento. Componentes autosuficientes, lo más independiente dentro de lo que sea posible.
  - Control distribuido. No cuello de botella, que no haya un único punto de falla.
  - Operación activo-activo o activo-pasivo. activo-pasivo es que un elemento esté funcionando y otro espera a que falle para reemplazarlo, en activo-activo, los dos están funcionando.

Reflexiones finales:

- No existe la bala de plata.
- El contexto del problema puede determinar la arquitectura del sistema.

- Los patrones son una guía para el diseño de un sistema.
- La solución puede ser una mezcla de patrones.

Nota

Con "No existe la bala de plata" se refiere a que no existe una sola solución a un requerimiento de diseño. No existe la solución perfecta, siempre se deben de sacrificar requerimientos no funcionales.
Siempre se pueden mezclar patrones dependiendo de lo que uno priorice en el diseño del sistema.

Evaluaciones Pasadas:

El control es individual

1. *(15 puntos)* Analizar los cinco factores que deben ser considerados cuando se quiere incluir en un sistema el atributo de calidad Seguridad.
   Autenticación, Autorización, Encriptación, Integridad, No repudio

2. *(15 puntos)* Describir detalladamente un sistema en el que usted aconsejaría utilizar el modelo de control basado en eventos y justificar su elección.
   Juego RPG con múltiples usuarios controlando su respectivo personaje. Cada usuario genera eventos con las acciones del personaje que deben ser manejadas.

3. *(30 puntos)* Se requiere desarrollar un sistema que permita ingresar y reportar la votación obtenida por cada candidato en cada una de las mesas receptoras de sufragios en todo el país. Las mesas receptoras están agrupadas en locales de votación a cargo de un jefe de local. Entre otras funciones, el jefe de local es el encargado de recolectar las actas de votación de cada mesa receptora e ingresar los resultados al sistema. Para ello, dispone de un notebook con conexión a internet.

Se requiere, ademas, que el sistema genere un informe con los resultados disponibles hasta ese momento y consolidados por cada candidato, con una periodicidad preestablecida en el sistema. Este reporte debe considerar varios niveles de agrupación de los resultados, tales como, a nivel de la mesa receptora, del local de votación, de la comuna, del distrito (para los diputados), de la región senatorial (para los senadores) y, finalmente, a nivel país (para el presidente).
Considerando que las elecciones son en 23 días mas, se pide lo siguiente:
a) analizar y justificar tres requerimientos no funcionales que el sistema debe satisfacer
- Seguridad, Confiabilidad, Performance

b) analizar y justificar cual de las arquitecturas de software genéricas va a usar para desarrollar el sistema
- Repositorio: ingreso independiente de la votación, obtención de reportes centralizados
c) describir brevemente la funcionalidad de tres de los componentes que, a su juicio, son los mas importantes del sistema a desarrollar.
- autenticación de usuarios
- recepción y encriptación de la votación (en cada local)
- sincronización con repositorio central

* * * * *

Clase Pre-Solemne 2:

1. Analice tres consideraciones que justifican la utilización de un patrón de arquitectura en el desarrollo de un sistema.
   a. Es una guia y apoyo
   b. Se ha probado un montón de veces
2. Analizar el patrón de tubos de filtros desde el punto de vista de lo siguentes atributos de calidad:
   a. Mantenibilidad: Los filtros al ser independientes son mantenibles ya que los cambios que se hagan en uno no afectarán al resto de los mismos. es fácil agregar una nueva funcionalidad y modificarlas, pero tener en cuenta que sucede mientras no se modifiquen las interdaces.
   b. Seguridad: Cada tubo puede impelentar su propia seguridad en su entrada o poner un filtro que permita autenticación y codificación (codif. al principio y al final)
   c. Escalabilidad: Lo que se refiere es poder procesar más información. Por lo que paralelizar el mismo flujo para una gran cantidad de datos es una aplicación.
   d. Felxibilidad: Filtros pueden ser conectados según las necesidades de procesamiento
   e. Eficiencia: Comunicación directa entre filtros a trabés de los tubos. Procesos paralelos.
3. Teniendo como base de analisis de proyecto del curso en el que está trabajando y considerando que se desea implementarlo en una aruitectura diferente a SOA, se le pode lo siguiente:
   a. Escoger una arquitecrtura y justificar su elección .
   b. Analizar las ventajas y desventajas del patrón de arquitectura asociado
   c. Hacer el diagrama global de componentes principales del sistema.