

INGENIERÍA DEL SOFTWARE

PARCIAL I

Pregúntelo para que te tome tu hermana, tu vieja o tu tía, que no tienen ni la más puta idea de ingeniería del software

Índice:

- Introducción ----- 2
- Análisis y especificación - - - 6
de requerimientos
- Arquitectura ----- 14
- Diseño ----- 20
- Diseño detallado ----- 33

INTRODUCCIÓN (cap. 1)

¿Cuál es el dominio del problema de la ingeniería del software?

Software de nivel industrial, es decir, aquel que no solo resuelve problemas a ciertos usuarios, sino también del que podrían depender grandes sistemas o negocios y cuya falla podría significar importantes pérdidas, directa o indirectamente.

Este debe ser de alta calidad, fuertemente testeado, documentado y mantenido una vez entregado.

Es grande, complejo y su desarrollo requiere la descomposición en fases.

¿Qué es la ingeniería del software?

aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software

¿Cuál es el problema fundamental de la IS? La satisfacción de las necesidades del usuario a través del software.

¿Cuál es la diferencia fundamental entre las fallas mecánicas y las de software?

SW
↓
Software

Fallas mecánicas \Rightarrow consecuencia del uso/desuso/ago del tiempo

Fallas de SW \Rightarrow consecuencia de errores introducidos en el desarrollo. Siempre existieron, solo se manifiestan tarde.

¿Cuáles son los desafíos de la IS?

Factores que afectan al enfoque elegido para resolver el problema.

Escala: debe considerarse el tamaño del SW a desarrollar, mientras más grande más hay que formalizar los métodos para su desarrollo. El SW también debe ser fácilmente adaptable, respecto a respuesta y rendimiento, ante el aumento/diminución de usuarios o requerimientos.

Calidad y productividad: IS es impulsada por tres factores: costo, schedule (calendario) y calidad

productividad capture ambos
mayor productividad \Rightarrow menor costo y menor tiempo

Por otro lado, la calidad se define en base a seis atributos principales:

- suele ser general (\leftarrow confiabilidad)
- Funcionalidad
 - Usabilidad
 - Eficiencia
 - Mantenibilidad
 - Portabilidad
- } los objetivos de calidad del SW deben definirse al inicio del proceso y el desarrollo deberá velar por ese objetivo

Consistencia y rechabilidad: cómo garantizar que los resultados extraídos quedan restringidos y así mantener cierto grado de consistencia en la C&P

Cambio: el software que desarrolla la IS debe ser rápidamente adaptable a los habituales cambios del entorno (usuarios, clientes, empresas, organizaciones, etc.).

Entonces \Rightarrow El desafío de la IS es consistentemente producir software de alta calidad y productividad, adaptable a los cambios, para resolver problemas de gran escala.

Definir cada uno de los atributos de calidad

Funcionalidad: capacidad de proveer funciones que cumplen con las necesidades establecidas o implícitas.

Confiabilidad: capacidad de realizar las funciones requeridas bajo las condiciones establecidas durante un tiempo específico.

Usabilidad: capacidad de ser comprendido, aprendido y usado.

Eficiencia: capacidad de proveer desempeño adecuado respecto al consumo de recursos usados.

Mantenibilidad: capacidad de modificación del sistema con el propósito de corregir, mejorar o actualizar.

Portabilidad: capacidad de ser adaptada a distintos entornos sin aplicar más acciones que las provistas para ese propósito.

¿Cuál es el enfoque de la IS?

Separar el proceso de desarrollo del producto desarrollado, para centrarse en el proceso.

Un proceso adecuado implica gran C&P.

(calidad y productividad)

¿Cuáles son las fases del proceso de desarrollo? ¿Qué ventajas tiene esta sección?

Fases:

- 1- Análisis y especificación de los requerimientos
- 2- Arquitectura
- 3- Diseño
- 4- Codificación
- 5- Testing
- 6- Entrega e instalación

Ventajas:

Permite verificar calidad y progreso en momentos definidos, o sea, luego de terminar cada fase, ya que c/u fase tiene una salida definida.

ANÁLISIS Y ESPECIFICACIÓN DE LOS REQUERIMIENTOS

¿Cuál es la entrada y salida de esta fase?

Entrada: Ideas de los clientes

Salida: Documento formal que especifica lo que el sistema debe hacer \Rightarrow SRS

¿Qué es y para qué es necesaria la SRS?

La SRS es el documento formal que será la salida de la fase de especificación de requerimientos.

Este describe qué debe hacer el sistema (no cómo).

Sirve para reducir la brecha comunicacional entre el cliente y el desarrollador, asegurar calidad en el producto final, reducir los costos de desarrollo y tener una referencia para la validación del producto final.

¿Qué es el proceso de requerimientos y cuáles son sus fases básicas?

Secuencia de pasos a seguir para convertir las necesidades de los clientes en la SRS.

3 fases básicas:

1 - Análisis del problema: entender qué necesita proveer el sistema

2 - Especificación de los requerimientos: se especifican los requerimientos en un documento (SRS)

3 - Validación: se asegura que los especificados sean efectivamente todos los requerimientos del software.

Este proceso no es lineal.

¿Cuál es el objetivo del análisis del problema?

Lograr una buena comprensión de las necesidades, requerimientos y restricciones del sistema.

NOMBRE y explica los distintos métodos para el análisis del problema

Enfoque informal: no hay metodología definida y no se construye un modelo formal del sistema.

La info se obtiene a través del análisis, observación, interacción con clientes y se traduce directamente a los SRS.

Método de análisis **ESTRUCTURADO**

Se enfoca en las funciones que realiza el sistema, lo ve como un set de transformadores de datos por los que fluye la info.

Pasos:

1.- Dibujar el diagrama de contexto: el sistema es tratado como un solo proceso con sus respectivas entradas, salidas, func., sumideros, etc.

2 - Dibujar el DFD del sistema existente: se refina el diagrama de contexto. Un DFD es una representación gráfica del flujo de datos del sistema.

3 - Dibujar el DFD del sistema progreso e identificar la barrera nombre-máquina: Se agregan al DFD todos los procesos (automatizados o no) y se identifica cuáles procesos se automatizarán y cuáles no. La barrera nombre-máquina estará entre la implementación de ambos.

Modelado orientado a objetos: el sistema es visto como objetos que interactúan entre sí o con el usuario a través de los servicios que proveen. Se modela usando diagrama de clases

Prototipado: se construye un sistema parcial prototípico que ayuda a visualizar cómo será el sistema final.

Dos enfoques

- Descriptivo: el prototipo se descarta luego de terminar esta fase
- Evolutivo: se genera el prototipo de manera que evolucione al sistema final.

¿Cuáles deben ser las características de una SRS?

Correcta: todos los requerimientos en la SRS son requerimientos deseados por el cliente.

Completa: todos los requerimientos deseados por el cliente están en la SRS

No ambigua: los requerimientos tienen un solo significado

Consistente: ningún requerimiento contradice a otro

Verificable: para cada requerimiento hay un proceso efectivo de garantizar que se cumple

Rastreable: puede determinarse el origen de cada requerimiento y cómo se relacionan con el software

● **herencia adicional:** qué elementos del SW satisfacen el requerimiento

● **nacia otras:** qué requerimiento satisface los elementos del SW

Modificable: permite incorporar cambios fácilmente manteniendo coherencia y consistencia

ordenadas respecto a importancia y estabilidad: tiene bien definido el orden de prioridades

¿Cuál es el objetivo de validar la SRS? ¿Cómo se valida?

Objetivo:

Asegurar

que ésta refleja de forma clara y cercana las necesidades del cliente. Se hace una revisión entre cliente, usuario, autor y desarrollador. Se deben usar checklists.

¿Cuáles son los errores más comunes a la hora de especificar requisitos?

- **Omisión:** algún requerimiento no está en la SRS
- **Inconsistencia:** hay requerimientos que se contradicen
- **Hechos incorrectos:** algún req. no es correcto
- **Ambigüedad:** hay requerimientos con múltiples significados

¿Para qué sirven las métricas?
¿Cuáles se usan en esta fase?

Proveen info cuantitativa para la administración del proyecto, permiten controlar su desarrollo.

Se usa el **QUNTO** función,
que estimá el tamaño

Y algunas métricas de calidad
directas: estiman el valor
de los atributos de calidad
indirectas: evalúan la efectivi-
dad de las métricas del control de
calidad.

Explique QUNTO función

Es una métrica que define el tamaño
en términos de funcionalidad

Hay 5 parámetros que logran capturar
la funcionalidad total del sistema

- entradas externas
- salidas externas
- archivos lógicos internos
- archivos de interfaz externa
- transacciones externas

Se clasifican todas las funciones del
sistema en algunos de estos parámetros
y se la vuelve a clasificar en Simple,
Promedio o compleja. Hay una
tabla que asigna un peso numérico
a cada función según su clasificación.

Luego se calcula

$$UPF = \sum_{i=1}^5 \sum_{j=1}^3 w_{ij} C_{ij}$$

peso

de la función

cent. de funciones
de tipo i con
complejidad j

Una vez obtenido el punto función no ajustado (UPF), se lo ajusta a la complejidad del entorno:

Se evalúan 14 características, (como comunicación de datos, reutilidad, facilidad para la operación y facilidad para la instalación) en una escala de 0 a 5 y con eso se calcula el factor de ajuste

$$CAF = 0,65 + 0,01 * \sum_{i=1}^{14} \gamma_i$$

evaluación
en la
escala de

Con eso calculado, obtenemos el punto función

$$PF = CAF * UPF$$

c/ característica

ARQUITECTURA DEL SOFTWARE

¿Qué es la arquitectura del software?

Es la estructura del sistema que comprende los elementos del SW, las propiedades externamente visibles de esos elementos y las relaciones entre ellas.

Es el diseño del sistema al más alto nivel.

¿Cuál es el rol de la arquitectura?

Entendimiento y **la presentación** del **comunicación**. Sistema a un alto nivel, esconde complejidades y facilita su entendimiento. Esto, a su vez, facilita la comunicación entre todos los actores involucrados (autores, clientes, usuarios, desarrolladores).

Reuso: La arqui debe elegirse de modo que los componentes queden reutilizables. Esto incrementa la productividad.

Construcción y **evolución**: La división en componentes guía el desarrollo del sistema. A medida que evoluciona el sistema la arqui ayuda a ver qué partes del sistema hay que modificar y el impacto de eso tendrá.

Análisis: permite analizar el sistema a alto nivel.

¿Qué es una vista de la arquitectura?
¿Qué tipos hay?

Una vista de la arquitectura es una representación del sistema como complejo por elementos y relaciones entre ellos.

Qué elementos será la vista dependerá de lo que la vista quiera resaltar.

La mayoría de las vistas propuestas son de alguno de estos 3 tipos:

Vista de módulos: colección de unidades de código, cada una implementa una funcionalidad del sistema.
Elemento principal: módulos.

Vista de componentes y conectores:
Elemento principal: unidades de ejecución
Estas interactúan con otras a través de conectores

Vista de asignación de recursos:
Cómo las distintas unidades del sistema asignan recursos, especifica las relaciones entre elementos del software y elementos del entorno.

• ¿Qué es un estilo arquitectónico?

Para la visión CAC, ¿cuáles son?

(componentes y conectores) → que será útil

Es una estructura general para la arquitectura de cierta clase de problema.

Define una familia de estructuras que satisfacen las restricciones de ese estilo.

• Tubos y Filtros: adecuado para sistemas que realizan fundamentalmente transformaciones de datos.

Un solo componente ⇒ Filtros

" " Connector ⇒ tubos

La info llega a un filtro, este la transforma y la pasa a otro filtro a través de un tubo.

Restricciones

- Filtros → Sólo produce y consume dato
 - no conoce la identidad de otros filtros
 - debe hacer buffering y sincronización

- Tubos → son unidireccionales
 - solo uno entre dos filtros:

llegar a uno a la entrada de otro

• Estilo de datos.

Comparados

Dos tipos de componentes:

- Repositorio de datos: provee almacenamiento permanente

- Usuarios de datos: acceden a los datos del repositorio, realizan acciones y vuelven a colocar los datos en el repositorio.

Se comunican solo a través del repositorio

Un solo conector \Rightarrow lectura/escritura

Variaciones

- Estilo difusión: todos los cambios en el repositorio se avisan a todos los usuarios

- Estilo repositorio: no se avisa, repositorio pasivo

• Client-Servidor:

Dos componentes: Cliente y Servidor

Solo se comunica con el

Servidor, es él quien siempre inicia la comunicación

Un conector \Rightarrow solicitud/respuesta

En general tiene estructura multi-nivel
3 niveles

- Nivel cliente: cliente

- " intermedio: lógica de negocio

del servidor

- Nivel base de datos: donde reside la info

• Publicar-Suscribir:

Dos componentes: los q' publican

los q' suscriben

Conectar: publicar/suscribir

Se publican un evento \Rightarrow Se invoca

la(s) componente(s) suscritas a él

• Peer-to-Peer: solo un componente,

el componente le que da servicio a otra

• Proceso q' se comunican:

Componente: 2 procesos \Rightarrow se comunican entre ellos

Conectar: mensajes $\xleftarrow{\text{a través de}}$

• ¿Qué es la diferencia entre
Arquitectura y diseño?

Arquitectura \Rightarrow diseño de muy alto nivel,
se enfocan en los componentes principales

Diseño \Rightarrow visto de módulos del sistema, estos módulos
se transformarán en el código de los componentes.

¿Qué es ATAM?

Debido a que la arqui tiene impacto en los atributos no funcionales del sistema, estos deben evaluarse.

ATAM (Architectural tradeoff analysis method) es una técnica elaborada y formal para la evaluación de la arquitectura.

Analiza la arqui para ciertos propósitos, identifica dependencias entre propósitos y hace un análisis de compromisos.

Pasos

1.- Recolección de escenarios: se recolectan los escenarios de interés para el análisis interacciones del sistema

2.- Realizar requerimientos y/o restricciones: se define qué se espera del sistema en el escenario, especificando los niveles deseados para el atributo de interés

3.- Describir las visas: se recopilan las visas que serán evaluadas

4.- Análisis específico de c/ atributo: se analizan las visas bajo los distintos escenarios separadamente para el atributo de interés.

5- Identificar puntos de sensibilidad y compromiso:

↓

elementos que tienen mayor impacto en un atributo

elementos que son puntos de sensibilidad para varios atributos.

DISEÑO DEL SOFTWARE

¿Cuáles son los criterios para evaluar el diseño?

- **Corrección:** fundamental. Un diseño es correcto si implementa todos los requerimientos y es sensible dentro las restricciones
- **Eficiencia:** uso apropiado de los recursos del sistema.
- **Simplicidad:** diseño simple fácil de comprender \Leftrightarrow impacto directo en la mantenibilidad. Facilita reading, de recepción y corrección de bugs y modificar el código

EF

SIMP

no son independientes.
Buscar equilibrio.

¿Cuáles son los principios fundamentales del diseño?

Diseño \Rightarrow proceso creativo.

No hay serie de pasos para crearlo.

Sí hay principios que seguir

Partición y jerarquía

el diseño se divide el problema en partes que sean manejables, solucionables y modificables por separado.
Estas partes se comunican y cooperan entre sí, hay que tratar de mantenerlas lo más independientes posible una de la otra.

El particionando genera jerarquía entre las partes.

Abstracción

describir el comportamiento externo de los componentes sin detallar sus interiores

los tipos

de componentes existentes:

Componentes \Rightarrow cajas negras

proveer comportamiento externo, ocultar detalle

en el proceso de diseño:

componentes no existen,

para decidir cómo interactúan solo es necesario el comportamiento externo.

dos mecanismos de abstracción

- **funcional:** especifica el módulo según la función que realiza (base para metodologías orientadas a funciones)
- **de objetos:** el sistema es un conjunto de objetos que proveen servicios. Solo estos son visibles fuera del objeto.

Base P/ metodologías orientadas a objetos

Modularidad

Un sistema es modular si consiste de componentes discretos que pueden implementarse separadamente y el cambio en una no tiene gran impacto en las demás.

Resulta de la conjunción de la abstracción y el particionado.

DISEÑO ORIENTADO A FUNCIONES

¿Qué es el diseño orientado a funciones?

El sistema es visto como una función que transforma entradas en salidas deseadas. Los módulos tmb se especifican como funciones o procesos

(funciones)
¿Qué es acoplamiento?
¿Cuáles son sus tipos?

El acoplamiento es el grado de dependencia que hay entre los módulos. Es un concepto inter-modular.
Debe ser **bajo**.

Ese se da por

- **Tipo de conexión:** tiene que ver con la complejidad y oscuridad de las interfaces de un módulo disminuye:

solo las entradas definidas en un módulo son usadas por otro (info de módulo a módulo solo se pasa a través de parámetros)

Comunicación:

usamos interfaces indirectas y oscuras (se usan atributos y operaciones internas al módulo, se usan variables compartidas)

- **Complejidad de cada interface:** considera la complejidad de los parámetros más compleja \Rightarrow más acoplamiento
 - **Tipo de flujo de información:** con qué info se comunican los módulos
- bajo acoplamiento \rightarrow la comunicación es sólo a través de info de datos o sólo info de control

alto acoplamiento \rightarrow

comunicación híbrida (info de datos + info de control)

(funciones)
¿Qué es cohesión? ¿Qué tipos hay?

Relación que hay entre los elementos del mismo módulo. Concepto
Debe ser alta. intra-modular.
Tipos (del de más baja cohesión al de más alta cohesión)

Causal: la relación no tiene significado lógico: hay ciertas lógicas en la relación temporal: los elementos se ejecutan al mismo tiempo.

Procedural: elementos en una misma unidad procedural

Comunicacional: elementos operan sobre el mismo dato

Secuencial: las salidas de un elemento es la entrada de otro

Funcional: todos los elementos trabajan juntos para cumplir una sola función

¿Qué es el diagrama de estructura?
¿Para qué sirve?

Notación gráfica para especificar la estructura estética del sistema. Representa los módulos jerárquica y sus interconexiones.

Sirve para que el diseñador pueda representar de forma rápida y compacta sus decisiones y así poder evaluarlas y modificarlas.

¿Qué es la metodología de diseño estructurado?

Pautas que ayudan al diseñador en el proceso de diseño.

Objetivo: especificar módulos de funciones y sus conexiones significativas en una estructura jerárquica con bajo acoplamiento y alta coherencia.

Pasos:

1. Reformular el problema como un DFD: captura el flujo de datos del sistema progreso.

2. Identificar entradas y salidas más abstractas:

MAI: elementos de datos en el DFD que están más distantes de la entrada real pero que aún quedan considerarse entradas.

MAO: idem. pero con las salidas.

Separar las distintas funciones del sistema:

- Subsistema q' realiza principalmente la entrada " " " " las transformaciones
- " " " " " la presentación de la salida.

3. Realizar el primer nivel de factorización: especificar el módulo principal, un módulo de entrada subordinado

por c/ MAI, un módulo de salida subordinado por c/ MAO, un módulo transformador subordinado por c/ transformador central.

4 - Factorizar los módulos de entrada, salida y transformadores:

- Módulos de entrada y salida: los transformadores que producen las MIAZ y las MAO son tratados como transformadores centrales. Sobre cada uno se requiere el paso 3.

- Módulos transformadores: Se determinan los subtransformadores que compuestos hacen al transformador.

5 - Mejorar la estructura:

Usando heurísticas y verificación

¿Qué son las heurísticas de diseño?

Reglas generales / Principios básicos para mejorar la estructura del sistema.
Típicamente ver con el tamaño del módulo, cantidad de flechas de entrada y salida, alcance del efecto y del control de un módulo.
"Por cada módulo, el alcance del efecto debe ser un subconjunto del de control"

¿Cuál es el objetivo de la verificación del diseño? asegurar que el diseño implemente los requerimientos

¿Cuáles son las métricas del diseño orientado a funciones?

- **Tamaño**: Cant. de módulos y sus jerarquías
- **Complejidad**: módulos simples es mejor
- **Métricas de red**: se enfoca en el diagrama de estructura. Es bueno si el módulo tiene un sólo módulo invocador
- **Métricas de especialidad**: cantidad de módulos dependientes de él que tiene otro módulo
- **Métricas de flujo de información**: Mide complejidad en función de los info q se envíen/sale, módulos q llaman/llama.
$$DC = \text{fan-in} * \text{fan-out} + \text{inflow} * \text{outflow}$$

Se calculan el promedio y desv. estandar de todos los módulos $\Rightarrow DC > \text{media} + \text{estándar} \Rightarrow$ ^{propenso a} error
 $\text{media} < DC < \text{media} + \text{estándar} \Rightarrow$ complejo
Normal en caso contrario.

¿Qué es el diseño orientado a objetos?

Objeto: definir las clases del sistema a definir y las relaciones entre esas

Estado: comportamiento dinámico

(obligados)
¿Qué es acoplamiento?
¿Típicos?

se der cls
POR INHERENCIA: métodos de
una clase invocan a otros
aumenta

métodos acceden a partes internas
de otros métodos, manipulan variables
de otras clases
disminuye

métodos se comunican a través
de parámetros

de componentes:
aumenta

una clase tiene variables de
otra clase

disminuye

las variables de una clase en otra
son atributos o parámetros de un método
de herencia:

aumenta

subclases modifican/eliminan declaraciones
de un método o modifican su comportamiento

disminuye

las subclases agregan variables de
instancia o métodos

(obligados)
¿Qué es cohesión? ¿Típos?

- de método: porque los elementos de un método están juntos en ese método
- aumentar

c) método implementar una sola función, todos sus elementos trabajan juntos para llevarla a cabo

- de clase: porque distintos atributos y métodos están en la misma clase
- aumentar regresión

clase \Rightarrow único concepto

disminuir

clase \Rightarrow varios conceptos

- de herencia: porque distintas clases están juntas en la misma jerarquía
- aumentar

clases jerarquizadas en consecuencia de la generalización-especialización.

¿Qué principios debe cumplir el diseño orientado a objetos?

Explicarlos brevemente

Principio abierto-cerrado

"Las extensiones deben ser abiertas para extenderse y cerradas para modificarse" B-Royer

El comportamiento del sistema
debe extenderse pero el código
existente no debe modificarse.
Se satisface con el correcto uso
de herencia y polimorfismo.

Principio de sustitución de LISKOV

"Un programador utiliza un objeto
de la clase C debería permanecer
inalterado si se reemplaza ese
objeto por uno de una subclase
de C"

Hubermann

Si se cumple con este principio,
tmb con el de Player

Principio de Responsabilidad Única:
Una clase debe tener una sola razón por la cual cambiar, es decir, una sola responsabilidad

Principio de segregación de interfaces:
Las interfaces deben ser específicas y centrarse en requerimientos de los clientes que las usan

Principio de inversión de dependencias:
Las clases deben depender de las abstracciones y no de las implementaciones concretas

¿Qué es OMT?

PASOS:

1 - **Producir diagrama de clases:**
obtenido en el análisis

2 - **Producir el modelo dinámico:**
especificar como cambia el estado de los objetos frente a un evento.
Derivar las operaciones de los objetos

3 - **Producir el modelo funcional:**
describir las operaciones que tienen lugar en el sistema, especificar cómo computar las salidas a partir de las entradas

4 - **Definir clases y operaciones (internas):** se evalúan

criticaránse las clases para ver si son necesarias en su forma actual, teniendo en cuenta cuestiones de implementación

5 - Optimizar: agregar asociaciones redundantes, guardar atributos derivados, usar hijos genéricos, ajustar la herencia.

• ¿Cuáles son las métricas en el diseño orientado a objetos?

- **Métodos heredados por clases:** mide complejidad de la clase. Depende de la cant. de métodos en la misma y su complejidad

- **Profundidad del árbol de herencia:** Una clase muy baja debajo de la jerarquía dificulta la predicción de su comportamiento.

- **Cantidad de hijos:** cant. de subclases inmediatas de una clase. Cuantificice el reuso.

- **Acoplamiento entre clases:** cantidad de clases a las qd. una clase está acoplada. Cuantificice modifiabilidad y propagación de errores.

- **Reservas para una clase:** cantidad de métodos qd. qd. ser invocados como respuesta a un mensaje recibido por un objeto de una clase.

DISEÑO DETALLADO

¿Qué se hace en el diseño detallado?

Se especifica la lógica que implementará cada uno de los módulos del diseño de nivel intermedio.

¿Qué es el PDL? ¿Ventajas? ¿Desventajas?

Process Design Language. Aquí se expresa el diseño en un lenguaje que sea preciso y no ambiguo y que pueda ser convertido fácilmente a implementación.

Síntesis de un lenguaje de programación estructurado con vocabulario de lenguaje natural.

Ventajas:

- Fácil de integrar con código fuente, lo hace fácil de mantener
- Permite la declaración de datos y procedimientos
- Forma más barata y efectiva de cambiar la arquitectura

Desventajas:

- No explica funcionalidad de manera comprendible
- La notación solo es comprendible para personas con manejo del PDL.