Introducción a la Ingeniería del Software

Programación Modular
Ingeniería en Telecomunicación

Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga

Contenido

La Crisis del Software

La Crisis del Software

Muchos proyectos software presentan **deficiencias**:

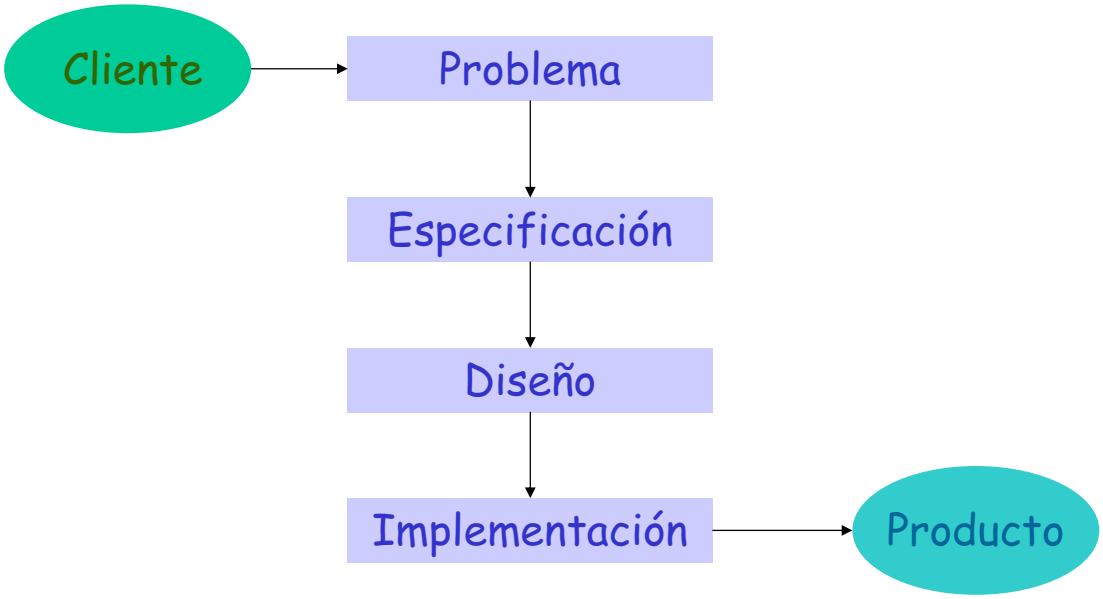
- Retraso en la entrega
- Falta de fiabilidad
- Coste excesivo
- Ineficiencia
- Mantenimiento problemático
- Falta de adaptabilidad
- Escasa portabilidad
- Carencia de documentación, ...

¿Por qué es tan difícil desarrollar software?

Desarrollar software puede ser un gran desafío intelectual:

- Problemas grandes, complejos y **muy variados**
- **Formalismos** inadecuados
- Gran diferencia entre la **teoría** y la **práctica**
- Imposibilidad de utilizar **aproximaciones**

Fases de desarrollo de un proyecto

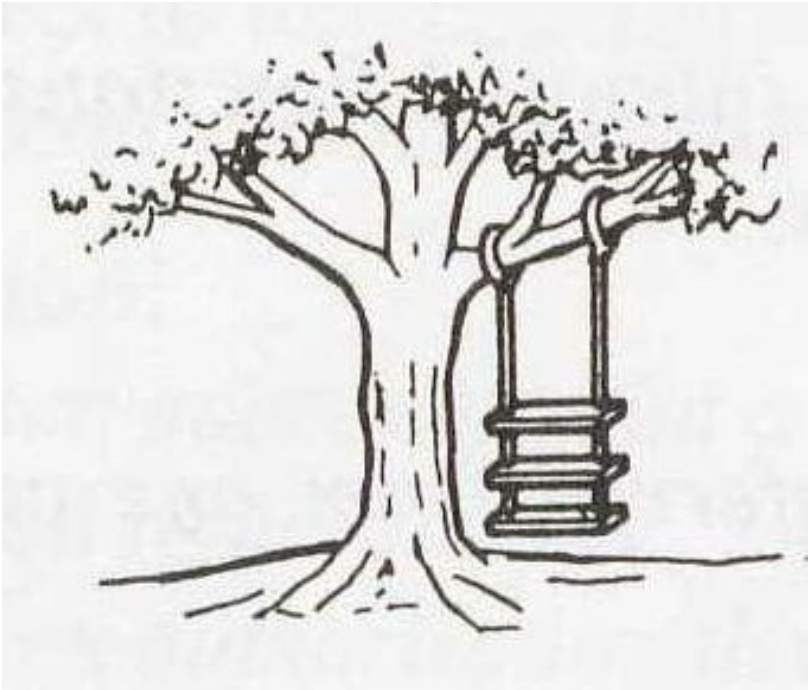


El desarrollo indisciplinado conduce a errores

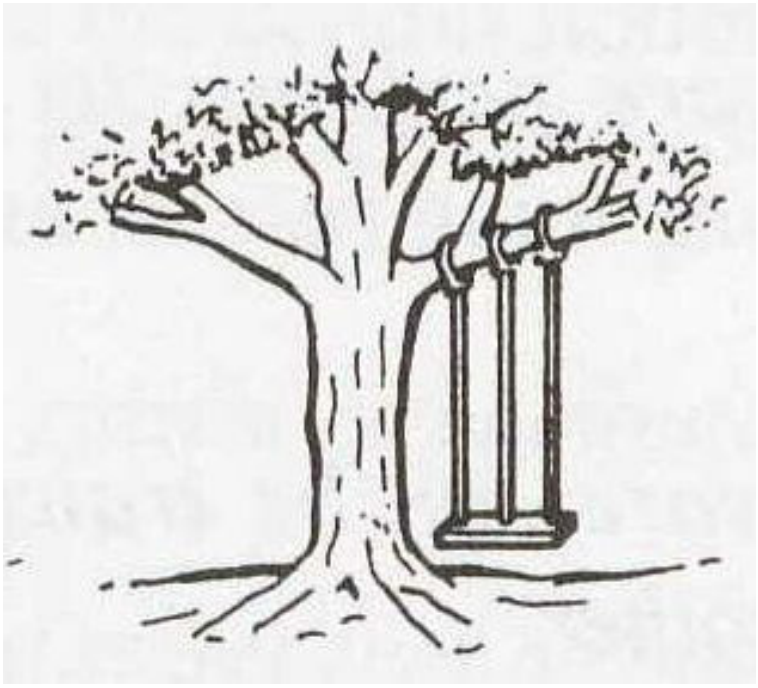
Un proyecto desarrollado de forma indisciplinada tiene muchas posibilidades de **fracasar**...

Ejemplo: diseñar un columpio

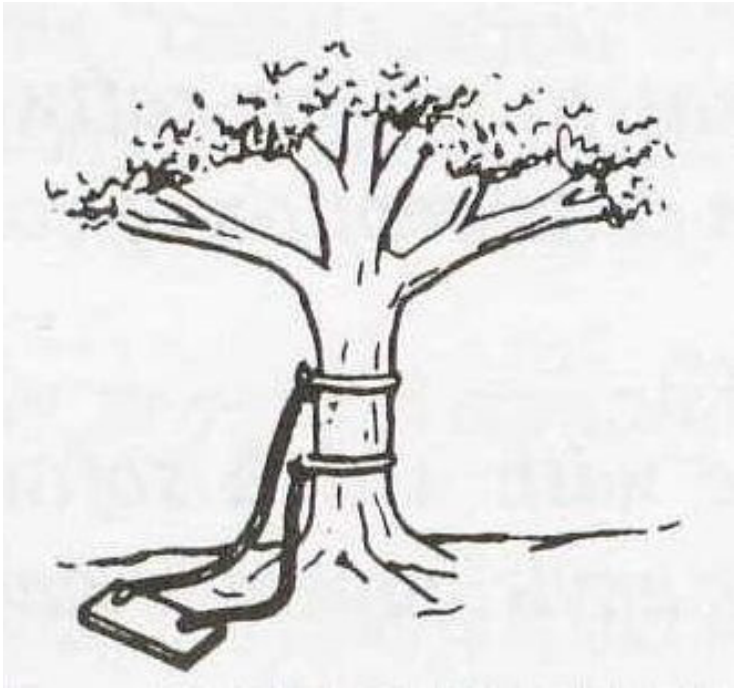
Lo que aparece en la definición del problema...



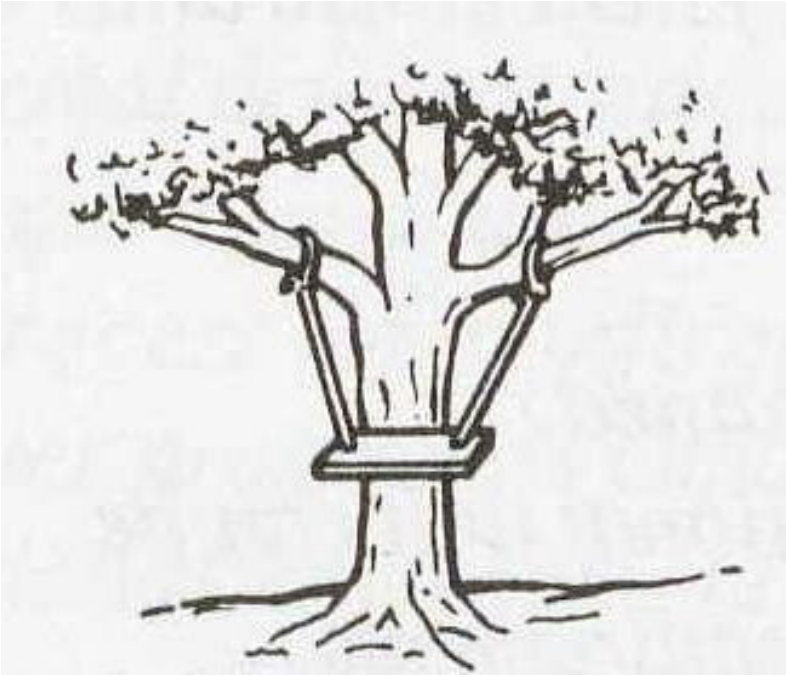
Lo que aparece en la especificación...



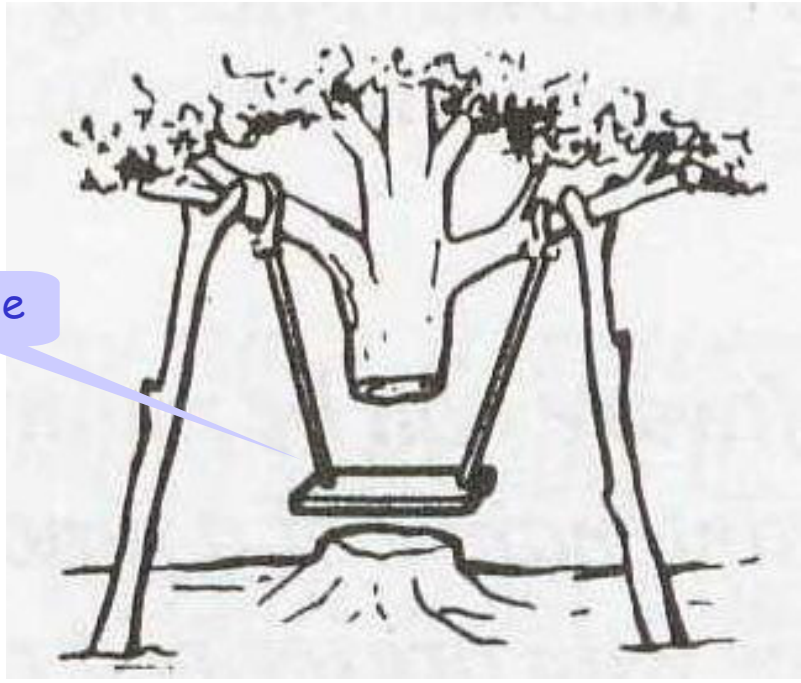
Lo que aparece en el diseño...



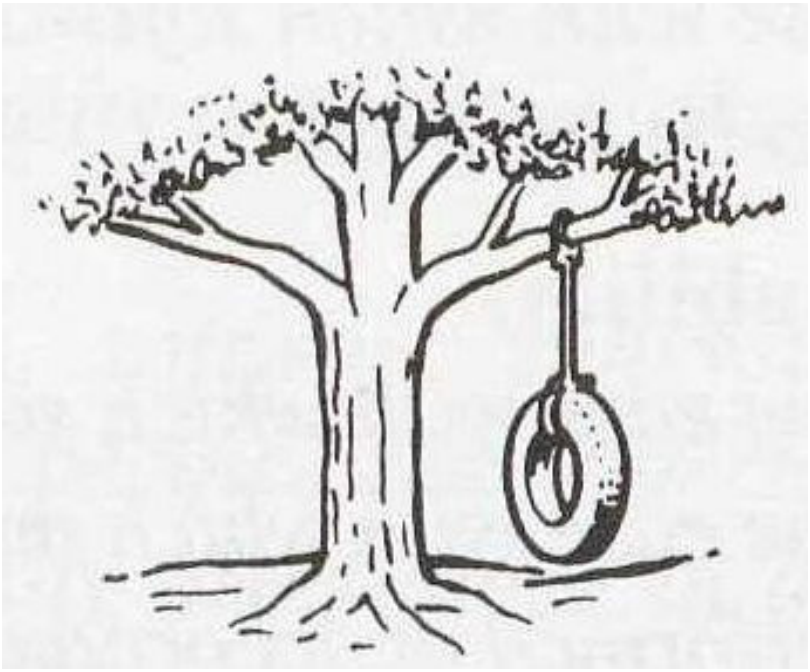
Lo que aparece en la implementación...



Lo que se entrega al cliente...



Lo que el cliente quería...



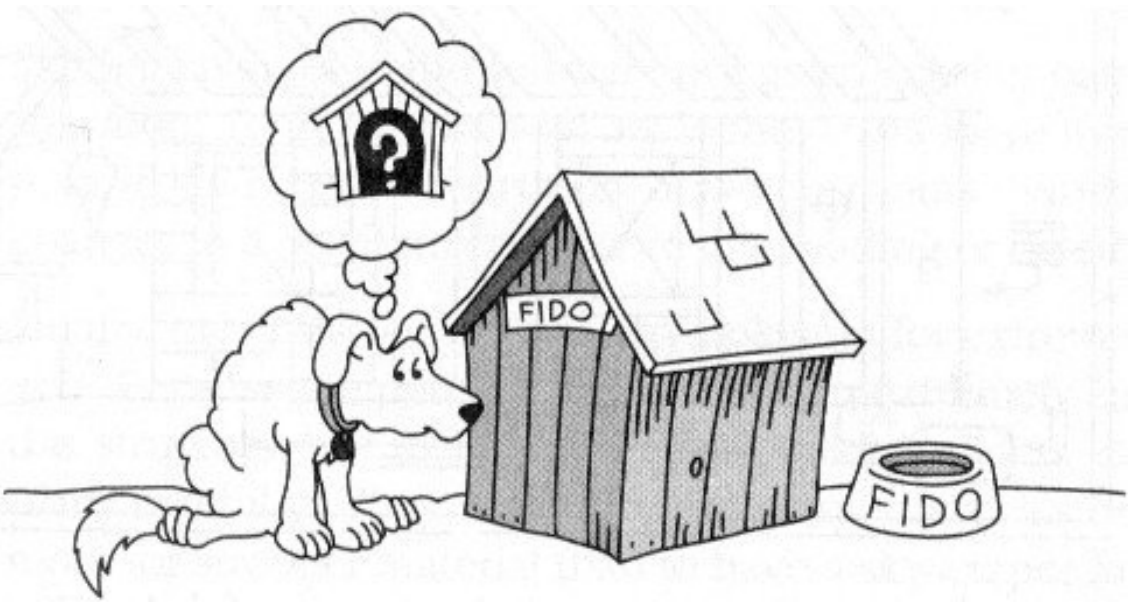
Construcción de Software

Desarrollar software es como construir un edificio: hay mucho que hacer antes del "verdadero" trabajo...

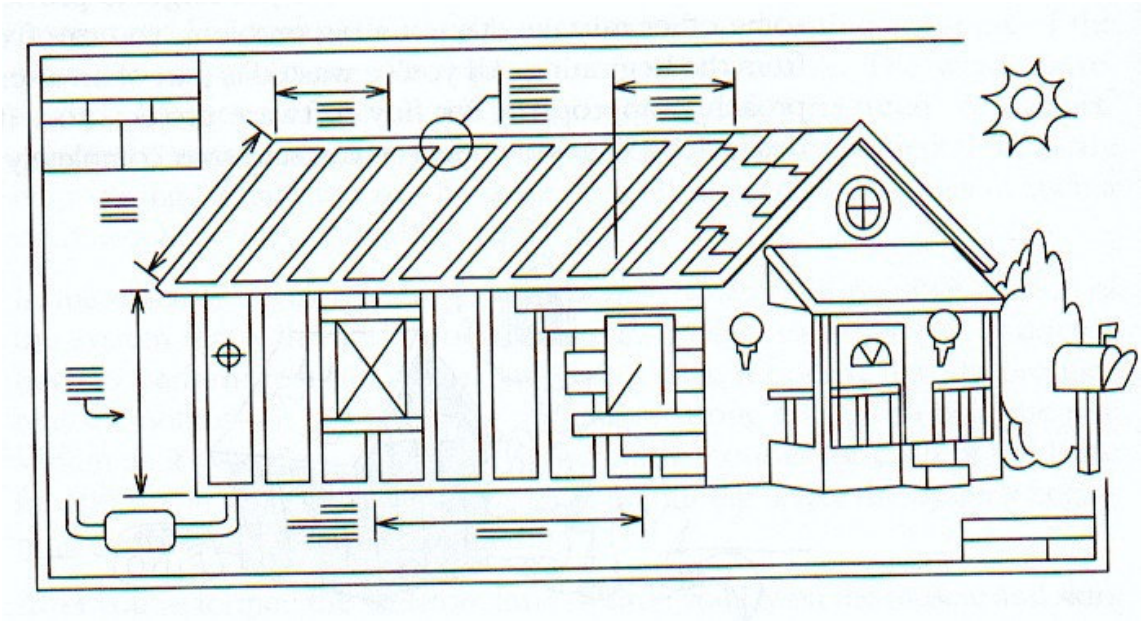
- Planificar minuciosamente
- Elegir materiales
- Establecer y respetar una temporización
- Inspeccionar frecuentemente la obra
- Los errores son muy costosos de reparar
- La dificultad depende del tamaño

Los problemas de organización y gestión son tan complicados como los problemas técnicos

El coste del error depende del proyecto...



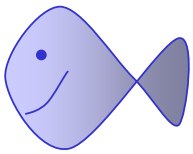
Si el proyecto es importante, planifica...



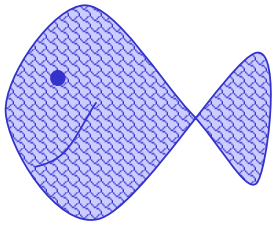
Metodologías de Diseño

En un ecosistema contaminado...

plancton radioactivo

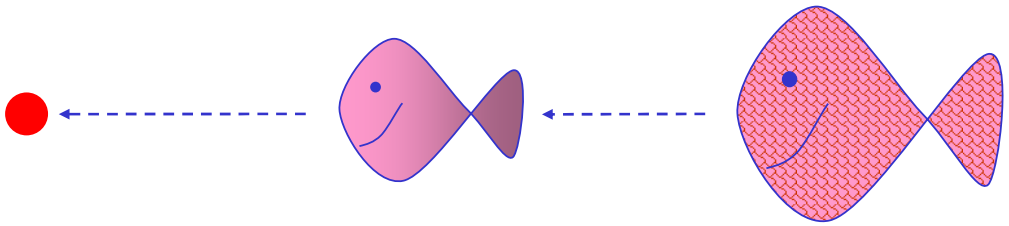


pez con parásitos



pez con descamación

Las enfermedades se propagan...



plancton radioactivo

pez radioactivo con parásitos

pez descamado radioactivo con parásitos

Cada fase puede introducir errores...

Problema mal planteado

Especificación incorrecta

Diseño inadecuado

Implementación errónea

Los errores se propagan...

Problema mal planteado

↓

Especificación incorrecta del problema mal planteado

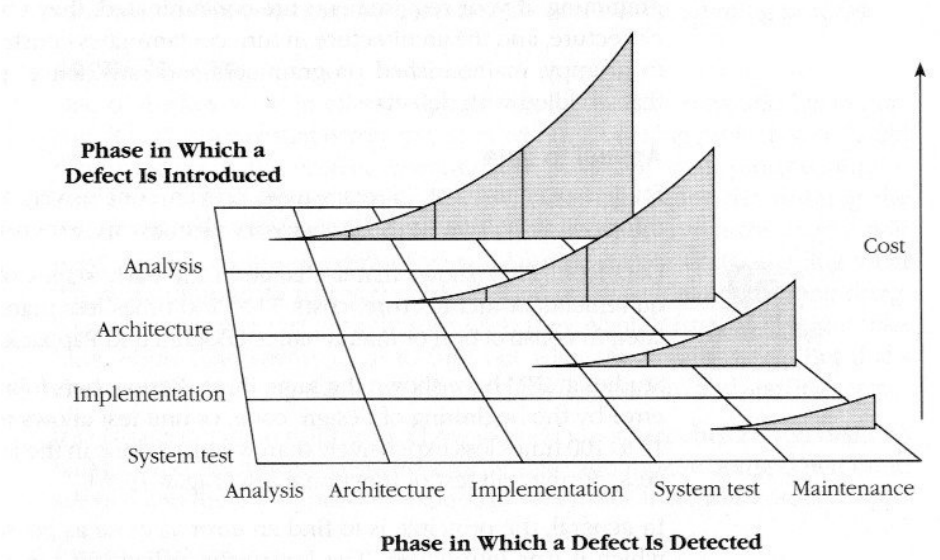
↓

Diseño inadecuado de la especificación incorrecta del problema mal planteado

↓

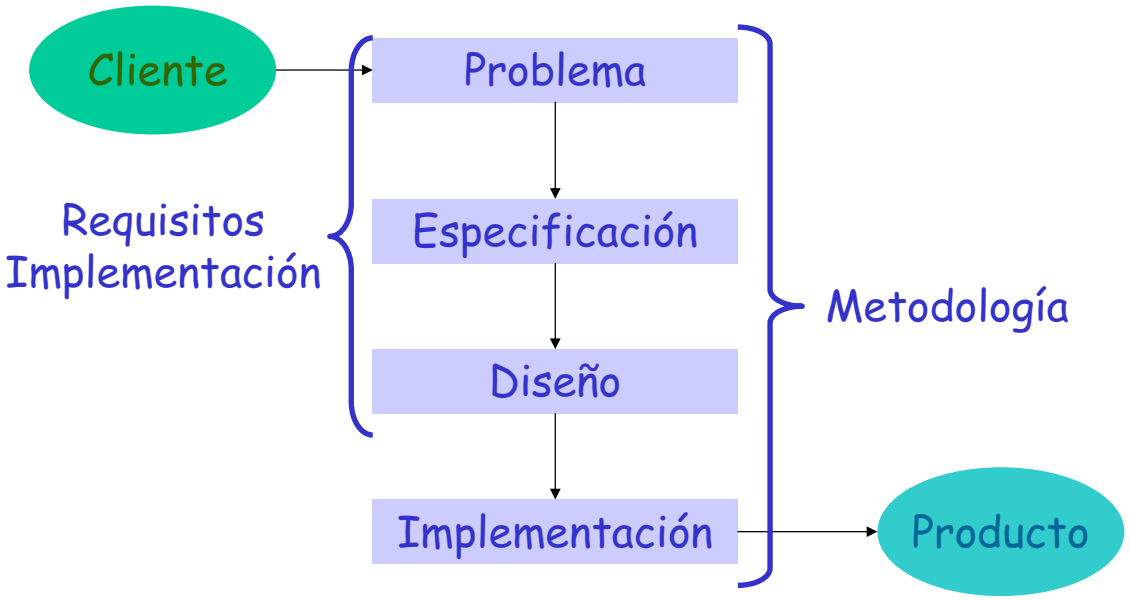
Implementación errónea del diseño inadecuado de ...

El coste de arrastrar los errores...



No empieces a codificar hasta que sepas lo que estás haciendo

Las metodologías planifican el desarrollo...



Fase 1: definición del problema original...

- En lenguaje natural
- Desde el punto de vista del usuario
- Sin referirse a una posible solución

¿Qué problema tengo que resolver?

Carrera espacial, años 60:

"Los astronautas no pueden tomar notas con los bolígrafos en ausencia de gravedad"

La visión de la NASA

- Problema:

"¿Cómo hacer que un bolígrafo funcione sin gravedad?"

- Solución:

"Bolígrafo espacial de avanzado diseño y gran coste"

La visión de la Agencia Espacial Soviética

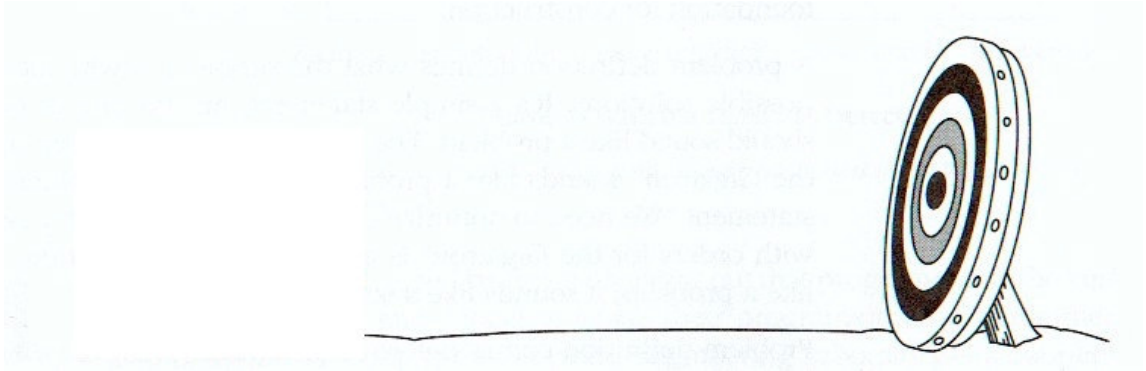
- Problema:

"¿Cómo tomar notas si no hay gravedad?"

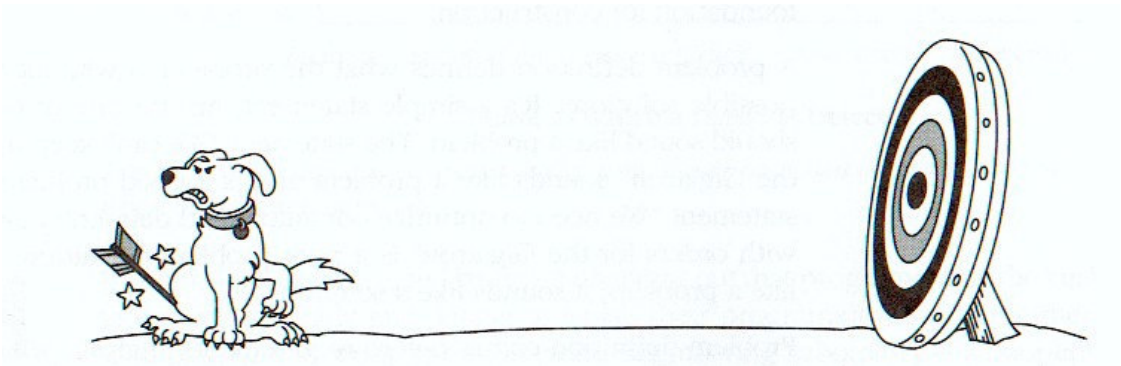
- Solución:

"Usando lápices..."

Define bien el problema antes de empezar...



Asegúrate de que sabes cuál es el problema...



Fase 2: especificación de la solución...

- Describe en detalle **qué** hace el sistema
- No describe **cómo** se hace
- Debe ser correcta
- Debe ser completa (contempla todos los casos)
- Emplea diagramas y notaciones formales
- Debe acomodar cambios (**se producirán**)

"Las especificaciones son como el agua, es más fácil construir sobre ellas cuando se han congelado"

¿Qué hace el sistema?

- Problema:

  Ordenar un vector de N elementos

- Especificación:

  Dado X[1..N], obtener Y[1..N] tal que:

  1) $\forall$ i $\exists$ j / Y[i] = X[j]
  2) $1 \leq i \leq N : Y[i] \leq Y[i+1]$

La especificación mejora tu puntería...

Fase 3: diseño de la solución...

- Describe **cómo** funciona el sistema
- Define la **estructura** del sistema:
  - ✓ qué **componentes** existen
  - ✓ qué **papel** juega cada componente
  - ✓ cómo se **relacionan** los componentes
- Justifica las **decisiones de diseño**
- Emplea diagramas y notaciones formales
- Debe acomodar cambios (**se producirán**)
- Independiente del lenguaje, el S.O. y la máquina
- Guía la implementación

¿Cómo lo hace el sistema?

- Problema:

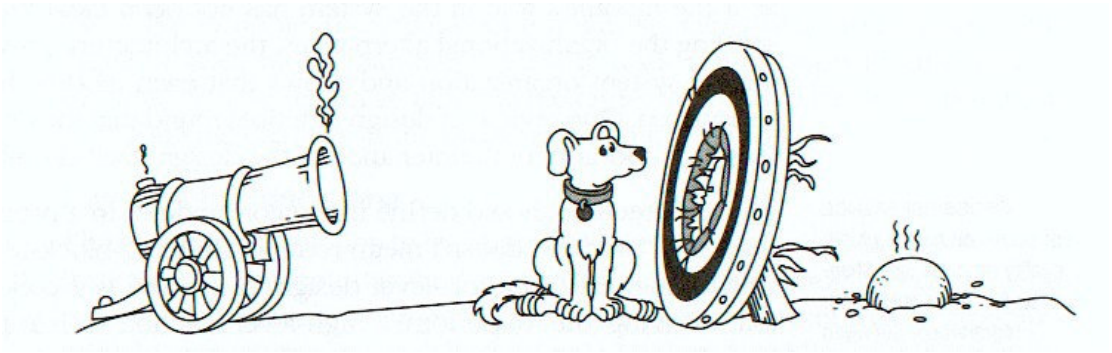
  Ordenar los números de la lotería primitiva

- Diseño:

  Almacenamos los números extraídos en un array y aplicamos quicksort...

  La solución es más complicada que el problema...

El diseño templa tu fuerza...



Calidad del Software

Calidad del Software

Las metodologías mejoran la **calidad del Software**:

- internamente (desarrolladores)
- externamente (usuarios)

Factores Internos de Calidad del Software

De cara al **desarrollador**, el software debe ser:

- Comprensible
- Legible
- Mantenible
- Flexible
- Portable
- Reutilizable
- Comprobable

Factores Externos de Calidad del Software

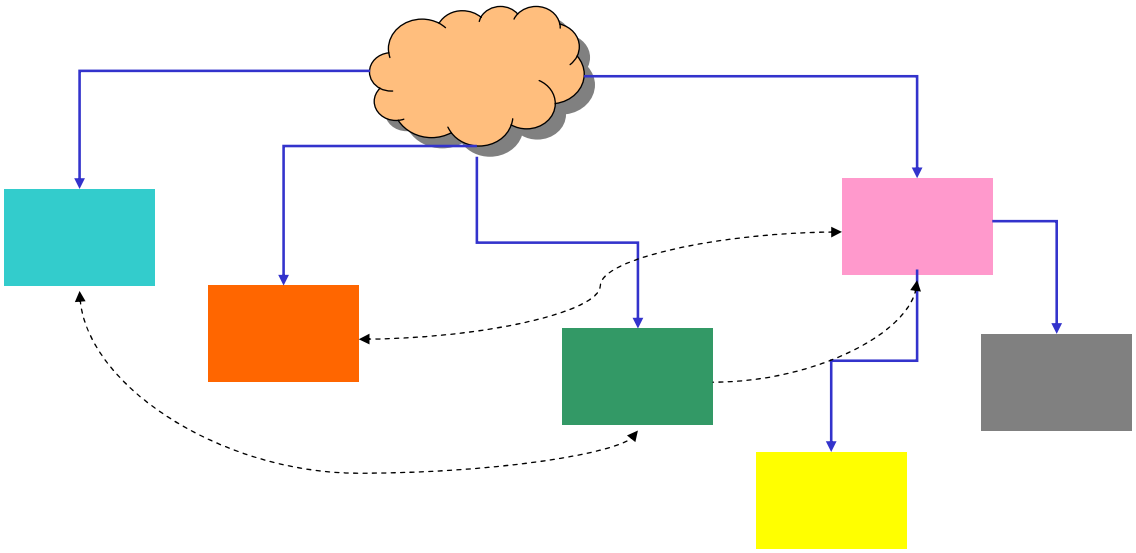
De cara al **usuario**, el software debe ser:

- Correcto
- Preciso
- Fácil de usar
- Eficiente
- Seguro
- Robusto

Principios de Diseño

Diseño Descendente Modular

Descomponer el sistema en **módulos** interconectados entre sí



Cuestiones que surgen...

- ¿qué criterio de descomposición emplear?
- ¿qué es exactamente un módulo?
- ¿cómo identificar un módulo?
- ¿cómo conectar los módulos?
- ¿cómo se comunican los módulos?

Las diferentes respuestas dan lugar a diferentes **metodologías de diseño**

Metodologías de Diseño

Metodología	Módulo
Abstracción Procedimental	Subprograma
Abstracción de Datos	Tipos
Diseño Estructurado	Proceso
Diseño Orientado a Objetos	Clase

Propiedades comunes de los módulos

Un módulo es siempre un **contenedor** de "recursos" con las siguientes propiedades:

- Ocultamiento de información
- Cohesión
- Acoplamiento

Ocultamiento de Información

IDEA -> dado un módulo, distinguir entre:

- **qué** hace y **cómo** lo hace
- su **uso** y su **funcionamiento**

MÉTODO -> descomponer los módulos en dos partes:

- Interfaz
- Implementación

Interfaz

- Parte **pública**, **visible** del módulo
- Determina **qué** servicios se ofrecen al usuario
- Indica el modo de **uso** ("instrucciones")
- Orientado al **usuario**

Implementación

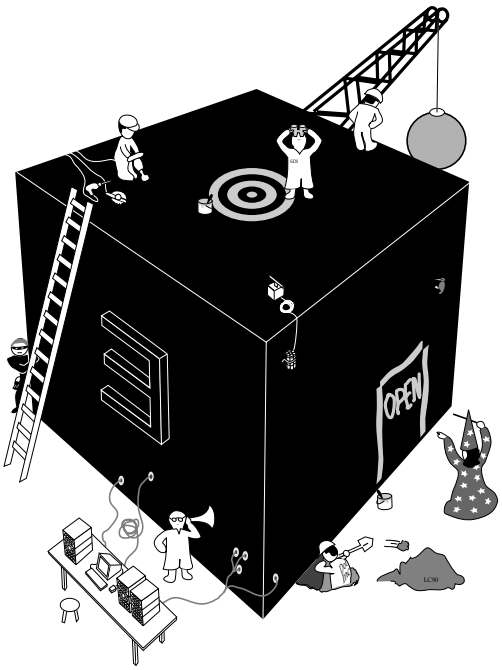
- Parte **privada**, **oculta** del módulo
- Determina **cómo** funcionan los servicios ofrecidos
- Oculta detalles no relevantes para el usuario
- Sólo el implementador puede acceder a la implementación

Interfaz vs. Implementación

Interfaz
(visible, público)

Implementación
(oculto, privado)

Nadie sabe qué contiene una caja negra...



Ejemplo: Abstracción Procedimental (I)

Módulo = Subprograma

```
ALGORITMO Ordenar(ES TVector A)
```

Interfaz

```
// Ordenar A por burbuja
VAR N i,j, ...;
INICIO
  PARA i=1 HASTA tam(A) HACER
     ...
  FINPARA
FIN
```

Implementación

Ejemplo: Abstracción Procedimental (II)

El usuario está **aislado** de cambios en la implementación

```
ALGORITMO Ordenar(ES TVector A)
```

Interfaz

```
// Ordenar A por quicksort
VAR N pivote, ...;
INICIO
  SI tam(A)>1 ENTONCES
    Partir(A,pivote);
    ...
  FINSI
FIN
```

Implementación

Cohesión

- Propiedad **intra-modular**
- Mide la relación entre los contenidos de un módulo
- Diversos grados, de fuerte a débil
- Objetivo: Maximizar la cohesión

Ejemplo de cohesión débil

```
ALGORITMO R IRPF(E TEmpleado P)
VAR R irpf;
INICIO
  irpf= ...
  Escribir(irpf);
  Devolver irpf;
 FIN
```

Interfaz

Implementación

Escribir el IRPF **limita** los usos del módulo.
Ej: listado de nóminas, irpf promedio de la empresa,...

Acoplamiento

- Propiedad **inter-modular**
- Mide la relación entre los módulos de un sistema
- Diversos grados, de fuerte a débil
- Objetivo: Minimizar el acoplamiento

Ejemplo de Acoplamiento fuerte (I)

Programar un juego de ajedrez:

```
TTablero tablero;
TTurno   turno = {maquina, humano};

ALGORITMO Movimiento_Maquina();
ALGORITMO Movimiento_Humano();
```


Ejemplo de Acoplamiento fuerte (II)

```
ALGORITMO Movimiento_Maquina()
INICIO
  SI (turno == maquina) ENTONCES
     mov= calcular_mov(tablero);
     mover(mov,tablero);
     turno= humano;
  FINSI
FIN
```

Interfaz

Implementación

Ejemplo de Acoplamiento fuerte (III)

```
ALGORITMO Movimiento_Humano()
INICIO
  SI (turno == humano) ENTONCES
     mov= leer_mov(tablero);
     mover(mov,tablero);
     turno= maquina;
  FINSI
FIN
```

Interfaz

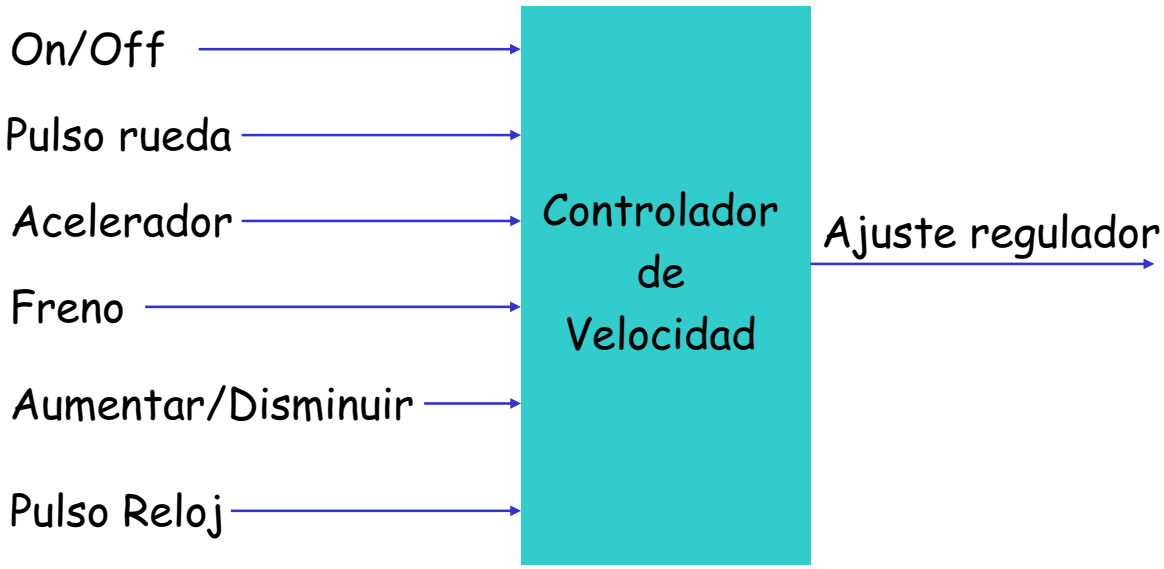
Implementación

La máquina no puede jugar contra sí misma u otras máquinas...

Un Caso de Estudio

Un controlador de velocidad de un coche

Objetivo: mantener una velocidad constante



Entradas y salida del controlador

On/Off: indica si el controlador está apagado o encendido

Pulso rueda: se recibe un pulso por cada rotación de la rueda

Acelerador: el controlador se desconecta si se pisa el acelerador

Freno: el controlador se desconecta si se pisa el freno

Aumentar/Disminuir: modificar la velocidad mantenida

Pulso reloj: se recibe un pulso cada milisegundo

Ajuste regulador: indica cuánto combustible se inyecta al motor

Detalles del funcionamiento del controlador

- El controlador debe mantener la **velocidad deseada**
- Los pulsos de la rueda y del reloj se emplean para calcular la **velocidad actual**
- Cuando se **enciende** el controlador, se toma la **velocidad actual** como **velocidad deseada**
- La velocidad deseada se puede **modificar**
- El regulador se ajusta según la **diferencia** entre la velocidad deseada y la actual
- El controlador se **desconecta** cuando el conductor pisa el freno o el acelerador

Diseño Estructurado

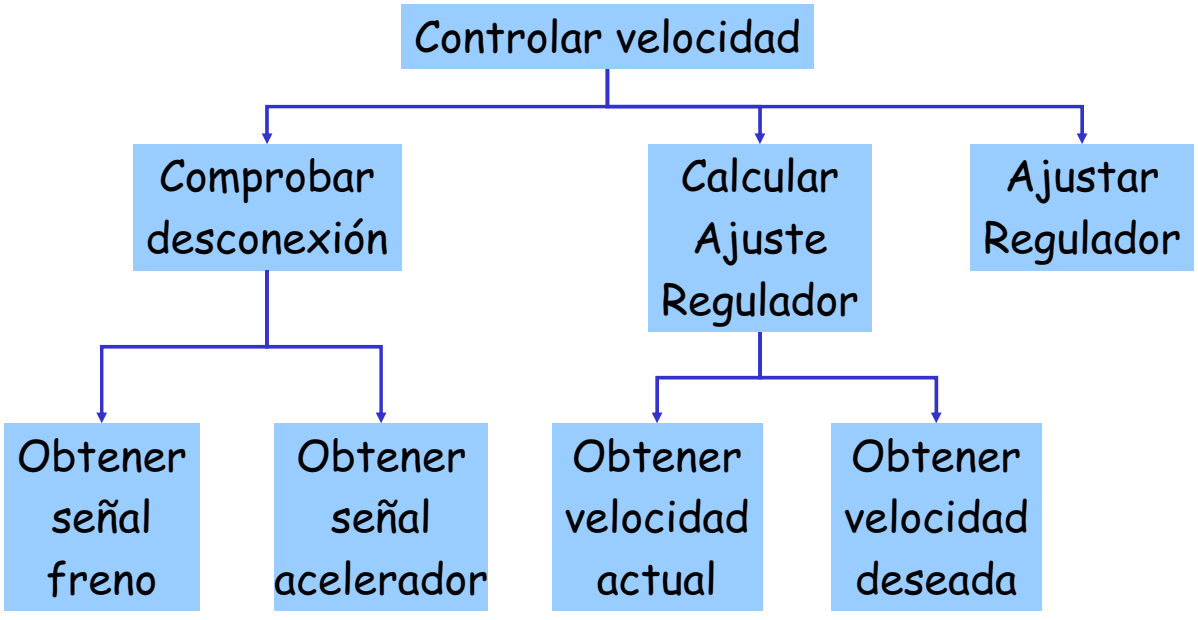
Claves del Diseño Estructurado

Diseño centrado en tareas (**entrada-proceso-salida**)

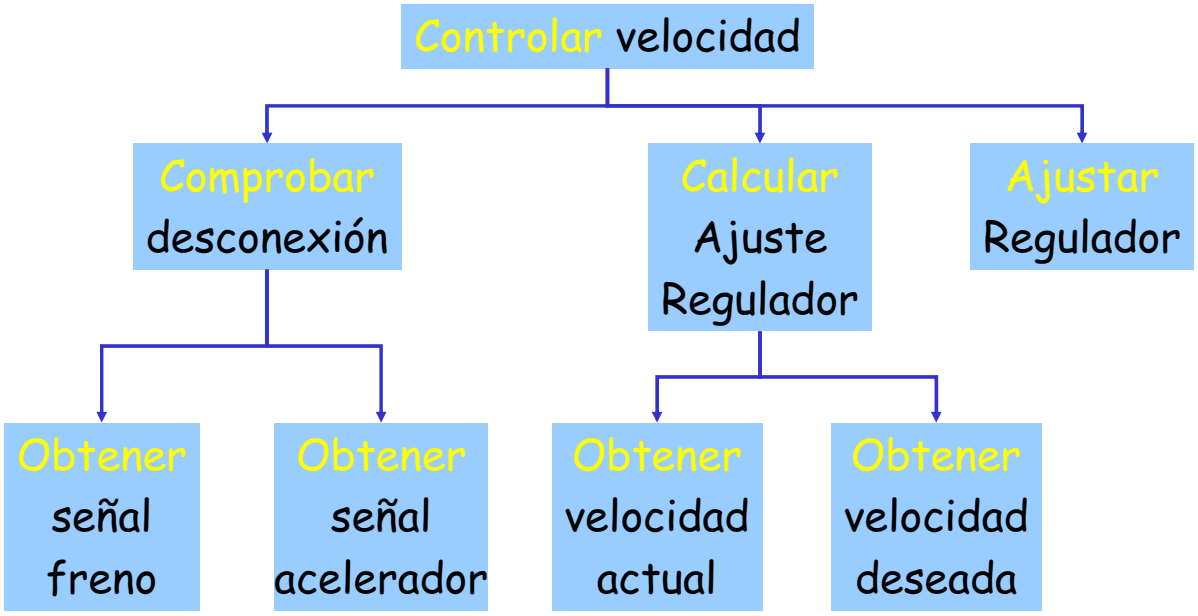
Identificar qué tarea realiza el sistema

1. Descomponer la tarea global en varias subtareas
2. Repetir el diseño para cada subtarea, hasta llegar a subtareas elementales

Descomposición Estructurada



Nos centramos en la acciones (verbos)



Diseño Orientado a Objetos

Claves del Diseño Orientado a Objetos

Diseño centrado en los objetos

1. Identificar los objetos ("personajes")
2. Determinar las acciones que realiza cada objeto ("el papel del personaje")
3. Determinar las acciones que cada objeto requiere de los demás ("relaciones entre personajes")
4. Repetir el diseño para cada objeto que no sea elemental

Descomposición Orientada a Objetos