

---

# Tema 3. Diseño e implementación

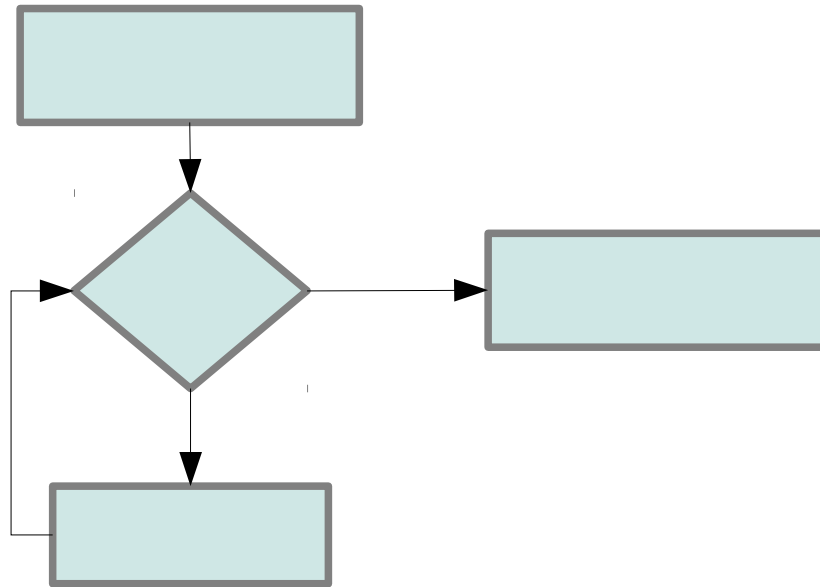
3.1. Introducción al diseño

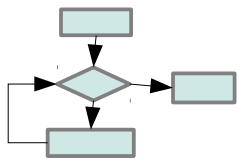
3.2. Diseño de la arquitectura

3.3. Diseño de los casos de uso

3.4. Diseño de la estructura de objetos

# Tema 3.1: Introducción al diseño





## Tema 3.1 Introducción al diseño.

3.1.1 Definición y características (R.S. Pressman cap. 9)

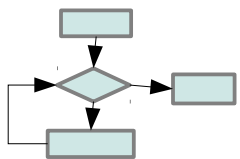
3.1.2 Principios de diseño (R.S. Pressman cap. 9)

3.1.3 Herramientas de diseño (R.S. Pressman pag 242-246).

3.1.4 Métodos de diseño.

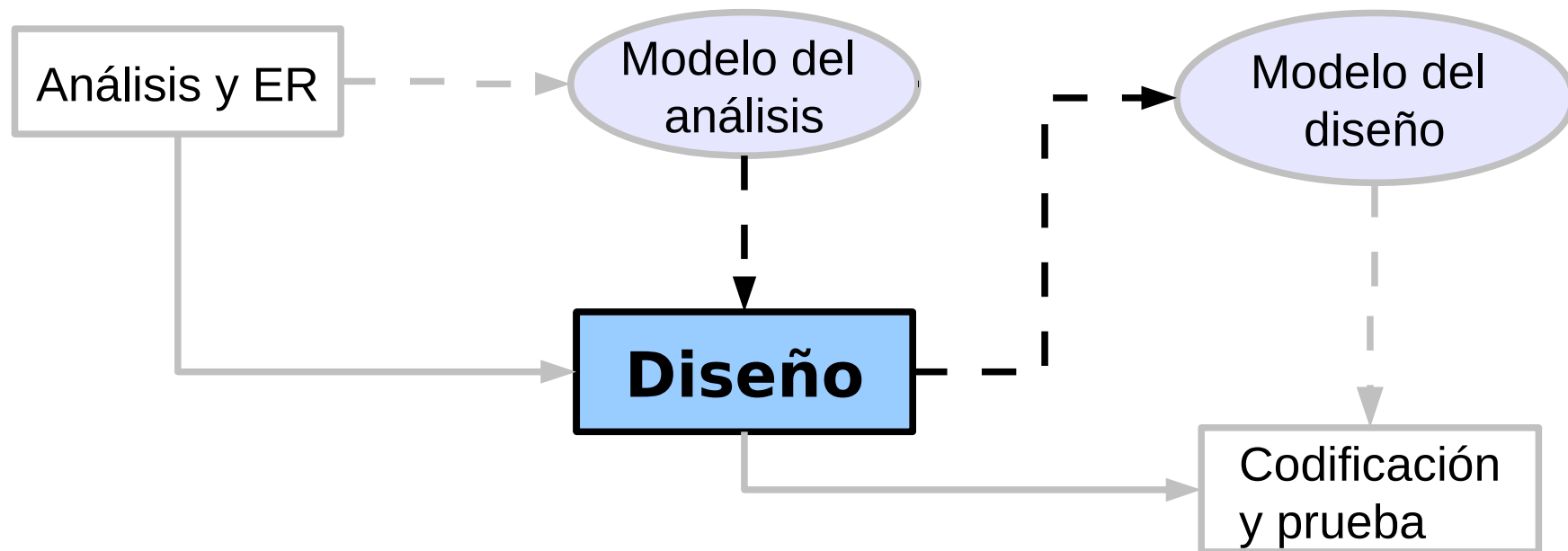
3.1.5 Modelo de diseño (J. Arlow, I. Neustadt cap. 16).

3.1.6. Tareas del diseño (J. Arlow, I. Neustadt cap. 16).

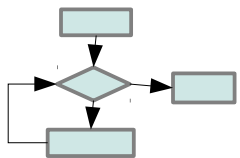


# Definición de diseño

El **diseño** es el proceso de aplicar distintos métodos, herramientas y principios con el propósito de definir un dispositivo, proceso o sistema con los suficientes detalles como para permitir su realización física.



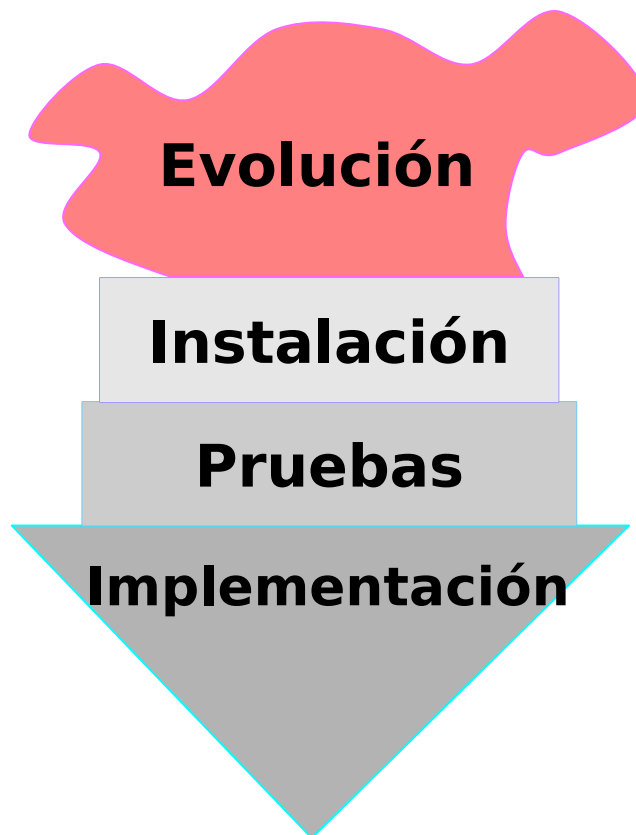
El **diseño de software** es el proceso de aplicar métodos, herramientas y principios de diseño, para traducir el modelo del análisis a una representación del software (modelo del diseño) que pueda ser codificada.



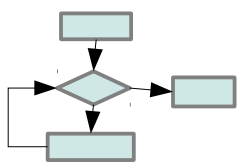
# El diseño es fundamental



**“Con Diseño”**



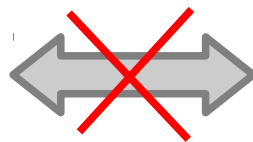
**“Sin Diseño”**



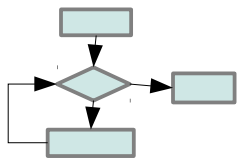
# Sus características

- El diseño implica una **propuesta de solución** al problema especificado durante el análisis.
- Es una **actividad creativa** apoyada en la experiencia del diseñador.
- Apoyado por principios, técnicas, herramientas, ...
- Es una tarea **clave para la calidad** del producto software.
- **Base para el resto** de las etapas del desarrollo (figura anterior).
- Debe ser un **proceso de refinamiento**.
- El diseño va a garantizar que un programa funcione correctamente.

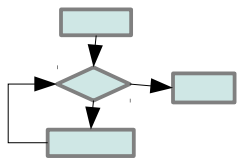
Hacer que un  
programa funcione



Hacer que funcione  
**CORRECTAMENTE**



- Modularidad.
- Abstracción
- Ocultamiento de información
- Independencia modular



# Modularidad: divide y vencerás

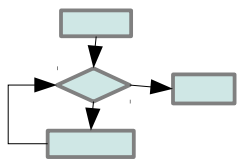
*"Un sistema software debe estar formado por piezas (Módulos), que deben encajar perfectamente, que interactúan entre sí para llevar a cabo algún objetivo común".*

Un **Módulo Software** es una unidad básica de descomposición de un sistema software y representa una entidad o un funcionamiento específico.

```
<Nombre> //que lo identifique  
    (<inicio>)  
        // contenido del módulo  
    (<fin>)
```

Pueden ser Módulos: una función, una clase, un paquete...

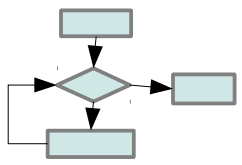




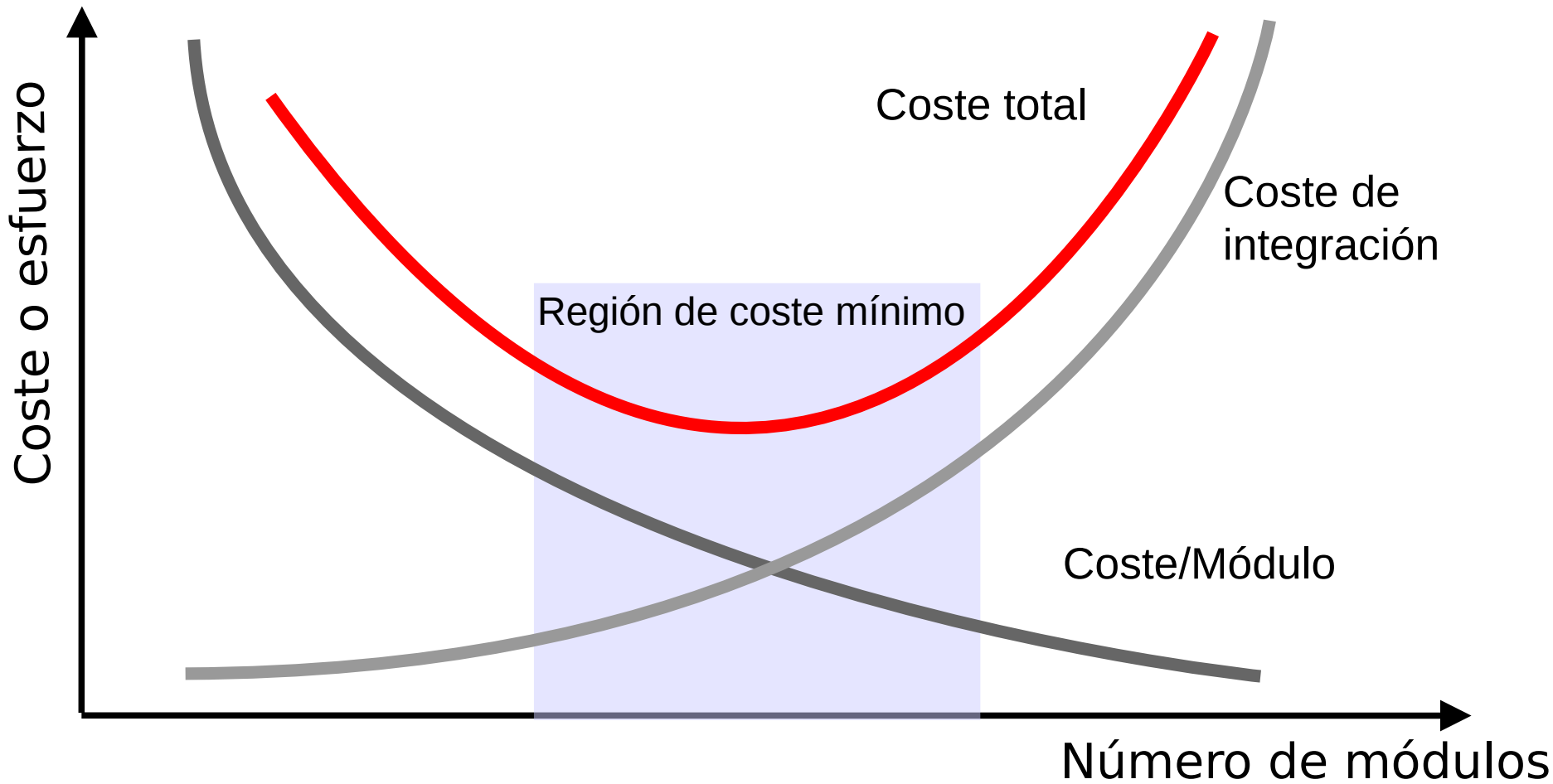
# Ventajas de la modularidad

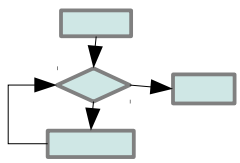
Los módulos:

- Son mas fáciles de entender y de documentar que todo el subsistema o sistema.
- Facilitan los cambios.
- Reducen la complejidad.
- Proporcionan implementaciones más sencillas.
- Posibilitan el desarrollo en paralelo.
- Permiten la prueba independiente.
- Facilitan el encapsulamiento.



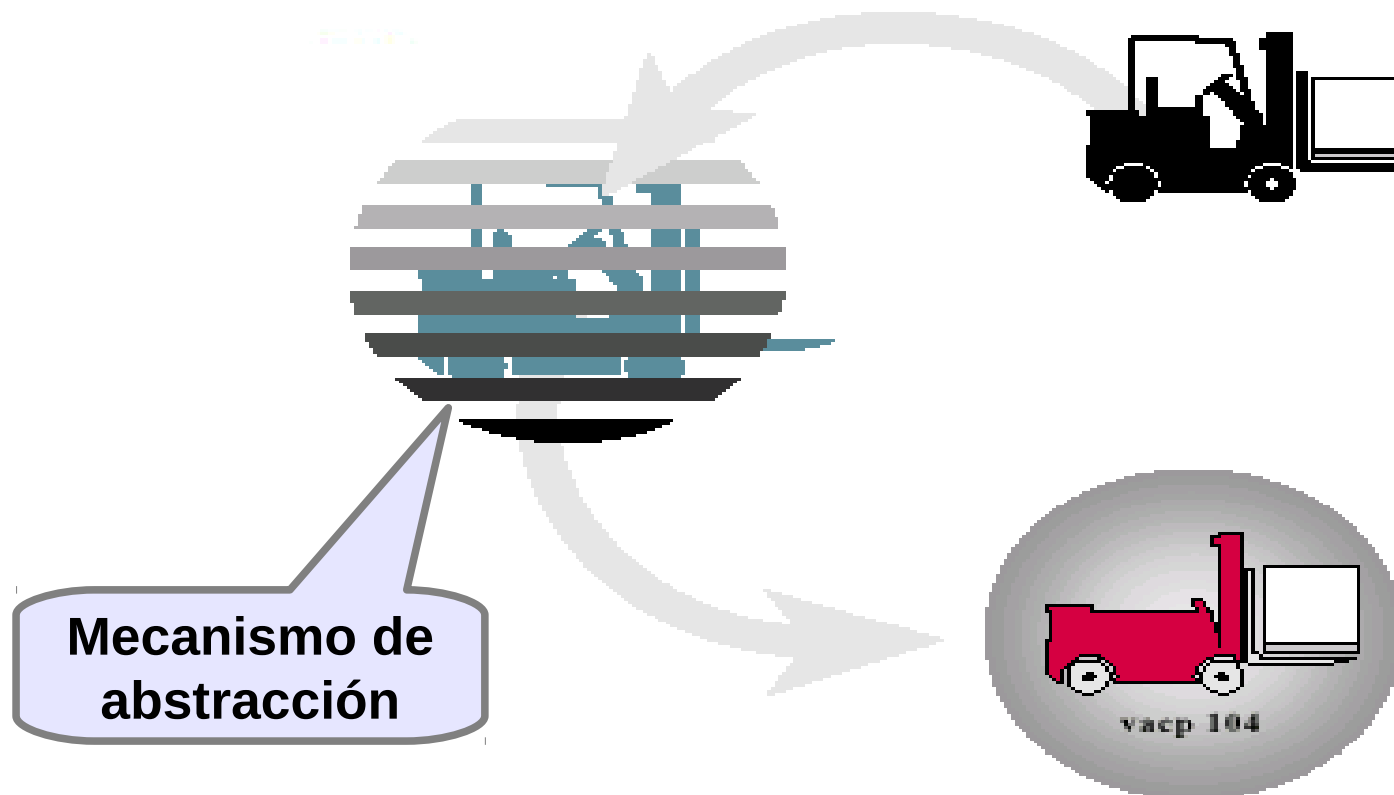
# Grado adecuado de modularidad

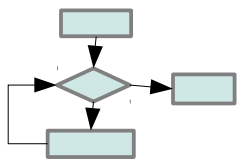




# Abstracción

*“Mecanismo que permite determinar qué es relevante y qué no lo es en un nivel de detalle determinado, ayudando a obtener la modularidad adecuada para ese nivel de detalle”*



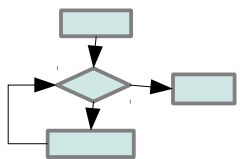


# Mecanismos de abstracción

## Mecanismos de abstracción en el diseño:

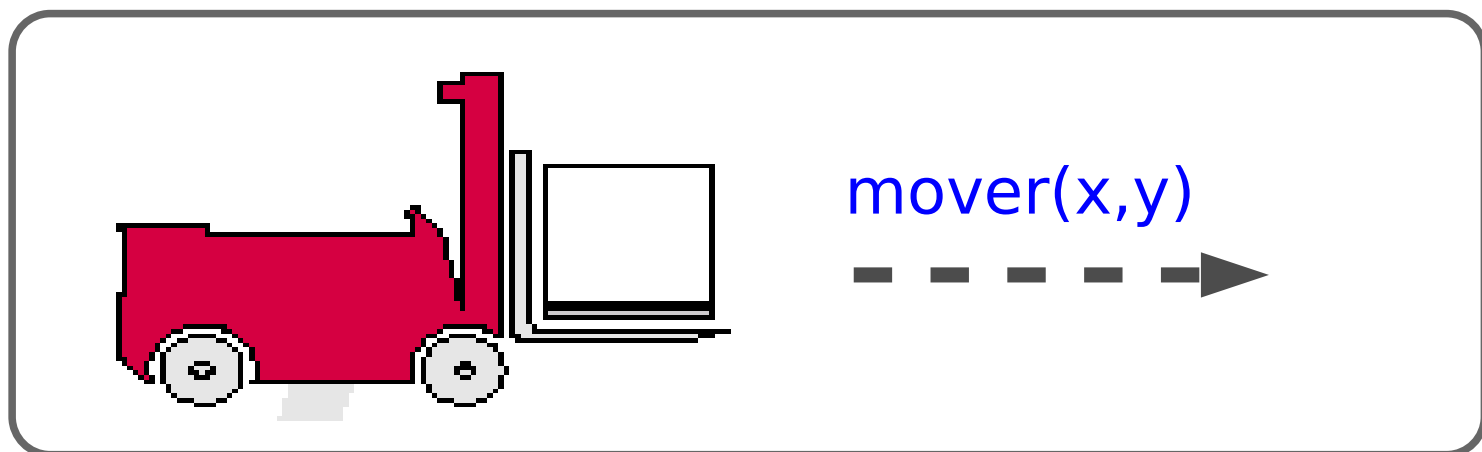
- Abstracción procedimental.
- Abstracción de datos.
- Abstracción de control.

Al proceso de ir incorporando detalles al diseño conforme vayamos bajando el nivel de abstracción se denomina **REFINAMIENTO**, propuesto por N. Wirth en 1971 (R.S. Pressman de 2006 pag.257).



# Abstracción procedimental

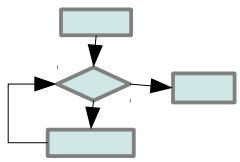
Se abstrae sobre el funcionamiento para conseguir una estructura modular basada en procedimientos.



Nivel de  
detalle muy  
elevado

**mover(x,y)**

*Descripción  
general del  
movimiento  
de la carretilla*



# Abstracción procedimental

**void moverLaCarretilla(float x, float y)**

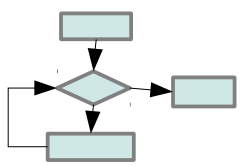
**//Precondición:** los valores de x e y deben estar dentro de los límites de movimiento de la carretilla.

**//Poscondicion:** la carretilla se ha desplazado desde la posición en la que se encontraba hasta la posición indicada por las coordenadas x e y.

{ *“Descripción del algoritmo de movimiento”* }

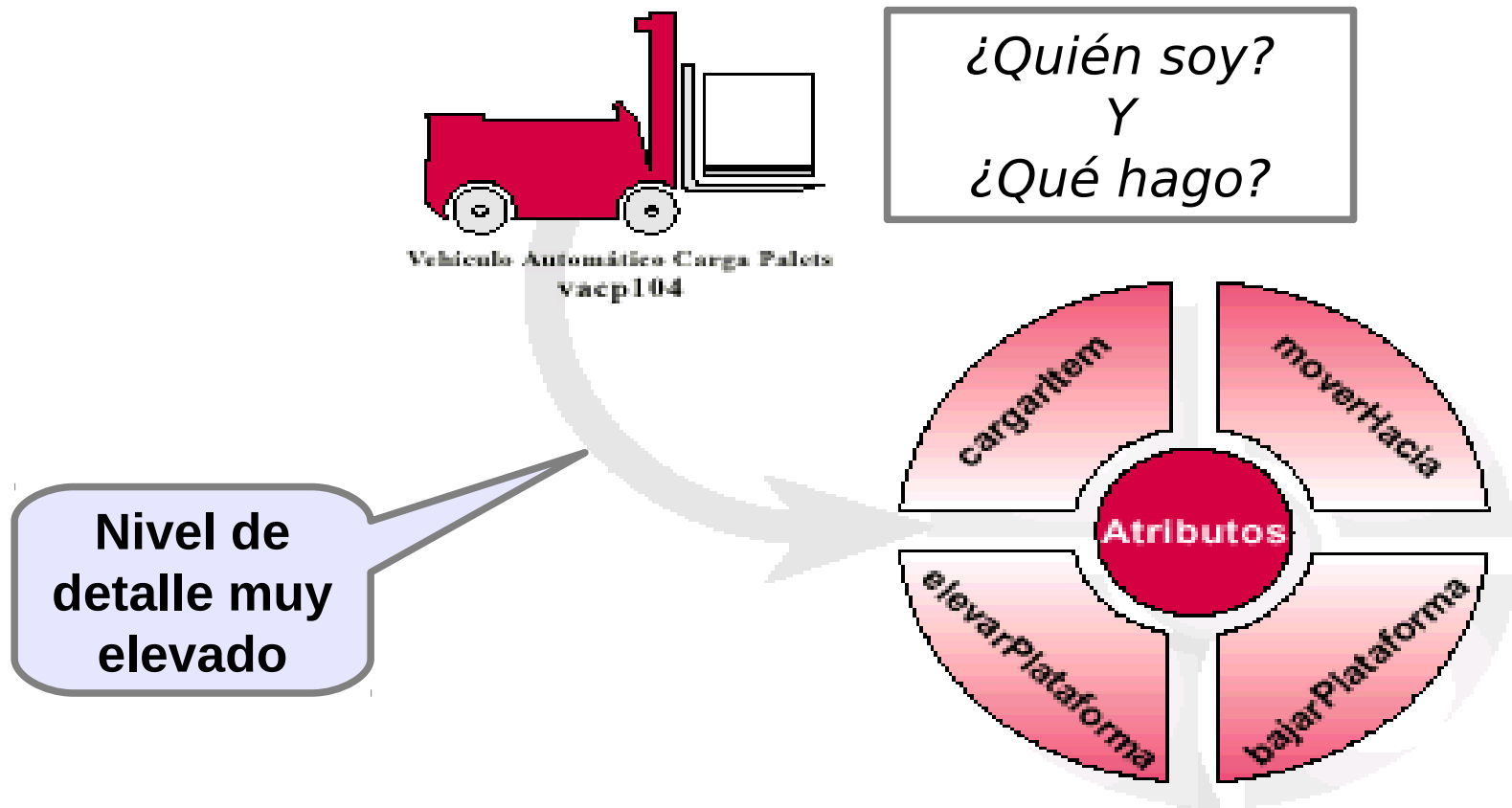
**Especificación  
de la  
abstracción  
procedimental**

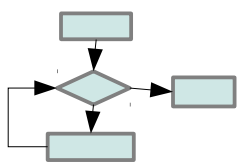
**Mediante un proceso de refinamiento llegamos a un nivel de detalle muy bajo o detallado**



# Abstracción de datos

Se abstrae tanto el funcionamiento como los atributos que definen el estado de una entidad, para obtener una estructura modular basada en el estado y funcionamiento de una entidad u objeto.





# Abstracción de datos

**Mediante un proceso de refinamiento llegamos a un nivel de detalle muy bajo o detallado**

**Nombre:** Carretilla

**Atributos que definen su estado:**

posicionX:float // posición X del plano en la que se encuentra la carretilla.

posicionY:float // posicion Y del plano en la que se encuentra la carretilla.

pesoMaximo:float // peso máximo que admite la carretilla

posiciónPala:float // posición en la que se encuentra la pala de carga

**Funcionalidad de los objetos carretilla:**

```
void moverLaCarretilla(x:float, y:float)
```

```
//Precondición: ....
```

```
//Poscondicion: ....
```

```
void cargarItem(peso:float )
```

```
//Precondición: ....
```

```
//Poscondicion: ....
```

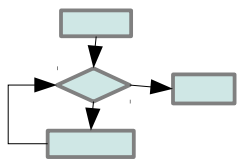
```
void moverPlataforma(z:float )
```

```
//Precondición: ....
```

```
//Poscondicion: ....
```

**¿Cuáles serían sus precondiciones y postcondiciones ?**





# Abstracción de control

Mecanismo que permite abstraer sobre el flujo de control de cualquier proceso en general.

```
10 hay datos?;
```

```
no
```

```
goto 20;
```

```
si
```

```
leer registro;
```

```
procesar registro;
```

```
guardar modificaciones;
```

```
goto 10;
```

```
20 // continuar
```

Mientras haya datos

{

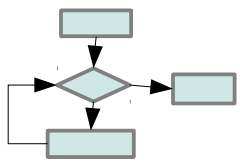
leer registro;

procesar registro;

guardar modificaciones;

}

Otros ejemplos: semáforos en SO e iteradores sobre colecciones

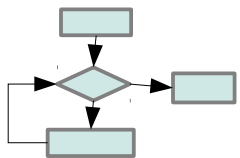


# Ocultamiento de información

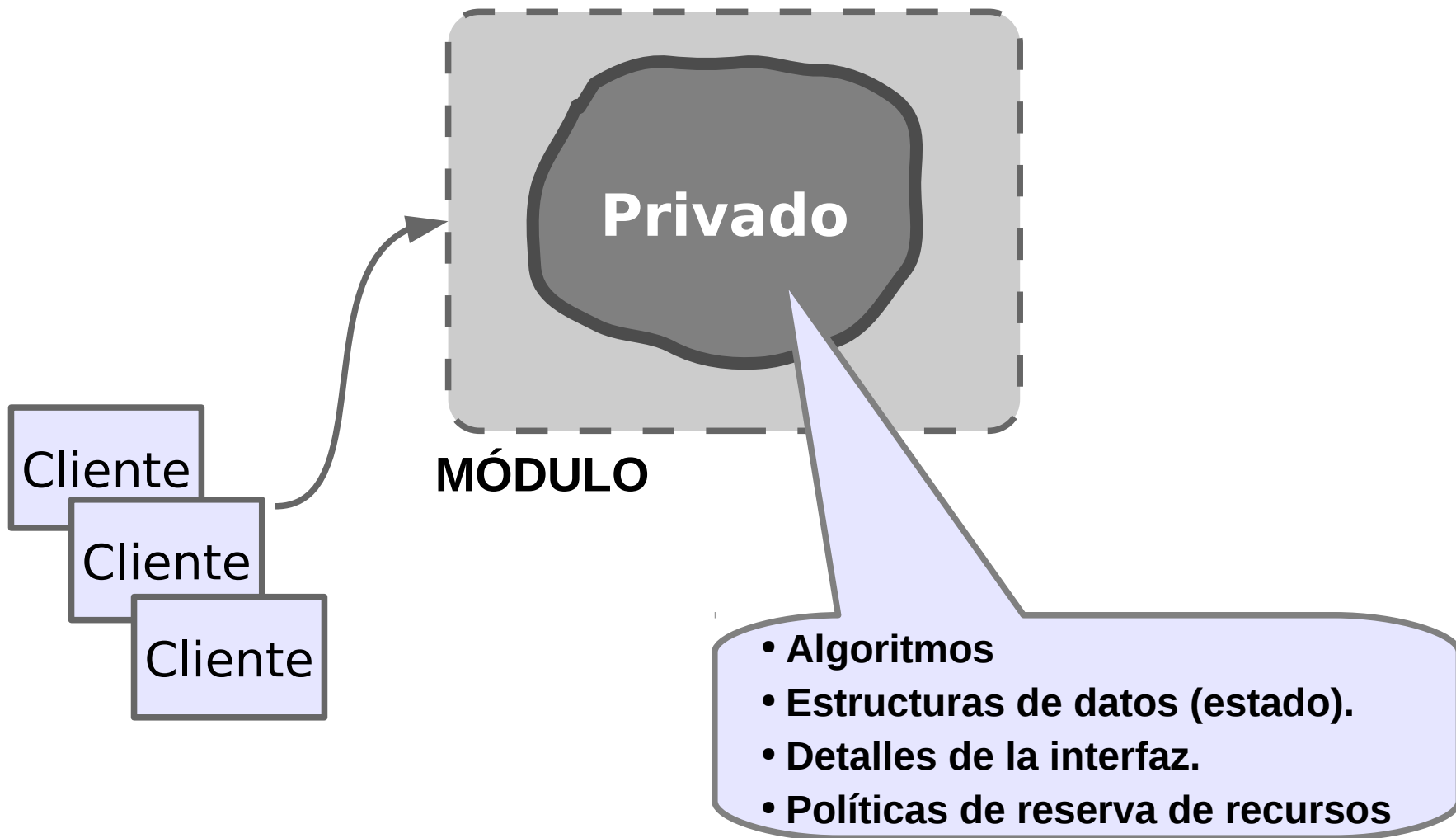
*“Un módulo debe especificarse y diseñarse de forma que la información (procedimientos y datos) que está dentro del módulo sea inaccesible para otros módulos que no necesiten de esa información”.*

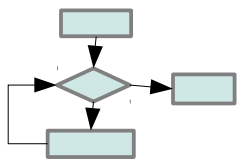
*Beneficios del ocultamiento de información:*

- Reduce la probabilidad de “efectos colaterales”.
- Limita el impacto global de las decisiones de diseño locales.
- Enfatiza la comunicación a través de interfaces controladas.
- Disminuye el uso de datos globales.
- Potencia la modularidad.
- Produce software de alta calidad.



# Ocultamiento de información





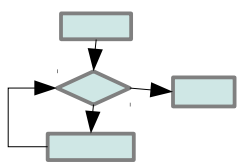
# Independencia modular: cohesión

*“Un módulo debe presentar un alto nivel de cohesión”*

**Cohesión:** Grado que tiene un módulo para centrarse en la realización de un solo objetivo y todos sus elementos deben estar ahí para llevar a cabo ese objetivo y ningún otro.

La alta cohesión proporciona módulos fáciles de entender, reutilizar y mantener.

- Si el módulo es un **procedimiento**: (Verlo en S. L. Pfleeger de 2002 p. 254-260 y R.S. Pressman de 2001 p. 229-231)
- Si el módulo es una **clase**, ésta presenta un nivel alto de cohesión si modela un solo concepto abstracto con un pequeño conjunto de responsabilidades íntimamente relacionadas y todas sus operaciones, atributos y asociaciones están para implementarlas”



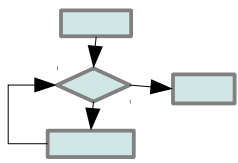
# Independencia modular: acoplamiento

*“Un módulo debe presentar un nivel de acoplamiento, con los demás módulos, lo más bajo posible”*

**Acoplamiento:** Medida de interdependencia entre módulos dentro de una estructura de software.

El acoplamiento entre módulos es indispensable y fundamental, hay que:

- Tratar de reducirlo siempre que sea posible.
- Mostrarlo de forma explícita en todos los modelos del diseño.
- Si el módulo es un **procedimiento**: (verlo en S. L. Pfleeger de 2002 p. 254-260 y R.S. Pressman de 2001 p. 229-231)
- Si el módulo es una **clase**, ésta debería relacionarse (mediante herencia, asociación o dependencia) sólo con la clases que necesite para llevar a cabo sus responsabilidades



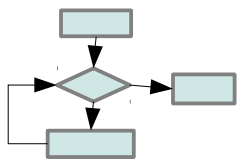
# Herramientas de diseño

Las **herramientas de diseño** son los instrumentos que ayudan a representar los modelos de diseño de software.

Algunas de las más usuales:

- Diagramas de UML: de clase, de interacción, de paquetes, de componentes, de despliegue...
- Cartas de estructura.
- Tablas de decisión.
- Diagramas de flujo de control:
  - Organigramas estructurados
  - Diagramas de NS.
- Lenguajes de diseño de programas (LDP).

VER Seminario. Herramientas de diseño diseño

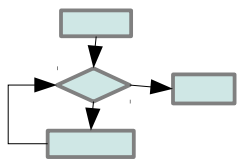


# Características de los métodos de diseño

Un **método de diseño** nos va a permitir obtener diseños de forma sistemática, dándonos las herramientas, las técnicas y los pasos a seguir para llevar a cabo el diseño.

Todo método de diseño **debe poseer**:

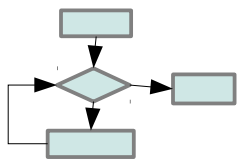
- **Principios** en los que se basa.
- **Mecanismos de traducción** del modelo de análisis al modelo de diseño.
- **Herramientas** que permitan representar los componentes funcionales y estructurales.
- **Heurísticas** que nos permitan refinar el diseño.
- Criterios para evaluar la calidad del diseño.



# Principales métodos de diseño

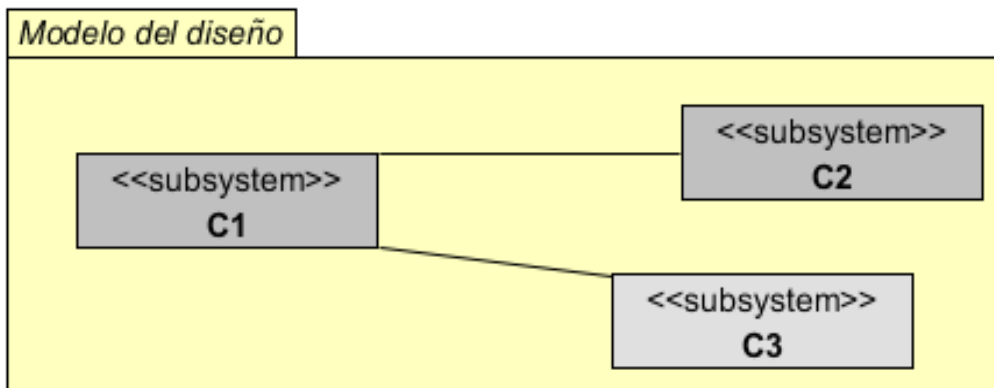
- **SSD** (Diseño estructurado de sistemas).
- **JSD** (Desarrollo de sistemas de Jackson).
- **ERA** (Entidad-Relación-Atributo).
- **OMT** (Técnicas de modelado de objetos).
- **Metodo de Booch** (Método de diseño basado en objetos)
- **Métodos orientado a objetos**: En la actualidad existe una gran variedad de métodos orientado a objetos, aunque la mayoría de ellos usan como herramienta de modelado UML y como proceso de desarrollo el PU.



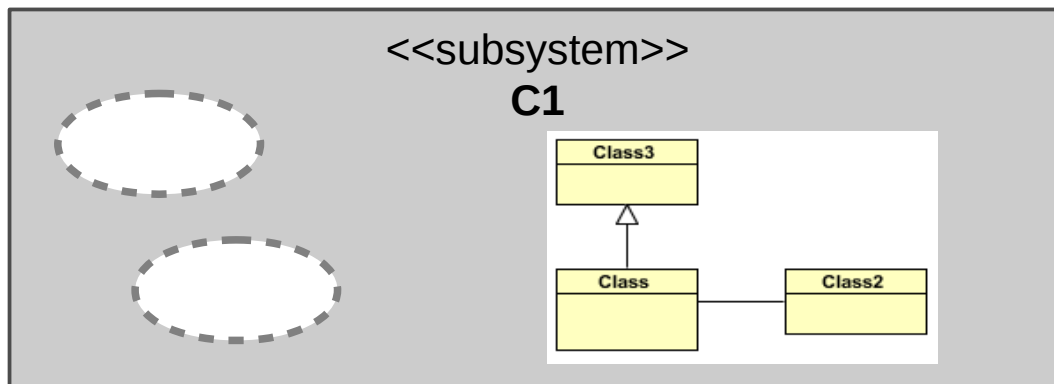


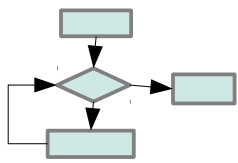
# Contenido del Modelo de diseño

A nivel general está formado por varios **subsistemas** de diseño junto con las **interfaces** que requieren o proporcionan estos subsistemas.



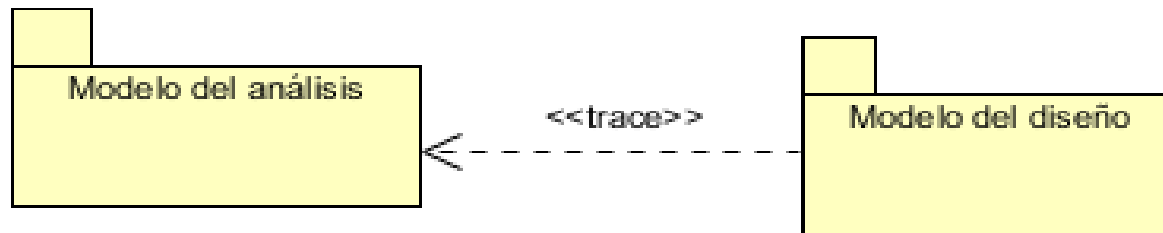
Cada subsistema de diseño a su vez pueden contener diferentes tipos de elementos de modelado del diseño, principalmente **realización de casos de uso-diseño** y **clases de diseño**.





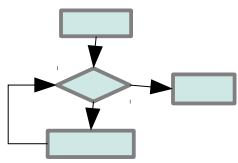
# Relación con el modelo del análisis

El modelo del diseño puede considerarse como una elaboración/refinamiento del modelo del análisis, en los que todos los artefacto de éste están mejor definidos e incorporan detalles técnicos que permiten su implementación.



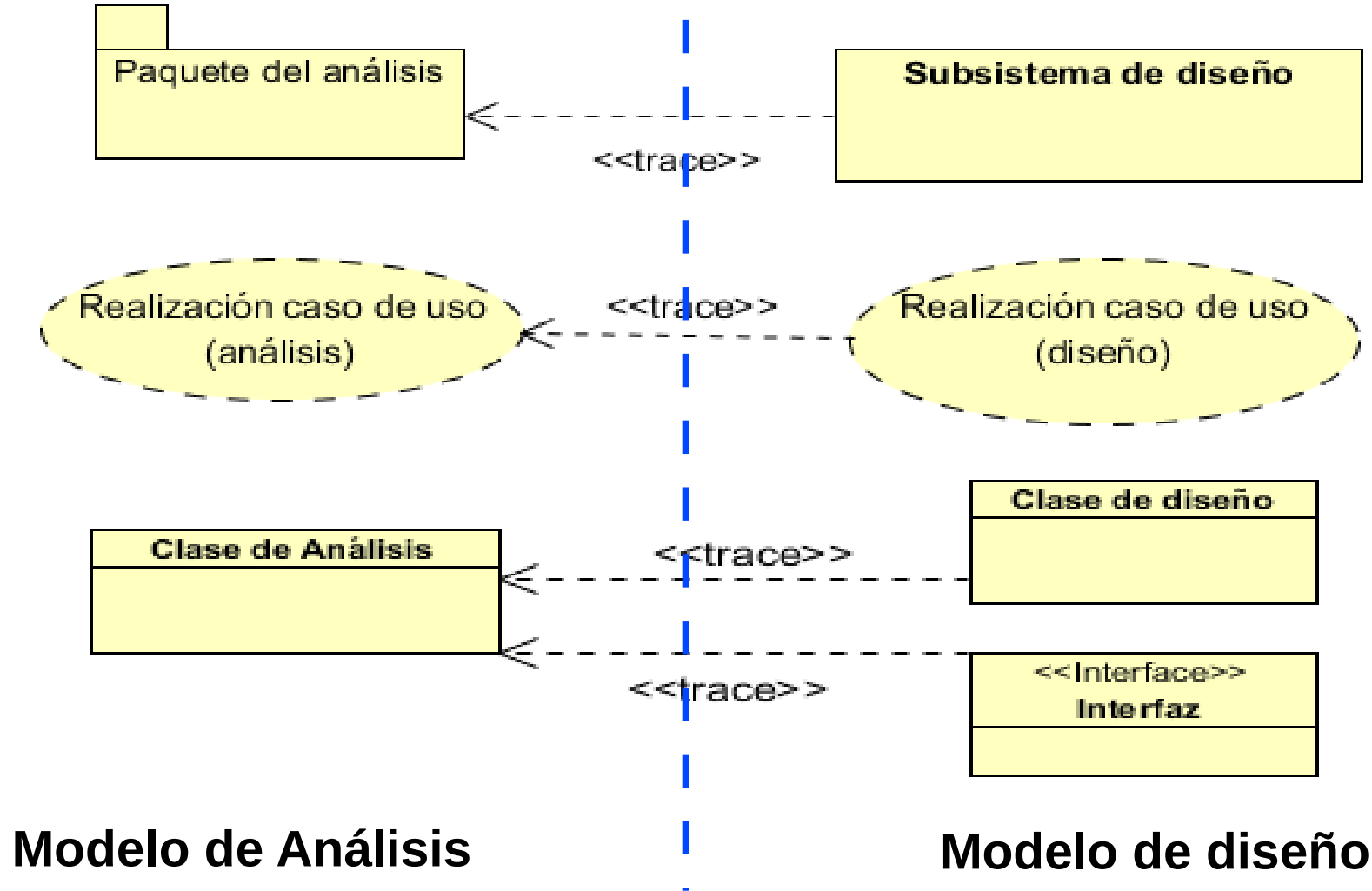
¿Qué hacer con estos dos modelos?

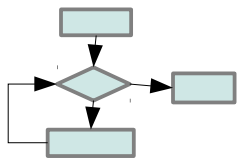
Estrategia	Consecuencias
A) Modificar el modelo del análisis	Se tiene un solo modelo pero se pierde la vista del análisis
B) Igual que A y usar una herramienta que nos recupere el modelo del análisis	Se tiene un solo modelo, pero la vista recuperada puede no ser satisfactoria
C) Congelar el modelo del análisis y hacer una copia para continuar con el diseño	Se tienen dos modelos que no van al mismo ritmo.
D) Mantener los dos modelo	Se tiene dos modelos al mismo ritmo, pero hay una sobrecarga de mantenimiento.



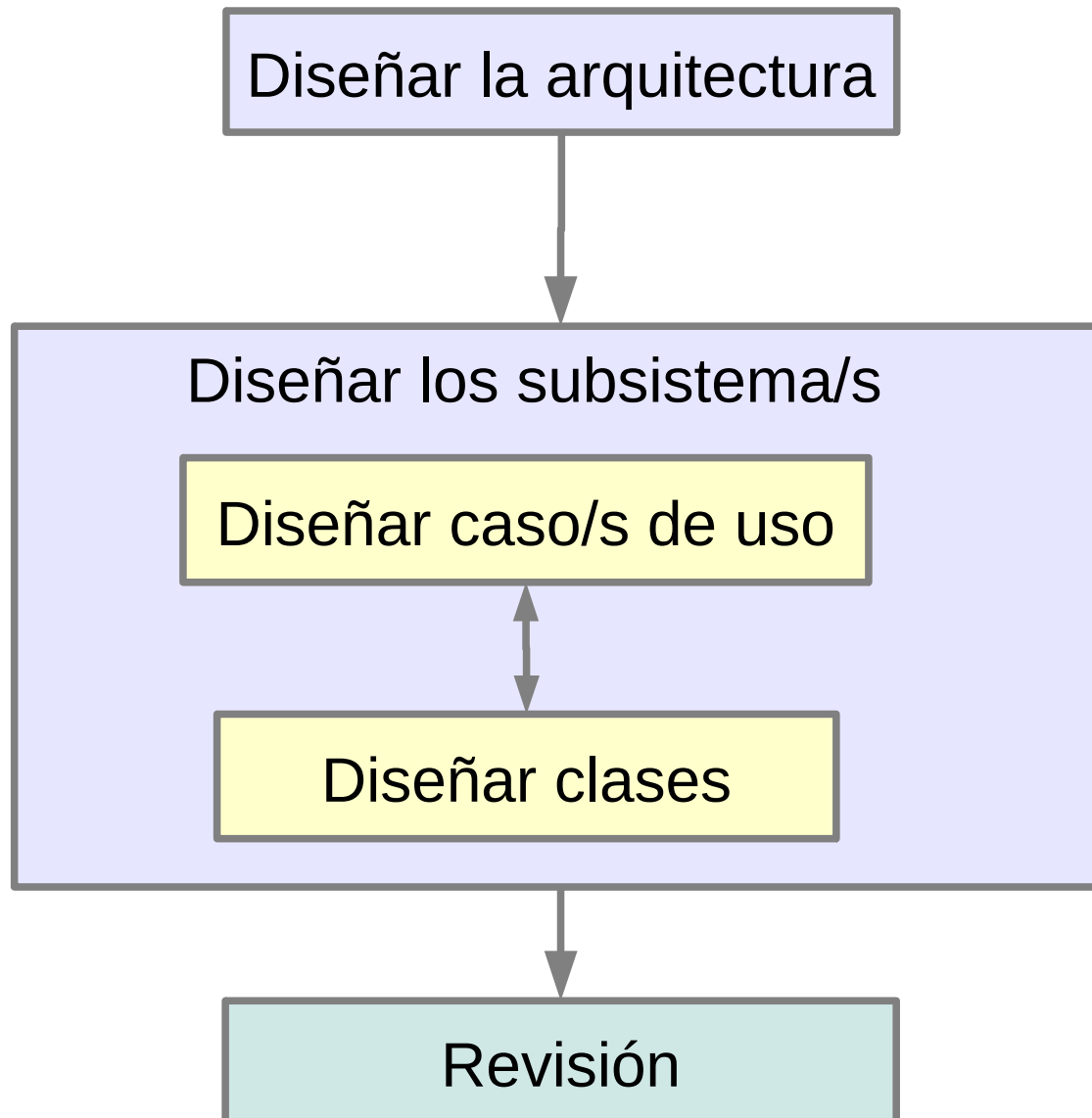
# Relación con el modelo del análisis

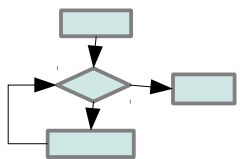
Trazabilidad entre los distintos artefacto del análisis y del diseño



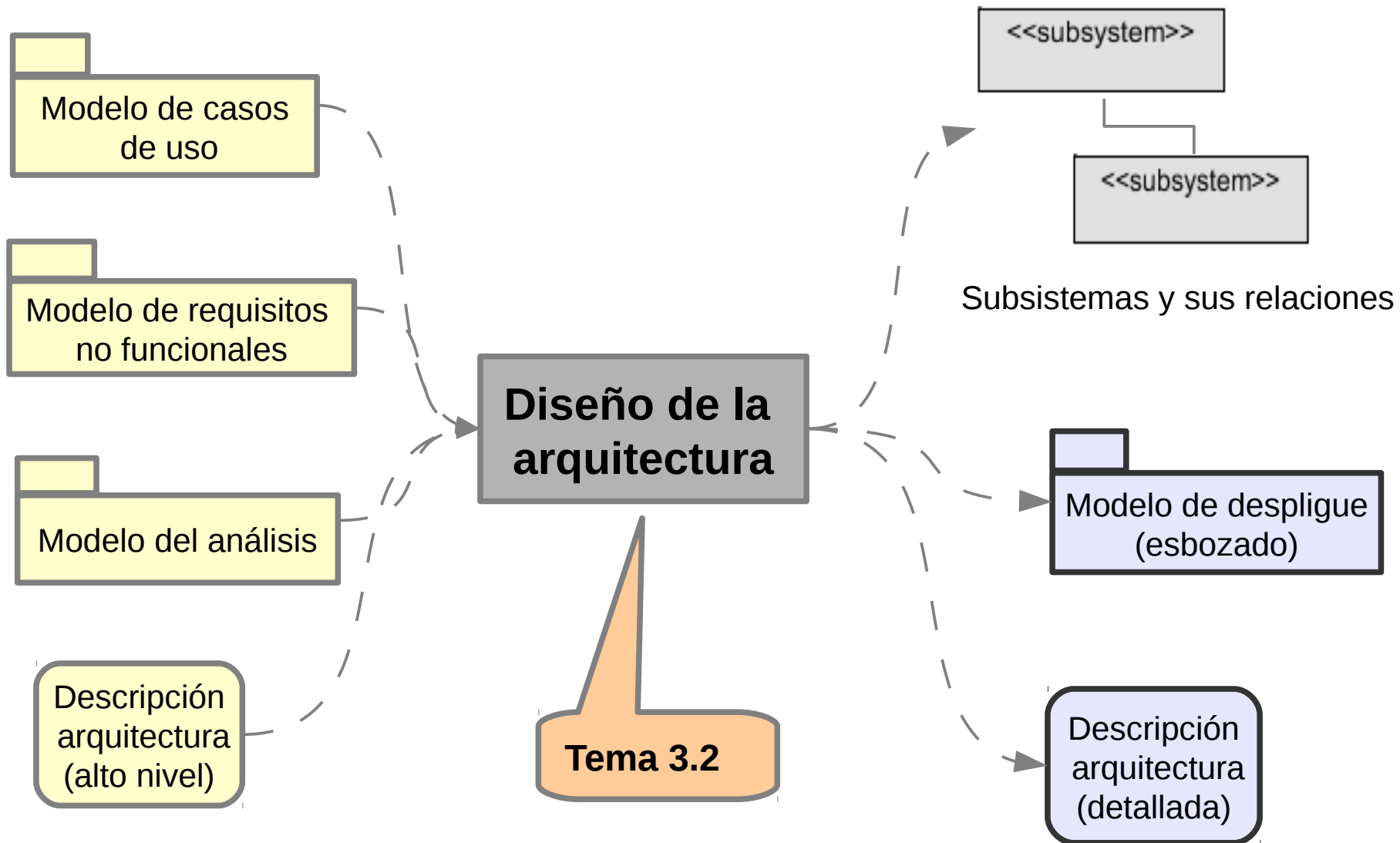


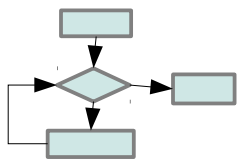
# Actividades y relación entre éstas





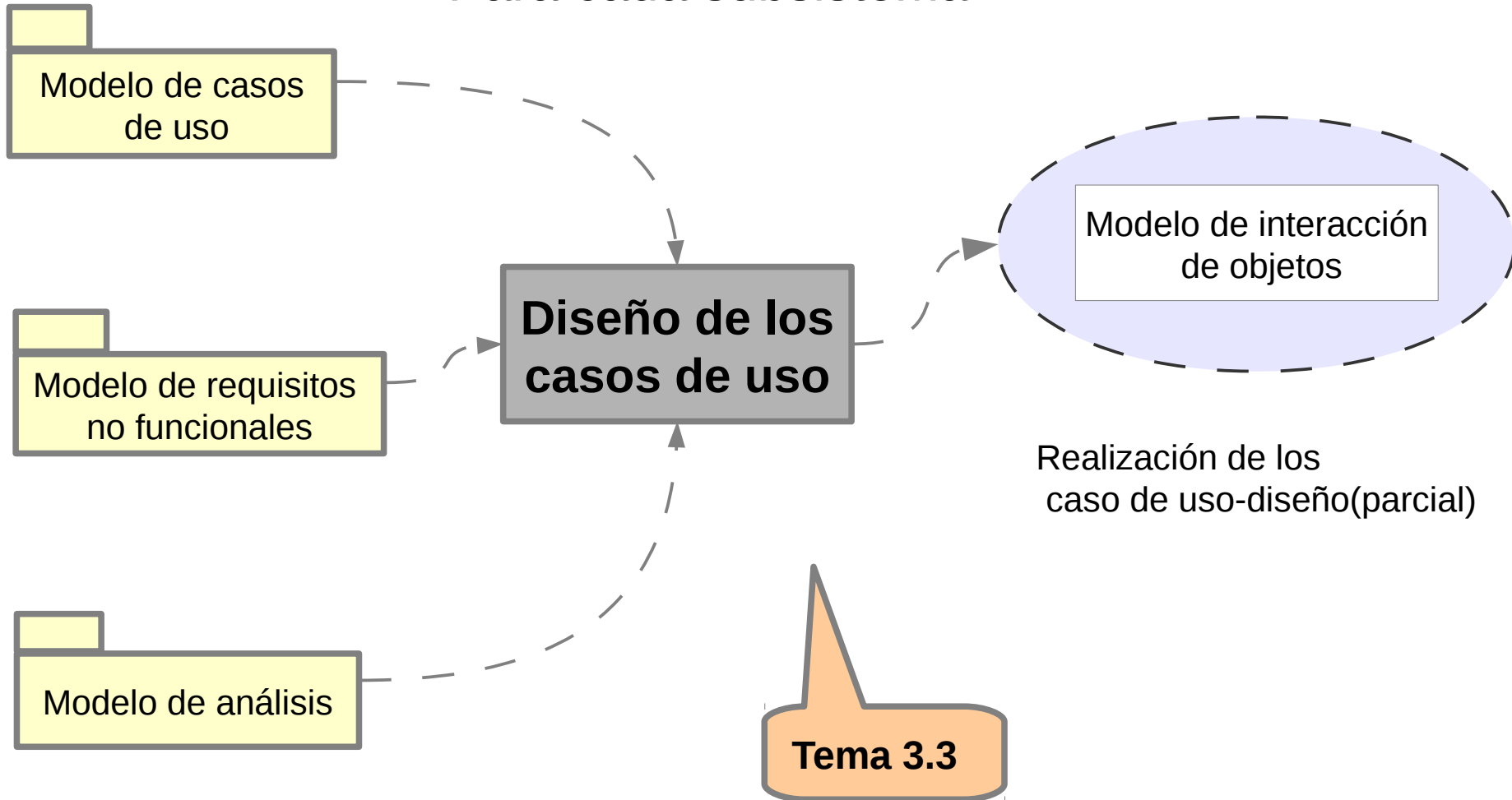
# Relación con el modelo de diseño

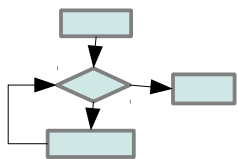




# Relación con el modelo de diseño

Para cada subsistema





# Relación con el modelo de diseño

