





# DISEÑO

Proceso Unificado





“modelamos el sistema y encontramos su forma para que soporte **todos** los requisitos incluyendo los no funcionales”

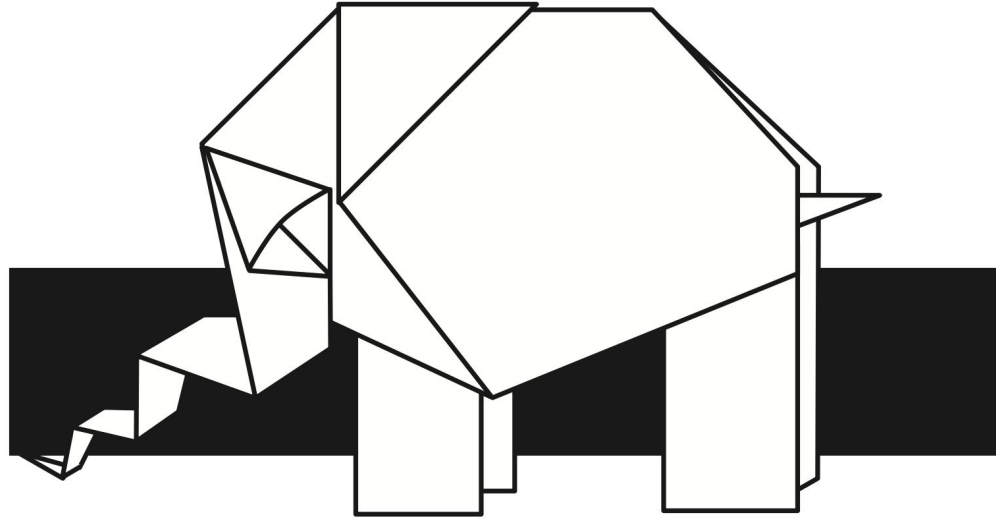


# Propósito

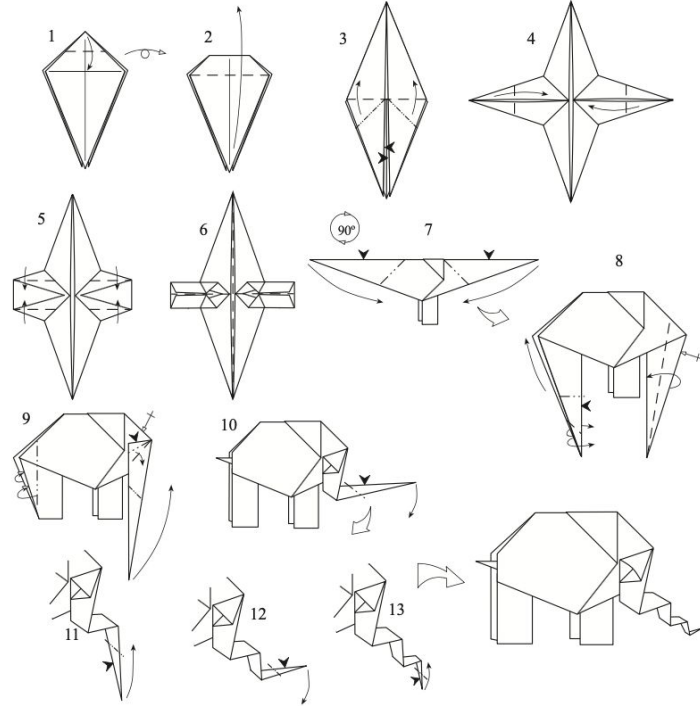
- Comprensión profunda de los requisitos no funcionales, restricciones de los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia, etc.
- Crear un punto de partida para la implementación.
- Descomponer los trabajos de implementación en partes más manejables.
- Capturar las interfaces entre subsistemas antes en el ciclo del software.
- Poder visualizar y reflexionar sobre el diseño.
- Crear una abstracción de la implementación.

# Ejemplo: Requerimiento

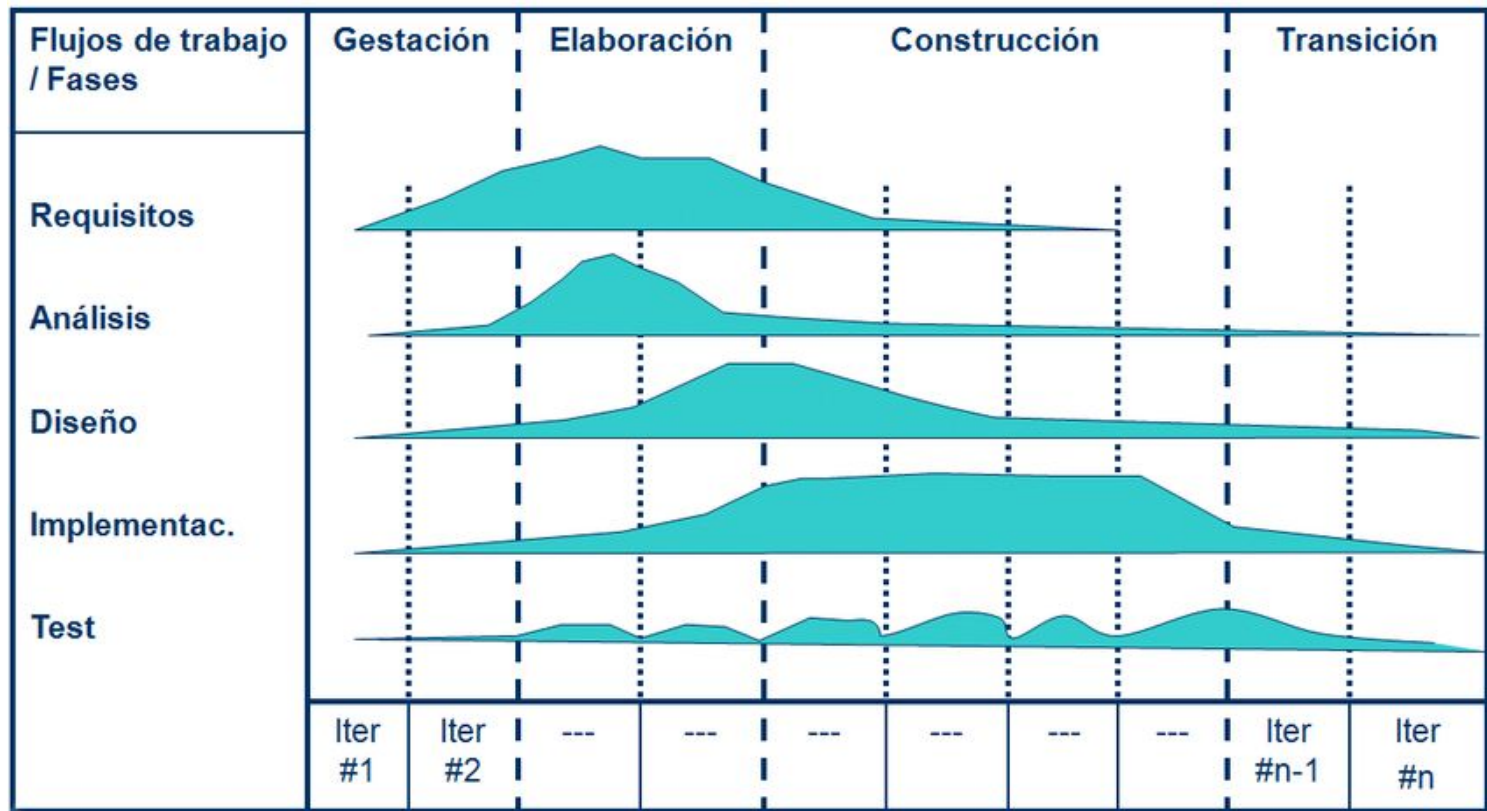
Crear un elefante de papel



# Ejemplo: Diseño

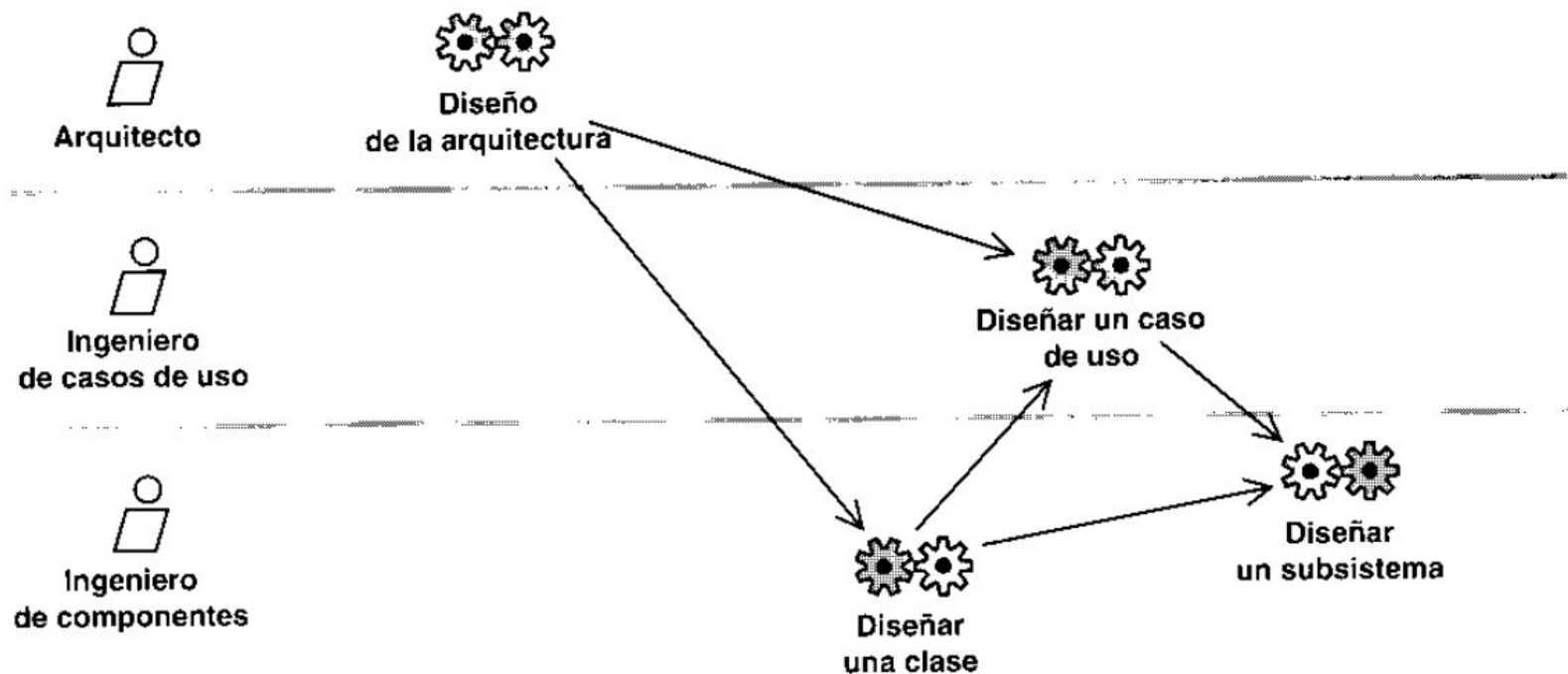


Modelo de análisis	Modelo de diseño
Modelo conceptual, porque es una abstracción del sistema y permite aspectos de la implementación.	Modelo físico, porque es un plano de la implementación.
Genérico respecto al diseño (aplicable a varios diseños).	No genérico, específico para una implementación.
Tres estereotipos conceptuales sobre las clases: Control, Entidad e Interfaz.	Cualquier número de estereotipos (físicos) sobre las clases, dependiendo del lenguaje de implementación.
Menos formal.	Más formal.
Menos caro de desarrollar (ratio al diseño 1:5).	Más caro de desarrollar (ratio al análisis 5:1).
Menos capas.	Más capas.
Dinámico (no muy centrado en la secuencia).	Dinámico (muy centrado en las secuencias).
Bosquejo del diseño del sistema, incluyendo su arquitectura.	Manifiesto del diseño del sistema, incluyendo su arquitectura (una de sus vistas).
Creado principalmente como “trabajo de a pie” en talleres o similares.	Creado principalmente como “programación visual” en ingeniería de ida y vuelta; el modelo de diseño es realizado según la ingeniería de ida y vuelta con el modelo de implementación (descrito en el Capítulo 10).
Puede no estar mantenido durante todo el ciclo de vida del software.	Debe ser mantenido durante todo el ciclo de vida del software.
Define una estructura que es una entrada esencial para modelar el sistema —incluyendo la creación del modelo de diseño.	Da forma al sistema mientras que intenta preservar la estructura definida por el modelo de análisis lo más posible.



Diseño en el ciclo de vida

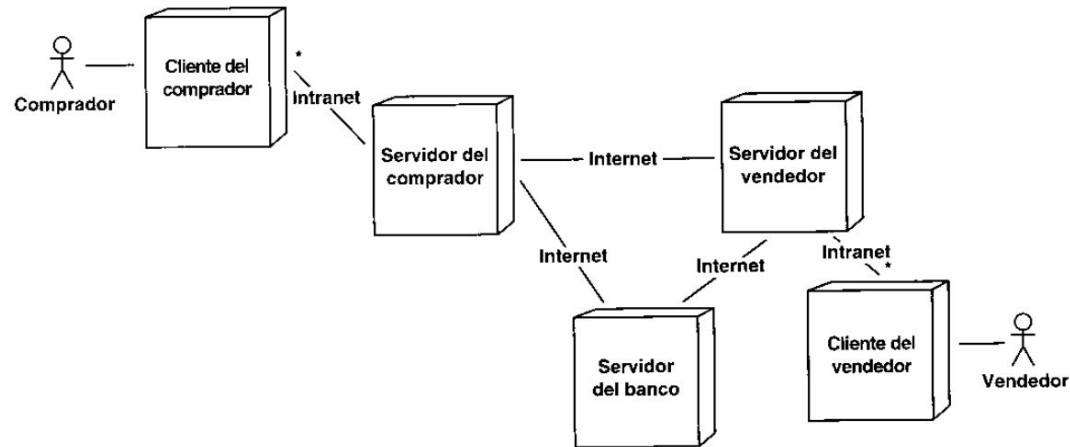
# Flujo de Trabajo





# Diseño de la Arquitectura

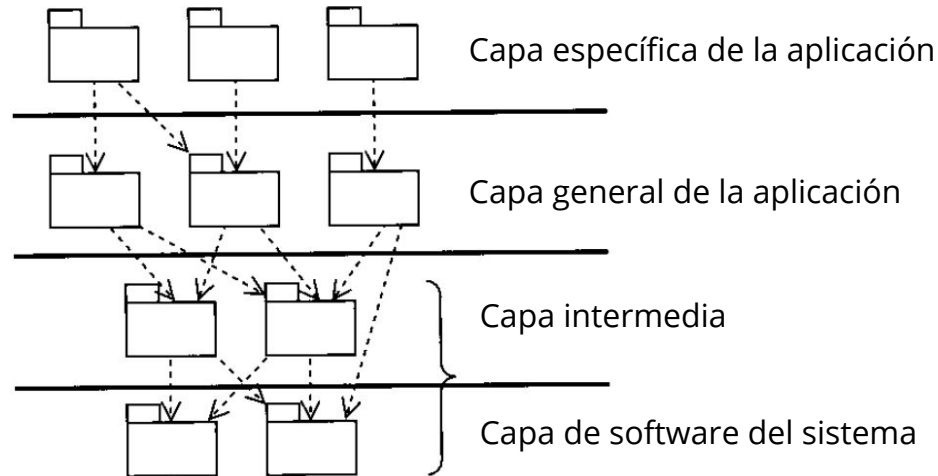
1- Identificar los nodos y la configuración de la red (Diag. de Despliegue)

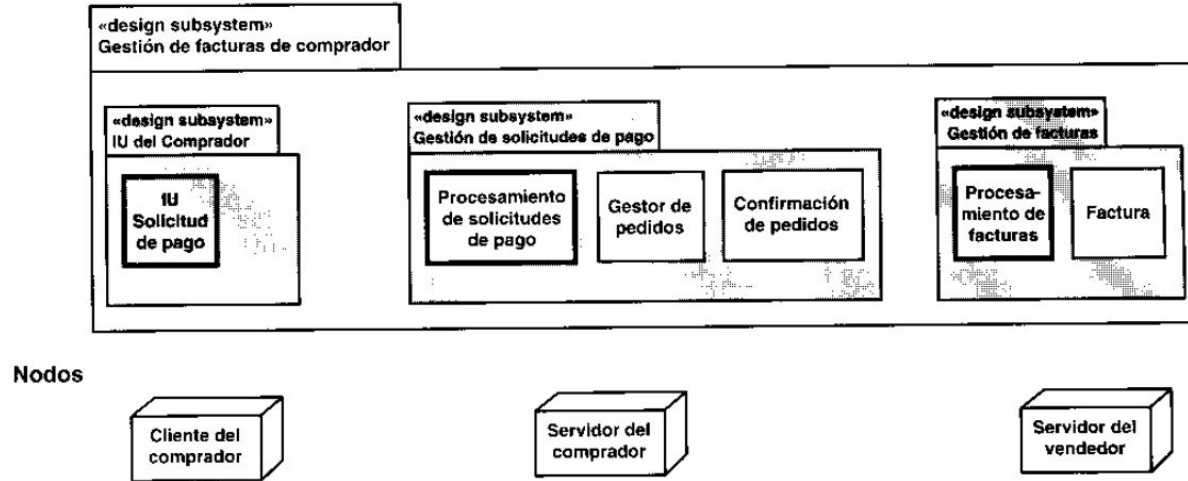


**Figura 9.18.** Diagrama de despliegue para el sistema Interbank.

# Diseño de la Arquitectura

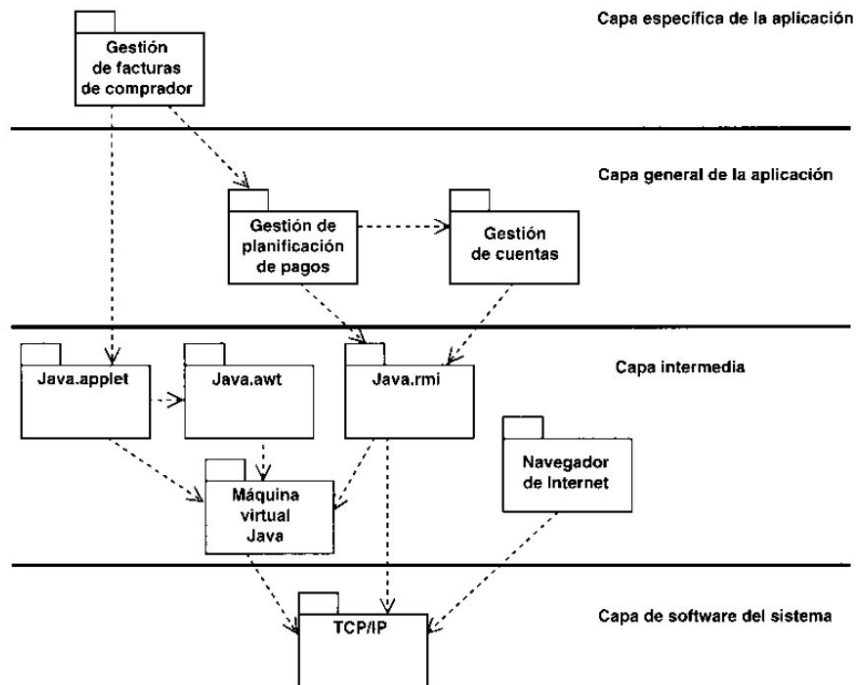
## 2- Identificar subsistemas y sus interfaces





**Figura 9.23.** Un subsistema descompuesto recursivamente en tres nuevos subsistemas para tratar su distribución.

Dividimos los subsistemas para distribuirlos en nodos



**Figura 9.26.** Dependencias y capas de algunos de los subsistemas del sistema Interbank.

Ejemplo

# Diseño de la Arquitectura

3 - Identificación de clases de diseño relevante para la arquitectura.

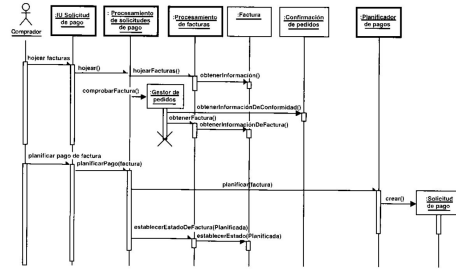
- A. Clases que surgen a partir de las clases de análisis
- B. Clases activas

4 - Identificar los mecanismos genéricos de diseño:

- A. Persistencia
- B. Distribución transparente
- C. Características de Seguridad
- D. Recuperación de Errores
- E. Transacciones
- F. Etc.

# Diseño de un CASO DE USO

1. Identificar las clases de diseño participantes.
2. Descripción de las interacciones entre objetos de diseño.



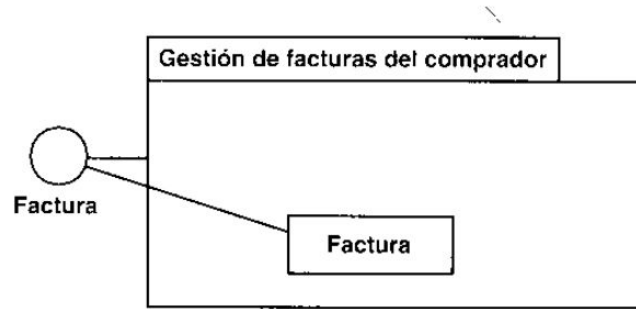
3. Identificar los subsistemas e interfaces participantes.
4. Descripción de interacciones entre subsistemas.
5. Captura de requisitos de implementación

# Diseño de una CLASE

1. Esbozar la clase de diseño
2. Identificar operadores, atributos, asociaciones, agregaciones
3. Descubrir métodos
4. Descubrir estados
5. Tratar requisitos especiales (ej: la factura debe ser accedida de distintos puntos)

# Diseño de un Subsistema

1. Mantenimiento de las dependencias entre subsistemas
2. Mantenimiento de las interfaces proporcionadas por subsistemas
3. Mantenimiento de contenidos de los subsistemas.



**Figura 9.45.** La clase Factura proporciona la interfaz Factura que proporciona el subsistema Gestión de Facturas del Comprador.



# Seguridad

```
boolean tienePermisoPara(String acción, Usuario usuario)

class Seguridad(){

    public tienePermisoPara(String acción, Usuario usuario){

        return true;

    }

}
```