



# INGENIERÍA DEL SOFTWARE I

## Tema 14

### *Pruebas de Sistemas Orientados a Objetos*

*Univ. Cantabria – Fac. de Ciencias*

*Laura Sánchez*



### Objetivos del Tema

- Reforzar los aspectos de pruebas aprendidos en el tema 5, para el caso de sistemas orientados a objetos.
- Aprender las principales estrategias y métodos de pruebas para este tipo de sistemas.
- Aprender a diseñar casos de pruebas para OO.



## Contenido

---

- Recordatorio
- Introducción
- Estrategias
  - Pruebas de unidad
    - Valores interesantes
  - Pruebas de integración
  - Pruebas de validación
- Diseño de casos de prueba
- Métodos de pruebas



## Bibliografía

---

- Básica
  - Jacobson, I., Booch, G., and Rumbaugh, J. (2000): El Proceso Unificado de Desarrollo. Addison-Wesley.
    - Cap. 11.
  - Pressman, R. (2005): Ingeniería del Software: Un Enfoque Práctico. 6<sup>o</sup> Edición. McGraw-Hill.
    - Caps. 13 y 14.



## Recordamos...

En el tema 5 se decía que **Verificación y Validación (VV)** es un conjunto de procedimientos, actividades, técnicas y herramientas que se utilizan, paralelamente al desarrollo de software, para asegurar que un producto software resuelve el problema inicialmente planteado.

**Verificación** ¿estamos construyendo correctamente el producto?

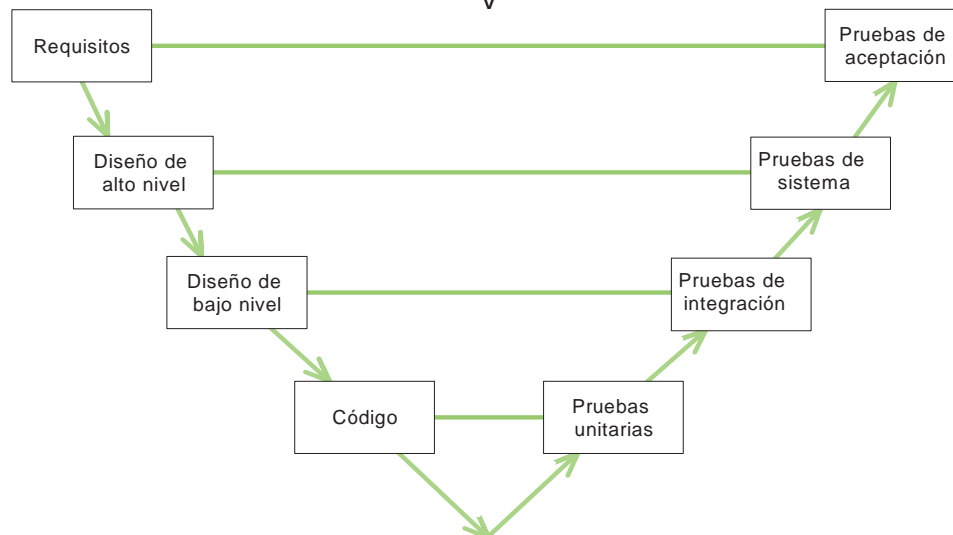
**Validación** ¿estamos construyendo el producto correcto?

- Las **pruebas** (testing) son una familia de técnicas de **VV**.
- **En el tema 5 se estudiaban las pruebas en el método convencional, en este tema se estudian para los sistemas orientados a objetos.**



## Recordamos...

Método tradicional: Modelo en V





## Introducción

---

- La OO introduce nuevos aspectos
  - Reutilización, abstracción, conexiones y relaciones entre clases, diseño del sistema, diseño de objetos... hasta su implementación.
  - La naturaleza de los programas OO cambian las estrategias y tácticas de prueba



## Introducción

---

- Pruebas OO
  - Características
    - Arquitectura software OO -> subsistemas organizados por **capas** que encapsulan clases que colaboran entre sí.
    - Necesario probar el sistema OO en diferentes niveles.
    - En cada etapa los modelos pueden probarse para descubrir errores y **evitar que se propaguen** a la siguiente iteración.



## Introducción

- Antes que el código, se generan los modelos de análisis y diseño del sistema.
- Los **modelos de análisis y diseño** son similares en estructura y contenido al programa OO, por tanto las **pruebas comienzan** con la revisión de estos modelos.

Paso inicial: verificación de modelos de análisis y diseño

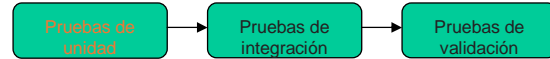


## Introducción

- Para los modelos de análisis y diseño hay que comprobar:
  - Exactitud:
    - ¿qué es? conformidad del modelo con el dominio del problema
    - ¿cómo se comprueba? Evaluación de los expertos en el tema
  - Compleción:
    - ¿qué es? El modelo incluye todas las partes necesarias para su correcta definición (nombre de los métodos, tipos de datos, parámetros..)
    - ¿cómo se comprueba? Evaluación de los expertos en el tema
  - Consistencia
    - ¿qué es? Las relaciones entre las entidades se respetan en todas las partes del modelo
    - ¿cómo se comprueba? Listas de comprobación



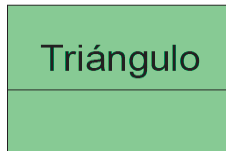
## Estrategias – Pruebas de Unidad



### • Pruebas de unidad

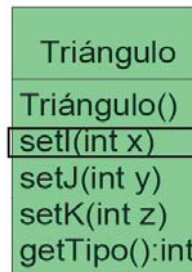
- ¿cuál es el concepto más parecido a módulo en OO? **La clase**
- Las pruebas de unidad en OO se realizan:

#### Nivel de clase



- 1) Probar cada uno de los métodos
- 2) Distintos niveles de cobertura

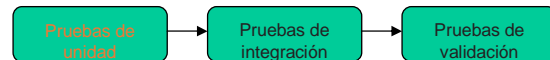
#### Nivel de método



- 1) Caja negra
- 2) Caja blanca



## Estrategias – Pruebas de Unidad

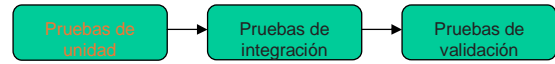


### • Pruebas de unidad

- ¿cómo se prueban las clases? **Con casos de prueba**
- ¿qué es un caso de prueba?
  - Construcción de una instancia de la clase que se va a probar
  - Ejecución de una serie de servicios sobre ésta
  - Comprobación del resultado (oráculo)

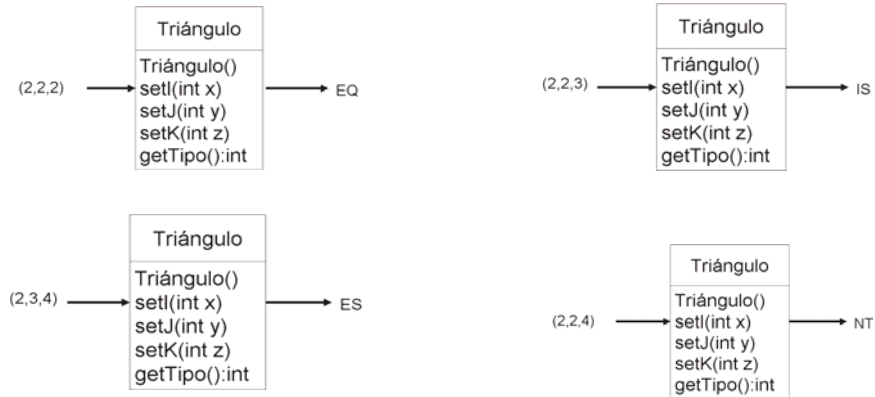


# Estrategias – Pruebas de Unidad

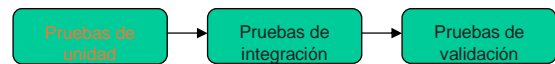


## • Pruebas de unidad (a nivel de método)

### ■ Pruebas de caja negra

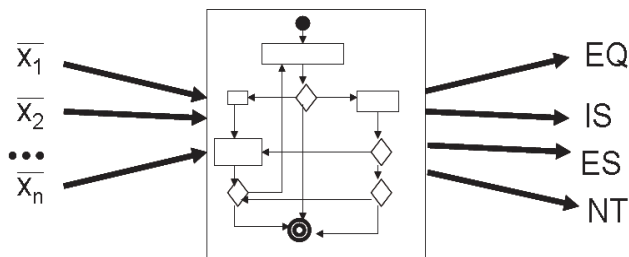


# Estrategias – Pruebas de Unidad



## • Pruebas de unidad (a nivel de método)

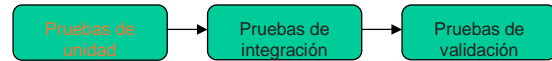
### ■ Pruebas de caja blanca



-Encontrar fragmentos del programa que no son ejecutados por los casos de prueba

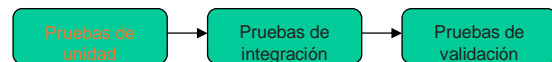
-Crear casos de prueba adicionales que incrementen la cobertura

-Determinar un valor cuantitativo de la cobertura

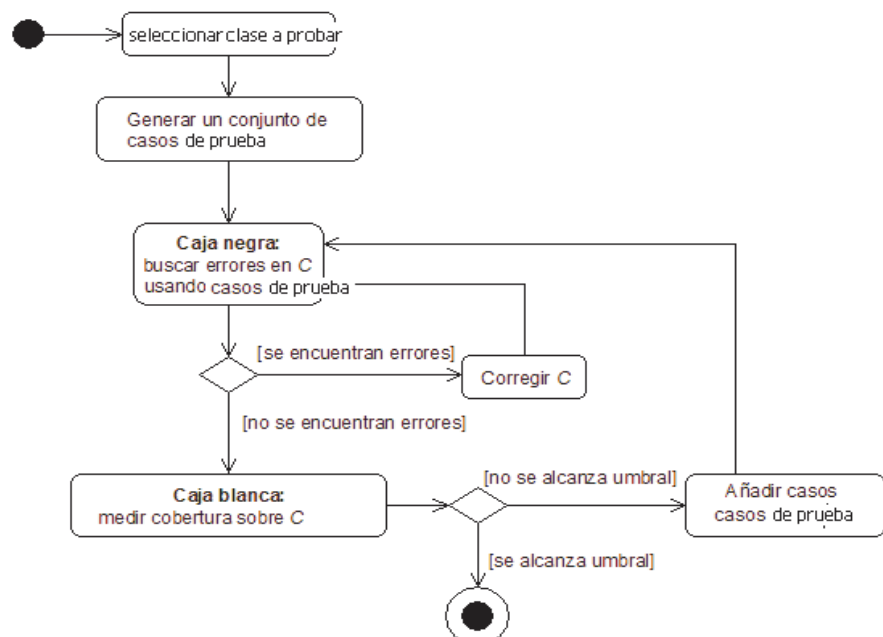


## • Método de caja blanca

- Criterios de cobertura (Mas detalle en el tema 5):
  - Cobertura de sentencias
  - Cobertura de decisiones
  - Cobertura de condiciones
  - Cobertura de decisión/condición
  - Cobertura de condición múltiple



-La idea es combinar los métodos de caja negra con los de caja blanca





- **Los valores interesantes:**

- Para diseñar correctamente los casos de prueba, éstos deberán usar **“valores interesantes”**.
- Se considera un valor interesante aquel que permita recorrer la **mayor cantidad** posible de código fuente en función del **criterio de cobertura** elegido



```

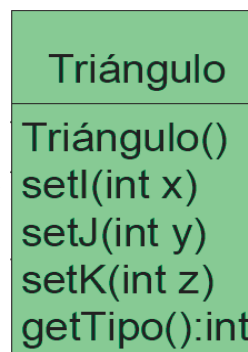
public int getTipo() {
    if (i==j) { tipo=tipo+1; }
    if (i==k) { tipo=tipo+2; }
    if (j==k) { tipo=tipo+3; }
    if (i<=0 || j<=0 || k<=0) {
        tipo=Triangulo.NO_TRIANGULO;
        return tipo;
    }
    if (tipo==0) {
        if (i+j<=k || j+k<=i || i+k<=j) {
            tipo=Triangulo.NO_TRIANGULO;
            return tipo;
        } else {
            tipo=Triangulo.ESCALENO;
            return tipo;
        }
    }
    if (tipo>3) {
        tipo=Triangulo.EQUILATERO;
        return tipo;
    } else if (tipo==1 && i+j>k) {
        tipo=Triangulo.ISOSCELES;
        return tipo;
    } else if (tipo==2 && i+k>j) {
        tipo=Triangulo.ISOSCELES;
        return tipo;
    } else if (tipo==3 && j+k>i) {
        tipo=Triangulo.ISOSCELES;
        return tipo;
    } else {
        tipo=Triangulo.NO_TRIANGULO;
        return tipo;
    }
}

```

- **Los valores interesantes.**

¿Qué conjuntos de tres valores recorren todas las sentencias del método getTipo()?

- (2,2,2)
- (0,2,4)
- (2,3,5)
- (2,3,4)
- (2,5,2)
- (2,2,5)
- (5,2,2)
- (2,2,4)



100% cobertura de sentencias



## Estrategias – Valores Interesantes

- **Técnicas de obtención de valores “interesantes”.**
  - Para un ejemplo real, el conjunto de valores a probar para conseguir cobertura total de sentencias puede ser muy grande.
  - Existen técnicas para proponer valores realmente interesantes
    - Clases de equivalencias
    - Valores límite



## Estrategias – Valores Interesantes

- **Técnicas de obtención de valores “interesantes”.**
  - Clases de equivalencia:
    - A una clase de equivalencia pertenecen todos los valores que provocan el mismo comportamiento de la clase bajo prueba
    - Esta técnica divide el dominio de los valores de entrada en un número finito de clases en clases de equivalencia
  - Valores límite:
    - Complementa a la anterior
    - En lugar de elegir cualquier valor de la clase de equivalencia, se seleccionan los valores situados en los límites de las clases de equivalencia



## Estrategias – Valores Interesantes

- **Técnicas de obtención de valores “interesantes”.**

Ejemplo:

- Clase equivalencia 1: (2,2,2)  
(3,3,3) (4,4,4)...
- Clase equivalencia 2: (2,2,5)  
(1,3,5) (2,3,6)...
- Clase equivalencia 3: (2,5,2)  
(1,5,3) (2,6,3)...

```
public int getTipo() {
    if (i==j) { tipo=tipo+1; }
    if (i==k) { tipo=tipo+2; }
    if (j==k) { tipo=tipo+3; }
    if (i<=0 || j<=0 || k<=0) {
        tipo=Triangulo.NO_TRIANGULO;
        return tipo;
    }
    if (tipo==0) {
        if (i+j<=k || j+k<=i || i+k<=j) {
            tipo=Triangulo.NO_TRIANGULO;
            return tipo;
        } else {
            tipo=Triangulo.ESCALENO;
            return tipo;
        }
    }
    if (tipo>3) {
        tipo=Triangulo.EQUILATERO;
        return tipo;
    } else if (tipo==1 && i+j>k) {
        tipo=Triangulo.ISOSCELES;
        return tipo;
    } else if (tipo==2 && i+k>j) {
        tipo=Triangulo.ISOSCELES;
        return tipo;
    } else if (tipo==3 && j+k>i) {
        tipo=Triangulo.ISOSCELES;
        return tipo;
    } else {
        tipo=Triangulo.NO_TRIANGULO;
        return tipo;
    }
}
```



## Estrategias – Valores Interesantes

- **Criterios de cobertura para valores:**
  - Grado en que los diferentes valores interesantes seleccionados se utilizan en la batería de casos de prueba:
    - 1-wise: se satisface cuando cada valor interesante de cada parámetro se incluye al menos en un caso de prueba
    - 2-wise: se satisface cuando cada par de valores interesantes se incluye al menos en un caso de prueba.



## Estrategias – Valores Interesantes

### • Criterios de cobertura para valores:

- **1-wise:** se satisface cuando cada valor interesante de cada parámetro se incluye al menos en un caso de prueba.

Ejemplo: valores interesantes para cada lado del triángulo  $\{0,1,2,3,4,5\}$ . Habría que probar con cada uno de esos valores al menos una vez en cada posición:

- $(0,1,2)$  en los próximos casos no habría que probar con el 0 en la 1<sup>o</sup> posición, ni con el 1 en 2<sup>o</sup> posición, ni con 2 en 3<sup>o</sup> posición
- $(1,0,2) - (2,3,4) - (2,4,3) - (3,5,0) - (4,2,1)$
- $(3,0,5)$  se repite el 0 en 2<sup>o</sup> posición. Es necesario utilizarlo para probar con el 5 en 3<sup>o</sup> posición
- $(5,0,1)$ ...etc



## Estrategias – Valores Interesantes

### • Criterios de cobertura para valores:

- **2-wise:** se satisface cuando cada par de valores interesantes se incluye al menos en un caso de prueba.

•A partir de esta tabla, habrá que ir construyendo casos de prueba que vayan visitando cada par de valores el menor número posible de veces

i/j	i/k	j/k
(0, 0)	(0, 0)	(0, 0)
(0, 1)	(0, 1)	(0, 1)
(0, 2)	(0, 2)	(0, 2)
(0, 3)	(0, 3)	(0, 3)
(1, 0)	(1, 0)	(1, 0)
(1, 1)	(1, 1)	(1, 1)
(1, 2)	(1, 2)	(1, 2)
(1, 3)	(1, 3)	(1, 3)
(2, 0)	(2, 0)	(2, 0)
(2, 1)	(2, 1)	(2, 1)
(2, 2)	(2, 2)	(2, 2)
(2, 3)	(2, 3)	(2, 3)
(3, 0)	(3, 0)	(3, 0)
(3, 1)	(3, 1)	(3, 1)
(3, 2)	(3, 2)	(3, 2)
(3, 3)	(3, 3)	(3, 3)



## Estrategias – Pruebas de Integración



- **Pruebas de integración**

¿Mismas estrategias que en estructurado? **No**, porque OO no tiene una estructura de control jerárquico.

### Nuevos paradigmas en OO:

- **Pruebas basadas en hilos:**
  - Integra las clases requeridas para responder a una entrada o suceso al sistema (diagramas de secuencia de UML)
  - Cada hilo se integra y prueba individualmente
- **Pruebas basadas en el uso:**
  - Se comienza probando las clases más independientes
  - Se continúa con las más dependientes hasta probar todo el sistema
- **Pruebas de agrupamiento**
  - Se selecciona un agrupamiento de clases colaboradoras
  - Se prueba diseñando las clases de pruebas que revelan errores en las colaboraciones



## Estrategias – Pruebas de Integración



- **Pruebas de integración**

### Ejemplo “Pruebas basadas en hilos”:

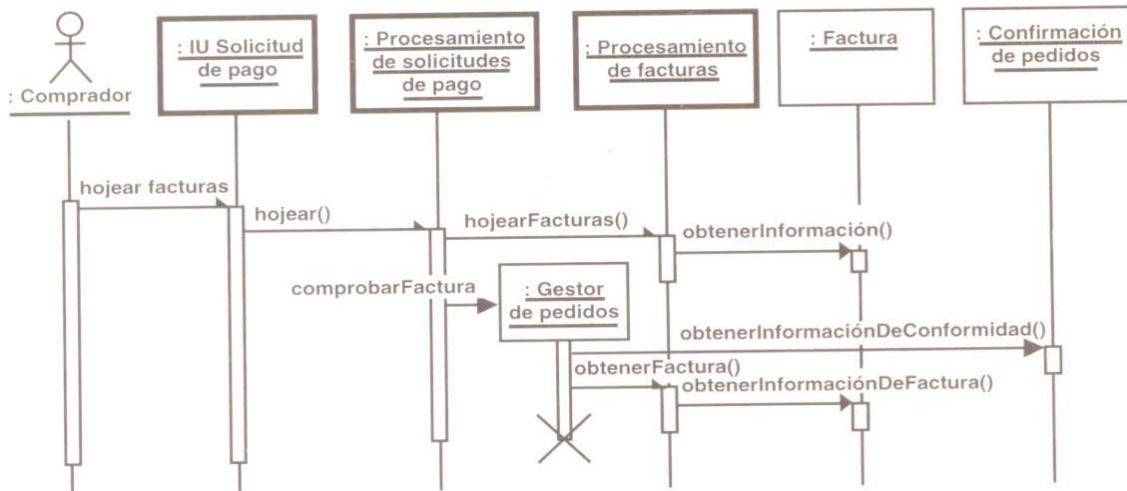
- Al haber varias secuencias diferentes en el diagrama de secuencias dependiendo, por ejemplo, del estado inicial del sistema y de la entrada del actor.
- El caso de pruebas que se deriva de un diagrama de secuencia debería describir cómo probar una secuencia interesante en el diagrama, tomando el estado inicial del sistema



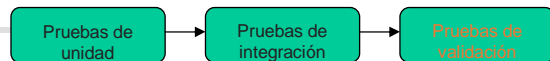
## Estrategias – Pruebas de Integración



- **Pruebas de integración**



## Estrategias – Pruebas de Validación



- **Pruebas de validación**

- Se centra en acciones visibles al usuario y salidas reconocibles desde el sistema
- Los casos de prueba se obtienen a partir de los casos de uso (son parte del modelo de análisis) ya que tienen una gran similitud de errores con los revelados en los requisitos de interacción del usuario
- ¿Qué métodos de prueba se utilizan?
  - **Caja negra**



## Estrategias – Pruebas de Validación

- **Ejemplo prueba de Validación:**
  - Caso de Prueba Pagar 300 – Bicicleta Montaña:

### Entrada:

- Existe pedido válido de Bicicleta de Montaña y se ha enviado a un vendedor
- El precio es 300 €incluyendo gastos envío
- El comprador ha recibido una confirmación de pedido (ID 12345)
- El comprador ha recibido una factura de 300 €y que debería estar en el estado pendiente. La factura debería apuntar a una cuenta (22-222-2222) la cual debería recibir el dinero. Esta cuenta tiene un saldo de 963.000 €y su titular debería ser el vendedor
- La cuenta 11-111-1111 del comprador tiene un saldo de 350 €

### Resultado:

- El estado de la factura debería ser puesto a cerrada, indicando así que ha sido pagada.
- Saldo de la cuenta 11-111-1111 = 50 €
- Saldo de la cuenta 22-222-2222= 963.350 €

### Condiciones:

- No se permite que otras instancias casos de uso accedan a las cuentas durante el caso de prueba



## Diseño de casos de prueba

- Identificar cada caso de prueba y asociarlo explícitamente con la clase a probar
- Declarar el propósito de la prueba
- Desarrollar una lista de pasos a seguir
  - Declarar una lista de estados del objeto a probar
  - Lista de mensajes y operaciones que se ejecutarán
  - Lista de excepciones que pueden ocurrir
  - Lista de condiciones externas necesarias para conducir las pruebas
  - Información adicional para facilitar la comprensión e implementación



## Diseño de casos de prueba

---

### **Prueba convencional**

Entrada – Proceso - Salida

### **Prueba OO**

Secuencia de operaciones para probar los estados de la clase



## Métodos de prueba

---

- **A nivel de clases**
  - Verificación al azar
  - Pruebas de partición
    - Partición basada en estados
    - Partición basada en atributos
    - Partición basada en categorías
- **A nivel de interclases**
  - Verificación al azar
  - Pruebas de partición
  - Basadas en el escenario
  - Pruebas de comportamiento



## Métodos de pruebas

---

- A nivel de clase
  - **Verificación al azar:** consiste en probar en orden aleatorio diferentes secuencias válidas de operaciones.
    - Ejemplo: para un sistema bancario, un orden aleatorio de operaciones sería abrir-configurar-depositar-retirar-cerrar



## Métodos de pruebas

---

- **A nivel de clase**
  - **Pruebas de partición:**
    - **Partición basada en estados:** clasifica las operaciones de clase en base a su habilidad de cambiar el estado de la clase.
      - Ejemplo: para la clase cuenta de un sistema bancario, las operaciones depositar o retirar cambian el estado, las de consulta de saldo no.
    - **Partición basada en atributos:** clasifica las operaciones basada en los atributos que usa.
      - Ejemplo: operaciones que hacen uso del atributo "LimiteCredito" y las que no
    - **Partición basada en categorías:** clasifica las operaciones de acuerdo a la función genérica que llevan a cabo
      - Ejemplo: operaciones de inicialización (abrir, configurar), operaciones de consulta (saldo, resumen)...



## Métodos de pruebas

---

- **A nivel de interclase:**
  - Es necesario verificar las colaboraciones entre las clases
  - Las técnicas de partición y al azar son iguales a las de a nivel de clase
  - ¿cómo se llevan a cabo las pruebas interclase?
    - Para cada clase cliente, generar secuencias de operaciones al azar. Así se enviarán mensajes a otras clases servidoras
    - Para cada mensaje determinar la clase colaboradora y la operación correspondiente en el servidor
    - Para cada operación invocada por el servidor determinar los mensajes que transmite
    - Para cada mensaje determinar el siguiente nivel de operaciones invocadas e incorporarlas a la secuencia de pruebas



## Métodos de pruebas

---

- **A nivel de interclase:**
  - Pruebas derivadas de los modelos de comportamiento.
    - Los diagramas de transición de estados pueden derivar una secuencia de pruebas
    - Se persigue alcanzar una cobertura total de estados.
    - La secuencia de operaciones debe causar que la clase haga transiciones por todos los estados
      - Ejemplo: abrir-preparar cuenta-depositar-retiro-cerrar



## En resumen...

---

- Las pruebas del diseño de sistemas estructurado son distintas a los sistemas orientados a objetos debido a las características particulares de cada uno.
- No todas las pruebas se realizan sobre artefactos software, sino también sobre los modelos.
- En los sistemas software, las pruebas son unitarias, de integración y de validación.
- Para los sistemas reales, es importante minimizar el número de casos de prueba que se realizan.