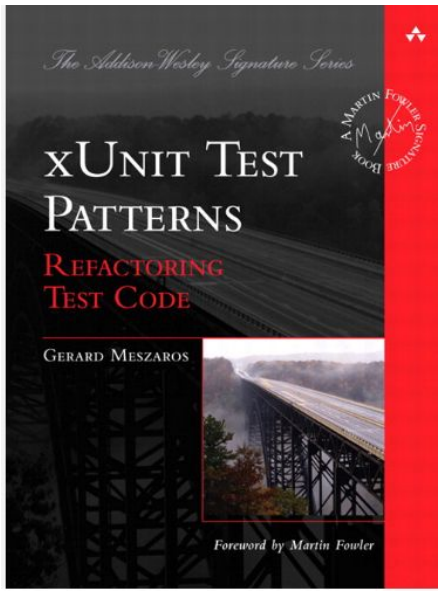


Test Doubles

(Pattern Language)

Federico Balaguer

Programación Orientada a Objetos 2

2025

“If debugging is the process of removing software bugs, then programming must be the process of putting them in.”
— **Edsger W. Dijkstra**

Test de unidad

Testeo de la *mínima unidad de ejecución/funcionalidad*.

En OOP, la mínima unidad es un método.

Objetivo: aislar cada parte de un programa y mostrar que funciona correctamente.

Cada test confirma que un método produce el output esperado ante un input conocido.

Es como un contrato escrito de lo que esa unidad tiene que satisfacer.

Paradoja del Testing

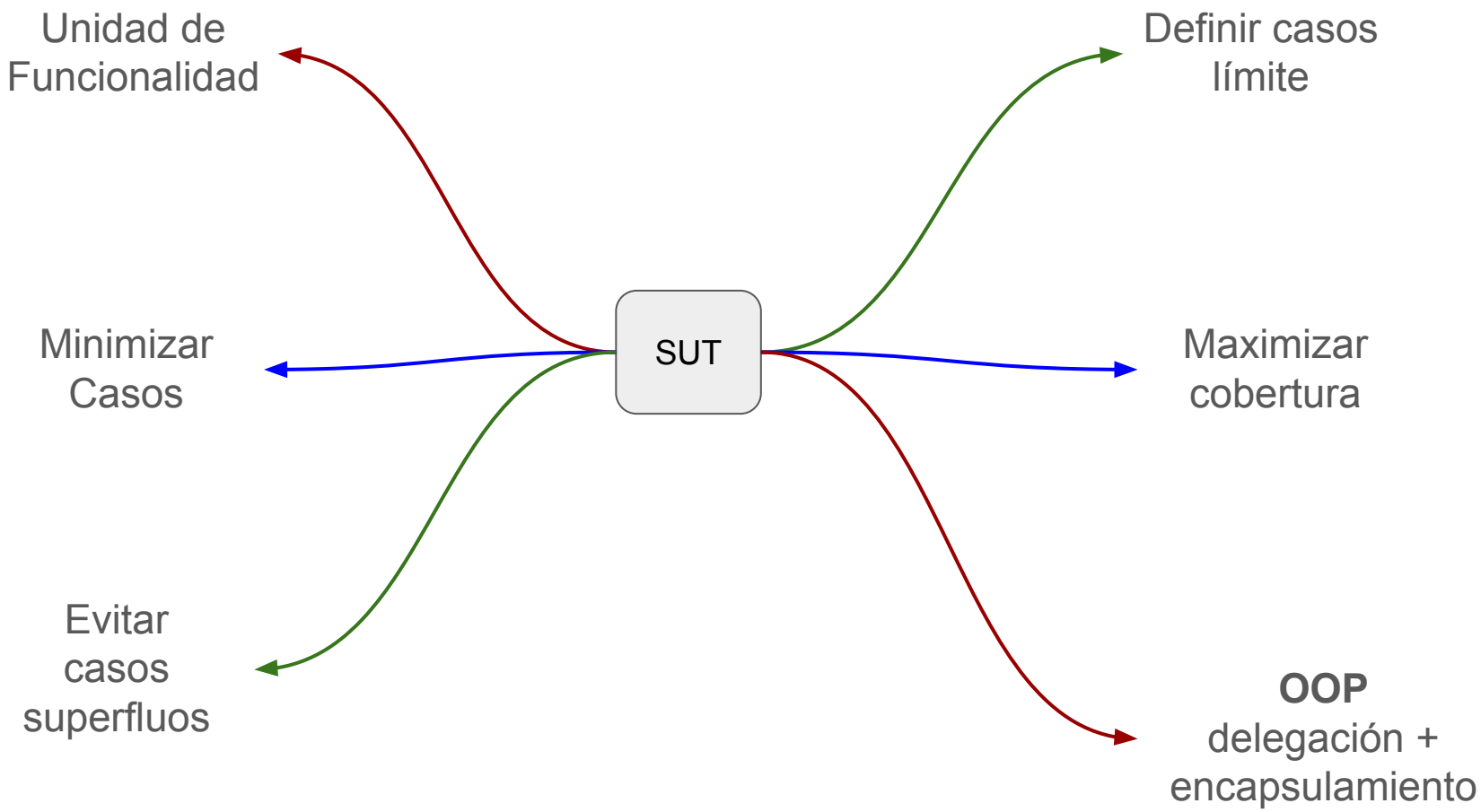
- Escribir casos de testing es deseable
- Escribir casos de testing costoso y aburrido
- Testear todos los métodos no es práctico

```
public String getName(){
  return Name;
}
```

```
public String setName(String s){
  Name=s;
}
```

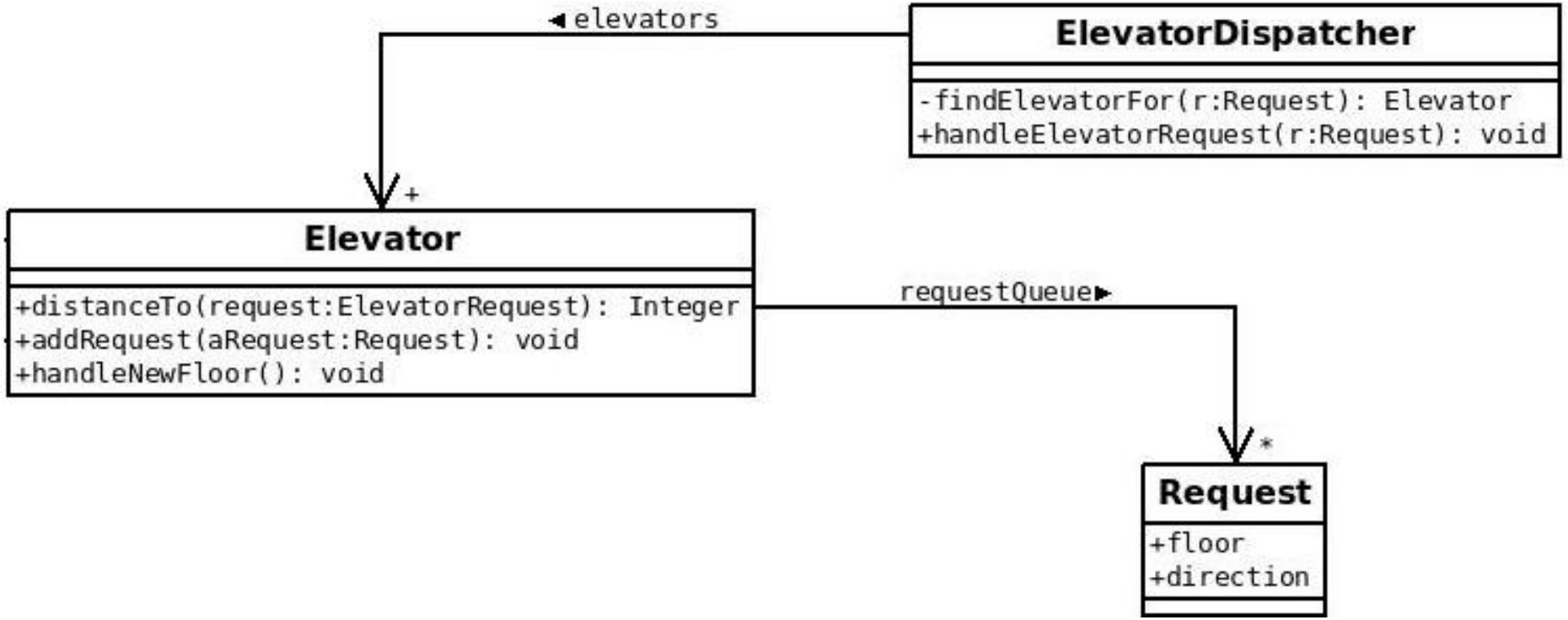
Objetivo: min. los casos y max. 'cobertura'

Fuerzas del problema



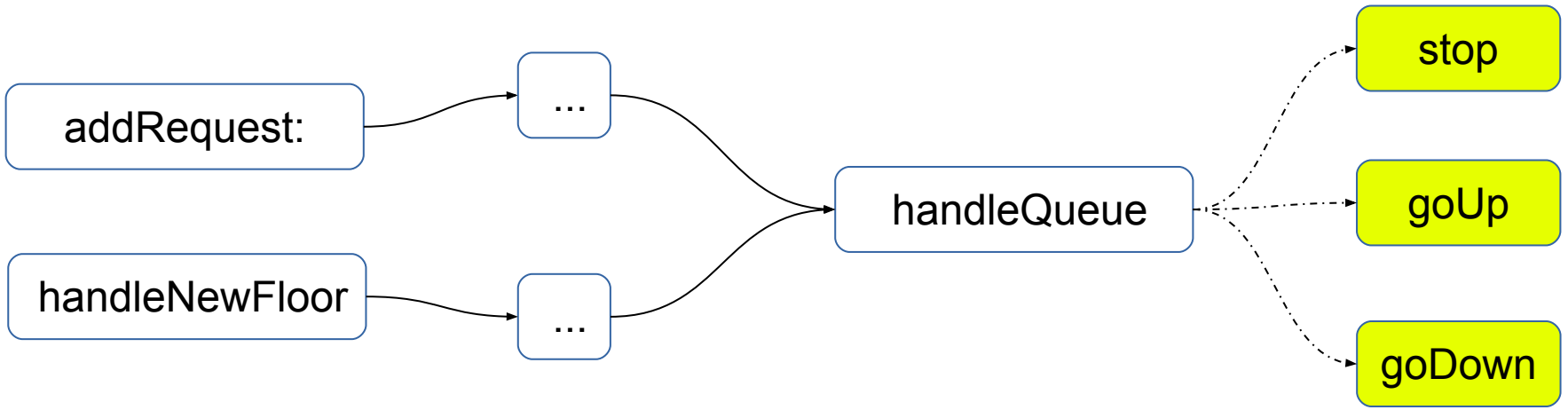
SUT: System Under Testing

Ascensores



Cómo hace el ascensor para moverse?
Cómo hace para el ascensor saber en qué piso está?

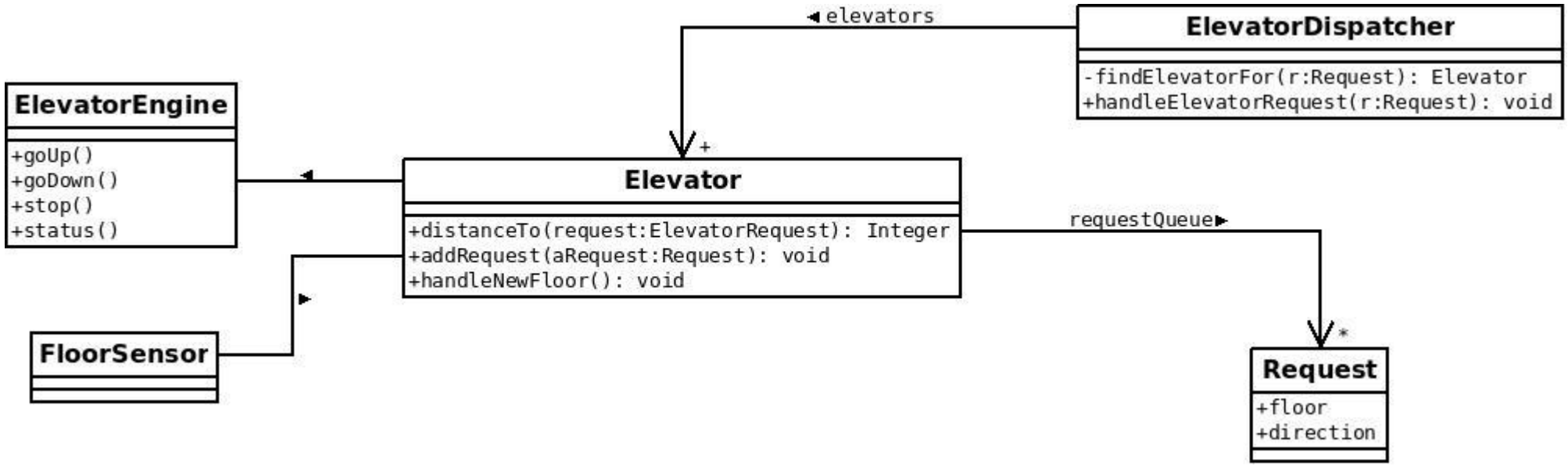
Ascensor




```
public void handleQueue(){
    if (requestQueue.first()==currentFloor) {
        this.stop();
        this.firstRequestDone();
    }
    else{ if (requestQueue.first()< currentFloor)
        this.goDown();
      else {this.goUp();}

    }
}
```

Ascensores



El SUT depende de ElevatorEngine y FloorSensor …

- Sabemos si funcionan Ok?
- que hacemos si no estan disponibles?

Test Double. Xunit Test Patterns -G. Meszaros-

Test Double

- Problema general
  - Es necesario realizar pruebas de un “SUT” que depende de un módulo u objeto
  - El módulo u objeto requerido no se puede utilizar en el ambiente de la pruebas
  - Las pruebas pueden ejercitar
    - Configuraciones válidas del sistema
    - “Salidas indirectas” del sistema
    - Lógica del sistema
    - Protocolos
- Test Double es un lenguaje de patrones

Lenguajes de Patrones

- Test Double
  - Crear un objeto que es una maqueta (polimórfica) del objeto o módulo requerido
  - Utilizar la maqueta según se necesite
- Rangos de implementación
  - **Cascarón vacío → Simulación**
  - Se generan diferentes patrones que se aplican a cada caso

Test Double

- Patrones (**Cascarón vacío → Simulación**)
  - **Test Stub**: cascarón vacío. Sirve para que el SUT envíe los mensajes esperados
  - **Test Spy**: Test Stub + registro de mensajes recibidos
  - **Mock Object**: test Stub + verification of mensajes recibidos
  - **Fake Object**: imitación. Se comporta como el módulo real (protocolos, tiempos de respuesta, etc)

Ascensores

- Métodos goUp, goDown, stop
  - Invocan mensajes del “motor”
  - Recibe eventos del “sensor de piso”

Con cada TestDouble se puede:

- **Stub**: recibe los mensajes que envía el Ascensor
  - testear distanceTo(:Request)
- **Spy**: guarda registro de los comando del Ascensor
  - verificar los métodos invocados en el motor
- **Mock**: comprueba la validez de los comandos que envía el Ascensor
- **Fake**: simula el comportamiento del motor → generando eventos del sensor de piso

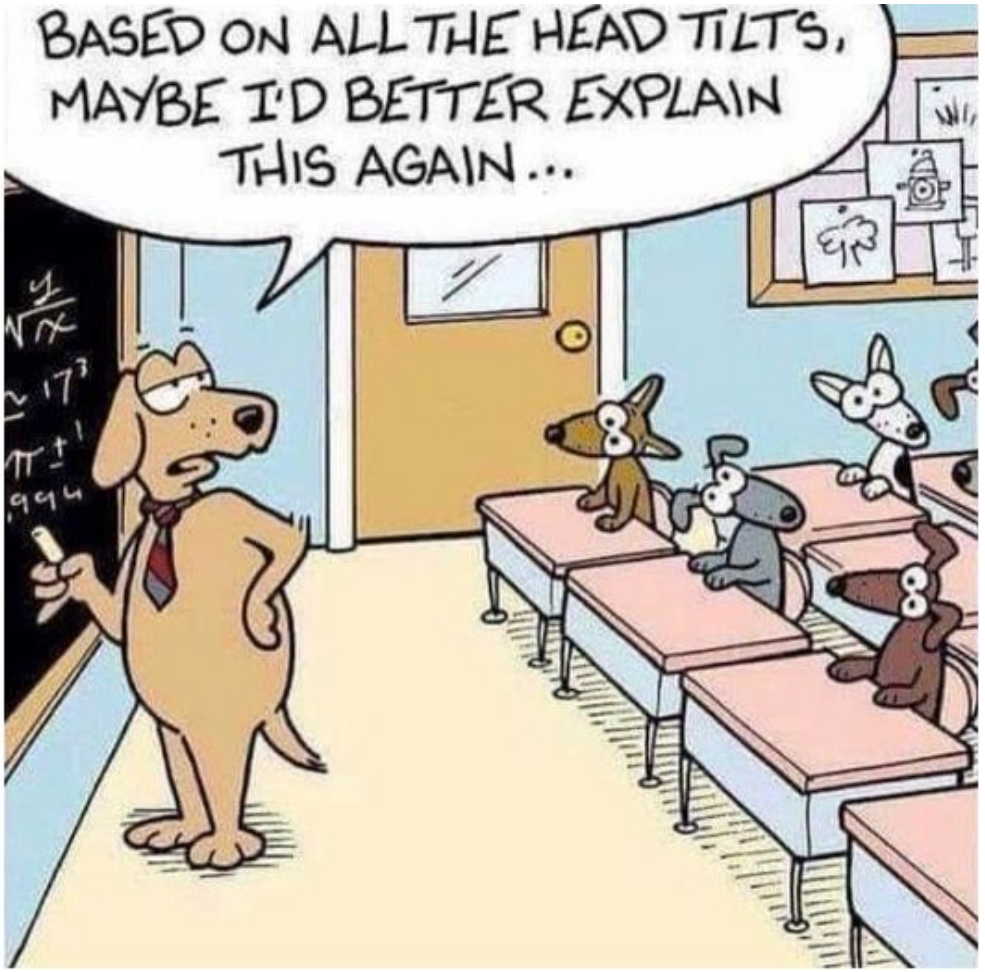
Test Double

- Implementar clases según sea necesario
  - Es necesario verificar funcionalidad de dependencia
  - Objetos que no están disponibles para probar
- Implementación:
  - Test Stub: simple y barato
  - Fake object: demanda análisis, threading, requiere mantenimiento

…¿y estos que son stubs, spies, mocks o fakes?

Test Doubles

- **Stub**: recibe los mensajes, no hace nada.
- **Spy**: guarda registro de mensajes recibidos.
- **Mock**: comprueba la validez de los mensajes recibidos
- **Fake**: simula el comportamiento en tiempo y forma



Comm Check...: The Final Flight of Shuttle Columbia

Michael Cabbage, William Harwood, 2008