Técnicas de diseño de casos de prueba

Entornos de desarrollo

Diapositivas realizadas por Aitor Ventura Edo

IES El Caminás
Curso 2025-2026

INTRODUCCIÓN

Dificultad del diseño de casos de prueba

Imposibilidad de prueba exhaustiva:

Ejemplo: En un programa que suma dos números de 0 a 99, habría 10,000 combinaciones posibles para probar, además de entradas inválidas.

Este número crece exponencialmente con la complejidad del software.

Equilibrio necesario:

Se deben seleccionar casos representativos que equilibren:

- Confianza en detectar errores.
- Uso eficiente de los recursos disponibles.

Idea fundamental del diseño

1. Elegir casos representativos que permitan detectar defectos sin necesidad de probar todas las posibilidades.
2. Si los casos seleccionados no detectan defectos, se gana confianza en que el software es funcional.
3. Desafío: Decidir qué casos son los más relevantes.

CAJA BLANCA

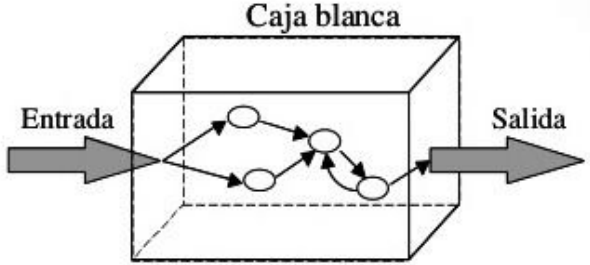
Pruebas de caja blanca

¿Qué son las pruebas de caja blanca?

Las pruebas de caja blanca son una técnica de pruebas de software que se centra en evaluar la estructura interna, lógica y diseño del código.

El objetivo principal es verificar que las rutas lógicas y los componentes internos del programa funcionen como se espera.

"Caja blanca" se refiere a que el probador tiene acceso al código fuente del software y puede examinarlo directamente.



Pruebas de caja blanca

¿Qué se analiza en las pruebas de caja blanca?

Se realizan verificaciones en diferentes aspectos internos del software, incluyendo:

- Rutas de ejecución: Se evalúan los diferentes caminos que puede tomar la ejecución del programa según las condiciones establecidas en el código.
  - Ejemplo: Verificar qué ocurre si una condición if es verdadera o falsa.
- Condiciones y decisiones: Comprueba que todas las decisiones (if, switch) se evalúan correctamente para todas las posibilidades de entrada.
- Bucles: Verifica el funcionamiento de los bucles (for, while):
  - ¿El bucle se ejecuta el número esperado de veces?
  - ¿Qué ocurre si el bucle no se ejecuta en absoluto?

Pruebas de caja blanca

¿Qué se analiza en las pruebas de caja blanca?

Se realizan verificaciones en diferentes aspectos internos del software, incluyendo:

- Estructuras de datos: Analiza el correcto manejo de estructuras de datos internas, como arrays, pilas, colas, etc.
- Cobertura del código: Se mide qué porcentaje del código ha sido probado mediante métricas como:
  - Cobertura de sentencias: ¿Se ejecutaron todas las líneas del código al menos una vez?
  - Cobertura de condiciones: ¿Se evaluaron todas las condiciones posibles?
  - Cobertura de rutas: ¿Se probaron todas las combinaciones de rutas posibles?

Beneficios de caja blanca

Beneficios de las pruebas de caja blanca

1. **Detección de errores internos:** Encuentra defectos en la lógica del código que no serían detectados solo con pruebas funcionales.
2. **Optimización del código:** Identifica partes del código innecesarias o redundantes.
3. **Cobertura completa:** Asegura que todas las partes del código se prueben.
4. **Análisis preventivo:** Detecta vulnerabilidades de seguridad al revisar cómo se manejan las entradas y salidas.

Limitaciones de caja blanca

Limitaciones de las pruebas de caja blanca

1. **Coste en tiempo y recursos:** Requiere un conocimiento profundo del código fuente, lo que puede aumentar el tiempo necesario para diseñar y ejecutar las pruebas.
2. **Escalabilidad limitada:** En sistemas grandes, probar todas las rutas y condiciones puede ser impracticable debido al número de combinaciones.
3. **Dependencia del conocimiento técnico:** Requiere que el probador sea un desarrollador o tenga habilidades avanzadas en programación.
4. **No detecta errores de integración o requisitos:** Estas pruebas no garantizan que el software cumpla con los requisitos del cliente, ya que solo se enfocan en la lógica interna.

Técnica del camino básico

Técnica del camino básico:

La técnica del camino básico permite identificar y probar todas las rutas independientes de un programa, asegurando que se cubra la lógica interna del código. Los pasos son los siguientes:

1. Analizar el código.
2. Crear el diagrama de flujo de control.
3. Calcular la complejidad ciclomática.
4. Identificar los caminos independientes.
5. Diseñar los casos de prueba.
6. Ejecutar y validar.

Ejemplo de técnica del camino básico

Para explicar la ténica del camino básico vamos a utilizar un ejemplo:

```java
public static double calcularTotalConDescuento(double[] precios) {
        double totalConDescuento = 0;

        for (int i = 0; i < precios.length; i++) {
            double precio = precios[i];

            if (precio > 50) {
                // Aplica un 20% de descuento
                totalConDescuento += precio * 0.8;
            } else {
                // Sin descuento
                totalConDescuento += precio;
            }
        }

        return totalConDescuento;
    }
```

Paso 1: Analizar el código

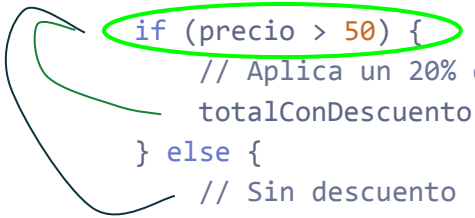
Identifica las condiciones, bucles y rutas del programa.

```java
public static double calcularTotalConDescuento(double[] precios) {
        double totalConDescuento = 0;

        for (int i = 0; i < precios.length; i++) {
            double precio = precios[i];

            if (precio > 50) {
                // Aplica un 20% de descuento
                totalConDescuento += precio * 0.8;
            } else {
                // Sin descuento
                totalConDescuento += precio;
            }
        }

        return totalConDescuento;
    }
```



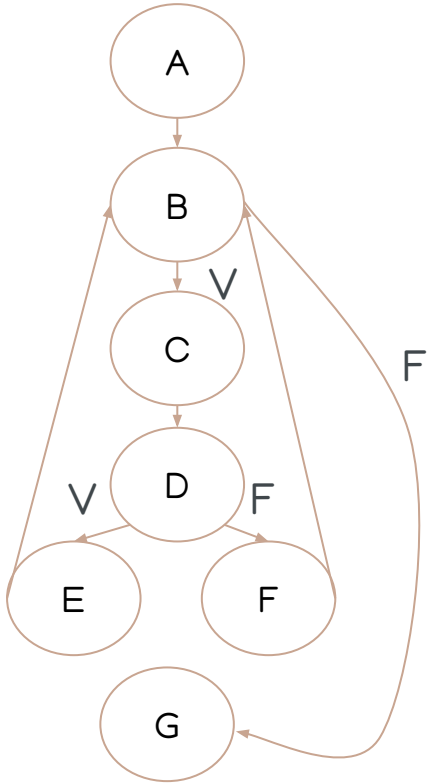
Paso II: Crear el Diagrama de Flujo de Control

```java
public static double calcularTotalConDescuento(double[] precios) {
    double totalConDescuento = 0;

    for (int i = 0; i < precios.length; i++) {
        double precio = precios[i];

        if (precio > 50) {
            // Aplica un 20% de descuento
            totalConDescuento += precio * 0.8;
        } else {
            // Sin descuento
            totalConDescuento += precio;
        }
    }

    return totalConDescuento;
}
```



Paso III: Calcular la complejidad ciclomática.

La complejidad ciclomática es una métrica utilizada para medir la complejidad lógica de un programa. Sirve como un indicador de la cantidad de caminos independientes que existen en el código fuente, es decir, los posibles flujos de ejecución.

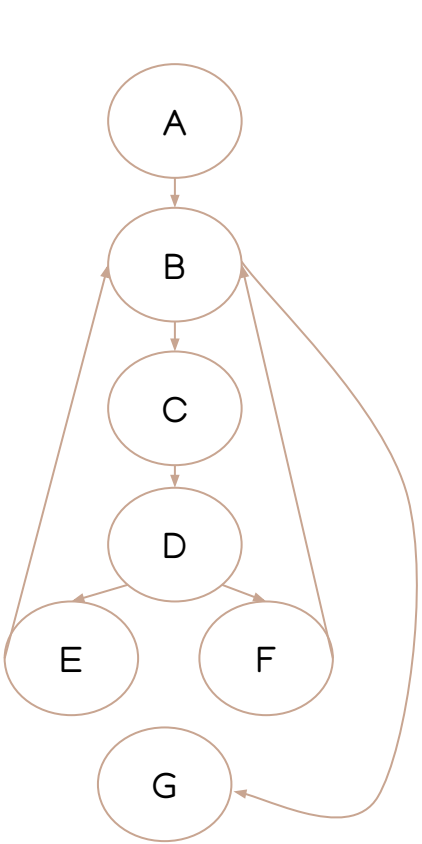
La fórmula básica para calcular la complejidad ciclomática es: $M = E - N + 2$.

Siendo E: número de aristas, N: número de nodos y M: complejidad ciclomática.

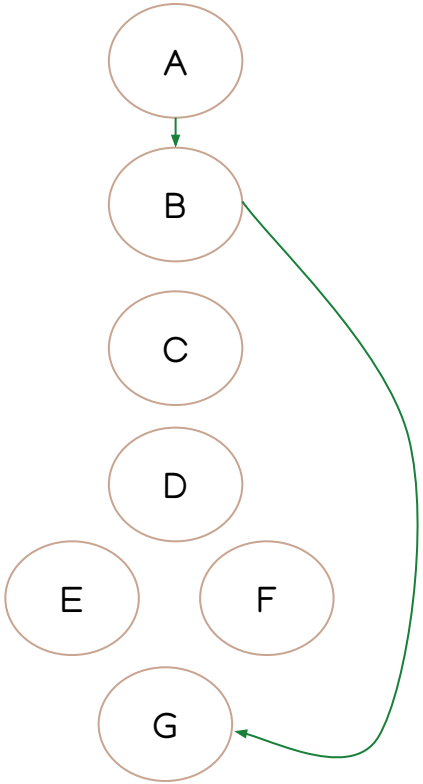
En nuestro ejemplo: M = 8 – 7 + 2 = 3

Por lo que hay 3 caminos independientes en el código. Con esos caminos pasaremos por todos los nodos.

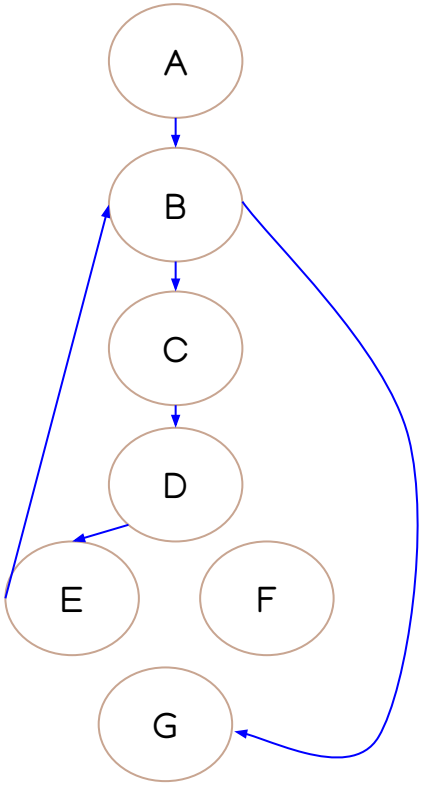
Paso IV: Identificar los caminos independientes.



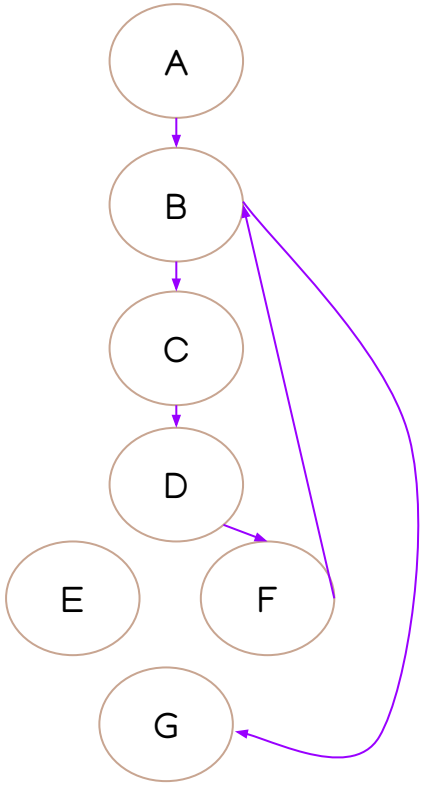
Camino 1



Camino 2



Camino 3



Paso V: Diseñar los casos de prueba

Camino 1

- Descripción: Array vacío
- Entrada: {}
- Salida esperada: 0.0

Camino 2

- Descripción: Precios mayores a 50 (descuento)
- Entrada: {60.0, 100.0}
- Salida esperada: 60×0.8+100×0.8=128.0

Camino 3

- Descripción: Precios menores o iguales a 50
- Entrada: {20.0, 50.0}
- Salida esperada: 20+50=70.0

Paso VI: Ejecutar y validar

Camino 1

- Descripción: Array vacío
- Entrada: {}
- Salida esperada: 0.0

```
La entrada es []
El precio final para el camino es 0.0
La entrada es [60.0, 100.0]
El precio final para el camino es 128.0
La entrada es [20.0, 50.0]
El precio final para el camino es 70.0
```

Camino 2

- Descripción: Precios mayores a 50
- Entrada: {60.0, 100.0}
- Salida esperada: 60×0.8+100×0.8=128.0

Camino 3

- Descripción: Precios menores o iguales a 50
- Entrada: {20.0, 50.0}
- Salida esperada: 20+50=70.0

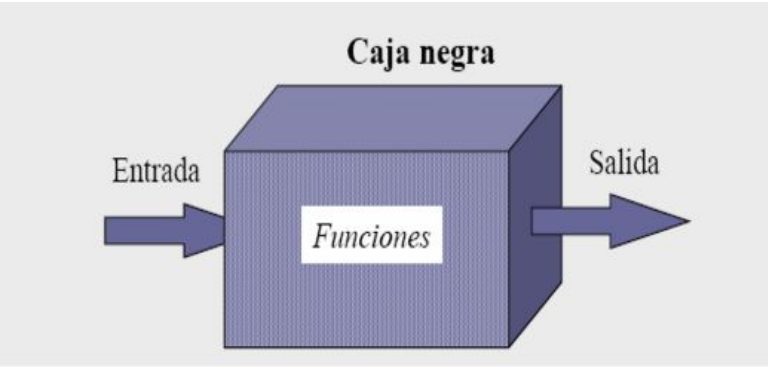
CAJA NEGRA

Pruebas de caja negra

¿Qué son las pruebas de caja negra?

Las pruebas de caja negra se enfocan en evaluar el comportamiento del software basándose en sus especificaciones, entradas y salidas, sin analizar su implementación interna.

Se seleccionan subconjuntos estratégicos de casos de prueba para detectar defectos de manera eficiente.



Particiones equivalentes

Particiones equivalentes

Dividen las entradas en clases de equivalencia, probando solo valores representativos que extrapolen resultados a otros valores de la misma clase.

Ejemplo: Una aplicación valida la edad de un usuario, aceptando valores entre 18 y 65 años.

Clases de equivalencia:

1. Valores válidos: 18-65 → 33
2. Valores fuera del rango inferior: menores de 18 → 6
3. Valores fuera del rango superior: mayores de 65 → 82

Definición de clases de equivalencia

Tipo de entrada	N° clases válidas	N° clases inválidas
Rango de valores Ej. [20..30]	1: valor en rango (25)	2: valor por debajo y otro por encima del rango (15 y 40)
Conjunto finito de valores Ej. {2, 4, 6, 8}	1: valor en el conjunto (4)	2: valor fuera del conjunto, por debajo y por encima (1 y 10)
Condición booleana (T/F) Ej. “debe ser una letra”	1: valor evaluado a cierto (“j”)	1: valor evaluado a falso (“?”)
Conjunto de valores admitidos Ej. {opción1, opción2, opción3}	3 en este ejemplo: tantos como valores admitidos {opción1, opción2, opción3}	1: valor no admitido (opción4)

Análisis de valores límite

Análisis de valores límite

Se eligen valores en los límites de las clases de equivalencia, donde suelen ocurrir errores.

Ejemplo: Una aplicación valida la edad de un usuario, aceptando valores entre 18 y 65 años.

Casos de prueba seleccionados:

- 18 (límite inferior válido)
- 65 (límite superior válido)
- 17 (justo debajo del límite inferior)
- 66 (justo por encima del límite superior)

Definición de valores límite

Tipo de entrada	N° clases válidas	N° clases inválidas
Rango de valores Ej. [20..30]	4: valor en los límites del rango (20, 21, 29 y 30)	2: valor justo por debajo y justo por encima del rango (19 y 31)
Conjunto finito de valores Ej. {2, 4, 6, 8}	4: valores mínimo y máximo en el conjunto y valores directamente adyacentes dentro del conjunto (2, 4, 6 y 8)	2: valor justo por debajo y justo por encima del conjunto (1 y 9)

Pruebas aleatorias

Se generan entradas aleatorias para probar un sistema, sin seguir un patrón específico.

Ejemplo: Para una calculadora que acepta dos números enteros y realiza operaciones básicas (+, −, *, /), se generan combinaciones aleatorias:

Entradas generadas aleatoriamente:

- 25 y 5 con operación +
- −12 y 0 con operación /
- 9 y −3 con operación *
- 100 y 50 con operación −