# 5.1. Comprobación y Depurado

Egin klik testua gehitzeko



## Introducción

- Hay que comprobar que un programa funcione adecuadamente
- 2 Formas
  - **Validación (*Validation*):** Proceso para determinar que el programa funciona como se pretende
  - **Depurado (*Debugging*):** Proceso de comprensión y corrección de errores

## Validación

- Determina si un programa funciona de manera adecuada
- No determina dónde o por qué falla
- La validación de un programa se puede hacer mediante:
  - **Verificación (*Verification*):** Proceso formal o informal que determina el correcto funcionamiento del programa con todas las entradas posibles
  - **Comprobación (*Testing*):** Proceso de ejecución de un conjunto de casos de prueba y comparación de los resultados obtenidos y los resultados esperados



## Comprobación

- **Comprobaciónes de tipo caja negra (*Black-Box Testing*)**
  - Los casos de prueba se generan considerando sólo la especificación
- **Comprobaciones de tipo caja blanca (*Glass-Box Testing*)**
  - Los casos de prueba se generan teniendo en cuenta los distintos caminos que se pueden encontrar en el diseño o la implementación

## Comprobaciones de Tipo Caja Negra

- Hay que determinar todos los caminos a partir de la especificación
- Para ello analizaremos
  - Precondiciones
  - Postcondiciones



## Comprobaciones de Tipo Caja Negra

```
/**
 * raizCuadrada
 * PRE: x>= 0 && .00001 < epsilon < .001
 * POST: devuelve sq tal que x - epsilon <= sq*sq <= x +
 * epsilon
 **/
public static float raizCuadrada (float x, float epsilon)
```

- Caminos
  - X = 0 && .00001 < epsilon < .001
  - X > 0 && .00001 < epsilon < .001

## Comprobaciones de Tipo Caja Negra

```
/**
 * esPrimo
 * POST: devuelve true si x es primo y false si no
 **/
public static boolean esPrimo (int x)
```

- Caminos
  - Conjunto de primos
  - Conjunto de no primos



## Comprobaciones de Tipo Caja Negra

- Excepciones
  - Si el programa o subprograma puede lanzar excepciones hay que hacer pruebas con los casos que provocan esa situación
  - Ejemplo:

- Caminos
  - $x \in a$
  - $x \notin a$
  - a es *null*

```
/**
 * buscaElemento
 * POST: Si a es null  lanza NullPointerException. Si no,
 * devuelve i tal que a[i]=x. Si no lanza NotFoundException
 **/
public static int buscaElemento (int[] a, int x) throws
        NotFoundException, NullPointerException
```

## Comprobaciones de Tipo Caja Negra

- Además de comprobar los valores *típicos* (valores entre el mínimo y máximo especificados,…) hay que comprobar valores *límite*
  - Raíz cuadrada de 0
- Errores que se pueden detectar
  - **Lógicos:** Se ha omitido el tratamiento de un valor límite
  - **Omisión de condiciones que pueden provocar un error del lenguaje o hardware:** Overflow,…

Conviene comprobar valores que se aproximen a los límites
- raizCuadrada:
  - Epsilon --> .001
  - Epsilon --> .00001

## Comprobaciones de Tipo Caja Negra

- Valores límite
  - Strings
    - Cadena vacía
    - Cadena con un único carácter
  - Arrays
    - Array vacío
    - Array con un único elemento

# Comprobaciones de Tipo Caja Negra

- Otro tipo de condición límite surge cuando cuand
  eleme

```
public static void concatenarCadena (StringBuffer a,
StringBuffer x)  throws NullPointerException
{
StringBuffer cadena =new StringBuffer( “abc”);
concatenarCadena (cadena, cadena);
    {
        a = a + x.charAt(x.length() -1);
        x = x.substring(0,x.length() -1);
    }
}
```



# Comprobaciones de Tipo Caja Blanca

- Las pruebas de tipo caja negra no son suficientes
- Se basan en la implementación
- Analizar todos los caminos encontrados en el código

## Comprobaciones de Tipo Caja Blanca

```
public static void maximoDeTres (int x, int y, int z)
{
    if (x > y)
        if (x > z)
            return x;
        else
            return z;
    else if (y > z)
        return y;
    else
        return z;
}
```

- $n^3$ posibilidades ($n$ rango de enteros)
- Caminos
  - 3, 2, 1
  - 3, 2, 4
  - 1, 2, 1
  - 1, 2, 3

## Comprobaciones de Tipo Caja Blanca

- Bucles → número muy elevado de caminos

```
j = k;
for (int i = 1; i <= 100; i++)
    if (Tests.pred (i*j)) j++;
```

- Simplificar → número reducido de caminos

```
j = k;
for (int i = 1; i <= 2; i++)
    if (Tests.pred (i*j)) j++;
```

# Comprobaciones de Tipo Caja Blanca

- Bucles con número fijo de iteraciones
  - Pruebas con 2 iteraciones
  - La 2ª iteración permite localizar errores de reinicialización de variables
- Bucles con número variables de iteraciones
  - Pruebas con 0,1 y 2 iteraciones
- Programas recursivos
  - Pruebas con 0 y 1 llamada recursiva



# Ejemplo de Comprobación

```
/**
 * esPalindromo
 * POST: lanza       NullPointerException si cadena es null. Si
 * no, devuelve true  si la cadena se lee igual de derecha a
 * izquierda y de izquierda a derecha. En caso contrario
 *  devuelve false. Ej: "deed" y "" son palindromos
 **/
public static boolean esPalindromo (String cadena)
                        throws NullPointerException {
      int inf = 0;
      int sup = cadena.length() - 1;
      while (sup > inf) {
            if (cadena.charAt(inf) != cadena.charAt(sup))
                  return false;
            else {
                  sup--;
                  inf++;
            }
      }
      return true;
}
```

## Ejemplo de Comprobación

E

- Caja Negra
  - Especificación
    - *Null*
    - Palíndromo
    - No palíndromo
  - Valores límite
    - Cadena vacía
    - Cadena con un carácter

- *null*
- “”
- “d”
- “deed”
- “ceed”



## Ejemplo de Comprobación

E

- Caja Blanca
  - NullPointerException
  - Sin ejecutar bucle
  - *False* en la primera iteración
  - *True* en la primera iteración
  - *False* en la segunda iteración
  - *True* en la segunda iteración

- Falta el caso *False* en la segunda iteración
  - “aaba”
- Añadir cadenas de longitud impar para el resto de los casos