ENTORNOS DE DESARROLLO UD-5

PRUEBAS DE CAJA NEGRA

ÍNDICE

[INTRODUCCIÓN 3](#_Toc132887630)

[CASO PRÁCTICO DE CAJA NEGRA 3](#_Toc132887631)

[A. DETERMINAR LAS CLASES DE EQUIVALENCIA 4](#_Toc132887632)

[B. ANÁLISIS DE LOS VALORES LÍMITE 5](#_Toc132887633)

[C. CONJETURA DE ERRORES 6](#_Toc132887634)

[D. CASOS DE USO, RESULTADOS ESPERADOS Y ANÁLISIS 6](#_Toc132887635)

# INTRODUCCIÓN

El propósito de las pruebas de caja negra o funcionales es comprobar si las salidas que devuelve la aplicación las esperadas en función de los parámetros de entrada.

Texto

Descripción generada automáticamente

Este tipo de prueba no consideraría, en ningún caso el código desarrollado, ni el algoritmo, ni la eficiencia, ni si hay partes del código innecesarias, etc. Estos aspectos son comprobados en las pruebas de caja blanca.

Dentro de las pruebas funcionales, podemos indicar los siguientes tipos:

* **Particiones equivalentes**: la idea de este tipo de pruebas funcionales es considerar el menor número posible de casos de pruebas, para ello, cada caso de prueba tiene que abarcar el mayor número posible de entradas diferentes. Lo que se pretende, es crear un conjunto de clases de equivalencia, donde la prueba de un valor representativo de la misma, en cuanto a la verificación de errores, sería extrapolable al que se conseguiría probando cualquier valor de la clase.
* **Análisis de valores límite**: en este caso, a la hora de implementar un caso de prueba, se van a elegir valores de entrada aquellos que se encuentra en el límite de las clases de equivalencia.
* **Pruebas aleatorias**: consiste en generar entradas aleatorias para la aplicación que hay que probar. Se suelen utilizar generadores de prueba, que son capaces de crear un volumen de casos de prueba al azar, con los que será alimentada la aplicación.
* **Conjetura de errores**: trata de generar casos de prueba que la experiencia ha demostrado generan típicamente errores. En valores numéricos, un buen ejemplo es comprobar si funciona correctamente con el valor 0, ya que si es utilizado como denominador en alguna división podría generar un error en nuestro programa.

# CASO PRÁCTICO DE CAJA NEGRA

La técnica para determinar los casos de prueba de caja negra se realiza completando los siguientes pasos:

1. Determinar las clases de equivalencia.
2. Determinar un análisis de valores límite.
3. Conjetura de errores.
4. Generar los casos de uso necesarios para probar las clases validas y no validas.
5. Establecer los datos de entrada y los resultados esperados.
6. Lanzar una ejecución del programa por cada caso de uso y comparar los resultados obtenidos con los esperados para determinar la corrección del código.

## DETERMINAR LAS CLASES DE EQUIVALENCIA

La técnica de clases de equivalencia es un tipo de prueba funcional, donde en cada caso de prueba se agrupa el mayor número de entradas posibles. A partir de aquí, se asume que la prueba de un valor representativo de cada clase, permite suponer que el resultado que se obtiene con él, será el mismo que con cualquier otro valor de la clase.

Los pasos para seguir para identificar las clases de equivalencia son:

1. Identificar las condiciones de las entradas del programa, es decir, restricciones de formato o contenido de los datos de entrada.
2. A partir de ellas, identificar clases de equivalencia que pueden ser:
   1. De datos válidos.
   2. De datos no válidos o erróneos.

Existen algunas **reglas heurísticas** que ayudan a identificar las clases:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

En cualquier caso, si se sospecha que ciertos elementos de una clase no se tratan igual que el resto de esta, deben dividirse en clases menores.

## ANÁLISIS DE LOS VALORES LÍMITE

La experiencia indica que los casos de prueba que exploran las condiciones límite de un programa producen un mejor resultado para detectar defectos.

El AVL (Análisis de valores límite) es una técnica de diseño de casos de prueba que complementa a la de particiones de equivalencia.

La principal diferencia se encuentra en el tratamiento que tienen las clases de equivalencia de rango de valores y de número finito y consecutivo de valores. Ahora la prueba se realizará sobre los valores límite de los rangos.

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

En las pruebas AVL también habría que generar casos de prueba atendiendo a clases de equivalencia de los datos de salida.

## CONJETURA DE ERRORES

La experiencia en la fase de pruebas indica que existen ciertos valores de entrada que típicamente son generadores de errores, y qué en ocasiones, pasan desapercibidos en las técnicas de clases de equivalencia y de valores límite.

La conjetura de errores considera esos datos y define nuevos casos de prueba a los que someter a los programas. Se trata de una técnica menos metódica que las anteriores, tiene mucho más que ver con la intuición y experiencia del programador.

Una prueba típica en conjetura de errores es probar el valor de entrada 0 para datos numéricos por si pudiera participar como denominador en alguna división durante la ejecución del programa.

## CASOS DE USO, RESULTADOS ESPERADOS Y ANÁLISIS

Ahora toca definir los datos de entrada al programa. Al conjunto de entradas al programa utilizados para cada ejecución se le denomina caso de uso. Los casos de uso se generarán a partir de las clases de equivalencia, valores límite y conjeturas de errores obtenidos en los apartados anteriores.

Este proceso consta de las siguientes fases:

1. Numerar las clases de equivalencia.
2. Crear casos de uso que cubran todas las clases de equivalencia válidas. Se intentará agrupar en cada caso de uso tantas clases de equivalencia como sea posible.
3. Crear un caso de uso para cada clase de equivalencia no válida.

Además, toca definir los resultados previstos en cada ejecución. Cuando posteriormente se lance la ejecución del programa para cada caso de uso, los resultados obtenidos serán comparados con los esperados y así determinar la corrección del código.