TEMA 3: Pruebas del Software

Módulo

Entornos de Desarrollo

para los ciclos

Desarrollo de Aplicaciones Web

Desarrollo de Aplicaciones Multiplataforma

3.1 Casos de prueba

Buena parte del tiempo de desarrollo de software se dedica a probar el código que se ha generado previamente. Cada prueba realizada sobre el código puede detectar errores (a menudo los detecta) que tienen que ser reparados.

Un ***caso de prueba*** se compone de entradas y resultados (o comportamiento) esperados. Cada caso de prueba incluirá también unas condiciones de ejecución. Una vez ejecutado el caso de prueba se analiza el comportamiento y se comparan los resultados con lo que se esperaba para determinar si el sistema ha pasado o no la prueba.

Ejemplo 1: Caso de prueba para un programa que calcula la división de dos números de entrada llamados dividendo y divisor.

Datos de entrada		***Resultado o comportamiento esperado***
dividendo	***divisor***	***dato de salida***
8	2	4

Si decidimos usar este caso de prueba entonces ejecutaremos el programa aportando los datos de entrada indicados: ***8 como dividendo*** y ***2 como divisor***. Tras ejecutarlo observamos el resultado obtenido y lo comparamos con el resultado esperado: ***4***. Si son iguales significa que el programa ha superado la prueba. Si no son iguales habrá que detectar y solucionar el error.

Ejemplo 2: Caso de prueba para un programa que calcula la división de dos números de entrada llamados dividendo y divisor.

Datos de entrada		***Resultado o comportamiento esperado***
dividendo	***divisor***	***dato de salida***
8	0	mensaje de error informativo

En este caso el resultado esperado tras ejecutar el programa no es un resultado numérico, sino un comportamiento: un mensaje de error informativo. Si la ejecución termina con un comportamiento distinto significa que hay un fallo que reparar.

Cuando diseñamos ***una prueba tendremos que incluir varios casos de prueba, de forma que se cubran todos los grupos de posibilidades***. No podremos probar todas posibles entradas para el programa de la división. Pero si podemos crear un caso de prueba de cada situación distinta: con números enteros, con números decimales, con números negativos, con ceros, etc.

3.1.1 Caja blanca y caja negra

Se distinguen dos tipos de pruebas:

- ***De caja blanca***: (o estructurales) Se analiza la estructura interna del programa inspeccionando el código, por eso, en las pruebas de caja blanca se incluyen casos de prueba para que:
  - ***Todas las sentencias se ejecutan al menos una vez***. Si hay un bucle habrá que incluir casos de prueba en los que la ejecución entra en el bucle y casos de prueba cuya ejecución se salta el bucle.
  - Todas l***as condiciones se ejecutan en su parte verdadera y en su parte falsa***. Habrá al menos dos casos de prueba: uno que pasa por la parte en la que se cumple la condición y otro que pasa por la parte en la que no se cumple esa condición.
  - Los bucles se ejecutan en sus límites. Todos los programadores, incluso los más experimentados, pueden tropezar en los límites de los bucles. Si un bucle se ejecuta exclusivamente para valores que van desde ***n*** hasta ***m***, tendremos que incluir casos de prueba que llegan al bucle con los valores ***n-1***, ***n***, ***m*** y ***m+1***. Observa que se trata del primer y último valor que entran al bucle y de los valores adyacentes que no entran.
  - Todas las estructuras de datos se usan para asegurar su validez. Por ejemplo, si nuestro programa incluye algún ***array***, la prueba no estará completa si no incluye un caso de prueba que use ese ***array***.
- ***De caja negra***: (o de comportamiento) Sólo se comprueba la funcionalidad, es decir, si la salida es adecuada en función de los datos de entrada, sin fijarse en el funcionamiento interno. En las pruebas de caja negra también serán necesarios múltiples casos de prueba que incluyan todos los grupos de entradas que tienen comportamientos distintos.

3.2 Secuencia de pruebas

La prueba completa del software requiere los siguientes pasos:

- ***Prueba de unidad***: Se centra en cada unidad de software, en cada módulo del código fuente. Aquí es donde se utilizan las pruebas de caja blanca y de caja negra. Probaremos cada método, cada clase, etc.
- ***Prueba de integración***: A partir de los módulos probados individualmente, se prueba el funcionamiento conjunto. Hay dos enfoques:
  - ***Big bang***: Consiste en ser estrictos en el orden de probar primero todos los módulos por separado antes de pasar al conjunto. Con este enfoque los errores se acumulan y suele ser más complicado.
  - ***Integración incremental***: Se van incorporando módulos poco a poco y probando su integración. Los errores son más fáciles de localizar así.
- ***Prueba de validación***: En el entorno real de trabajo y con intervención del usuario final. Se basa en determinar si se cumplen los requisitos y se usan para ello pruebas de caja negra de dos tipos:
  - ***Prueba alfa***: Con el usuario final en el lugar de desarrollo. El desarrollador observa y registra los errores y problemas.
  - ***Prueba beta***: Con el usuario final en su propio lugar de trabajo. El desarrollador no está presente, pero será informado por el usuario.
- ***Prueba del sistema***: Para verificar funcionalidad y rendimiento. Se prueba su integración con otros elementos del sistema. Incluye las siguientes:
  - ***Prueba de recuperación***: Se fuerza el fallo del software y se comprueba que la recuperación es satisfactoria.
  - ***Prueba de seguridad***: Se intenta verificar que el sistema está protegido contra accesos ilegales.
  - ***Prueba de resistencia o estrés***: Enfrenta al sistema con situaciones que demandan gran cantidad de recursos: accesos simultáneos, uso de memoria, etc.

Todas las pruebas que se realizan deben de estar documentadas incluyendo calendarios, objetivos, resultados, personal responsable y los detalles de cada prueba. Naturalmente también se deben acompañar de conclusiones y, en su caso, instrucciones para solucionar los problemas detectados.

3.3 Pruebas habituales

3.3.1 Prueba del camino básico (caja blanca)

La prueba del camino básico es una prueba de caja blanca que permite obtener una ***medida de la complejidad*** del diseño de un procedimiento (por ejemplo, un método ***Java***). A partir de esa medida se establece un ***conjunto básico de caminos*** de ejecución y, finalmente, se diseñan casos de prueba para cada uno de esos caminos. Se trata de conseguir que, durante la prueba, todas las sentencias del programa se ejecutan al menos una vez, por eso se trata de una prueba de caja blanca.

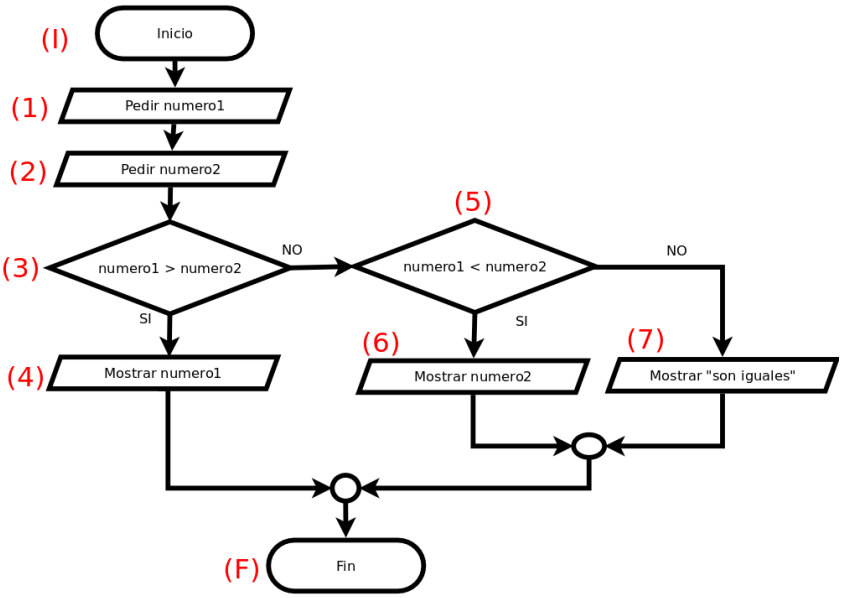
GRAFO DE FLUJO

En un grafo de flujo los nodos son círculos que representan un grupo de sentencias que siempre se ejecutan juntas y las flechas indican la dirección que sigue el flujo entre los nodos. Podemos obtener el grafo de flujo a partir de un diagrama de flujo o a partir del código.

Ejemplo 1: Grafo de flujo a partir de un diagrama de flujo (Mayor)

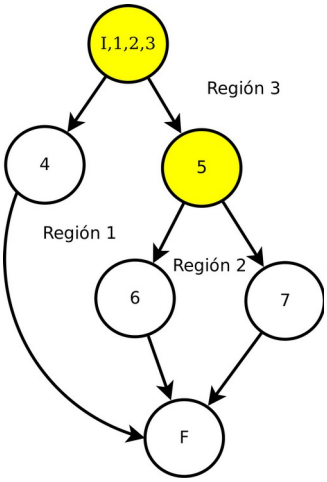
Aquí tenemos el diagrama de flujo para un algoritmo que solicita dos números al usuario y dice cual de ellos es mayor, o si son iguales.

Para obtener el grafo de flujo se numera cada elemento del diagrama (lectura, escritura, condición, etc), incluido el inicio ***(I)*** y el final ***(F)***. Posteriormente, al dibujar el grafo, agrupamos en un solo nodo todos los elementos del diagrama que siempre se ejecutan juntos.



La siguiente imagen muestra el grafo de flujo. Tenemos:

- ***Nodos***: cada uno de los círculos. En el ejemplo hay 6 nodos.
- ***Aristas***: cada una de la flechas. En el ejemplo hay 7 aristas.
- ***Nodos predicado***: son los nodos de los que salen dos o más aristas (son los nodos que contienen una condición). En el ejemplo hay 2 nodos predicado (los marcados en amarillo).
- ***Regiones***: cada una de las áreas encerradas por nodos y aristas. El área exterior del grafo es una región más. En el ejemplo hay 3 regiones.



Ejemplo 2: Grafo de flujo a partir de código o pseudocódigo (Sumador)

En realidad, no hay diferencias en la construcción del grafo a partir de un diagrama de flujo, pseudocódigo o código. En el código (o pseudocódigo) se numera cada elemento y se procede de la misma forma.

En el siguiente ejemplo tenemos un método ***Java*** en el que se pide al usuario que introduzca dos enteros positivos y devuelve la suma. Antes de sumar comprueba que los valores introducidos son, efectivamente, positivos. Cada línea de código ha sido numerada (incluyendo inicio y fin), pero observa que la línea que tiene ***la condición compleja tiene dos números***. Esto es porque, aunque nosotros lo escribimos como una única condición, en realidad se comprueban una tras otra.

Nota: Hay que tener especial cuidado con las condiciones complejas porque tendrán que representarse mediante varios nodos, tantos como condiciones simples equivalentes.

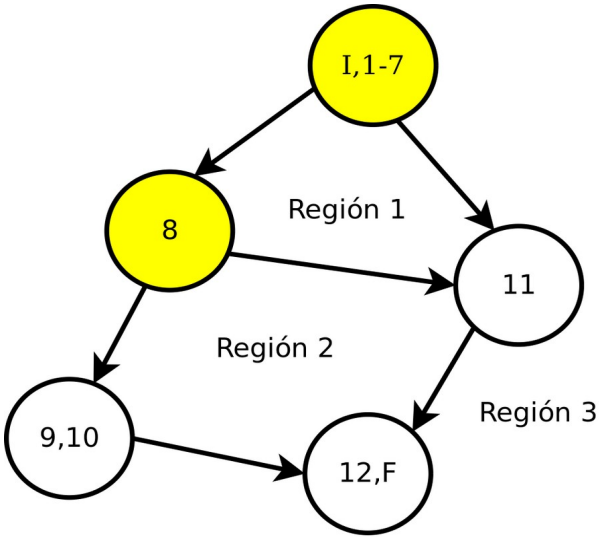
```
(I)  public void suma () {
     (1)     Scanner teclado = new Scanner(System.in);
     (2)     int a, b, resultado;

     (3)     System.out.println("Introduce un número positivo");
     (4)     a = teclado.nextInt();
     (5)     System.out.println("Introduce otro número positivo");
     (6)     b = teclado.nextInt();
     (7)y(8) if ((a>0) && (b>0)) {
     (9)             resultado = a + b;
     (10)            System.out.println("El resultado es: "+resultado);
             } else {
     (11)            System.out.println("No son positivos");
             }
     (12)    teclado.close();
(F)  }
```

El número ***(7)*** corresponde a la primera condición: ***a>0***. Si cuando el compilador evalúa esta condición obtiene un valor ***falso***, no evaluará la segunda porque independientemente de su valor, el camino a seguir ya está determinado.

El número ***(8)*** corresponde a la segunda condición: ***b>0***. Esta condición sólo será evaluada en caso de que la primera tenga un valor de ***verdadero***.

El grafo de flujo, que aparece en la siguiente imagen, tiene:

- 5 nodos
- 6 aristas
- 2 nodos predicado (en amarillo)
- 3 regiones



COMPLEJIDAD CICLOMÁTICA

Mide la complejidad de un programa y coincide con el número de caminos distintos que puede seguir el flujo. Contar el número de caminos puede ser complicado, sobre todo cuando aumenta el tamaño y complejidad del grafo. Por eso nos ayuda saber que ese número coincide con:

- El número de ***caminos*** distintos.
- El número de ***regiones*** del grafo.
- El número de aristas, restando el número de nodos y sumando 2: (***aristas-nodos+2***)
- El número de nodos predicado más 1: (***predicados + 1***)

Una complejidad ciclomática por encima del valor 20 en un procedimiento entraña un riesgo alto porque indica que se trata de un programa complejo. Si pasa de 50 hablamos de "muy alto riesgo" o incluso, de programas no testeables. Para evitar estos valores podemos ayudarnos de la programación modular, es decir, dividir el programa en varios módulos o subprogramas.

Ejemplo 1: Complejidad ciclomática del programa Mayor.

El grafo tiene 6 nodos, 7 aristas, 2 nodos predicado y 3 regiones. Si aplicamos las fórmulas veremos que la complejidad ciclomática del ejemplo 1 es ***3***, es decir, hay 3 caminos distintos y son:

- ***camino 1***: I, 1, 2, 3, 4, F
- ***camino 2***: I, 1, 2, 3, 5, 6, F
- ***camino 3***: I, 1, 2, 3, 5, 7, F

Ejemplo 2: Complejidad ciclomática del Sumador

Este grafo tiene 5 nodos, 6 aristas, 2 nodos predicado y 3 regiones. Aplicando cualquiera de las fórmulas expuestas anteriormente concluimos que tiene ***3 caminos*** distintos y por tanto, tendremos que elaborar 3 casos de prueba. Los caminos son:

- ***camino 1***: I, 1-7, 8, 9-10, 12, F
- ***camino 2***: I, 1-7, 8, 11, 12, F
- ***camino 3***: I, 1-7, 11, 12, F

CASOS DE PRUEBA