

ÍNDICE

1. PLANIFICACIÓN DE LAS PRUEBAS 1

2. PRUEBAS UNITARIAS..... 2

2.1. PROCEDIMIENTOS Y CASOS DE PRUEBA..... 3

2.2. TIPOS DE PRUEBAS 3

2.3. PRUEBAS DE REGRESIÓN 5

3. DOCUMENTACIÓN DE LA PRUEBA..... 5

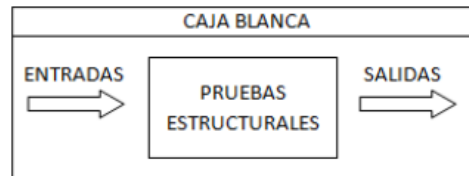
4. NORMAS DE CALIDAD..... 6

5. VALIDACIÓN 7

ACTIVIDAD. EJEMPLO 7

1. Introducción

Las pruebas estructurales son el conjunto de pruebas de la caja blanca. Con este tipo de pruebas, se pretende verificar la estructura interna de cada componente de la aplicación, con independencia de su funcionalidad.



Las pruebas estructurales no pretenden asegurar la corrección de los resultados producidos, su función es comprobar que se van a ejecutar todas las instrucciones del programa, que no hay código no usado, comprobar que los caminos lógicos del programa se van a recorrer, etc.

Este tipo de pruebas, se basan en unos criterios de cobertura lógica, cuyo cumplimiento determina la mayor o menor seguridad en la detección de errores.

Los criterios de cobertura son:

- **Cobertura de sentencias:** se han de generar casos de pruebas suficientes para que cada instrucción del programa sea ejecutada al menos una vez.
- **Cobertura de decisiones:** se trata de crear los suficientes casos de prueba para que cada opción, resultado de una comprobación lógica del programa, se evalúe al menos una vez a cierto y otra a falso. En la decisión MIENTRAS (A and B), habrá casos de prueba donde (A and B) sea verdadero y donde (A and B) sea falso.
- **Cobertura de condiciones:** se trata de crear los suficientes casos de prueba para que cada elemento de una condición, se evalúe al menos una vez a falso y otra a verdadero. En la decisión MIENTRAS (A and B), habrá casos de prueba donde A sea falso, A sea verdadero, B sea falso y B sea verdadero.
- **Cobertura de condiciones y decisiones:** consiste en cumplir simultáneamente las dos anteriores.
- **Cobertura del camino de prueba:** se pueden realizar dos variantes, una indica que cada bucle se debe ejecutar sólo una vez, ya que hacerlo más veces no aumenta la efectividad de la prueba y otra que recomienda que se pruebe cada bucle tres veces: la primera sin entrar en su interior, otra ejecutándolo una vez y otra más ejecutándolo al menos dos veces.

2. Caso práctico de Caja Blanca

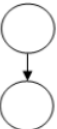
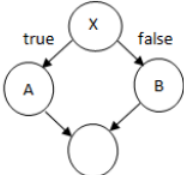
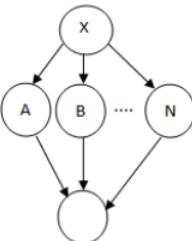
La técnica para determinar los casos de prueba de caja blanca que garantiza cobertura de sentencias, decisiones/condiciones y de caminos.

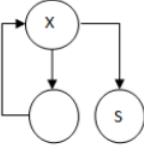
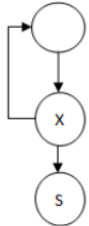
Se realiza completando los siguientes pasos:

1. Crear un grafo que represente el condigo a probar.
2. Calcular la complejidad ciclomática o de McCabe del grafo obtenido.
3. Determinar tantos caminos (recorridos del grafo) como la complejidad ciclomática calculada.
4. Generar un caso de uso por cada camino, determinando sus datos de entrada y los resultados esperados.
5. Lanzar una ejecución del programa por cada caso de uso y comparar los resultados obtenidos con los esperados para comprobar la corrección del código.

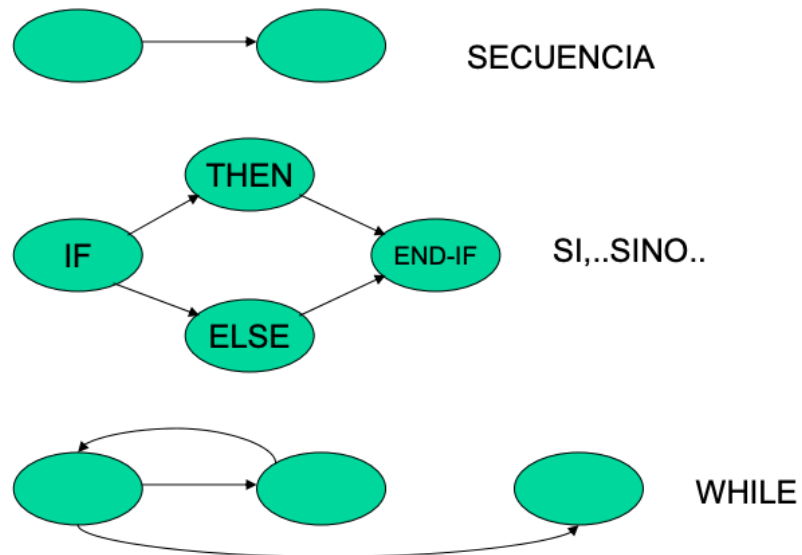
2.1. Obtención del Grafo

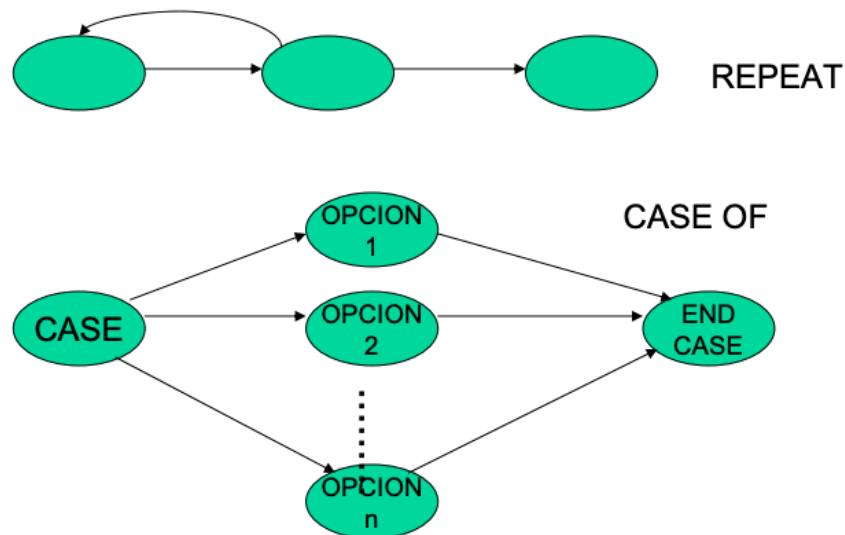
Se trata de **crear un grafo** en base al tipo de instrucciones que vayamos encontrando en nuestro código. Los tipos de estructuras principales que aparecen en los programas son secuencias de instrucciones, condiciones e iteraciones. Éstas se representan como sigue en el grafo.

Estructuras básicas	
Secuencia	 <pre>String sNota = "10"; System.out.println("Tu nota es: " + sNota);</pre>
Condición	 <pre>int iNota = 3; if(iNota >= 5) { System.out.println("Enhorabuena. Superado."); } else if (iNota < 5) { System.out.println("La proxima vez sera"); }</pre>
Selección múltiple	 <pre>switch (iNota) { case 1: case 2: case 3: case 4: { System.out.println("La proxima vez sera"); break;} default: { System.out.println("Enhorabuena. Superado."); } }</pre>

Iteración	 <pre> int iNumSal = 2; while(iNumSal > 0) { System.out.println("Hola !!!!!"); iNumSal--; } </pre>
Do Iteración	 <pre> int iNumSal = 2; do { System.out.println("Hola !!!!!"); iNumSal--; } while(iNumSal > 0); </pre>

Otra forma de verlo





Los grafos se construyen a partir de nodos y aristas. Los nodos representan secuencias de instrucciones consecutivas donde no hay alternativas en la ejecución o condiciones a evaluar, que en función del resultado hará que la ejecución siga una dirección u otra.

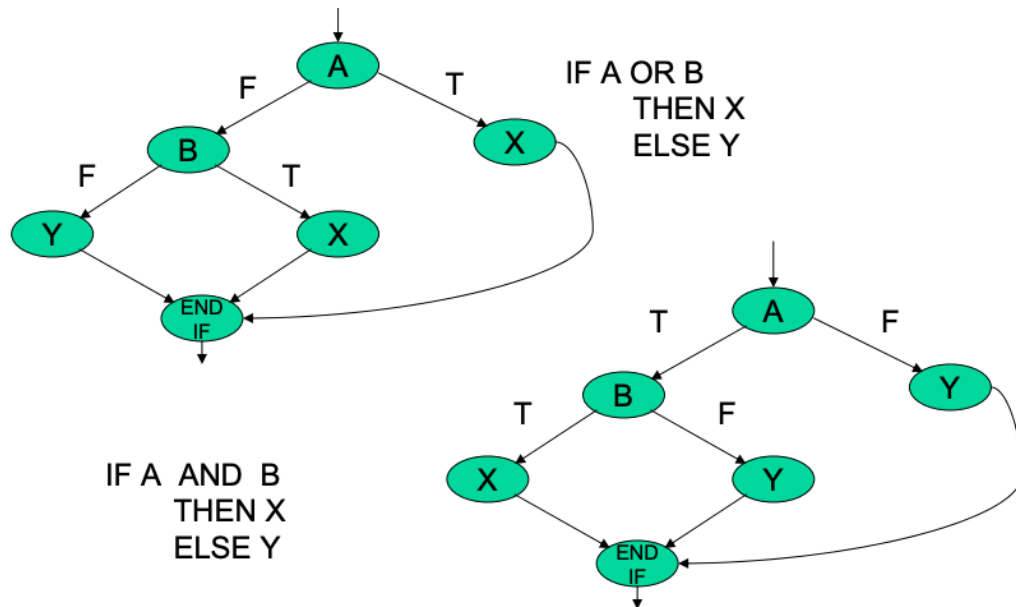
Las aristas son las encargadas de unir los nodos. En el caso de que las decisiones tengan múltiples condiciones, habrá **que separar cada condición en un nodo** como sigue:

Estructuras de Decisión	
And (&&)	<pre> int iNota = 6; boolean bPrimerio = true; if (iNota >= 5 && bPrimerio) { System.out.println("Enhorabuena. Pasas a 2."); } else { System.out.println("La proxima vez sera"); } </pre>
Or ()	<pre> int iNota = 6; if (iNota == 5 iNota > 5) { System.out.println("Enhorabuena. Superado."); } else { System.out.println("La proxima vez sera"); } </pre>

Algunos **consejos útiles** al crear grafos son:

1. Separar todas las condiciones.
2. Agrupar sentencias 'simples' en bloques.
3. Numerar todos los bloques de sentencias y también las condiciones.

EJEMPLO



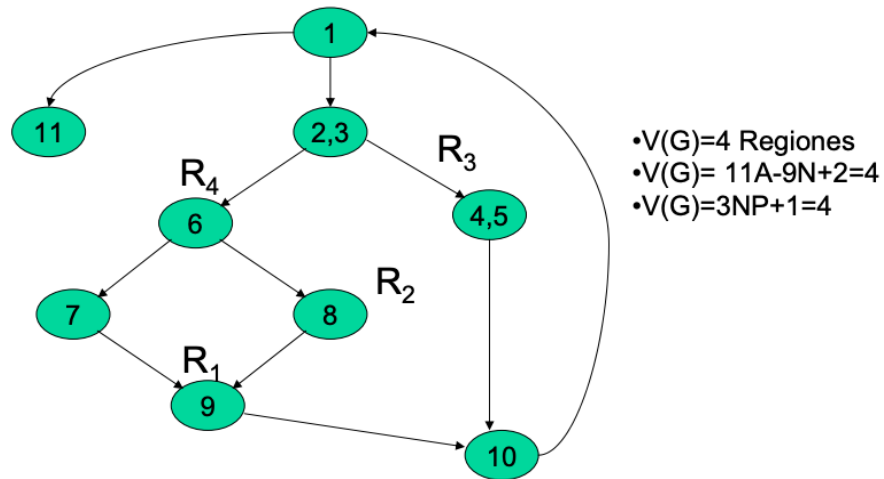
2.2. Complejidad de McCabe o ciclomática.

La **complejidad de McCabe o ciclomática** es una medida del software que aporta una valoración cuantitativa de la **complejidad lógica** de un programa.

A partir del grafo se determina su complejidad ciclomática. Es posible hacerlo por tres métodos diferentes, pero todos ellos han de dar el mismo resultado.

- $V(G) = a - n + 2$, siendo a el número de arcos o aristas del grafo y n el número de nodos.
- $V(G) = r$, siendo r el número de regiones cerradas del grafo (incluida la externa).
- $V(G) = c + 1$, siendo c el número de nodos de condición.

EJEMPLO



2.3. Caminos de prueba

El número de caminos de prueba debe ser igual a la complejidad calculada. Consiste en hacer recorridos desde el inicio hasta el final del método con el que se esté trabajando. Se irán registrando los nodos por los que va pasando la ejecución del camino.

Cada nuevo camino deberá aportar el paso por nuevas aristas/nodos del grafo. La definición de caminos se hará desde los más sencillos a los más complicados.

CAMINOS BÁSICOS:

- Camino 1: 1-11
- Camino 2: 1-2-3-4-5-10-1-11
- Camino 3: 1-2-3-6-8-9-10-1-11
- Camino 4: 1-2-3-6-7-9-10-1-11