PRUEBAS

CODE UD7 – Tema 1

IES Plurilingüe Antón Losada Diéguez

Adrián Fernández González

Tabla de contenido

Pruebas

1. Introducción

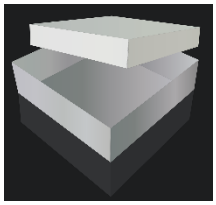
Las pruebas son una parte fundamental en el desarrollo de todo software. Estas suelen realizarse durante la propia creación del sistema para detectar errores y después, para analizar si cumple con lo estipulado, ver su rendimiento y si se conecta correctamente con el hardware sobre el que va a funcionar.

Como es lógico, aunque su creación y uso es algo que conlleva tiempo, es vital detectar y corregir todos los posibles fallos antes de que el sistema llegue a un entorno real y pueda causar daños.

2. Pruebas funcionales

Las pruebas funcionales se utilizan para ver si el sistema hace lo que tiene que hacer y funciona correctamente.

2.1. Caja blanca



Las pruebas de caja blanca son aquellas en las que se conoce el sistema, se sabe cómo funciona y qué lo conforma.

Permiten analizar cada parte del sistema y detectar posibles fallos en cada sección del código.

Son las que se suelen realizar mientras se programa y por parte del propio equipo de desarrollo.

2.1.1 Camino básico

La prueba del camino básico es una de las más habituales de tipo caja blanca. Esta permite analizar todas las posibles rutas que puede tomar un programa dependiendo de los valores de sus variables. Esto asegura una comprobación total del sistema y el poder detectar los caminos más largos, aquellas combinaciones de valores que conllevan un mayor número de líneas de código a ejecutar.

Para ello, se realizan siempre los mismos pasos:

1.- Se crea un grafo del sistema en el que se represente el flujo del mismo, los caminos que toma en base a los bucles y condiciones y el inicio y fin.

2.- Se calcula la complejidad ciclomática del grafo teniendo en cuenta los nodos y las aristas.

Complejidad = aristas – nodos + 2

3.- Se seleccionan un número de caminos posibles igual a la complejidad ciclomática empezando por el más extenso y añadiendo una arista nueva en cada uno.

4.- Se establecen las entradas necesarias para esas rutas y las salidas esperadas.

5.- Se realizan las pruebas de cada camino para verificar que está todo correcto y detectar errores.

2.1.2. Estructuras de control

Estas pruebas se centran en los bucles y las estructuras condicionales exclusivamente, ya que es donde se suelen encontrar la mayoría de los errores.

Son sencillas de realizar y detectan muchos errores, pero no permiten ver más allá de esos bloques de código ni la conexión entre las partes del sistema.

Hay tres tipos fundamentales, de condiciones, de bucles y de flujo.

2.1.2.1. Prueba de condiciones

En esta, se prueban todos los posibles valores de las variables verificando si sigue el camino correcto.

Son similares a las de camino básico, pero con todas las posibles opciones de una única estructura de control.

2.1.2.2. Prueba de bucles

En esta se prueban los bucles en los tres momentos clave, un valor normal, el último valor posible y el valor que rompe el bucle. Por comodidad se toma el penúltimo valor, el último y el valor que hace romper el bucle.

En caso de estar anidados, se haría esto por cada uno de los bucles desde el más interno al más externo.

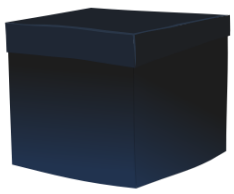
Con solo tres valores se verifica cualquier bucle y el comportamiento en todas sus opciones.

2.1.2.3. Prueba de flujo

Esta prueba se utiliza para probar un flujo en concreto de una parte del sistema para detectar sus posibles fallos.

Una vez establecido el flujo a analizar, se establecen la serie de valores de entrada de las estructuras de control para seguir esa ruta en concreto y así verificar si dicho camino es correcto.

2.2. Caja negra



Las pruebas de caja negra son aquellas en las que no se conoce el funcionamiento interno, simplemente se toman como una caja opaca en la que se conoce qué admite y qué devuelve.

Estas se centran en analizar la funcionalidad, si hace lo que tiene que hacer y cómo se conectan las distintas partes del sistema entre sí para realizar la funcionalidad para la que fueron diseñadas.

Suelen realizarse al terminar una parte del sistema para verificar su adecuación a los requisitos y su integración con el resto. Pueden ser realizadas por el propio equipo de desarrollo o por equipos independientes, ya que no necesitan ver el interior del programa.

2.1.1. Particiones de equivalencia

Estas pruebas están pensadas para minimizar el número de pruebas necesarias para analizar todo el sistema.

Primero se analizan los valores posibles que admite el bloque de código y sus resultados esperados.

Luego, se establecen los conjuntos de valores que devuelven el mismo resultado y se agrupan.

Después, se realiza lo mismo con los no válidos, ya que también hay que verificar cómo reacciona el sistema ante datos erróneos. (Lanzando excepciones, mostrando mensajes de error, etc.)

Una vez establecidos los valores, se prueba uno de cada grupo, de tal forma que se verifican todas las posibilidades con el menor número de pruebas.

2.1.2. Análisis de valores límite

Estas pruebas se basan en la estadística que dice que la mayoría de los errores se encuentran en los primeros valores y los últimos.

Para ello y puesto que se deben probar valores válidos e inválidos, se toman el primer y último valor posible y los anteriores y siguientes a estos, probando así 4 valores válidos y 2 inválidos.

Esto se realiza para cada rango de posibles valores.

3. Pruebas de rendimiento o no funcionales

Las pruebas de rendimiento o no funcionales se usan para ver el rendimiento, para conocer los cuellos de botella, el consumo de recursos y la eficiencia del sistema y sus partes.

Existen múltiples tipos de pruebas, cada una diseñada para probar el sistema desde un punto de vista.

Se denominan también no funcionales porque no permiten ver si una parte del sistema hace lo que tiene que hacer, solo si lo que hace lo hace de forma eficiente.

3.1. Pruebas de carga

Estas pruebas se realizan para verificar que el sistema es capaz de manejar la carga de trabajo para la que fue diseñada.

Una vez establecidos los valores aceptables, se realizan las simulaciones determinadas y se mide el resultado.

Por ejemplo, un sistema ha de soportar 100 usuarios simultáneos con unos tiempos de respuesta máximos de 1s. Establecidos los valores aceptables, se simulan 100 usuarios y se miden los tiempos de respuesta.

3.2. Pruebas de capacidad

Estas pruebas tienen el objetivo de conocer los límites del sistema y qué limita el rendimiento del mismo.

Se suelen realizar incrementando la carga de una tarea de forma secuencial hasta llegar al punto en el que el sistema empieza a salirse de los límites aceptables y así descubrir hasta dónde es capaz de llegar.

Si se realizan unas acciones de forma precisa, se puede conocer qué es lo que limita ese rendimiento.

Por ejemplo, se simulan usuarios realizando una misma acción concreta y se mide el tiempo de reacción. Se va incrementando el número de usuarios hasta que el sistema se sale de los parámetros aceptables. Luego, se realiza la misma simulación, pero con otra acción y así sucesivamente con todas. Una vez realizadas todas, se conocerá el rendimiento de cada acción y aquellas que requieren menos usuarios para colapsar, son las más costosas.

3.3. Pruebas de estrés

Son pruebas en las que se sobrecarga el sistema por encima de sus capacidades para ver cómo reacciona.

3.4. Pruebas de estabilidad

Estas pruebas están pensadas para medir cómo reacciona el sistema ante una carga media durante un largo periodo de tiempo, para detectar posibles pérdidas de eficiencia a largo plazo y posibles errores de consumo de recursos.

Este tipo de pruebas permiten detectar posibles recursos que no fueron liberados

3.5. Pruebas aisladas

Es habitual realizar pruebas del sistema entero, pero en sistemas complejos es importante probar cada parte del sistema de forma independiente, sobre todo si el sistema es grande.

Al probar cada parte, se pueden detectar más fácilmente los cuellos de botella y localizar todo tipo de fallos.

3.6. Pruebas de regresión

Estas pruebas son las que se realizan cuando se actualiza un sistema o se implanta uno nuevo para medir el rendimiento de ambos.

Se realizan en paralelo las mismas pruebas al anterior sistema y al nuevo, comparando los resultados en cada aspecto.

4. Pruebas de usabilidad

Estas pruebas se centran en la interfaz de usuario, en analizar si es amigable, fácil de usar y si los usuarios potenciales la entienden y les parece cómoda.

También permiten detectar las zonas del sistema al que más se accede y cuales tienden a ser ignoradas.

Con todo esto se puede conocer qué partes deberían mejorarse, dónde establecer los menús y secciones más importantes o reestructurar el diseño ineficaz.

5. Crear un diagrama de flujo de un método

Para crear un diagrama de flujo hay que tener en cuenta que cada instrucción conforma un nodo, cada asignación de valores es un nodo e incluso los parámetros de un método son un nodo.

A mayores, siempre hay un nodo inicio y un nodo fin.

Cada paso de una instrucción a otra es una arista y aquellas instrucciones iterativas y alterativas tendrán distintos caminos dependiendo de los valores de la condición.

5.1. Condiciones

Las condiciones, ya sea en una instrucción alterativa o una iterativa, conforman un nodo con una arista por cada opción posible.

En caso de que una condición esté formada por varias operaciones lógicas unidas mediante AND u OR, habrá que tratarlas como condiciones independientes anidadas según el operador que las une.

5.1.1. Condición AND

En una condición AND, cada uno de los operandos de la condición es un nodo y se valorarán en cascada cuando den *true*. Si alguno de ellos es *false*, se saltará al caso contrario directamente, solo si todos son *true* entrará.

5.1.2. Condición OR

En una condición OR, cada uno de los operandos es un nodo y se valorarán en cascada cuando alguno sea *false*. Si alguno es *true*, entra en su interior, si todos son *false*, saltará al caso contrario.

5.2. Instrucciones alterativas

Las instrucciones alterativas tendrán dos ramas si son un *if* o múltiples si son un *switch*.

Con los switch, se creará una rama independiente por cada caso posible teniendo en cuenta que los que estén combinados se tomarán como uno solo y los que no tengan un *break* se enlazarán.

En caso de los if, seguirá las mismas reglas que en las condiciones comentadas anteriormente.

5.3. Bucles

En los bucles, el flujo entra en el cuerpo y vuelve atrás en función de la condición del bucle.

5.3.1. While

En el *while*, la condición es un nodo y el interior otro que estarán conectados de forma cíclica del segundo al primero y es el primero el que saltaría a lo siguiente, tras el bucle, en caso de que la condición no se cumpla.

5.3.2. Do-while

El bucle *do-while* es similar al *while*, con la salvedad de que primero entra en el cuerpo, luego la condición.

5.3.3. For

En el *for*, primero va el nodo de la asignación del valor i, luego el nodo/s de la condición, luego el/los nodo/s del cuerpo y por último el nodo del incremento del valor que volverá al de la condición. Si la condición no se cumple, irá al siguiente nodo tras el bucle.

5.4. Parámetros de entrada

Los propios parámetros de entrada de un método forman un nodo conjunto, ya que la obtención de los mismos es una instrucción.

5.5. Asignaciones y declaraciones

Las asignaciones de valores conforman un nodo y se agrupan si están de forma conjunta.

Las declaraciones se obvian o se estipulan en el mismo nodo que la primera asignación, formando parte del mismo nodo.

5.6. Entradas y salidas

Las entradas y salidas conforman un nodo, pudiendo agruparse si son consecutivos.

5.6. Return

Cada return es un nodo en si mismo, independientemente de si se devuelve directamente un valor o una variable.

5.7. Consideraciones finales

Hay que tener en cuenta que, todos los elementos lineales, es decir, aquellos que no crean ramas, aportan 1 nodo y 1 arista, por lo que da igual el número de ellos que haya, ya que, según la fórmula de la complejidad, a cada arista se le resta un nodo.

Otra forma de calcular esta complejidad es, una vez dibujado el diagrama, contar el número de áreas creadas (zonas rodeadas de aristas) y sumarle uno, dejando más claro la irrelevancia de los elementos lineales.