Técnicas para la verificación, prueba, y documentación de programas

Alberto García Díaz

31 de octubre de 2011

Índice

1. Introducción

Una estrategia de prueba de software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que dan como resultado una correcta construcción del sistema.
Una estrategia de prueba de software proporciona una guía para el desarrollo de sistemas, para la organización del control de calidad y para el cliente.
Cualquier estrategia de prueba debe incorporar:

- Planificación de la prueba.
- Diseño de casos de prueba.
- Ejecución de pruebas.
- Agrupación y evaluación de los resultados.

2. Prueba de programas

2.1. Definición de prueba de programa

Sobre las pruebas de programa podríamos decir que es el proceso de ejecución de un programa para descubrir un error.
El hecho de probar es garantía de calidad, se realiza una revisión final con respecto a los requerimientos iniciales, sobre el diseño y sobre la codificación.
Un buen caso de prueba será aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta el momento.
Una prueba tendrá éxito si descubre un error no detectado hasta entonces.

2.1.1. Principios básicos

Las pruebas las deben realizar personas distintas a las que codificaron el programa.
Los casos de prueba deben ser escritos para entradas válidas e inválidas o no esperadas.
La probabilidad de encontrar errores adicionales en una sección del programa es proporcional al número de errores ya encontrados.

2.1.2. Métodos generales de diseño de pruebas

Tipo Caja Blanca Lo que se hace examinar la estructura interna del programa.

Tipo Caja Negra Los casos de prueba se diseñan considerando únicamente las entradas y las salidas.

2.2. Fases de prueba

Las diferentes fases que el proceso de pruebas ha de seguir en el desarrollo de un sistema de software pueden verse en la siguiente tabla:

Nivel	Fase de prueba
Codificación	Prueba de unidad o unitaria
Diseño	Prueba de integración
Requerimientos	Prueba de validación
Sistema	Prueba de sistema

2.2.1. Pruebas de unidad

Destinada a probar la menor unidad en el diseño de software: el módulo. Se realiza en la fase de codificación.

Estructuras de datos locales Se prueba que los tipos de datos sean consistentes, que los valores estén inicializados, control de los desbordamientos, etc.

Interfaz de módulo Parámetros entre módulos o funciones o métodos. Uso de variables globales, apertura y cierre de ficheros, etc.

Estructuras de control Se prueban los posibles errores en el flujo del programa: Inicializaciones correctas, terminaciones de bucle inesperadas, variables de control de bucle modificados incorrectamente, etc.
Entre las pruebas de caja blanca destaremos la prueba de camino básico y la prueba de bucle.

Manejo de errores Hay que controlar los errores provocados por el usuario final y los producidos eventualmente. En un buen diseño de software hay que describir el error claramente, el mensaje debe estar relacionado con el error.

Prueba de condiciones límite Los errores tienden a darse en los límites del dominio de una estructura, por tanto hay que comprobar que el software responde cuando se provocan condiciones límite.

Eficiencia del código Esto afecta tanto a la velocidad de ejecución como a los requisitos de memoria. La entrada/salida también se ralentiza. Además del buen diseño ayudan las opciones de optimización de los compiladores.

2.2.2. Pruebas de integración

Una vez comprobado cada módulo por separado, la siguiente prueba va dirigida a probar la correcta interconexión de los módulos. Los problemas que pueden aparecer es la modificación de variables globales, efectos laterales percibidos en el paso de parámetros.
Hay dos estrategias para llevar a cabo dichas pruebas:

Integración ascendente Construcción y prueba de los módulos de más bajo nivel.

Integración descendente Primero se realiza el programa principal y luego se van incorporando el resto de módulos. Los módulos de más alto nivel, y por tanto, más utilizados, se probarán más.

Conclusiones La selección de una u otra estrategia dependerá del software.

Inconvenientes de la integración ascendente El programa principal no existe hasta el final del desarrollo y se dedica mucho tiempo a módulos que luego quizás no se aprovechen.

Inconvenientes de la integración descendente Surgen problemas cuando se requiere un módulo inferior para probar adecuadamente los de nivel superior y es difícil encontrar datos de entrada del módulo de nivel más alto que sirvan para probar un módulo inferior.

Muchas veces se opta por una solución mixta.

2.2.3. Pruebas de validación

Son pruebas para comprobar que el software funciona conforme lo exigido por el cliente, esto es, si se ajusta a los requerimientos.
En este paso se llevan a cabo pruebas de caja negra con el fin de detectar:

- Funciones incorrectas.
- Funciones nuevas a tener en cuenta.
- Errores de interfaz.

- Errores en los accesos a ficheros.
- Errores de inicialización y terminación.

Se llevan a cabo dichas pruebas entre el equipo de prueba y los usuarios finales.

2.2.4. Pruebas de sistema

La finalidad es clara, el software entregado debe quedar en perfecto estado de funcionamiento. En este punto se controla:

- Detalles acerca del hardware.
- Se fuerzan errores en el sistema para comprobar recuperaciones.
- Se prueban los mecanismos de protección (claves, seguridad de ficheros, etc.)
- Nivel de rendimiento del sistema.
- Pruebas de resistencia, de estrés, situaciones anormales...

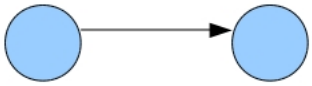
3. Técnicas de pruebas de programas

3.1. Pruebas de camino básico

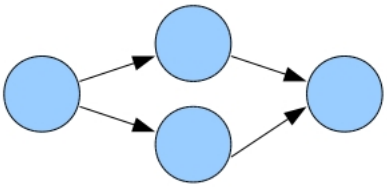
Propuesto por McCabe, se trata de encontrar la complejidad lógica de un módulo; complejidad ciclomática. Para calcularla hay que realizar 4 etapas:

3.1.1. Etapa 1

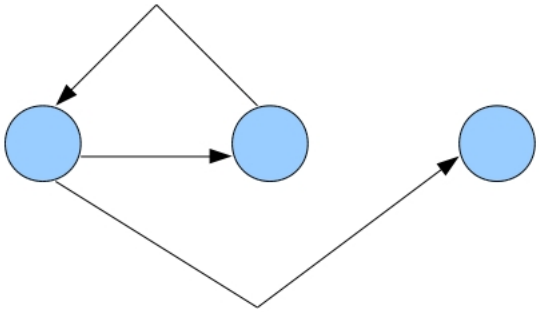
Usando el diagrama o el propio código fuente del programa se genera un grafo.



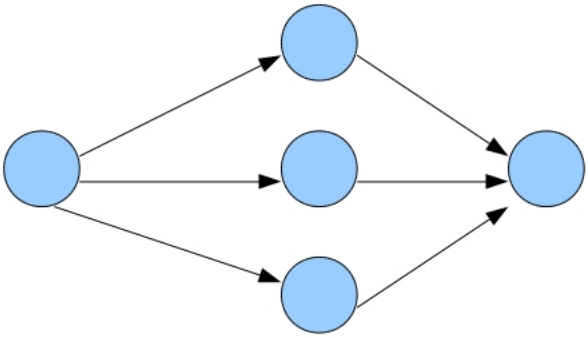
Secuencia



Decisión



Iteración o bucle



Selección múltiple

3.1.2. Etapa 2

Cálculo de la complejidad ciclomática del grafo, que se puede realizar contando las áreas delimitadas cerradas más el área exterior.

3.1.3. Etapa 3

Determinar el conjunto básico de caminos independientes, que es el número de caminos básicos, que además coincide con la complejidad ciclomática del mismo.

3.1.4. Etapa 4

Se preparan las pruebas para la ejecución de los caminos del conjunto básico.

3.2. Pruebas de bucle

Otra técnica de caja blanca para comprobar la validez de las estructuras iterativas de un módulo.

3.2.1. Bucles simples

Si `n` es el número máximo de iteraciones (repeticiones) permitidas, entonces hay que probar:

- Saltarse el bucle.
- Pasar una sola vez.
- Pasar dos veces.
- Pasar `m` con `m<n`.
- Pasar `n-1` y `n+1` veces.

3.2.2. Bucles anidados

Ocurre cuando al menos hay un bucle dentro de otro. En este caso hay que:

- Poner todos los bucles, excepto el más interior a valores mínimos.
- Realizar pruebas de bucle simple en el más interior, manteniendo el resto con valores mínimos.

- Ir a los bucles más exteriores realizando dichas pruebas y dejando los bucles interiores con valores medios.

3.2.3. Bucles concatenados

Un bucle a continuación de otro. Si no hay relación entre ellos, es decir, las variables de control son independientes, aplicar las pruebas de bucle simple. Si sí son dependientes, se aplicarán las pruebas de bucles anidados.

4. Documentación de programas

4.1. Introducción

En el código escrito del programa, deberemos seguir unas normas que tienen como finalidad que nuestro programa sea **claro**, **legible** y **bien documentado**.
Podemos englobar dichas normas en:

4.2. Estilo de escritura

Se aplican las siguientes normas a la hora de codificar:

- Espacios en blanco en la separación de expresiones. Separar mediante líneas en blanco los diferentes bloques de código.
- Uso de paréntesis en expresiones complicadas.
- Separadores visuales entre bloques del programa, módulos, etc.
- Identificadores significativos.
- Evitar complicadas comparaciones condicionales.
- Eliminar comparaciones con condiciones negativas.
- Evitar gran anidamiento de bucles o condiciones.

4.3. Declaraciones de datos

Se deben estandarizar las declaraciones de datos aunque el lenguaje de programación no las tenga o imponga.
Se deben comentar las estructuras de datos complejas y aplicar nombres significativos; ni muy largos ni muy cortos.

4.4. Comentarios

Los comentarios se usarán para la clarificar la estructura de cada módulo, así como la función realizada por cada bloque de instrucciones, las variables, etc.
Los tipos de comentarios son:

4.4.1. De prólogo

Se incluyen al principio de cada módulo, tarea, etc. Describen la interfaz, describen ejemplos de llamadas de función o métodos, describen los argumentos, etc.

4.4.2. Descriptivos

Describen en lenguaje natural el funcionamiento de instrucciones o bloque de ellas. No deben parafrasear el código simplemente.