PRU
Prueba de software: principios y técnicas

Introducción 2
Casos de prueba 4
Principios de la prueba del software 5
Procesos de prueba 7
Tipos de pruebas 8
Técnicas de prueba de caja blanca 9
Técnicas de prueba de caja negra 16
Pruebas de integración 23
Pruebas de "alto nivel" 24
Pruebas de regresión 27
Documentación de las Pruebas 28
Bibliografía 29

Introducción

Definición y Objetivos

Dos definiciones:

1. Proceso de verificar que el programa se ejecuta correctamente.
2. Proceso de ejecución de un programa para encontrar errores

¿Cuál es la correcta?

Objetivos

- Proceso de ejecución de un programa con la intención de descubrir errores
- Un buen caso de prueba es aquel que tiene una alta probabilidad de descubrir un error no encontrado hasta entonces
- Por tanto, una prueba tiene **ÉXITO** si descubre un error no detectado
- La prueba no puede asegurar la ausencia de errores, solamente demostrar que existen

Cuando se realizan las pruebas:

- Al principio, labor “destructiva” hacia el programa que se prueba
- A largo plazo, se mejora la calidad del software, labor “constructiva”

Casos de prueba

Un conjunto de **entradas**, **condiciones de ejecución** y **resultados esperados** desarrollados para un objetivo particular.

- Se **diseñan**: aplicando técnicas (como las de caja blanca y caja negra).
- Se **ejecutan:**
  - el software se ejecuta siguiendo el caso de prueba
  - se comparan las salidas obtenidas con los resultados esperados → se determina si existe error.

Principios de la prueba del software

1 La **salida deseada** es una parte NECESARIA para cada caso de prueba

2 Un programador debe evitar probar su propio programa

Al programar se construye

Al probar se destruye (a corto plazo)

Además: Se descubren mejor los errores debidos al mal entendimiento del diseño o especificación ("dos ojos ven más que uno"). Analogía: Crítica de un libro

No es imposible que el programador pruebe, pero no conveniente

Lo anterior no es aplicable a la depuración

3. Una organización no debe probar sus propios programa (idealista)

Consecuencia: Equipos dedicados a pruebas

Independientes

Especializados

4. Inspeccionar MUCHO **los resultados** de cada prueba

5 Escribir casos de pruebas para

- Entradas **inválidas e inesperadas**
- además de para las **válidas y esperadas**

6 Intentar ver:
   - Si no hace lo que se supone que debe de hacer
   - Si hace lo que no se supone que debe de hacer (Examinar **efectos laterales**)

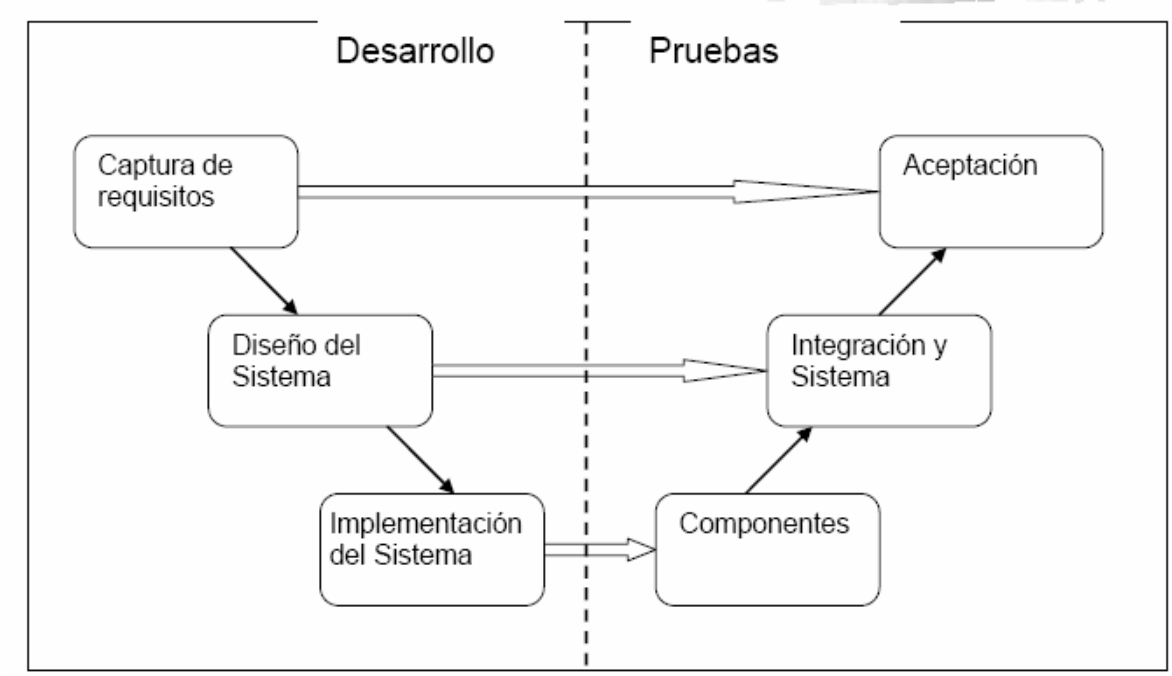
7 Evitar usar caso de prueba de usar y tirar, a no ser que el programa sea de usar y tirar

   Guardar siempre los casos de prueba para:
   - No perder tiempo reinventando casos de prueba
   - Posibilitar efectuar pruebas de regresión

8 No plantear una prueba asumiendo que no se van a encontrar errores

9 La probabilidad de existencia de errores en una sección de programa es proporcional al número de errores ya encontrados en esa sección

Procesos de prueba



Además, cada tipo de pruebas debe:

- Inicialmente, **planificarse** (plan de pruebas).
- Posteriormente **ejecutarse** (informe de pruebas).

Tipos de pruebas

Respecto a la técnica aplicada para su **obtención**.

Pruebas de caja blanca

Conducidas por la estructura lógica (interna) del programa

Problema: Muchos errores quedan sin detectar:

- No se prueba la especificación
- No se detectan ausencias

Pruebas de caja negra

Conducidas por las entradas y salidas

Problema: Infinitas posibilidades para las entradas

Técnicas de prueba de caja blanca

Cobertura

La cobertura mide el grado en que un conjunto de casos de prueba ejercita la estructura del programa.

Varias posibilidades:

- Sentencias
- Decisiones
- Condiciones
- Caminos

Proceso

1. Elegir un nivel de cobertura.
2. Examinar el código e identificar casos de prueba suficientes para garantizar la cobertura elegida.

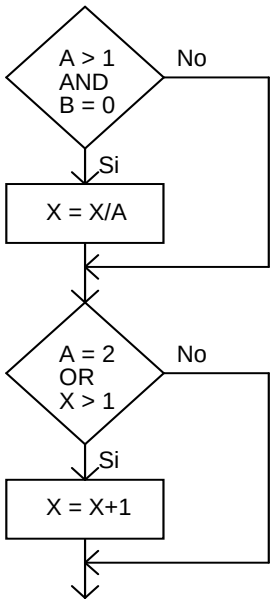
Proceso (variante)

Variante del proceso cuando ya tenemos un plan de pruebas para el programa.

1. Elegir un nivel de cobertura.
2. Examinar si el plan de pruebas existente ya tiene el nivel de cobertura elegido.
3. Si no se garantiza la cobertura, identificar casos de prueba adicionales para garantizarla.

Cobertura de sentencias

Cada sentencia ha de ser ejecutada al menos una vez



Caso de prueba:

A=2, B=0, X=3

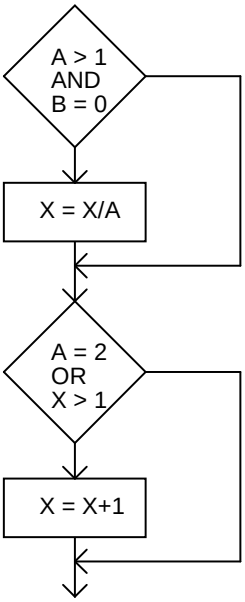
¿Qué pasaría si por error, la primera decisión ha de ser OR?

¿Y si en la segunda ha de ser X>0?

Cobertura de decisiones

Los casos de prueba han de hacer que cada decisión tenga los valores CIERTO y FALSO

Incluye cobertura de sentencias



Casos de prueba:

A=3, B=0, X=3

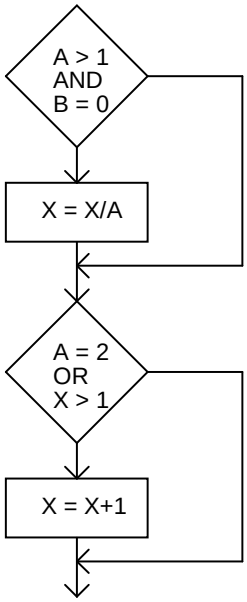
A=2, B=1, X=1

¿Qué pasaría si en la segunda decisión ha de ser x<1 en vez de x>1?

Cobertura de condiciones

Cada condición toma todos los posibles valores al menos una vez

Suele ser superior a las anteriores



A	B	X	A>1	B=0	A=2	X>1
1	0	3	F	C	F	C
2	1	1	C	F	C	F

¿Los casos anteriores incluyen los criterios anteriores?

- Decisiones
- Sentencias

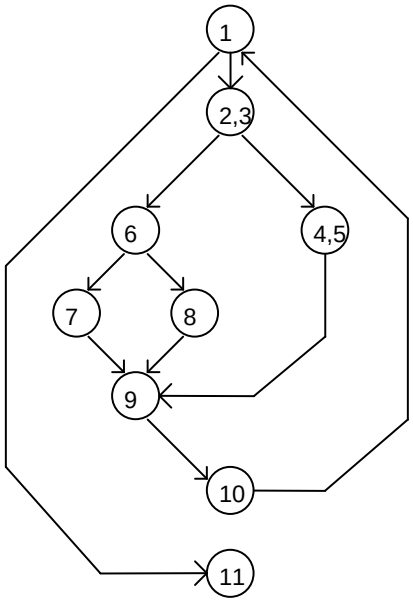
Prueba del camino básico: Complejidad ciclomática

Objetivo

Recorrer todos los caminos independientes

El número de caminos lo indica el valor de la complejidad ciclomática (McCabe)

Grafo:

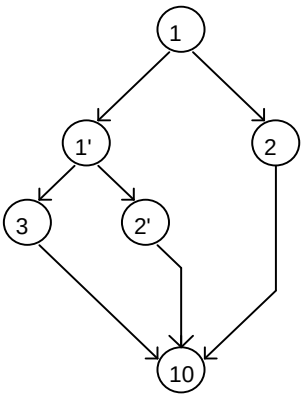



```
 1  WHILE NOT final DO
 2    leer
 3    IF campo1=0 THEN
 4      procesar
 5      incrementar conta.
 6    ELSE IF campo1=1 THEN
 7      reinic. conta.
      ELSE
 8      procesar
 9    END IF
10  END WHILE
11
```

Cálculo

Fórmula	V(G)
Número de regiones del grafo	4
Aristas - Nodos + 2	11 - 9 + 2
Número Predicados + 1	3 + 1

Cuando hay más de una condición por decisión, cada una de ellas cuenta como un predicado (nodo)



V(G) = 3

```
1   IF a OR b THEN
2     x
    ELSE
3     y
4   END IF
```

```
1   IF a THEN
2     x
1' ELSE IF b THEN
2'    x
    ELSE
3     y
4   END IF
```

Excepción

Sentencias CASE (switch)

Utilidad

- No solamente para determinar el número de caminos.
- Mide la CALIDAD del código

  Un criterio de calidad de uso general:

  V(G) < 10 Módulo aceptable

  V(G) >=10 Módulo rechazado

- Puede usarse como criterio de selección de casos de prueba: al menos uno por camino.

Importante

- Fácil uso en una organización
- Fácilmente automatizable

Técnicas de prueba de caja negra

Se centran en los requisitos funcionales del software a probar

Básicamente, solamente se conocen las Entradas y Salidas

Partición en clases de equivalencia

Consisten en particiones (clases de equivalencia) de las entradas identificando grupos donde se pueden encontrar errores

Los grupos de entradas se clasificarán en VÁLIDOS e INVÁLIDOS

Para cada grupo identificado, se seleccionan uno o varios casos de prueba (ver a continuación)

Proceso

1. Identificar y numerar cada clase
2. Escribir el mínimo número de casos de prueba para cubrir todas las clases VÁLIDAS
3. Escribir un caso de prueba para CADA UNA de las clases INVÁLIDAS (objetivo: evitar el enmascaramiento de los casos de pruebas)

Identificación de clases de equivalencia

Tipo de Condición	Clases Válidas	Clases inválidas
Rango de valores (1 .. 999)	1 .. 99	<1, >999
Enumeración de valores (de uno a seis)	1 .. 6	<1, >6
Conjunto (rojo, verde)	uno por cada valor (rojo, verde)	uno por un valor no válido (gris)
Condición booleana (debe ser letra)	Condición cierta (letra)	Condición no cierta (dígito)

Si hay razones para creer que los elementos de una clase no se tratarán de la misma forma, dividir la clase en otras más pequeñas

Ejemplo

Chequeo sintáctico de una sentencia de declaración

INTEGER a [,a]

- cuatro variables como máximo
- identificador de 6 caracteres máximo
- el primer carácter una letra

Clases de equivalencia

Entrada	Clases Válidas	Clases Inválidas
Nombre Variable (contenido)	(1) Tiene letras (2) Tiene dígitos	(3) Tiene otros caracteres
Existe variable	(4) SI	(5) NO
Comienza por letra	(6) SI	(7) NO
Tamaño nombre	(8) 1 .. 6	(9) 0 (10) >6
Número de variables	(11) 1 .. 4	(12) 0 (13) >4

Casos de prueba

Entrada	Clases cubiertas
INTEGER ABC,A19,B	1, 2, 4, 6, 8, 11
INTEGER	5
INTEGER 213	7
INTEGER A?3	3
INTEGER ,A	9
INTEGER ABCDEFG	10
INTEGER	12
INTEGER A,B,C,D,E	13

Análisis de valores límite

Objetivo

Ejercitar las pruebas en los LÍMITES ó BORDES de las clases de equivalencia

Diferencia con la anterior

- En vez de seleccionar cualquier elemento de la clase se seleccionan varios (los límites)
- Además se tienen en cuenta las clases de equivalencia para las salidas

Determinación de valores límite

- Valores situados en los bordes
- Adyacentes a los situados en los bordes
- Valor típico

Tipo de Condición	Clases Válidas	Clases inválidas
Rango (-5 .. +5)	+5, +4, +2, -4, -5	+10, +7, +6, -6, -7, -10
Rango (-5.000 .. + 5.000)	+5.000, +4.999, +2.345, -4.999, -5.000	+12.345, +5.002, +5.001, -5.001, -5.002, -11.456
Enumeración de valores (de uno a diez)	1, 2, 5, 9, 10	0, 11, 12, 17
Conjunto ordenado (rojo, verde)	uno por cada valor (rojo, verde)	uno por un valor no válido (gris)
Condición booleana (debe ser letra)	Condición cierta (letra)	Condición no cierta (dígito)

Ejemplo

Para simplificar se tomarán solamente los bordes y un valor típico

Clases de equivalencia + valores límite

Entrada	Clases Válidas	Clases Inválidas
Nombre Variable (contenido)	(1) Una letra (2) Todo letras (3) Un dígito (3') Todo-1º dígitos	(5) Un carácter inválido (6) Todos inválidos (6') Todos dígitos
Existe variable	(7) SI	(8) NO
Comienza por letra	(9) SI	(10) NO
Tamaño nombre	(11) 1, (12) 4 (13) 6	(14) 0, (15) 7
Número de variables	(16) 1, (17) 4	(18) 0, (19) 5

Para las salidas: Devuelve un código de error

1. Sintaxis Correcta
2. Faltan Parámetros
3. Sobran Parámetros
4. Identificador muy largo
5. Identificador ilegal

Entrada	Clases válidas	Clases Inválidas
Código de salida	1,2,3,4,5	otro

Casos de prueba

Entrada	Casos cubiertos
INTEGER A21,AAA,A2	1, 2, 3, 7, 9, 3'
INTEGER A,AAAA,ABCDEF	11,12,13
INTEGER BCD	16
INTEGER B,C,D,E	17
INTEGER AB?C	5
INTEGER ?$@	6
INTEGER 12345	6'
INTEGER	8,14,18
INTEGER 5B1	10
INTEGER ABCDEFG	15
INTEGER AB,CD,E,GH,IJ	19

Se comprueba además que se cubren todas las clases válidas para las salidas

Intentar forzar la clase inválida de las salidas:

```
Linea = "INTEGER ABC,DEF"
CodigoSalida = -99
CheckSintaxis(linea,CodigoSalida)
Write(Linea)
Write(CodigoSalida)
```

Estrategias para las pruebas

- Rechazar o someter a inspección los módulos con valores de complejidad ciclomática alta
- Usar siempre análisis de valores límite
- Especial cuidado con las clases inválidas (sin olvidar las válidas)
- Añadir casos "por instinto"
- Determinar al final la cobertura lógica, añadir casos si queda algún camino por cubrir

Importante

Guardar los casos de prueba y las salidas obtenidas para facilitar las pruebas de regresión

Pruebas de integración

Realizadas sobre componentes previamente probados por separado

- Existen infinitas formas de implementar y probar de forma organizada
- Lo peor es hacerlo de forma desorganizada

Estrategia no incremental

- Probar cada módulo por separado
- Juntar todo
- Esperar que funcione

Problema: Cuando falla algo, ¿cómo se sabe dónde está el fallo?

Estrategia incremental

- Probar un módulo por separado
- Añadir otro
- Probar y depurar la combinación
- Repetir con otros módulos

De esta forma se puede detectar dónde están los fallos encontrados

Esto no excluye que se pruebe previamente cada módulo por separado

Requiere un plan de integración

Pruebas de "alto nivel"

Pruebas alfa

Conducidas por cliente en lugar de desarrollo

El equipo registra las anomalías

Pruebas beta

Se distribuye el producto a potenciales usuarios

Estos prueban el producto e informan de los fallos detectados

Pruebas de aceptación

El cliente certifica que el sistema es válido para él

Utilizar el plan de aceptación existente

Pruebas de sistema

Un tipo de pruebas de “alto nivel”: tiene en cuenta los objetivos: El producto formando un todo con su entorno

Usabilidad

- Los interfaces de usuario son adecuados a las características del usuario?
- Las salidas y diagnósticos son claros y comprensibles?
- El conjunto del interface de usuario mantiene la integridad conceptual?
- Excesivo número de opciones, u opciones que no serán utilizadas?
- Hay reconocimiento inmediato de cada entrada de datos?
- Es fácil de usar?

Seguridad

Security: Intentar violar toda protección del sistema

Safety: Intentar descubrir si el sistema puede afectar al entorno (importante en sistemas de control)

Volumen

Someter a gran cantidad de datos (base de datos, tablas muy grandes)

Probar con cantidades de datos no permitidas (p.e. disco lleno)

Rendimiento

Tiempos de respuesta

- En condiciones normales
- En las peores condiciones

Configuración

Al menos, probar con cada posible configuración hardware y dispositivos

Recuperación

Cómo se recupera el sistema ante fallos

- Software
- Hardware
- Comunicaciones

Documentación

Mediante inspección. Buscar discrepancias con el funcionamiento del sistema

Pruebas de regresión

Notificación y eliminación de errores:

1. Cuando se localiza un fallo, se documenta y se pasa al programador para que lo solucione
2. Una vez fijado el error, se modifica el código para eliminar la situación descrita
3. Generalmente no se vuelve a probar el programa modificado
4. Consecuencia: El software tiene los errores que tenía MÁS los derivados de la modificación

Solución: Cada vez que se modifica el software se deben de ejecutar otra vez todas las pruebas.

Este proceso se denomina Pruebas de Regresión.

Documentación de las Pruebas

Plan de Pruebas

Información detallada sobre los diferentes casos de prueba (planificados) para su posterior ejecución en el sistema:

- Tipos de pruebas planificadas
- Entorno operativo de ejecución de las pruebas
- Elementos auxiliares (p.e., ficheros, tablas, etc.) necesarios para la ejecución de los casos de prueba
- Responsables de ejecución
- Para cada caso de prueba:
    - Identificación y objetivo
    - Descripción detallada de las entradas
    - Modo de ejecución
    - Descripción detallada de las salidas esperadas

Informe de Pruebas

Información acerca de la ejecución de los casos de prueba contenidos en el Plan de Pruebas.

Para cada caso de prueba especificar la salida obtenida, y en caso de ser diferente a la esperada, documentar el error encontrado con el mayor nivel de detalle posible.

Estándares:

"IEEE 829-1998 for Software Test Documentation” [IEEE,98]

Bibliografía

- "The Art of Software Testing", Myers, G.J.

  John Wiley & Sons, 1979
- "Black-Box Testing. Techniques for Functional Testing of Software and Systems" Beizer, B.

  Ed.Wiley 1995
- "Testing Computer Software (2nd edition)" Kaner, C., Falk, J., Nguyen, H. Q

  Ed. Wiley, 1999
- "Lessons Learned in Software Testing" Kaner, C., Bach, J., Pettichord, B.

  Ed. Wiley, 2002
- "Software Testing" Patton, R.

  Ed SAMS 2001
- “IEEE Standard 829-1998 for Software Test Documentation”, accesible online en la url “http://standards.ieee.org"