Introducción a la Algorítmica y Programación (3300)

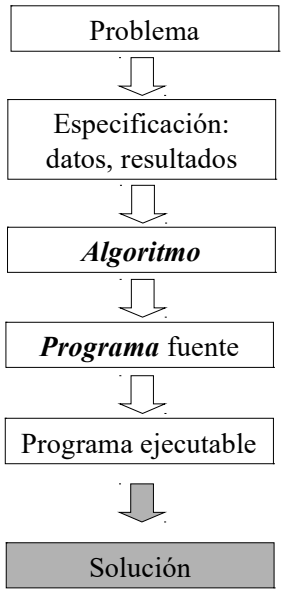
Prof. Ariel Ferreira Szpiniak - aferreira@exa.unrc.edu.ar
Departamento de Computación
Facultad de Cs. Exactas, Fco-Qcas y Naturales
Universidad Nacional de Río Cuarto

Teoría 11

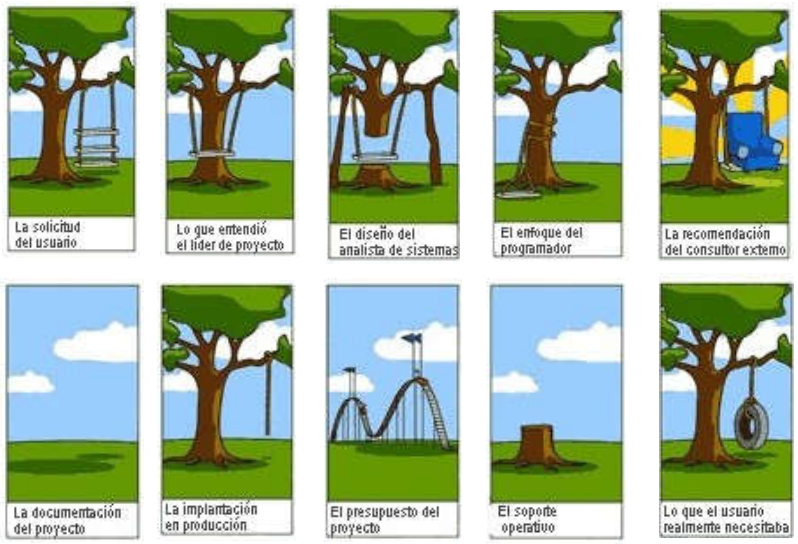
Ejecución y Prueba

Pasos para solucionar un problema

- **Análisis**
  El problema debe ser claramente especificado y entendido.
- **Diseño**
  Construcción de una solución general del problema y **chequeo de esa solución mediante la prueba de escritorio.**
- **Implementación**
  Traducción del algoritmo a un lenguaje de programación de alto nivel.
- **Compilación**
  Escritura del programa Pascal, usando, en nuestro caso, algún Compilador de Pascal.
- ***Ejecución y Prueba***
  Corrida y **prueba de funcionamiento del programa en la computadora.**



¿Por qué es importante respetar las etapas de resolución de un problema?



Retomamos los pasos para solucionar un problema para analizar en detalle la última etapa

Ejecución y Prueba

Corrida y prueba de funcionamiento del programa en la computadora.

*Ahora que ya conocemos la forma para lograr una solución modular a un problema usando acciones y funciones, veremos con un poco más de detalle como realizar las **pruebas** tanto de **módulos** de programa como de **programas** completos.*

*La prueba de un programa, o parte del mismo, es un chequeo más exhaustivo del que realizamos sobre el **algoritmo** con la prueba de **escritorio**.*

Ejecución y Prueba
Introducción

- Cuando ejecutamos el programa debemos realizar la prueba del mismo para saber si hace lo que realmente nos pidieron que haga.
- Se estima que la mitad del esfuerzo de desarrollo de un programa, tanto en tiempo como en costo, se invierte en la etapa de prueba.
- Especialistas en Ingeniería de Software estiman que por cada hora que se le dedica a la prueba (testing) de un programa (sistema) se ahorran diez horas de soporte o mantenimiento, además de ganar en calidad del producto. Cada error detectado y corregido supone un importante descenso en el tiempo de corrección de un programa tiempo después de haberlo entregado al usuario final.

Ejecución y Prueba
Introducción

¿Qué es probar?

- Es muy común que los programas tengan errores.
- El objetivo específico de la prueba es encontrar la mayor cantidad de errores posibles.
- La prueba puede mostrar la presencia de errores, pero nunca su ausencia (E. W. Dijkstra).
- Probar un programa es usarlo con la peor intención a fin de encontrarle fallas (G. J. Myers).

Ejecución y Prueba
Introducción

Prueba Exhaustiva

- Probar un programa es someterlo a todas las posible variaciones de los datos de entrada, tanto si son válidos como si no lo son.
- La prueba ideal es imposible. Es impensado probar el programa para todos los casos posibles.
- Hay que buscar alternativas que sean aceptables para encontrar errores.
- Para ello se hace la prueba o testeo del programa (testing).
- Hay que elegir bien los casos de prueba.

Ejecución y Prueba
Fases de prueba

La prueba de programas puede dividirse en diferentes etapas denominadas: **prueba de unidad**, **prueba de integración** y **prueba de aceptación**.

- **Prueba de unidad:** se plantea a pequeña escala, y consiste en ir probando uno a uno los diferentes módulos que constituyen un programa.
- **Prueba de integración:** se centran en probar la coherencia semántica entre los diferentes módulos. Se plantea a mayor escala, que puede llegar a dimensiones industriales cuando el número de módulos es muy elevado, o la funcionalidad que se espera del programa es muy compleja.
- **Prueba de aceptación:** es realizada por el usuario sobre la base del sistema completo.

Ejecución y Prueba

Prueba de Unidad

- **Fase informal**: la lleva a cabo el propio codificador en su escritorio, y consiste en ir ejecutando el código para convencerse de que funciona. De allí que ese tipo de pruebas reciban el nombre de **pruebas de escritorio**. Esta fase suele consistir en pequeños ejemplos que se intentan ejecutar.

- **Fase sistemática**: se empieza a buscar fallos siguiendo algún criterio: **caja negra o caja blanca**.
  - **Caja negra:** cuando prescinde de los detalles del código y se limita a lo que se ve desde el exterior. Intenta descubrir casos y circunstancias en los que el módulo no hace lo que se espera de él. Por ello se apoyan fuertemente en la especificación. Se buscan clases de equivalencia para reducir las cantidad de casos de prueba.
  - **Caja blanca:** cuando se analiza el código minuciosamente y se intenta que falle.

Ejecución y Prueba

Caja Blanca

- También llamadas pruebas estructurales. Con ánimo de "probarlo todo".

Cobertura: medida porcentual del código que hemos cubierto con la prueba.

Tipos de cobertura:

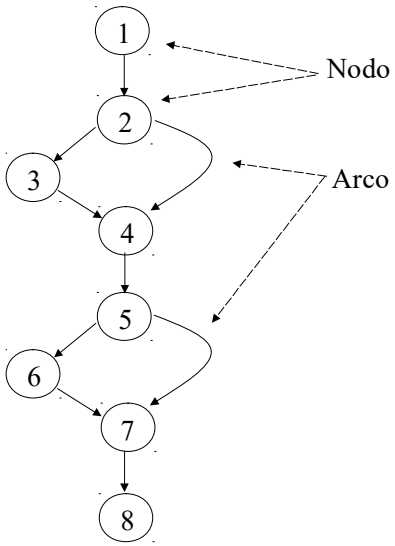
- **Cobertura de sentencias**: recorrer la mayor cantidad de sentencias del programa.
- **Cobertura de ramas o decisión**: recorrer todas las posibles salidas de los puntos de decisión.
- **Cobertura de condición**: analizar todas las situaciones posibles en condiciones lógicas complejas.
- **Cobertura de caminos**: se intenta cubrir todos los caminos del grafo de flujo de control.

Ejecución y Prueba

Caja Blanca

```
Algoritmo EjemploPrueba
Lexico
  a, b, x ∈ Z
Inicio
  Entrada:a b x        (1)
  si (a>-1) y (b=0)    (2)
  entonces
    x ← x/a            (3)
(4) fsi
  si (a=2) o (x>1)     (5)
  entonces
    x ← x+1            (6)
(7) fsi
  Salida:x             (8)
Fin
```

Grafo de flujo de control

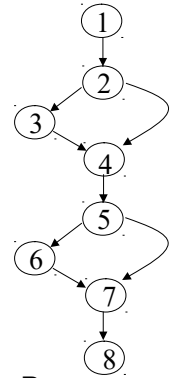


Ejecución y Prueba

Caja Blanca

Cobertura de sentencias

Como el número de sentencias de un programa es finito, se puede diseñar un plan de pruebas que vaya ejecutando más y más sentencias, hasta que hayamos pasado por todas, o por una inmensa mayoría. En engañosa en el caso de sentencias condicionales e iterativas.



- Todas las sentencias del programa deben ejecutarse
- **Equivale a cubrir todos los nodos del grafo de flujo de control**.
- En el ejemplo, debemos cubrir los nodos 1,2,3,4,5,6,7 y 8
- Por ejemplo, con el siguiente test lo logramos:
  - Test 1: a=2, b=0, x=1
- Es importante destacar que en el caso de condicionales **si entonces** (sin **sino**), la cobertura de sentencia puede ser conseguida sin ejecutar el **si** en falso ni una sola vez. Pero en un **si entonces sino**, hay que cubrir el caso verdadero y el falso.

Ejecución y Prueba

Caja Blanca

Cobertura de ramas o decisión

Refinamiento de la cobertura de sentencias.

Consiste en recorrer todas las posibles salidas de los puntos de decisión. Por ejemplo, en una sentencia condicional de 2 ramas, para conseguir una cobertura del 100% hay que ejecutar al menos 2 veces, una satisfaciendo la condición, y otra no.

Si lográramos una cobertura de ramas del 100%, esto llevaría implícita una cobertura del 100% de sentencias, pues toda sentencia está en alguna rama.

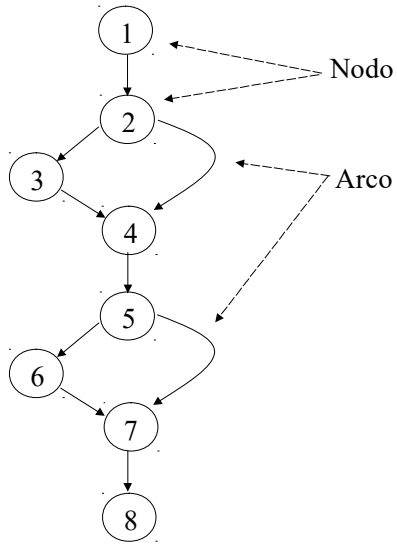
- Todas las decisiones deben satisfacerse al menos una vez por Verdadero y una vez por Falso.
- **Equivale a cubrir todos los arcos del grafo de flujo de control** en lugar de todos los nodos.

Ejecución y Prueba

Caja Blanca

```
Algoritmo EjemploPrueba
Lexico
  a, b, x ∈ Z
Inicio
  Entrada:a b x
  si (a>-1) y (b=0)
  entonces
    x ← x/a
  fsi
  si (a=2) o (x>1)
  entonces
    x ← x+1
  fsi
  Salida:x
Fin
```

Grafo de flujo de control



Ejecución y Prueba

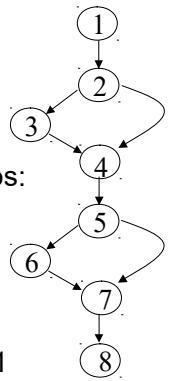
Caja Blanca

Cobertura de ramas o decisión

- En el ejemplo, debemos cubrir todos los arcos
- Por ejemplo, con los siguientes tests:

  Test 1: a=0, b=0, x=0 / Test 2: a=-2, b=0, x=2
- Ejecutamos los siguientes caminos, que cubren todos los arcos:

  Camino 1: 1,2,3,4,5,7,8 / Camino 2: 1,2,4,5,6,7,8
- Problema: una decisión puede estar compuesta de varias condiciones
  - Test 1: a=0, b=0, x=0. Condiciones: a>-1, b=0, a<>2, x<=1
  - Test 2: a=-2, b=0, x=2. Condiciones: a<=-1, b=0, a<>2, x>1

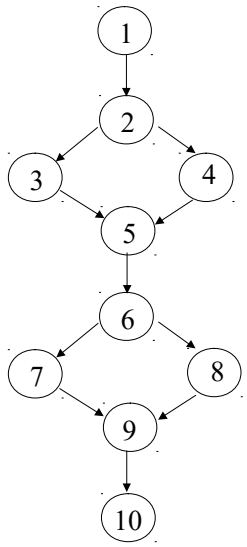


Ejecución y Prueba

Caja Blanca

```
Algoritmo EjemploPrueba2
Lexico
  a, b, x ∈ Z
Inicio
  Entrada:a b x
  si (a>-1) y (b=0)
  entonces
    x ← x/a
  sino
    x ← x/b
  fsi
  si (a=2) o (x>1)
  entonces
    x ← x+1
  sino
    x ← x+b
  Fsi
  Salida:x
Fin
```

Grafo de flujo de control



Ejecución y Prueba

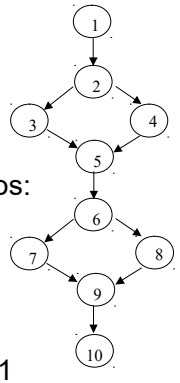
Caja Blanca

Cobertura de ramas o decisión

- En el ejemplo, debemos cubrir todos los arcos
- Por ejemplo, con los siguientes tests:

  Test 1: a=0, b=0, x=0 / Test 2: a=-2, b=0, x=2
- Ejecutamos los siguientes caminos, que cubren todos los arcos:

  Camino 1: 1,2,3,5,6,8,9,10 / Camino 2: 1,2,4,5,6,7,9,10
- Problema: una decisión puede estar compuesta de varias condiciones
  - Test 1: a=0, b=0, x=0. Condiciones: a>-1, b=0, a<>2, x<=1
  - Test 2: a=-2, b=0, x=2. Condiciones: a<=-1, b=0, a<>2, x>1



Ejecución y Prueba

Caja Blanca

Cobertura de condición

La cobertura de ramas o decisión puede tener problemas en el caso de sentencias condicionales con expresiones complejas.

```
si (condicion1 o condicion2) entonces
  acciones1
sino
  acciones2
fsi
```

Alcanzan 2 pruebas para cubrir las 2 ramas. Pero no estamos considerando otras combinaciones posibles de condicion1 y condicion2.

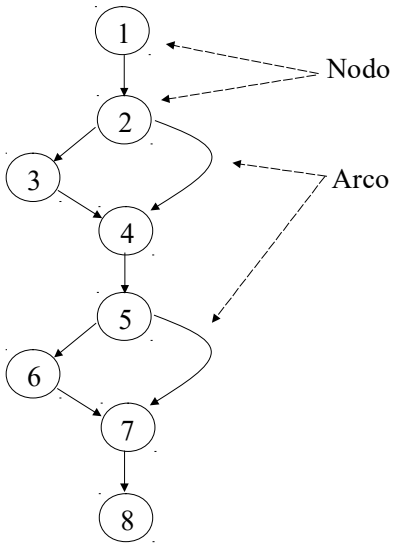
El criterio de cobertura de condición divide las expresiones lógicas complejas en sus componentes, e intenta cubrir todos los posibles valores de cada uno de ellos.

Ejecución y Prueba

Caja Blanca

```
Algoritmo EjemploPrueba
Lexico
  a, b, x ∈ Z
Inicio
  Entrada:a b x
  si (a>-1) y (b=0)
  entonces
    x ← x/a
  fsi
  si (a=2) o (x>1)
  entonces
    x ← x+1
  fsi
  Salida:x
Fin
```

Grafo de flujo de control



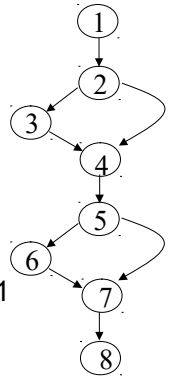
Ejecución y Prueba

Caja Blanca

Cobertura de condición

- Todas las condiciones, de todas las decisiones, deben satisfacerse al menos una vez por Verdadero y una vez por Falso.
- Para satisfacer condiciones hay que tener en cuenta:

  (a>-1) (a<=-1) (b=0) (b<>0) (a=2) (a<>2) (x>1) (x<=1)
- Por ejemplo:
  - Test 1: a=0, b=0, x=0. Condiciones: a>-1, b=0, a<>2, x<=1
  - Test 2: a=-2, b=0, x=2. Condiciones: a<=-1, b=0, a<>2, x>1
  - Test 3: a=2, b=1, x=0. Condiciones: a>-1, b<>0, x<=1, a=2

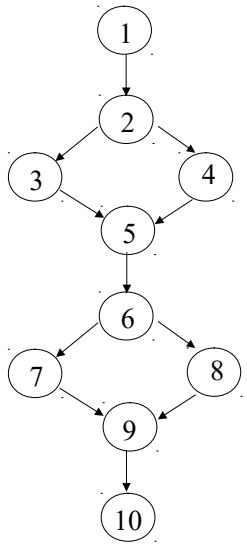


Ejecución y Prueba

Caja Blanca

```
Algoritmo EjemploPrueba2
Lexico
  a, b, x ∈ Z
Inicio
  Entrada:a b x
  si (a>-1) y (b=0)
  entonces
    x ← x/a
  sino
    x ← x/b
  fsi
  si (a=2) o (x>1)
  entonces
    x ← x+1
  sino
    x ← x+b
  Fsi
  Salida:x
Fin
```

Grafo de flujo de control



Ejecución y Prueba

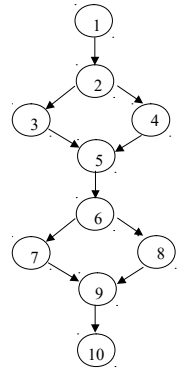
Caja Blanca

Cobertura de condición

- Todas las condiciones, de todas las decisiones, deben satisfacerse al menos una vez por Verdadero y una vez por Falso.

- Para satisfacer condiciones hay que tener en cuenta:
  (a>-1) (a<=-1) (b=0) (b<>0) (a=2) (a<>2) (x>1) (x<=1)

- Por ejemplo (completar los Test):
  - Test 1:
  - Test 2:
  - ….
  - ….
  - ….
  - ….

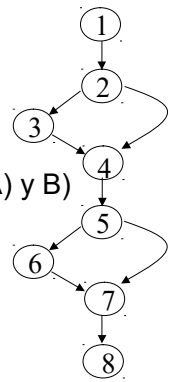


Ejecución y Prueba

Caja Blanca

Relación entre Cobertura de ramas (o decisión) y de condición

- Consideremos el siguiente si: **si** (A y B) …
  - Ramas: (A y B), no(A y B)
    - Puede faltar no(A) o no(B)
    - Ramas no implica condiciones
  - Condiciones: A, no(A), B, no(B) es decir: (A y no(B)) (no(A) y B)
    - Puede faltar (A y B) (no(A) y no(B))
    - Condiciones no implica ramas
- Ninguno garantiza el otro.
- Lo mismo sucede entre sentencias y condiciones



- Para superar este inconveniente se pueden integrar los dos tipos de cobertura, conocida como **cobertura de decisión/condición** o **cobertura de ramas/condición**.

Ejecución y Prueba

Caja Blanca

Cobertura de caminos

Se intenta cubrir todos los caminos del grafo de flujo de control. Lo ciclos dificultan esta tarea. Por tal motivo, en muchos casos se busca que cada ciclo se ejecute 0, 1 o más veces como mínimo (depende el tipo de ciclo). Cobertura de caminos es muy nombrado desde el punto de vista teórico, luego en la práctica se usa algún criterio que acote el mismo.
Las coberturas anteriores son insuficientes para los ciclos dado que un ciclo se ejecuta un cierto número de veces.

Para un ciclo de tipo **mientras** hay que pasar 3 pruebas

1. 0 ejecuciones
2. 1 ejecución
3. más de 1 ejecución (en caso de ser posible se puede buscar un caso promedio)

Ejecución y Prueba

Caja Blanca

Cobertura de caminos

Para un ciclo de tipo **repetir** o **iterar** hay que pasar 2 pruebas

1. 1 ejecución
2. más de 1 ejecución (en caso de ser posible se puede buscar un caso promedio)

En el caso de los ciclos **para** hay que pasar 2 pruebas

1. 0 ejecuciones
2. Cantidad total de ejecuciones.

   Nota: Como está definido el número de veces que se va a ejecutar, son más simples de analizar. Pero hay que examinar su contenido para descartar problemas como: alterar la variable de control, o el valor de alguna variable que se utilice en el cálculo del incremento o del límite de iteración.

Ejecución y Prueba

Caja Blanca

Aplicación práctica

- Se suele procurar alcanzar una cobertura cercana al 100% de sentencias. Es muy recomendable, aunque más costoso, conseguir una buena cobertura de ramas y de caminos.
- La ejecución de pruebas de caja blanca puede llevarse a cabo con un depurador (que permite le ejecución paso a paso), un listado del módulo, un conjunto de pruebas y un marcador para ir señalando por dónde vamos pasando.

Ejecución y Prueba

Caja Negra

- También llamadas pruebas funcionales.
- Se centran en lo que se espera de un módulo, es decir, intentan encontrar casos en que el módulo no respecta su especificación.
- El testeador se limita a suministrarle datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el módulo por dentro.
- Se apoyan en la **especificación** del módulo.
- El problema suele estar en los datos que se utilizan dado que el conjunto de datos posibles suele ser muy amplio.
- Una técnica algebraica utilizada es la de "clases de equivalencia".

Ejecución y Prueba

Caja Negra

Clases de equivalencia

- No existe una regla de aplicación universal, pero hay recetas:
  - Parámetros de entrada:
    - para un parámetro de entrada que deba estar comprendido en un cierto rango o que requiera un valor concreto; aparecen 3 clases de equivalencia: por debajo, en y por encima del rango.
    - para un parámetro de entrada que requiera un valor de entre los de un conjunto o un valor lógico; aparecen 2 clases de equivalencia.
  - Parámetros de salida: los mismos criterios que se aplican a los datos de entrada. Hay que intentar generar resultados en todas y cada una de las clases de equivalencia de los resultados.

Ejecución y Prueba

Caja Negra

Clases de equivalencia

- Ejemplo: utilizamos un entero para identificar el día del mes. Los valores posibles están en el rango [1..31].
  Hay 3 clases:
  1. números menores que 1
  2. números entre 1 y 31
  3. números mayores que 31

- Un buen número de errores aparecen en torno a los puntos de cambio de clase de equivalencia denominados "**frontera**" (o valores límite). Conviene probar usualmente con 2 valores por frontera: uno justo abajo y uno justo arriba.

Ejecución y Prueba

Prueba de integración

- Analiza la coherencia semántica entre los diferentes módulos, tanto estática como dinámica.
- Se va realizando por etapas, englobando progresivamente más módulos.
- Se pueden empezar en cuanto tenemos unos pocos módulos.
- En un diseño descendente (top-down) se empieza a probar por los módulos más generales; mientras que en un diseño ascendente (botton-up) se empieza a probar por los módulos de base.
- Diseño descendente:
  - Ventaja: estar pensando en términos de la funcionalidad global.
  - Desventaja: para cada prueba hay que "inventarse" algo sencillito que simule el papel de los módulos inferiores, que aún no están disponibles.
- Diseño ascendente:
  - Ventaja: evita tener que escribir módulos ficticios
  - Desventaja: se centra en el desarrollo más que en las expectativas del cliente.

Ejecución y Prueba

Prueba de integración

- Estas pruebas se pueden plantear desde un punto de vista **estructural** o **funcional**.

- **Pruebas estructurales**: similares a caja blanca pero a un nivel conceptual superior. Se refieren a llamadas entre módulos. Se trata de identificar todos los posibles esquemas de llamadas para lograr una buena cobertura.

- **Pruebas funcionales de integración**: similares a caja negra. Tratan de encontrar fallos en la respuesta de un módulo cuando su operación depende de los servicios prestados por otro módulo.
- En todas las pruebas funcionales se siguen utilizando las técnicas de partición en clases de equivalencia y análisis de casos frontera.

Ejecución y Prueba

Prueba de aceptación

- Las realiza el cliente.
- Son básicamente pruebas funcionales sobre el sistema completo,
- Buscan una cobertura de la especificación de requisitos y del manual del usuario.
- Muchas veces se aplican las técnicas denominadas **pruebas alfa** y **pruebas beta**.
  - **Prueba alfa**: se invita al usuario al lugar de desarrollo y se desarrolla una prueba en un entorno controlado, con un experto al lado para ayudarle y analizar los resultados.
  - **Prueba beta**: se hace luego de la prueba alfa. Se desarrollan en el lugar de trabajo del usuario. El usuario queda a solas con el producto y trata de encontrarle fallos que luego informa a los desarrolladores.
    Estas pruebas se usan cuando el producto se va a vender a muchas clientes.

Ejecución y Prueba

Otros tipos de pruebas

- **Recorridos (walkthroughs)**

Consiste en sentar alrededor de una mesa a los desarrolladores y a una serie de revisores que analizan el diseño del sistema, leen el programa línea a línea y piden explicaciones de todo lo que no está claro.

- **Aleatorias (random testing)**

Se basa en la hipótesis de que la probabilidad de descubrir un error es prácticamente la misma si se hacen una serie de pruebas aleatoriamente elegidas, que si se hacen siguiendo las instrucciones dictadas por criterios de cobertura (caja negra o blanca).

- **Solidez (robustness testing)**

Se prueba la capacidad del sistema para salir de situaciones provocadas por errores en el suministro de datos. Estas pruebas son importantes en sistemas con una interfaz al exterior, en particular cuando la interfaz es humana.

Ejecución y Prueba

Otros tipos de pruebas

- **Stress (stress testing)**

Se usa para saber hasta que punto soporta un sistema, bien por razones internas (cuantos datos podrá procesar), bien externas (es capaz de trabajar con un disco al 90% o una carga de la CPU del 90%, etc.)

- **Prestaciones (performance testing)**

Para los casos donde es importante el tiempo le lleva al sistema procesar tantos datos, o cuánta memoria consume, o cuánto espacio en disco utiliza, o cuántos datos transfiere por un canal de comunicaciones, etc.

- **Conformidad u Homologación (conformance testing)**

Es frecuente que organismos internacionales como ISO, ECMA, IEEE, J2EE, etc. elaboren especificaciones de referencia a las que los diversos fabricantes de software deben atenerse.

Ejecución y Prueba

Otros tipos de pruebas

- **Interoperabilidad (interoperability tesing)**

Involucran 2 o más productos buscando problemas de comunicación entre ellos.

- **Regresión (regression testing)**

En cada nueva versión se supone que o bien se corrigen defectos, o se añaden nuevas funciones, o ambas cosas. Una nueva versión exige una nueva pasada por las pruebas. Si se han sistematizado en una fase anterior, se vuelven a aplicar automáticamente, simplemente para comprobar que las modificaciones no provocan errores donde antes no los había.

- **Mutación (mutation testing)**

Consistente en alterar ligeramente el sistema, introduciendo errores, para averiguar si el conjunto de pruebas es capaz de detectarlo. En caso de no hacerlo significa que el conjunto de pruebas no es bueno.

Ejecución y Prueba

Depuración (debugging)

Casi todos los compiladores suelen llevar asociada la posibilidad de ejecutar un programa paso a paso, permitiéndole al operador conocer dónde está en cada momento, y cuánto valen las variables.

Ejecución y Prueba

Plan de pruebas

- Un plan de pruebas está constituido por un conjunto de pruebas.
- Un caso de prueba consta de 3 bloques de información:
  1. El propósito de la prueba
  2. Los pasos de ejecución de la prueba
  3. El resultado que se espera
- Respecto al orden de pruebas, una práctica frecuente es la siguiente:
  1. Pasar pruebas de caja negra analizando valores límite. Hay que analizar condiciones límite de entrada y de salida.
  2. Identificar clases de equivalencia de datos (entrada y salida) y añadir más pruebas de caja negra para contemplar valores normales (en las clases de equivalencia en que estos sean diferentes de los valores límite; es decir, en rangos amplios de valores).
  3. Añadir pruebas basadas en "presunción de error". A partir de la experiencia y el sentido común, se aventuran situaciones que parecen proclives a padecer defectos, y se buscan errores en esos puntos.
  4. Los casos de prueba para caja negra sirven también para caja blanca. No hace falta que volver a realizarlos sino medir la cobertura de caja blanca que se ha logrado con los casos de las fases previas y añadir más pruebas hasta lograr la cobertura deseada. Normalmente se busca una buena cobertura de ramas y de ciclos.

Bibliografía

- Mañas, J. (basado en). “Prueba de Programas”. pruebaProgramas.pdf (Aula Virtual).
- José A. Mañas. Prueba de Programas. 1994. http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm
- Glenford J. Myers. El Arte de Probar el Software (The Art of Software Testing) El Ateneo, 1983 (John Wiley & Sons, Inc. 1979).
- Barbee Teasley Mynatt. Software Engineering with Student Project Guidance.Prentice - Hall International Editions, 1990.
- Boris Beizer. Software Testing Techniques Van Nostrand Reinhold (N.Y.) 2a ed. 1990.
- Roger S. Pressman. Software Engineering: A Practitioner's Approach. McGraw-Hill Intl. Eds. 1987.