Pruebas de programas

Informática I - 2547100

Departamento de Ingeniería Electrónica
y de Telecomunicaciones

Facultad de Ingeniería

2015-2

Testing and debugging

- Los humanos son imperfectos, los programas también.
- Las **pruebas** permiten determinar si un programa funciona bien o no.
- La **depuración** ayuda a corregir un programa que no funciona bien.
- Los programas deben ser diseñados de manera que sus pruebas y depuración sean más fáciles.

Testing

Pruebas: ejecuciones múltiples de un programa con datos de entrada estratégicos para evidenciar fallos.

- *"Las pruebas de un programa pueden ser usadas para mostrar que hay fallos, pero nunca para mostrar su ausencia". E. Dijkstra.*
- En la mayoría de los casos, incluso los programas más simples, tienen millones de posibles datos de entrada.
  Es imposible probar todos los casos.

How to test a program?

Test suite: conjunto de datos de entrada que tiene una alta probabilidad de evidenciar fallos en un programa y que no tome demasiado tiempo ejecutarlo.

- Ideal: particionar el universo de todas las posibles entradas en subconjuntos que logren cubrir todas las opciones.
- Por lo general es difícil construir un buen test suite y depende mucho de la habilidad del programador.

Test suite example

```
def isBigger(a, b):
    '''Asume que a y b son int
       Retorna True si a es mayor que b
       y Falso de lo contrario'''
```

El universo de posibles entradas serían todas las combinaciones de dos números enteros. Una manera de crear subconjuntos sería:

a positivo, b positivo	a=0, b=0
a negativo, b negativo	a≠0, b=0
a positivo, b negativo	a=0, b≠0
a negativo, b negativo	

Probar un valor de cada rango daría una razonable garantía de que si existe un fallo, éste se evidenciaría.

Black-box testing

- Las pruebas de caja negra son diseñadas sin conocer la implementación de una función, solo su especificación.
- Permite que quien hace las pruebas sea diferente de quien hace el programa.
- Esta independencia tiende a mejorar la calidad de las pruebas.
- Las pruebas son independientes de los posibles cambios en la implementación del programa.

Black-box testing example

```
def sqrt(x, epsilon):
    '''Asume que x y epsilon son float,
       que x>=0 y que epsion>0
       Retorna resultado tal que:
       x-epsilon <= resultado**2 <= x+epsilon'''
```

x	epsilon	Razón
0,0	0,0001	Caso típico con x = 0
25,0	0,0001	Caso típico donde x tiene raíz entera
0,5	0,0001	Caso típico con x < 1
2,0	0,0001	Caso típico donde x tiene raíz irracional
2,0	0,5**64	Raíz irracional con valor muy pequeño para épsilon
0,5**64	0,5**64	x y épsilon muy pequeños
2,0**64	0,5**64	x muy grande y épsilon muy pequeño
0,5**64	2,0**64	x muy pequeño y épsilon muy grande
2,0**64	2,0**64	x y épsilon muy grandes

Black-box hints

Gran cantidad de fallos en los programas suceden en las condiciones de frontera:

- Listas: lista vacía, lista solo con un elemento, lista de listas.

igual para cadenas, tuplas y diccionarios

- Números: valores positivos y negativos, cero, valores muy grandes y muy pequeños.

Glass-box testing

- Las pruebas de caja blanca son diseñadas teniendo en cuenta la estructura interna de una función.
- Son más fáciles de diseñar debido a que la lectura del código puede revelar los casos críticos.
- Una buena metodología de pruebas debe incluir ambas estrategias: black-box y glass-box

Glass-box testing example

```
def isPrime(x):
    '''Asume que x es un entero no negativo
       Retorna True si x es primo
       y Falso de lo contrario'''
    if x <= 2:
        return False
    for i in range(2, x):
        if x%i == 0:
            return False
    return True
```

- Observando el código se ve que los valores de 0, 1 y 2 para x son tratados como casos especiales y por lo tanto deben ser probados.
- isPrime(2) no funciona correctamente
- Una prueba de black-box talvez no hubiera hecho esa prueba y el fallo no se hubiera notado.

Path-complete

Una prueba es de ruta-completa si cada posible ruta del código es explorada.

- Es fácil determinar las rutas que uno quiere probar y cuáles se están quedando por fuera.
- Los ciclos hacen que sea casi imposible probar exhaustivamente todas las rutas.

```
def isPrime(x):
    '''Asume que x es un entero no negativo
       Retorna True si x es primo
       y Falso de lo contrario'''
    if x <= 2:
        return False
    for i in range(2, x):
        if x%i == 0:
            return False
    return True
```

Glass-box hints

- if-else: pruebe ambas ramas de ejecución en todos los if-else
- for-while: pruebe casos en que
  - el ciclo nunca itere (ej: lista vacía)
  - que itere una vez
  - que itere más de una vez
  - que el ciclo se termine por cada una de las posibles razones

Testing methodology

- Pruebas de módulos: pruebas de funciones individuales o partes del programa
- Pruebas integradas: pruebas del programa completo (más difíciles)

Automatización de las pruebas

- Test drivers: programas hechos para ejecutar pruebas sobre otros programas automáticamente, inyectando los test suites
- Stubs: funciones que simulan las partes del programa que son necesarias para ejecutar la parte del programa que se está creando.