Práctica 4: Testing de Software

Maximiliano Cristiá

Ingeniería de Software 2
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

2024

Temario vocabulario y conceptos; testing estructural

Ejercicios

1. Discuta las diferencias entre validación y verificación y explique por qué la validación es un proceso particularmente difícil.
2. Una forma de trabajo común respecto del testeo de un sistema es testearlo hasta agotar el presupuesto destinado a esa fase del ciclo de vida y entregarlo a los clientes. Discuta el aspecto ético de esa forma de trabajo.
3. Explique la racionalidad detrás del testing estructural.
4. Determine las diferencias y semejanzas entre el testing estructural y el testing basado en especificaciones.
5. Genere conjuntos de prueba mínimos utilizando los siguientes criterios para los fragmentos de programas que se listan a continuación. En cada caso se indican las variables de entrada; todas las variables son de tipo entero.

   a) Cubrimiento de sentencias
   b) Cubrimiento de flechas
   c) Cubrimiento de condiciones

   En todos los casos indique si el conjunto seleccionado revela el error en el fragmento VI).

   I) Entradas: x, z

   ```
   if x > z
   then y := 3
   else y := 2
   fi;
   if x > z + 1
   then w := 3
   else w := 2
   fi;
   ```

   II) Entradas: x, z, a, b

   ```
   if x > z
   then y := 3
   else y := 2
   fi;
   ```

```
if a > b
then w := 3
else w := 2
fi;
```

iii) Entradas: x, z

```
if x > z
then y := 3
else x := z + 2
fi;
if x > z + 1
then w := 3
else w := 2
fi;
```

iv) Entradas: x, z, b

```
if x > z and x > 3
then a := 1
else a := 2
fi;
if a > b or z < x
then w := 1
else z := x
fi;
```

v) Entradas: x, z

```
if x > 0 or z > 0
then write("1")
else write("2")
fi;
if z > 0
then write("3")
else write("4")
fi;
```

vi) Entradas: x, z

```
if x > 0
then y := 5
else z := z - x
fi;
if z > 1
then z := z / x
else z:= 0
fi;
```

6. Considere el siguiente fragmento de programa que implementa la búsqueda lineal y cuyas variables de entradas son *num_items*, el arreglo *table* (de dimensión *num_items* y con índices desde 1 en adelante) y *desired_elem* (se supone $num\_items \geq 0$):

```
found := false;
if num_items <> 0 then
  counter := 1;
  while (not found) and counter < num_items do
    if table[counter] = desired_elem then found := true fi;
    counter := counter + 1
  done
fi;
if found
then write("the desired element exists in the table")
else write("the desired element doesn't exists in the table")
fi;
```

Genere conjuntos de prueba mínimos utilizando los siguientes criterios:

a) Cubrimiento de sentencias
b) Cubrimiento de flechas
c) Cubrimiento de condiciones

Indique si alguno de los criterios detecta el error que tiene el programa.

7. Considere el siguiente fragmento de programa, que es una variación del programa del ejercicio anterior donde se asigna el resultado de una expresión booleana a *found*:

```
found := false;
if num_items <> 0 then
  counter := 1;
  while (not found) and counter < num_items do
    found := (table[counter] = desired_elem);
    counter := counter + 1;
  done
fi;
if found
then write("the desired element exists in the table")
else write("the desired element doesn't exists in the table")
fi;
```

¿Cuántos casos de prueba de los que generó en el ejercicio anterior siguen sirviendo y cuántos no? ¿Hay algún criterio cuyo conjunto de prueba es el mismo? ¿Son los programas iguales o no? Si son iguales, ¿deberían haber cambiado los conjuntos de prueba generados anteriormente?

8. Sea $\langle C_i \rangle_{i \in [1,n]}$ la secuencia de todas las condiciones lógicas usadas en un programa P para gobernar el flujo de ejecución ordenadas según su punto de aparición en el programa. Por ejemplo, para el programa i) del ejercicio 5 tenemos $\langle C_i \rangle_{i \in [1,2]} = \langle x > z, x > z+1 \rangle$; o sea $C_1 \widehat{=} x > z$ y $C_2 \widehat{=} x > z+1$.

   El criterio de *asignación de valores de verdad* se define como sigue. Sea $t = \langle b_1, \ldots, b_n \rangle$ una asignación de valores de verdad a las condiciones $\langle C_i \rangle_{i \in [1,n]}$, es decir $\langle b_1, \ldots, b_n \rangle$ es un vector n-dimensional de valores booleanos cada uno de los cuales es asignado a cada condición de P. Por ejemplo para el programa mencionado en el párrafo anterior un t posible sería $\langle true, false \rangle$. Es decir, estamos pidiendo $(x > z) = true$ y $(x > z+1) = false$. En otras palabras t relaciona las condiciones (compuestas) de cada estructura de control de P (y no las condiciones de una misma estructura de control como lo hacen otros criterios de testing). De esta forma, el criterio de asignación de valores de verdad define un caso de prueba para cada asignación t posible. Cada uno de esos casos de prueba debe seleccionar valores para las variables de entrada de P de forma tal que cada C_i tenga el valor indicado en t_i. Por ejemplo, para el programa que venimos considerando tendríamos cuatro asignaciones: $\langle true, true \rangle$, $\langle true, false \rangle$, $\langle false, true \rangle$ y $\langle false, false \rangle$. Entonces, deberíamos generar un conjunto de prueba con suficientes casos como para cubrir las cuatro asignaciones (siempre que sea posible), por ejemplo: $\{\langle x = 2, z = 0 \rangle, \langle x = 2, z = 1 \rangle, \langle imposible \rangle, \langle x = 1, z = 1 \rangle\}$, donde $\langle imposible \rangle$ significa que es imposible generar un caso de prueba para la asignación $\langle false, true \rangle$.

   Genere conjuntos de prueba mínimos compatibles con el criterio de asignación de valores de verdad para todos los programas de los ejercicios anteriores. Indique si los conjuntos seleccionados revelan el error en el fragmento vi) del ejercicio 5 y el error en el programa del ejercicio 6.

9. Sea $\langle C_i \rangle_{i \in [1,n]}$ definido como en el ejercicio anterior. El criterio de *condiciones múltiples* puede definirse como sigue: cada conjunto de prueba debe recorrer todas las flechas del GFC y hacer todas las condiciones C_i verdaderas y falsas de todas las formas posibles, basándose en los valores de las proposiciones simples que las componen. Por ejemplo, si

C_5 es p and q, entonces debemos generar casos de prueba que hagan a p verdadero y q verdadero, p verdadero y q falso, etc.

Genere conjuntos de prueba mínimos compatibles con el criterio de condiciones múltiples para todos los programas de los ejercicios anteriores. Indique si los conjuntos seleccionados revelan el error en el fragmento VI) del ejercicio 5 y el error en el programa del ejercicio 6.

10. Si alguno de los criterios que usó para testear el programa del ejercicio 6 encontró el error, ¿se debió a una elección fortuita de los valores de entrada o cualquier conjunto de prueba que satisface el criterio hubiera contenido un caso de prueba que hubiera encontrado el error?

11. Considere el siguiente fragmento de un programa de ordenación donde las variables de entrada son n y el arreglo a (de dimensión $n > 0$).

```
for i in 2..n do
  x := a[i];
  a[0] := x;
  j := i - 1
  while x < a[j] do
    a[j + 1] := a[j];
    j := j - 1
  done;
  a[j + 1] := x
done;
```

Reescriba el programa de manera tal que pueda generar el GFC y genere conjuntos de prueba mínimos que satisfagan los criterios de:

a) Cubrimiento sentencias
b) Cubrimiento de flechas
c) Cubrimiento de condiciones
d) Asignación de valores de verdad
e) Cubrimiento de condiciones múltiples

12. *Para pensar, analizar y discutir*. En el enunciado del ejercicio 8 dijimos que es imposible generar un caso de prueba para la asignación $\langle false, true \rangle$ del programa I) del ejercicio 5. Rigurosamente hablando, esto habría que demostrarlo porque de lo contrario podríamos no estar testeando el programa en una situación importante. ¿Cómo lo podría demostrar? ¿A qué problema general de lógica corresponde? ¿Podría usar $\{log\}$ para hacer la demostración?

La situación que se da con el programa I) del ejercicio 5 se puede generalizar a situaciones mucho más complejas donde, por ejemplo, intervienen arreglos. ¿Qué herramientas necesitaría para que lo ayuden en la tarea? ¿Podría usar $\{log\}$? ¿Faltaría algo? ¿Se podría construir una herramienta que resuelva el problema para todos los casos?

¿Podría usar $\{log\}$ no solo para demostrar que cierta asignación es imposible sino también para encontrar valores de entrada para las que son posibles?