Práctica 4: Testing de Software

Maximiliano Cristiá
Ingeniería de Software
F.C.E.I.A. - U.N.R.

Noviembre 2010

1. Temario

- Vocabulario y conceptos
- Testing estructural basado en el flujo de control
- Testing funcional basado en especificaciones Z

Problemas

Vocabulario y conceptos

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. Volviendo sobre las cualidades del software vistas en la Unidad I de Análisis de Sistemas, clasifíquelas con respecto a las siguientes cuestiones relativas a V&V:
   - ¿Son subjetivas u objetivas?
   - ¿Son binarias o no?
   - ¿Son más o menos relevantes en diferentes aplicaciones y entornos?
4. Explique la racionalidad detrás del testing estructural.
5. Determine las diferencias y semejanzas entre el testing estructural y el testing basado en especificaciones.

Testing estructural basado en el flujo de control

1. Muestre que el procedimiento del Ejemplo 6.2 de la página 262 de [GJM91] contiene un error sutil: no funciona para tablas de un cierto tamaño. Determine dicho tamaño.

2. Considere el siguiente fragmento de programa

```
read(x); read(y);
if x > 0 or y > 0 then
  write("1");
else
  write("2");
endif;
if y > 0 then
  write("3");
else
  write("4");
endif;
```

   Genere casos de prueba utilizando el criterio de cubrimiento de sentencias.

3. Considere los dos fragmentos de programa siguientes:

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

```
found := false;
if num_items <> 0 then counter := 1;
  while (not found) and counter < num_items loop
    found := table[counter] = desired_elem;
    counter := counter + 1;
  endloop
endif;
if found then
  write("the desired element exists in the table");
```

```
else
  write("the desired element doesn't exists in the table");
endif;
```

Determine si el criterio de cubrimiento de flechas genera conjuntos de prueba diferentes en cada caso.

4. Sea $\{C_i\}$ el conjunto de todas las condiciones lógicas usadas en un programa P para gobernar el flujo de ejecución. El criterio de cubrimiento de flechas requiere que el conjunto de prueba $\{d_j\}$ deber ser tal que cada C_i sea falsa y verdadera para algún d_j al menos una vez.

   Para cada C_i sean D_i y $overline D_i$ los conjuntos de entradas que hacen que C_i sea verdadera y falsa, respectivamente. Luego el criterio de cubrimiento de flechas es satisfecho por conjuntos de prueba T que deben contener al menos un elemento de D_i y uno de $overline D_i$ para cada C_i. Explique por qué esto no determina una partición de D.

   Como consecuencia debe haber diferentes conjuntos de prueba con diferente cardinalidad que satisfagan el criterio de cubrimiento de flechas. Así, aparece el problema de encontrar conjuntos de prueba minimales que satisfagan el criterio. Para los siguientes fragmentos, encuentre conjuntos de prueba mínimos compatibles con el criterio de cubrimiento de flechas:

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

b)
```
if x > z then
  y := 3;
else
  y := 2;
endif;
if a > b then
  w := 3;
else
  w := 2;
endif;
```

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

5. Sea $\{C_i\}$ definido como en el problema anterior. Considere la definición del criterio de asignación de valores de verdad:

   Sea $t = \langle tr_1, \ldots, tr_n \rangle$ una asignación de valores de verdad a las condiciones $\{C_i\}$, es decir $\langle tr_1, \ldots, tr_n \rangle$ es un vector n-dimensional de valores booleanos que será asignado a cada condición de P. 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). Para cada una de estas asignaciones, sea D_t el subconjunto de D que hace todas las C_i sean verdaderas o falsas según la asignación. Muestre que $\{D_t\}$ es una partición de D.

6. Sea $\{C_i\}$ definido como en el problema anterior. El criterio de cubrimiento de condiciones múltiples puede definirse como sigue: cada conjunto de prueba debe hacer todas las condiciones C_i verdaderas o falsas de todas las formas posibles, basándose en los valores de las proposiciones simples que las componen. Por ejemplo, si C_5 es c_{51} **and** c_{52}, entonces debemos generar cuatro casos de prueba que hacen a c_{51} verdadera, c_{52} verdadera, c_{51} verdadera, c_{51} falsa, etc.

   Determine un conjunto de prueba (posiblemente minimal) que satisfaga el criterio de cubrimiento de condiciones múltiples para el siguiente fragmento de programa:

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

7. Verifique el siguiente programa utilizando el criterio de asignaciones de valores de verdad y determine si revela el error existente.

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

8. Considere el siguiente fragmento de un programa de ordenación:

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

Verifíquelo utilizando 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

Ayuda: considere a todas las componentes del arreglo como una única variable, `a`.

9. Calcule los conjunto de casos de prueba de los programas del problema 3 que satisfagan los criterios

   a) Cubrimiento de condiciones

   b) Asignación de valores de verdad

   c) Cubrimiento de condiciones múltiples

Testing basado en especificaciones Z

Consideraciones generales para resolver los ejercicios

- A menos que se indique lo contrario, todos los problemas deben resolverse utilizando Fastest. En todos los casos se debe consignar el *script* que se utilizó para resolver el problema.
- Siempre se deben aplicar las tácticas de testing que optimicen la cobertura de las principales alternativas funcionales expresadas en el modelo. Sin embargo, no se deben generar casos de prueba que no pongan a prueba significativamente la implementación.
- Siempre se debe justificar la elección de las tácticas de testing aplicadas.
- En estos problemas "caso de prueba" es equivalente a "caso de prueba abstracto".
- Se deben generar casos de prueba para todas las clases que sea satisfacibles.
- En ningún caso debe quedar una clase de prueba sin caso de prueba (a menos que haya algún error en la aplicación).
- Si una clase de prueba es insatisfacible y Fastest no la podó, siempre que sea posible, se debe agregar el teorema de eliminación más general posible a la respectiva biblioteca.

1. Aplicando al menos dos tácticas derive casos de test para una operación (especificada en Z) que actualice dos tablas relacionadas por un campo clave común a ambas.
2. **Base de datos cinematográficos.** Considere la especificación Z escrita en Análisis de Sistemas para este problema. Genere casos de prueba para las operaciones:

   a) Que obtienen todas las películas de un director dado

   b) Que modifican el nombre de un director de una película dada.

3. **Editor de textos.** Considere la especificación Z desarrollada en [Jac97] para este problema. Genere casos de prueba para todas las operaciones definidas.
4. **Banco.** Considere el modelo más simple desarrollado en el apunte de clase sobre Z de Análisis de Sistemas. Genere casos de prueba para todas las operaciones allí definidas.

5. Genere casos de prueba para la siguiente operación.

State

$$f : X \pfun \mathbb{Z}$$

Operation

$$\Delta State$$
$$new? : \mathbb{P}\, X$$
$$f' = f \cup \{x : X \mid x \in new? \setminus \operatorname{dom} f \bullet x \mapsto 0\} \setminus \{x : X \mid x \in new? \cap \operatorname{dom} f \bullet x \mapsto f\ x\}$$

6. Considere la siguiente especificación sobre parte de los requerimientos de un sistema de seguridad social.

$$[DNI, DOMICILIO]$$

SeguridadSocial

$$viveEn : DNI \pfun DOMICILIO$$
$$ingresosPorDomicilio : DOMICILIO \pfun \mathbb{Z}$$

SeguridadSocialInv

$$SeguridadSocial$$
$$\operatorname{ran} viveEn = \operatorname{dom} ingresosPorDomicilio$$

CambioDeDomicilio

$$\Delta SeguridadSocial$$
$$x? : DNI;\ nuevoDomicilio? : DOMICILIO;\ z? : \mathbb{Z}$$
$$x? \in \operatorname{dom} viveEn$$
$$nuevoDomicilio? \neq viveEn\ x?$$
$$viveEn' = viveEn \oplus \{x? \mapsto nuevoDomicilio?\}$$
$$ingresosPorDomicilio' = \textbf{if } viveEn\ x? \notin \operatorname{ran}(\{x?\} \rhd viveEn) \textbf{ then } \{viveEn\ x?\} \rhd ingresosPorDomicilio \oplus \{nuevoDomicilio? \mapsto z?\} \textbf{ else } ingresosPorDomicilio \oplus \{nuevoDomicilio? \mapsto z?\}$$

a) Aplique FND a la operación *CambioDeDomicilio* de forma tal de eliminar el **if** − **then** − **else** .

b) Genere casos de prueba para la operación en cuestión.

7. Genere casos de prueba para *Setuid*,

$[OBJECT, USER, CATEGORY]$

$PROCID == \mathbb{N}$

$root : USER$
$SECADMIN : CATEGORY$
$softtcb : \mathbb{P}\ OBJECT$

SecClass
$categs : \mathbb{P}\ CATEGORY$
$level : \mathbb{Z}$

Process
$usr, suid : USER$
$prog : OBJECT$

SecureFileSystem
$aprocs : PROCID \pfun Process$
$users : \mathbb{P}\ USER$
$usc : USER \pfun SecClass$

PSetuid
$\Delta Process$
$new? : USER$

$suid' = new?$
$usr' = usr$
$prog' = prog$

SetuidOk1
$\Delta SecureFileSystem$
$\Delta Process$
$PSetuid$
$pid? : PROCID$
$new? : USER$

$pid? \in \mathrm{dom}\ aprocs$
$new? \in users$
$(aprocs\ pid?).usr = root$
$SECADMIN \notin (usc\ new?).categs$
$(aprocs\ pid?).prog \notin softtcb$
$(aprocs\ pid?) = \theta Process$
$aprocs' = aprocs \oplus \{pid? \mapsto \theta Process'\}$
$users' = users$
$usc' = usc$

SetuidOk2
$\Delta SecureFileSystem$
$\Delta Process$
$PSetuid$
$pid? : PROCID$
$new? : USER$

$pid? \in \mathrm{dom}\ aprocs$
$new? \in users$
$(aprocs\ pid?).usr = root$
$SECADMIN \in (usc\ new?).categs$
$(aprocs\ pid?).prog \in softtcb$
$(aprocs\ pid?) = \theta Process$
$aprocs' = aprocs \oplus \{pid? \mapsto \theta Process'\}$
$users' = users$
$usc' = usc$

SetuidOk3
$\Delta SecureFileSystem$
$\Delta Process$
$PSetuid$
$pid? : PROCID$
$new? : USER$

$pid? \in \mathrm{dom}\ aprocs$
$new? \in users$
$(aprocs\ pid?).usr \neq root$
$SECADMIN \notin (usc\ new?).categs$
$new? = suidto\ (aprocs\ pid?).prog$
$(aprocs\ pid?) = \theta Process$
$aprocs' = aprocs \oplus \{pid? \mapsto \theta Process'\}$
$users' = users$
$usc' = usc$

SetuidE1
$\Xi SecureFileSystem$
$pid? : PROCID$
$new? : USER$

$pid? \in \mathrm{dom}\ aprocs$
$new? \in users$
$((aprocs\ pid?).usr = root$
$\quad \land\ SECADMIN \in (uscnew?).categs$
$\quad \land\ (aprocs\ pid?).prog \notin softtcb$
$\lor\ (aprocs\ pid?).usr \neq root$
$\quad \land\ SECADMIN \in (uscnew?).categs$
$\lor\ (aprocs\ pid?).usr \neq root$
$\quad \land\ new? \neq suidto\ (aprocs\ pid?).prog)$

$$SetuidE2 == [\Xi SecureFileSystem \mid pid? \notin \operatorname{dom} aprocs]$$
$$SetuidE3 == [\Xi SecureFileSystem \mid new? \notin users]$$
$$SetuidE == SetuidE1 \lor SetuidE2 \lor SetuidE3$$
$$SetuidOk == SetuidOk1 \lor SetuidOk2 \lor SetuidOk3$$
$$Setuid == SetuidOk \lor SetuidE$$

8. (No hacer con Fastest) Considere la especificación del problema 7. Diseñe una táctica de testing funcional que tenga en cuenta la definición axiomática. Generalice la táctica para cualquier especificación. Pruebe que su táctica genera franjas de nodos que son ortogonales y completas. Aplique su táctica a la especificación del problema 7.

9. Defina una partición estándar para el operador $\frown$. Cargue esta partición en Fastest.

10. Genere casos de prueba para la operación $ReservationOK1$.

$[PASSPORT]$

$$maxSeats : \mathbb{Z}$$
$$\hline$$
$$maxSeats > 0$$

El sistema se representa por medio de dos secuencias: *seats*, que almacena los números de pasaporte que tienen asiento reservado en el avión; y *waitingList*, que guarda los números de pasaporte que han solicitado una reserva pero están en lista de espera.

ARS
$$seats, waitingList : \operatorname{seq} PASSPORT$$

Se modela la operación para solicitar una reserva. Si aun hay lugar en el avión se le asigna un asiento al siguiente pasaporte. Hasta ese momento la lista de espera debe estar vacía.

$ReservationOK1$
$$\Delta ARS$$
$$p? : PASSPORT$$
$$\hline$$
$$\#seats < maxSeats$$
$$seats' = seats \frown \langle p? \rangle$$
$$waitingList' = waitingList$$

Si no hay más asientos pero se siguen haciendo reservaciones, se debe poner a los siguientes pasaportes en la lista de espera (pero esta no puede ser muy larga).

$ReservationOK2$
$$\Delta ARS$$
$$p? : PASSPORT$$
$$\hline$$
$$\#seats = maxSeats$$
$$\#waitingList < 2 * maxSeats$$
$$seats' = seats$$
$$waitingList' = waitingList \frown \langle p? \rangle$$

$$Reservation == ReservationOK1 \lor ReservationOK2$$

Referencias

[GJM91] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of sofware engineering.* Prentice Hall, Upper Saddle River, New Jersey, 1991.

[Jac97] Jonathan Jacky. *The Way of Z.* Cambridge University Press, 1997.