Tema 2: Diseño de tests y criterios de adecuación

Miguel Gómez-Zamalloa Gil

Dep. de Sistemas Informáticos y Computación. UCM

Especificación, Validación y Testing
Optativa de los Grados de la Facultad de Informática,
Curso 2015-2016

Índice

Tema 2: Diseño de tests y criterios de adecuación

- Introducción
- Estrategias de caja-blanca
- Estrategias de caja-negra

Introducción

- Realizar un testing completo es imposible o irrealizable (ver tema 1, slides 24-29)
- El problema a resolver es por tanto el siguiente:

Encontrar un subconjunto de casos de prueba de entre todos los posibles con el cual sea posible detectar la mayor cantidad posible de errores con una probabilidad alta, y con un coste razonable

Criterio de adecuación vs. estrategia de generación de tests

- **Estrategia de generación de tests:** Determina cómo se diseñan los tests, y cuándo acaba el proceso
- **Criterio de adecuación:** Mide la calidad de un conjunto de tests a posteriori
  - Ej.: Porcentaje de instrucciones del programa ejercitadas por los tests

Criterio de adecuación vs. estrategia de generación de tests

- **Estrategia de generación de tests:** Determina cómo se diseñan los tests, y cuándo acaba el proceso
- **Criterio de adecuación:** Mide la calidad de un conjunto de tests a posteriori
  - Ej.: Porcentaje de instrucciones del programa ejercitadas por los tests

Normalmente, la generación de tests va guiada por uno o varios criterios de adecuación, de manera que éstos se satisface(n) por construcción

Estrategias de generación de tests

Podemos distinguir tres clases de estrategias:

1. Generación aleatoria
2. Estrategias de caja-blanca $\Rightarrow$ Se usa el código fuente
3. Estrategias de caja-negra $\Rightarrow$ Se usan los requisitos o especificación

- Dependiendo del contexto es preferible usar una estrategia u otra, o combinarlas
  - *ArtOfST* recomienda primeramente generar tests usando estrategias de caja-negra y posteriormente generar tests suplementarios usando estrategias de caja-blanca

Estrategias de generación de tests

- La generación aleatoria es aparentemente poco efectiva pero aún así se utiliza mucho
  - En ciertos contextos sí puede resultar muy útil
  - Por ej.: generación/testing automático contra especificaciones formales
- También hay metodologías *mixtas*. Por ejemplo, es habitual partir de una generación aleatoria seguida de un proceso de refinamiento sucesivo
- En el resto del tema veremos estrategias de caja-blanca y estrategias de caja-negra

Estrategias de caja-blanca

- Statement coverage
- Decision coverage
- Condition coverage
- Condition/Decision coverage
- Multiple-condition coverage
- Path coverage
- Modified condition/decision coverage

(Apuntes de clase + ArtOfST págs. 43-52 + Mathur págs. 415-441)

Estrategias de caja-negra

Se pueden distinguir dos grandes categorías dependiendo del tipo de requisitos o especificación que tengamos:

1. Estrategias para especificaciones informales:
   - *Equivalence partitioning*
   - *Boundary-value analysis*
   - *Cause-effect graphing*
2. Estrategias para especificaciones formales (*Model-based testing*):
   - Máquinas de estados finitos
   - Lenguajes de modelado: UML, Z, B, etc.

Equivalence partitioning

- Un buen test-case debería cumplir:
  1. Representa a un conjunto de test-cases en el sentido de que supuestamente revela los mismos errores que ellos
  2. Reduce en más de uno el número de casos restantes necesarios para realizar un buen testing
- (1) sugiere que debemos particionar el dominio de cada entrada en clases de equivalencia (con entradas supuestamente equivalentes en cuanto a los errores que pueden detectar)
- (2) sugiere que se trate de minimizar el número de tests a base de maximizar el ńumero de condiciones que cada test ejercita

Equivalence partitioning

Se procede en dos fases:

1. Identificación de clases de equivalencia
2. Generación de los tests

1) Identificación de cases de equivalencia

- Identificar cada condición de entrada (frase de la especificación) y particionarla en dos o más grupos
- Al menos se debe distinguir entre *entrada válida* y *entrada no válida*
- Pautas a seguir:
  - Si se especifica un rango, identificar una clase válida y dos inválidas (una por arriba y una por debajo). Ej.: $c \in ['a'..'f']$
  - Si se especifica un número de elementos, identificar una clase válida y dos inválidas (ningún elemento y más elementos de los permitidos)
  - Si se especifica un conjunto de entradas y hay razones para pensar que cada uno se maneja de forma distinta, identificar una clase válida para cada entrada y otra clase inválida
  - Si hay razones para pensar que elementos de la misma clase de equivalencia se tratan de forma diferente entonces particionarla correspondientemente

2) Generación de los tests

1. Asignar un número o nombre a cada clase de equivalencia
2. Mientras no se hayan cubierto todas las clases válidas, escribir un nuevo test que cubra tantas clases válidas como sea posible
3. Mientras haya alguna clase inválida sin cubrir, escribir un nuevo test que cubra una, y solo una, clase inválida
   - Es habitual que un error enmascare otros errores
   - Ej.: Si se espera una entrada $l \in \{english, spanish\}$, $age \in [1.,199]$, el test $\{l = french, age = 0\}$ probablemente no ejercitará el supuesto chequeo de la edad, dando un error del estilo "french is not a valid language"

Ejemplo de Equivalence partitioning

(ArtOfST págs. 56-58)

Boundary-value analysis

- *Boundary-value analysis* funciona de forma similar a *Equivalence partitioning* y trata de mejorarlo
- La experiencia ha demostrado que los tests que ejercitan los valores límite de las condiciones de entrada y salida son muy útiles
  - Si en `tipoDeTriangulo` se escribe por error `a+b < c` el test $\{a = 1, b = 2, c = 3\}$ sería crucial
- Dos diferencias fundamentales respecto a *Equivalence partitioning*:
  1. Se seleccionan varios tests (normalmente 2 o 3) para cada clase de equivalencia. Se trata de estudiar las condiciones límite.
  2. No solo se tienen en cuenta las condiciones de entrada, sino también las condiciones de salida $\Rightarrow$ Clases de equivalencia para la salida
- Por tanto, de nuevo, el primer paso es la identificación de las clases de equivalencia, ahora también de salida

Pautas a seguir (a añadir a las expuestas en *Equiv. part.*):

- Si se especifica un rango, escribir tests válidos para los límites del rango, y tests inválidos para los primeros valores tras los límites
  - Ej.:$n \in [1.,199]$. Tests: $\{n = 1, n = 199, n = 0, n = 200\}$
- Si se especifica un número de elementos, escribir tests válidos para el mínimo y el máximo, y tests inválidos para uno más del máximo y uno menos del mínimo
  - Ej.: Fichero con entre 1 y 255 registros.
    Tests: {1 reg., 255 regs.,0 regs., 256 regs.}
- Usar las dos pautas anteriores también para las condiciones de salida

Ejemplo de Boundary-value analysis

(ArtOfST págs. 61-65)

Cause-effect graphing

- Una debilidad de *Boundary-value analysis* y *Equivalence partitioning* es que no exploran distintas combinaciones de condiciones de entrada
- El número de combinaciones puede ser astronómico
- Los *Cause-effect graphs* modelan dependencias entre distintas entradas y salidas
- Dichas dependencias pueden servir para seleccionar ciertas combinaciones evitando así la explosión combinatoria