

en realidad están tratando con más de un caso de uso. Cada una de estas posibles rutas debe describirse en un caso de uso independiente.

Ejemplo Consideremos un ejemplo del escenario "Retiro correcto de dinero (\$100) de un cajero automático con una comisión de \$5". Un caso de uso que describa este escenario podría parecerse al que se muestra a continuación.

Este caso de uso consta de un escenario principal (pasos 1 a 11), tres excepciones (etiquetadas 3A, 4A y 9A) y un flujo alternativo (9B). La numeración de estos eventos hace referencia al paso en el que pueden ocurrir. Si es posible más de una excepción o flujo alternativo en un solo paso, se indican con letras consecutivas: A, B, C, etc.

Caso de uso: UC-003-02

Nombre: retiro exitoso de \$100 de un cajero automático con una comisión de \$5.

Requisitos previos: el usuario ha iniciado sesión, la tarjeta de pago se reconoce como una tarjeta con una tarifa de retiro de \$5.

Pasos del escenario principal:

1. El usuario selecciona la opción de Retirar dinero.
2. El sistema muestra un menú con los montos de pago disponibles.
3. El usuario selecciona la opción \$100.
4. El sistema verifica si hay al menos \$105 en la cuenta del usuario.
5. El sistema solicita una copia impresa de la confirmación.
6. El usuario selecciona la opción NO.
7. El sistema muestra el mensaje Quitar tarjeta.
8. El sistema devuelve la tarjeta.
9. El usuario toma la tarjeta.
10. El sistema retira \$100 y transfiere \$5 de comisión a la cuenta bancaria.
11. El sistema muestra el mensaje Tomar dinero. El caso de uso finaliza, el usuario cierra la sesión, el sistema regresa a la pantalla inicial.

Condiciones finales: se retiraron \$100 del usuario, el saldo de la cuenta del usuario se redujo en \$105, el saldo de efectivo del cajero automático se redujo en \$100.

Fujos alternativos y excepciones:

3A - el usuario no selecciona ninguna opción durante 30 segundos (el sistema devuelve la tarjeta y vuelve a la pantalla inicial).

4A - el saldo de la cuenta es inferior a \$105 (mensaje Fondos insuficientes, devolver la tarjeta, registrar al usuario, volver a la pantalla inicial y finalizar el caso de uso).

9A - el usuario no toma la tarjeta durante 30 segundos (el sistema "extrae" la tarjeta, envía una notificación por correo electrónico al cliente y vuelve a la pantalla inicial).

9B - el usuario vuelve a insertar la tarjeta después de recibirla (el sistema devuelve la tarjeta).

Derivar casos de prueba a partir de un caso de uso
Hay un mínimo de lo que un evaluador debe hacer para probar un caso de uso. Este mínimo (entendido como cobertura total, 100% del caso de uso) es diseñar:

- Un caso de prueba para ejercitar el escenario principal, sin eventos inesperados.
- Suficientes casos de prueba para ejercitar cada excepción y flujo alternativo.

Entonces, en nuestro ejemplo, podríamos diseñar los siguientes cinco casos de prueba:

- CT1: implementación de los pasos 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
- TC2: implementación de los pasos 1, 2, 3A
- TC3: implementación de los pasos 1, 2, 3, 4A
- TC4: implementación de los pasos 1, 2, 3, 4, 5, 6, 7, 8, 9A
- TC5: implementación de los pasos 1, 2, 3, 4, 5, 6, 7, 8, 9B, 10, 11

Cada excepción y flujo alternativo debe probarse en casos de prueba separados para evitar el enmascaramiento de defectos. Discutimos este fenómeno en detalle en la Sección. 4.2.1.
Sin embargo, en algunas situaciones, por ejemplo, presión de tiempo, se puede permitir probar más de un flujo alternativo dentro de un caso de prueba.

Destacar que TC5 (flujo alternativo) nos permite conseguir el objetivo (llegamos al paso 11).
TC2, TC3 y TC4 finalizan antes debido a la ocurrencia de situaciones incorrectas durante la ejecución del escenario. Podemos diseñar los casos de prueba a partir del caso de uso, en forma de escenarios, describiendo la respuesta esperada del sistema en cada paso. Por ejemplo, el escenario para el caso de prueba TC3 puede verse como en la Tabla 4.12.

Observemos que los pasos consecutivos (acciones del usuario y respuestas esperadas) corresponden a los pasos 1, 2, 3 y 4A del caso de uso consecutivo. Observe también que el evaluador verificó el valor límite para el monto del retiro en este escenario: se suponía que toda la operación reduciría el saldo de la cuenta del usuario en \$105, mientras que el saldo de la cuenta declarado era \$104,99. Éste es un ejemplo típico de combinación

Tabla 4.12 Caso de prueba construido a partir de un caso de uso

Caso de prueba TC3: Retiro en cajero automático con fondos insuficientes en la cuenta (relativo al caso de uso UC-003-02, ocurrencia de la excepción 4A).		
Condiciones		
previas: • El usuario ha iniciado sesión • El sistema está en el menú principal • El retiro se reconoce como cargado con una comisión de \$5 (debido al tipo de tarjeta) • Saldo en cuenta: \$104.99 Paso Evento		
		Resultado Esperado
1	Seleccione la opción para Retirar dinero	El sistema pasa al menú de selección de importes.
2	Elige la opción \$100	El sistema encuentra que no hay fondos en la cuenta, muestra el mensaje Sin fondos, devuelve la tarjeta
Condiciones		
posteriores: • El saldo de la cuenta del usuario y el saldo de la cuenta bancaria no han cambiado • El cajero automático no retiró dinero • El cajero automático devolvió la tarjeta al usuario		
• El sistema volvió a la pantalla de bienvenida		

técnicas de prueba para incluir la verificación de muchos tipos diferentes de condiciones en un pequeño número de casos de prueba. Aquí, para verificar el correcto funcionamiento del sistema en una situación de fondos insuficientes en la cuenta, utilizamos la técnica del análisis del valor límite, de modo que la cantidad insuficiente era sólo 1 centavo menos que la cantidad que ya habría permitido el retiro correcto de dinero.

Cobertura

El estándar ISO/IEC 29119 no define una medida de cobertura para las pruebas de casos de uso. Sin embargo, el antiguo plan de estudios sí proporciona dicha medida, asumiendo que los elementos de cobertura son flujos de escenarios de casos de uso. Por lo tanto, la cobertura se puede definir como la relación entre el número de rutas de flujo verificadas y todas las rutas de flujo posibles descritas en el caso de uso.

Por ejemplo, en el caso de uso descrito anteriormente, hay cinco rutas de flujo diferentes: la ruta principal, tres flujos con manejo de excepciones y uno con flujo alternativo. Si el conjunto de pruebas contuviera solo los casos de prueba TC1 y TC5, la cobertura de casos de uso con estas pruebas sería $2/5 = 40\%$.

4.3 Técnicas de prueba de caja blanca

FL-4.3.1 (K2) Explicar la prueba de declaraciones.

FL-4.3.2 (K2) Explicar las pruebas de rama.

FL-4.3.3 (K2) Explicar el valor de las pruebas de caja blanca.

Las pruebas de caja blanca se basan en la estructura interna del objeto de prueba. Muy a menudo, las pruebas de caja blanca están asociadas con las pruebas de componentes, en las que el modelo del objeto de prueba es la estructura interna del código, representada, por ejemplo, en forma del llamado gráfico de flujo de control (CFG). Sin embargo, es importante señalar que las técnicas de prueba de caja blanca se pueden aplicar a todos los niveles de prueba, por ejemplo:

- En las pruebas de componentes (estructura de ejemplo: CFG)
 - En las pruebas de integración (estructuras de ejemplo: gráfico de llamadas, API)
 - En las pruebas de sistemas (estructuras de ejemplo: proceso de negocio modelado en BPMN idioma, 3 menú del programa)
- En las pruebas de aceptación (estructura de ejemplo: estructura de páginas del sitio web)

El programa de estudios de Foundation Level analiza dos técnicas de prueba de caja blanca: prueba de declaraciones y prueba de rama. Ambas técnicas están asociadas por naturaleza con el código, por lo que su área de aplicación es principalmente la prueba de componentes. Además de éstas, existen muchas otras técnicas de caja blanca (y normalmente más potentes), como:

³ Modelo y notación de procesos de negocio, BPMN: una notación gráfica utilizada para describir el negocio. procesos

- Pruebas MC/DC •

Pruebas de condiciones múltiples •

Pruebas de bucle •

Pruebas de ruta básica

Sin embargo, estas técnicas no se analizan en el programa de estudios del nivel básico.

Algunos de ellos se analizan en el programa de estudios de Nivel avanzado: Analista de pruebas técnicas [29]. Estas técnicas de prueba de caja blanca más potentes se utilizan a menudo cuando se prueban sistemas críticos para la seguridad (por ejemplo, software de instrumentos médicos o software aeroespacial).

4.3.1 Pruebas de declaraciones y cobertura de declaraciones

La prueba de declaraciones es la más simple y también la más débil de todas las técnicas de prueba de caja blanca. Implica cubrir declaraciones ejecutables en el código fuente de un programa.

Ejemplo Considere un fragmento de código simple:

```

1. ENTRADA x, y // dos números naturales
2. SI (x > y) ENTONCES
3.
    z := x - y
    DEMÁS
4.    z := y - x
5. SI (x > 1) ENTONCES
6.    z := z*2
7. REGRESAR z

```

Las declaraciones ejecutables están marcadas con los números del 1 al 7 en este código. El texto que comienza con los caracteres “//” es un comentario y no se ejecuta cuando se ejecuta el programa. En general, el código se ejecuta secuencialmente, línea por línea, a menos que se produzcan algunos saltos en el flujo de control (por ejemplo, en las sentencias de decisión 2 y 5 de nuestro código). La palabra clave ELSE es sólo una parte de la sintaxis de la instrucción IF-THEN-ELSE y por sí sola no se trata como una instrucción ejecutable (porque no hay nada que ejecutar aquí).

Después de tomar de la entrada dos variables, x e y, en la declaración 1, el programa verifica en la declaración 2 si x es mayor que y. Si es así, se ejecuta la declaración 3 (asignación a z de la diferencia x - y), seguida de un salto a la declaración 5. Si no, se ejecuta la declaración 4 (la parte “else” de IF-THEN-ELSE), en la que la diferencia y - x se asigna a la variable z, seguido de un salto a la declaración 5. En la declaración 5, se verifica si la variable x tiene un valor mayor que 1. Si es así, el cuerpo de la declaración IF-THEN (declaración 6) se ejecuta (duplicando el valor de z). Si la decisión en la declaración 5 es falsa, se omite la declaración 6 y el flujo de control salta después del bloque SI-ENTONCES, es decir, a la declaración 7, donde se devuelve el resultado (el valor de la variable z) y el programa termina. su funcionamiento.

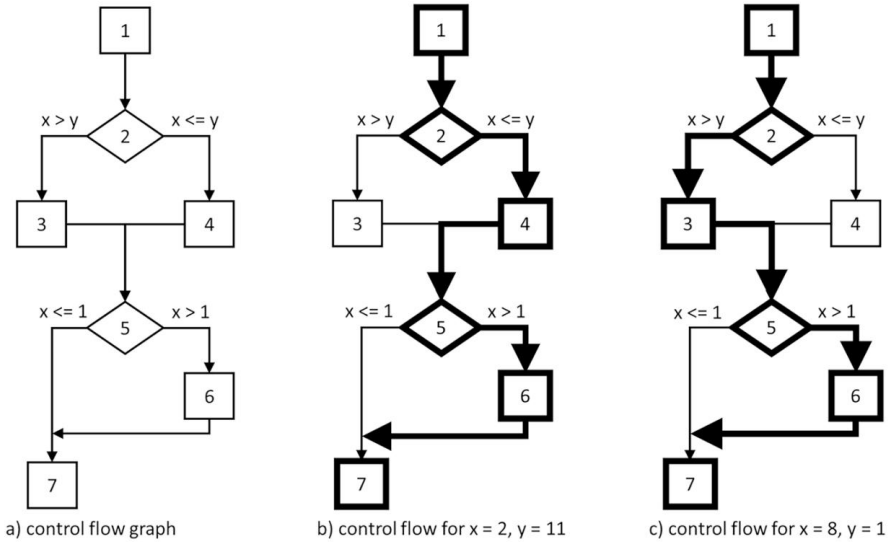


Fig. 4.12 CFG y ejemplos de flujo de control para diferentes datos de entrada

El código fuente se puede representar como un CFG. En la figura 4.12a se muestra un gráfico de este tipo para el código antes mencionado. Es un gráfico dirigido, en el que los vértices representan declaraciones y las flechas representan el posible flujo de control entre declaraciones. Las declaraciones de decisión se indican aquí con rombos, mientras que otras declaraciones se indican con cuadrados.

Considere dos casos de prueba para el código del ejemplo anterior:

- TC1: entrada: $x = 2, y = 11$; salida esperada: 18.
- TC2: entrada: $x = 8, y = 1$; resultado esperado: 14.

En la figura 4.12b, las flechas en negrita muestran el flujo de control cuando a la entrada se le dan los valores $x = 2, y = 11$. El flujo de control para TC1 será el siguiente (entre paréntesis, mostramos el resultado de la decisión en un momento dado). declaración de decisión):

$1 \rightarrow 2 (x \leq y) \rightarrow 4 \rightarrow 5 (x > 1) \rightarrow 6 \rightarrow 7$.


Por otro lado, para el TC2 con entrada $x = 8, y = 1$, se muestra el flujo de control en la Fig. 4.12c y será el siguiente:

$1 \rightarrow 2 (x > y) \rightarrow 3 \rightarrow 5 (x > 1) \rightarrow 6 \rightarrow 7$.

Enfaticemos una cosa muy importante en el contexto de las técnicas de prueba de caja blanca: los resultados esperados en los casos de prueba no se derivan del código. Esto se debe a que el código es precisamente lo que estamos probando, por lo que no puede ser su propio oráculo. Los resultados esperados se derivan de una especificación externa al código. En nuestro caso, podría ser así: "El programa toma dos números naturales y luego les resta los

uno más pequeño del más grande. Si el primer número es mayor que 1, el valor se duplica aún más. El programa devuelve el valor calculado de esta manera".

Cobertura

En las pruebas de declaraciones, los elementos de cobertura son las declaraciones ejecutables. Por de cobertura de  lo tanto, es el cociente de instrucciones ejecutables cubiertas por casos declaraciones de prueba por el número de todas las instrucciones ejecutables en el código analizado, generalmente expresado como porcentaje. Tenga en cuenta que esta métrica no tiene en cuenta declaraciones que no son ejecutables (por ejemplo, comentarios o encabezados de funciones). Reconsideremos el código fuente anterior y los casos de prueba TC1 y TC2. Si nuestro conjunto de pruebas contiene sólo TC1, su ejecución alcanzará una cobertura de 6/7 (aprox. 86%), ya que este caso de prueba ejercita seis instrucciones ejecutables diferentes de las siete que componen nuestro código (éstas son 1, 2, 4, 5, 6 y 7). TC2 también ejercita seis de los siete enunciados (1, 2, 3, 5, 6, 7), por lo que logra la misma cobertura de ca. 86%. Sin embargo, si nuestro conjunto de prueba contiene TC1 y TC2, la cobertura será del 100%, porque juntos, estos dos casos de prueba ejercitan las siete declaraciones (1, 2, 3, 4, 5, 6, 7).


4.3.2 Pruebas de sucursales y cobertura de sucursales

Una rama es un flujo de control entre dos vértices de un CFG. Una rama puede ser incondicional o condicional. Una rama incondicional entre los vértices A y B significa que después de que se completa la ejecución de la declaración A, el control debe pasar a la declaración B. Una rama condicional entre A y B significa que después de que se completa la ejecución de la declaración A, el control puede pasar a la declaración B, pero no necesariamente.

Las ramas condicionales surgen de vértices de decisión, es decir, lugares del código donde se toma alguna decisión de la que depende el curso de control posterior. Ejemplos de declaraciones de decisión son las declaraciones condicionales IF-THEN, IF-THEN-ELSE y SWITCH-CASE, así como declaraciones que verifican la llamada condición de bucle en los bucles WHILE, DO-WHILE o FOR.

Por ejemplo, en la figura 4.12a, las ramas incondicionales son 1→2, 3→5, 4→5 y 6→7, porque, por ejemplo, si se ejecuta la declaración 3, la siguiente declaración debe ser la declaración 5. otras ramas, 2→3, 2→4, 5→6 y 5→7, son condicionales. Por ejemplo, después de la ejecución de la sentencia 2, el control puede pasar a la sentencia 3 o a la sentencia 4. Cuál de estos casos ocurra dependerá del valor lógico de la decisión SI en la sentencia 2. Si $x > y$, el control irá a 3, pero si $x \leq y$, irá a 4.

Cobertura

En las pruebas de sucursales, los elementos de cobertura son sucursales, tanto condicionales como incondicionales. Por lo tanto, la  cobertura de sucursales se calcula como el número de sucursales que se ejercieron durante la ejecución de la prueba dividido por el número total de sucursales en el código. El objetivo de las pruebas de sucursales es diseñar una cantidad suficiente de casos de prueba que alcancen el nivel requerido (aceptado) de cobertura de sucursales. Como en otros casos, esta cobertura suele expresarse en porcentaje. Cuando se logra una cobertura completa de sucursales del 100%, significa que todas las sucursales en el código, tanto condicionales como

incondicional: se ejecutó durante las pruebas al menos una vez. Esto significa que hemos probado todas las transiciones directas posibles entre declaraciones en el código.

Ejemplo Consideremos nuevamente el código de ejemplo de la Sección. [4.3.1](#):

```
1. ENTRADA x, y // dos números naturales 2. SI (x > y) ENTONCES 3. z := x - y
MÁS

4.      z := y - x 5. SI
(x > 1) ENTONCES 6.
      z := z*2
7. REGRESAR z
```

En este código, tenemos ocho ramas:

1 → 2, 2 → 3, 2 → 4, 3 → 5, 4 → 5, 5
→ 6, 5 → 7, 6 → 7.

El caso de prueba TC1 (x = 2, y = 11, resultado esperado: 18) cubre las siguientes ramas:

- 1 → 2 (incondicional) • 2 → 4
- (condicional) • 4 → 5
- (incondicional) • 5 → 6
- (condicional) • 6 → 7
- (incondicional)

y logra una cobertura de $5/8 = 67,5\%$.

El caso de prueba TC2 (x = 8, y = 1, resultado esperado: 14) cubre las siguientes ramas:

- 1 → 2 (incondicional) • 2 → 3
- (condicional) • 3 → 5
- (incondicional) • 5 → 6
- (condicional) • 6 → 7
- (incondicional)

y logra también una cobertura de $5/8 = 67,5\%$.

Los dos casos juntos logran $7/8 = 87,5\%$ de cobertura de sucursales, porque todos las ramas están cubiertas excepto una, 5 → 7.

Ejemplo Considere el código y su CFG de la figura [4.13](#).

Un solo caso de prueba con entrada y = 3 logra una cobertura de declaración del 100%. Esto se debe a que el control pasará por las siguientes declaraciones (el valor de la variable x asignado en la declaración 4 y la decisión verificada en la declaración 3 se dan entre paréntesis):

1 → 2 (x := 1) → 3 (1 < 3) → 4 (x := 2) → 3 (2 < 3) → 4 (x := 3) → 3 (3 < 3) → 5.

```

1.  INPUT  y
2.  x=1;
3.  WHILE (x < y) DO
4.    x=x+1;
5.  PRINT ("End of the loop")
    END

```

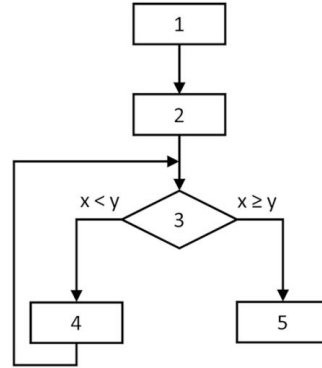


Fig. 4.13 El código fuente con un bucle while y su CFG

El mismo caso de prueba también logra una cobertura de sucursales del 100%. Hay cinco ramas en el código: $1 \rightarrow 2$, $2 \rightarrow 3$, $3 \rightarrow 4$, $3 \rightarrow 5$ y $4 \rightarrow 3$. Según el flujo de control, las ramas se cubrirán secuencialmente de la siguiente manera:

$1 \rightarrow 2$ (incondicional) $2 \rightarrow 3$ (incondicional) $3 \rightarrow 4$ (condicional) $4 \rightarrow 3$ (incondicional) $3 \rightarrow 4$ (condicional, cubierto anteriormente) $4 \rightarrow 3$ (incondicional, cubierto antes) $3 \rightarrow 5$ (condicional)

Ejemplo Considere el código de la figura 4.14a.

Su CFG, en el que cada vértice corresponde a un único enunciado, se muestra en la figura 4.14b. Hay nueve ramas en este gráfico:

• $1 \rightarrow 2$ (incondicional) •
 $2 \rightarrow 3$ (incondicional) • $3 \rightarrow 4$ (condicional) • $3 \rightarrow 5$ (condicional) • $5 \rightarrow 6$ (condicional) • $5 \rightarrow 9$ (condicional) • $6 \rightarrow 7$ (incondicional) • $7 \rightarrow 8$ (incondicional) • $8 \rightarrow 5$ (incondicional)

Para lograr una cobertura de sucursales del 100%, necesitamos al menos dos casos de prueba, por ejemplo:

CT1: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$

CT2: $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 5 \rightarrow 9$

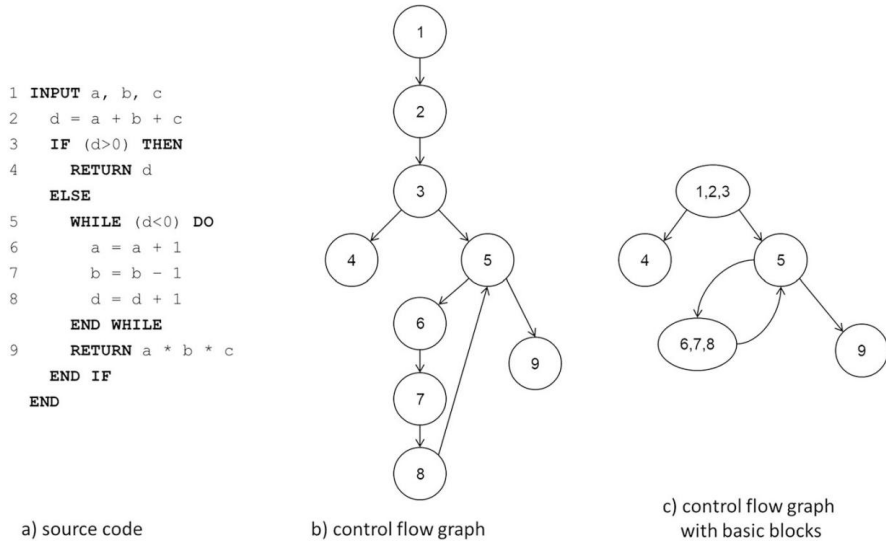


Fig. 4.14 Dos CFG para el mismo código

TC1 cubre tres de nueve sucursales, por lo que logra una cobertura de sucursales de $3/9 \approx 33,3\%$. TC2 cubre ocho de las nueve sucursales, por lo que logra una cobertura de sucursales de $8/9 \approx 89\%$. Las dos pruebas juntas cubren las nueve ramas, por lo que el conjunto de pruebas {TC1, TC2} logra una cobertura completa de las ramas, $9/9 = 100\%$.

En ocasiones, un CFG se dibuja de manera que incluya varias proposiciones en un único vértice, constituyendo lo que se denomina bloque básico. Un bloque básico es una secuencia de declaraciones tal que cuando una de ellas se ejecuta, todas las demás deben ejecutarse. Un CFG modificado de esta manera es más pequeño y más legible (no contiene "largas cadenas de vértices"), pero su uso afecta la medida de cobertura, ya que habrá menos aristas que en un CFG sin bloques básicos. Por ejemplo, la gráfica de la figura 4.14c es equivalente a la gráfica de la figura 4.14b, pero tiene sólo cinco aristas. El TC1 mencionado anteriormente logra en este caso $1/5 = 20\%$ de cobertura de sucursales, mientras que el TC2 logra $4/5 = 80\%$ de cobertura.

Si la cobertura se mide directamente sobre el código, equivale a la cobertura calculada para un CFG, en el que cada vértice representa una sola instrucción, ya que las ramas representan el flujo directo de control entre instrucciones individuales, no entre sus grupos (bloques básicos).

4.3.3 El valor de las pruebas de caja blanca

La prueba de sentencias se basa en la siguiente hipótesis de error: si hay un defecto en el programa, éste debe localizarse en una de las sentencias ejecutables. Y si logramos una cobertura del 100% del extracto, entonces estamos seguros de que un extracto con un defecto

Fue ejecutado. Esto, por supuesto, no garantizará que se produzca un fallo, pero al menos creará esa posibilidad. Considere un ejemplo simple de código que toma dos números como entrada y devuelve su multiplicación (si el primer número es positivo) o devuelve el primer número (si el primer número no es positivo). El código tiene el siguiente aspecto:

```
1. ENTRADA x, y 2. SI  
x>0 ENTONCES  
3.      x:= x+y  
4. REGRESAR x
```

En este programa, hay un defecto que involucra el uso incorrecto del operador de suma en lugar de multiplicación en la línea 3. Para los datos de entrada $x = 2$, $y = 2$, el programa revisará todas las declaraciones, por lo que cubrirá completamente la declaración al 100%. se logrará. En particular, la declaración 3 se ejecutará incorrectamente, pero dicho sea de paso, $2 * 2$ es igual a $2 + 2$, por lo que el resultado devuelto será correcto a pesar del defecto. Este sencillo ejemplo muestra que la cobertura de declaraciones no es una técnica sólida y que vale la pena utilizar otras más potentes, como las pruebas de cobertura de sucursales.

En cambio, en las pruebas de rama, la hipótesis del error es la siguiente: si hay un defecto en el programa, provoca un flujo de control erróneo. En una situación en la que hemos logrado una cobertura de sucursal del 100%, debemos haber ejercido al menos una vez una transición incorrecta (por ejemplo, el valor de decisión debería ser verdadero, pero era falso), para que el programa ejecute las declaraciones incorrectas, y esto posiblemente resultará en una falla.

Por supuesto, como en el caso de las pruebas de declaraciones, la cobertura total de la rama no garantiza que se produzca una falla, incluso si el programa toma el camino equivocado. En nuestro ejemplo, cuando agregamos un segundo caso de prueba con $x = 0$, $y = 5$, cubriremos adicionalmente la rama $2 \rightarrow 3$. Los dos casos de prueba juntos logran una cobertura de rama del 100%, pero aún así, el defecto en el código desaparecerá. no se identificará: los resultados reales en ambos casos de prueba serán exactamente los mismos que los resultados esperados.

Sin embargo, existe una relación importante entre las pruebas de declaraciones y las pruebas de rama:

Lograr una cobertura del 100% en sucursales garantiza el logro de una cobertura del 100% en el estado de cuenta.

En jerga científica, decimos que la cobertura de sucursal incluye la cobertura de declaración. Esto significa que cualquier conjunto de prueba que logre una cobertura de sucursal del 100%, por definición, también logra una cobertura de estado de cuenta del 100%. Por lo tanto, en tal caso no necesitamos verificar la cobertura del estado de cuenta por separado. Sabemos que debe ser al 100%.

La relación de subsunción inversa no es cierta: lograr una cobertura del 100% del estado de cuenta no garantiza que se logre una cobertura del 100% de las sucursales. Para demostrar esto, consideremos el fragmento de código proporcionado anteriormente. Este programa tiene cuatro declaraciones y cuatro ramas: $1 \rightarrow 2$, $2 \rightarrow 3$, $2 \rightarrow 4$ y $3 \rightarrow 4$. Para los datos de entrada $x = 3$, $y = 1$, el programa revisará las cuatro declaraciones, 1, 2, 3 y 4, por lo que este caso de prueba logrará una cobertura de declaración del 100%. Sin embargo, no hemos cubierto todas las ramas.

después de la ejecución de la sentencia 2, el programa pasa a la sentencia 3 (porque $3 > 0$), por lo que no cubre la rama $2 \rightarrow 4$ (que se ejecutaría en el caso de que la decisión en la sentencia 2 fuera falsa). Entonces, la prueba para $x = 3$, $y = 1$ logra una cobertura del 100% del estado de cuenta pero solo del 75% de cobertura de la sucursal.

Tenga en cuenta también que cada declaración de decisión es una declaración y, como tal, es parte del conjunto de declaraciones ejecutables. Por ejemplo, en el código anterior, la línea 2 es una declaración (y se trata como tal cuando utilizamos la técnica de cobertura de declaraciones). Entonces podemos decir que la cobertura del 100% de la declaración garantiza la ejecución de cada decisión en el código, pero no podemos decir que garantiza el logro de cada resultado de cada decisión (y por lo tanto no podemos decir que garantiza la cobertura de todas las ramas del código).

La ventaja clave de todas las técnicas de prueba de caja blanca es que durante las pruebas se tiene en cuenta toda la implementación del software, lo que facilita la detección de defectos incluso cuando la especificación del software no es clara o está incompleta. El diseño de la prueba es independiente de la especificación. Una debilidad correspondiente es que si el software no implementa uno o más requisitos, las pruebas de caja blanca pueden no detectar defectos resultantes que impliquen falta de funcionalidad [50].

Las técnicas de prueba de caja blanca se pueden utilizar en pruebas estáticas. Son muy adecuados para revisiones de código que aún no está listo para su ejecución [51], así como pseudocódigo y otra lógica de alto nivel que se puede modelar utilizando un diagrama de flujo de control.

Si sólo se realizan pruebas de caja negra, no hay forma de medir la cobertura real del código. Las pruebas de caja blanca proporcionan una medida objetiva de la cobertura y proporcionan la información necesaria para permitir que se generen pruebas adicionales para aumentar esa cobertura y posteriormente aumentar la confianza en el código.

4.4 Técnicas de prueba basadas en la experiencia

FL-4.4.1 (K2) Explicar los errores de adivinación.

FL-4.4.2 (K2) Explicar las pruebas exploratorias.

FL-4.4.3 (K2) Explicar las pruebas basadas en listas de verificación.

Además de las técnicas de prueba basadas en especificaciones y de caja blanca, existe una tercera familia de técnicas: técnicas de prueba basadas en la experiencia. Esta categoría contiene técnicas que se consideran menos formales que las discutidas anteriormente, donde la base de las acciones del evaluador siempre fue algún modelo formal (un modelo de dominio, lógica, comportamiento, estructura, etc.).

Las técnicas de prueba basadas en la experiencia utilizan principalmente el conocimiento, las habilidades, la intuición y la experiencia de los evaluadores trabajando con productos similares o incluso con el mismo producto anteriormente. Este enfoque facilita que el evaluador identifique fallas o defectos que serían difíciles de detectar utilizando técnicas más estructuradas. A pesar de las apariencias, los evaluadores suelen utilizar técnicas de prueba basadas en la experiencia. Sin embargo, es importante tener en cuenta que la eficacia de estas técnicas, al

su propia naturaleza dependerá en gran medida del enfoque y la experiencia de cada evaluador. El programa de estudios de Foundation Level enumera los siguientes tres tipos de técnicas de prueba basadas en la experiencia, que analizaremos en las siguientes secciones:

• Error al adivinar •

Pruebas exploratorias •

Pruebas basadas en listas de verificación

4.4.1 Error al adivinar

Descripción de la técnica La adivinación

de errores es quizás conceptualmente la más simple de todas las técnicas de prueba basadas en la experiencia. No está asociado a ninguna planificación previa, ni es necesario gestionar las actividades del evaluador asociadas al uso de esta técnica. El evaluador simplemente predice los tipos de errores posiblemente cometidos por los desarrolladores, defectos o fallas en un componente o sistema haciendo referencia, entre otras cosas, a los siguientes aspectos:

• Cómo ha funcionado hasta el momento el componente o sistema bajo prueba (o su versión anterior que ya esté funcionando en producción). •

Cuáles son los errores típicos que conoce el evaluador, cometidos por desarrolladores, arquitectos, analistas y otros miembros del equipo de producción. • Conocimiento

de Fallos que han ocurrido previamente en aplicaciones similares probadas.
por el evaluador o que el evaluador ha oído hablar

Los errores, defectos y fallas en general pueden estar relacionados con:

• Entrada (p. ej., entrada válida no aceptada, entrada no válida aceptada, parámetro incorrecto valor, parámetro de entrada faltante)

• Salida (p. ej., formato de salida incorrecto, salida incorrecta, salida correcta en el momento equivocado, salida incompleta, salida faltante, errores gramaticales o de puntuación) • Lógica (p. ej., faltan casos a considerar, casos duplicados a considerar, operador lógico no válido, falta condición, iteración de bucle no válida) • Cálculos (p. ej., algoritmo incorrecto, algoritmo

ineficaz, cálculo faltante, operando no válido, operador no válido, error de horquillado, precisión insuficiente del resultado, función incorporada no válida)

• Interfaz (p. ej., procesamiento incorrecto de eventos desde la interfaz, fallas relacionadas con el tiempo en el procesamiento de entrada/salida, llamada a una función incorrecta o inexistente, parámetro faltante, tipos de parámetros incompatibles) • Datos (p. ej., inicialización,

definición o declaración de una variable, acceso incorrecto a una variable, valor incorrecto de una variable, uso de una variable no válida, referencia incorrecta a un dato, escala o unidad incorrecta de un dato, dimensión incorrecta de los datos, índice incorrecto, tipo incorrecto de una variable, rango incorrecto de una variable, "error por uno" (ver Apartado 4.2.2))

Fig. 4.15 Calculadora de IMC
(fuente: calculadorasworld.com)

The image shows a web-based BMI calculator interface. At the top, it says "BMI CALCULATOR". Below this, there are two radio buttons: "Imperial" (unselected) and "Metric" (selected). There are two input fields: "Height" with the value "180" and a unit dropdown set to "cm", and "Weight" with the value "80" and a unit dropdown set to "kg". Below the input fields, a green box displays the result: "Your BMI is 24.69 (normal BMI)". At the bottom, there is a blue "Reset" button. The footer of the application says "Powered by CalculatorsWorld.com".

Existe una variación más organizada y metódica de esta técnica, llamada ataque de falla o ataque de software. Este enfoque implica la creación de una lista de posibles errores, defectos y fallas. El evaluador analiza la lista punto por punto e intenta hacer que cada error, defecto o falla sea apropiadamente visible para el objeto bajo prueba.

Muchos ejemplos de este tipo de ataques se describen en [52-54]. Esta técnica se diferencia de las demás en que su punto de partida es un evento "negativo", un defecto o falla específica, en lugar de algo "positivo" a verificar (como el dominio de entrada o la lógica de negocios del sistema).

Un evaluador puede crear listas de errores, defectos y fallas basadas en su propia experiencia; entonces serán las más efectivas. También pueden utilizar varios tipos de listas que están disponibles públicamente, por ejemplo, en Internet.

Ejemplo El evaluador está probando un programa para calcular el IMC (índice de masa corporal), que toma dos valores del usuario como entrada, peso y altura, y luego calcula el IMC. La interfaz de la aplicación se muestra en la Fig. 4.15.

El probador utiliza un ataque de falla utilizando la siguiente lista de defectos y fallas:

1. La ocurrencia de un desbordamiento de un campo de formulario.
2. La ocurrencia de la división por cero.
3. Forzar la aparición de un valor no válido.