Diseños de Casos de Pruebas: *Black Box* y *White Box*

Pruebas de Sistemas

UNIVERSIDAD SIGLO 21

Técnicas de diseño de casos de pruebas

En esta unidad se desarrollarán, una a una, las técnicas para aplicar en el diseño de casos de pruebas. “Estas técnicas quedan enmarcadas dentro del ***enfoque dinámico***, es decir, que apuntan a ejecutar una parte del sistema o el sistema completo para determinar si funciona según lo esperado”. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Existen diversos factores, para llevar adelante el ciclo de pruebas. Ellos son:

- *“Nivel de prueba*: esto marca el objetivo de la prueba, qué es lo que se espera alcanzar. Se recuerdan que los niveles son: nivel de componentes, nivel de sistemas, nivel de integración y nivel de aceptación”. (Uninotas, 2017, https://bit.ly/2KGaC4d).

- *“Enfoque de la prueba*, si es estático o dinámico” (Uninotas, 2017, https://bit.ly/2KGaC4d). Aunque en un proceso de desarrollo es claro que deben existir puntos de pruebas de ambos enfoques.

- *Objeto de prueba:* si es diseño, requerimientos, código o sistema lo que se está probando. Algunos objetos determinan un enfoque en particular y único, y otros objetos pueden aplicar los dos enfoques dependiendo lo que se desee evaluar. Por ejemplo, si el objeto bajo pruebas es el diseño al momento del diseño, previo a la codificación, toma el enfoque estático a través de una revisión. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Sin embargo, si el objeto es el código, puede tomar un enfoque estático a través de una revisión o inspección (por dar un tipo) o también el enfoque dinámico en cualquiera de los niveles de pruebas.

- *“Estrategia de prueba:* que está ligada al objeto que se usa para derivar los casos de pruebas, pudiendo ser de caja blanca o de caja negra”. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Basadas en especificaciones (Black Box)

El diseño de casos de pruebas de caja negra o también denominado pruebas funcionales, pruebas basadas en las especificaciones, o pruebas de comportamiento son las pruebas que se derivan de los requerimientos, las especificaciones, o directamente de la interfaz de usuario. (Uninotas, 2017, https://bit.ly/2KGaC4d)

"La selección de los casos de pruebas de caja negra está basada en un análisis de las especificaciones de un componente sin referenciar a detalles internos de la estructura lógica" (Uninotas, 2017, https://bit.ly/2KGaC4d).

Se denominan así porque el probador observa el objeto de prueba como una caja negra.

Puntualizando las características, se destaca que:

- Interesa qué hace el sistema y no el cómo lo hace.
- Los casos de pruebas están basados en la especificación de los requerimientos, sea funcional o no funcional.
- Se conducen a nivel de interfaz, probando el comportamiento de la siguiente manera: ante una entrada tal, una salida tal.
- Prueba todas las combinaciones de entrada y salida de datos
- La cobertura alcanzada con las pruebas es medida en porcentaje o cantidad de funcionalidad abarcada.

Este tipo de pruebas apuntan a verificar la corrección y la completitud de una función respondiendo a:

- Completitud: ¿están disponibles **todas** las funciones especificadas en el módulo?
- Correctitud: ¿son **correctos** los resultados de las funciones ejecutadas?

Roger S. Pressman (2006) da una lista de tipos de errores que pueden encontrarse por medio de las pruebas de caja negra:

1. Errores del comportamiento.

2. Errores en la interfaz.
3. Errores en las estructuras de datos o acceso a la base de datos.
   Errores en las funciones que desempeña el sistema, o funciones que están faltando. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Siempre se trabaja bajo la premisa de que la ejecución de casos de prueba no deberían ser ejecutados de manera redundante, pero, sin embargo, debe ser integral/completo. Esto significa probar lo menos posible, pero probar tanto como sea necesario.

Las técnicas de diseño de casos de pruebas vienen a colaborar con esto, ayudando al diseño de casos mínimo y necesario para las garantías de pruebas deseadas.

Partición de equivalencias

La partición en clases de equivalencia es lo que la mayoría de los probadores hacen de forma intuitiva.

Es un método sistemático que identifica clases de pruebas representativas a partir de valores de entrada y valores de salida de un programa a las que denomina clases de equivalencias.

Las reglas que se aplican para esta agrupación en clases de equivalencia de los rangos de valores son:

1. Todos los valores para los cuales se espera que el programa tenga un comportamiento común es una clase de equivalencia.
2. Las clases de equivalencia no pueden superponerse.
3. Las clases de equivalencia no pueden presentar ningún salto.

Una vez identificadas las clases de equivalencia, las mismas pueden ser divididas de forma adicional en:

- Clase de equivalencia válida: todos los valores dentro del rango de definición.

- Clase de equivalencia no válida: se distinguen dos casos para valores fuera del rango de definición. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Las clases de equivalencia pueden consistir en un rango de valores, por ejemplo: 0 < x < 10, o en un valor aislado, por ejemplo: x = Verdadero.

Como paso siguiente, una vez que se tienen todas las clases de equivalencia identificadas y considerando las inválidas también, se elige un valor representativo por cada uno de los rangos, o sea, uno de cada clase.

Cualquier valor que pertenezca a la clase puede ser un representante, aunque los óptimos son:

- Valores característicos, es decir, utilizados con frecuencia.
- Valores problemáticos, es decir, que son sospechosos de producir fallos.
- Valores límites, que está en la frontera del rango.

Se arman así los casos de pruebas que combinan los representantes de cada clase de equivalencia, utilizando un único representante de cada clase, pero con las siguientes consideraciones:

- Los representantes de clases de equivalencia válidas se pueden combinar.
- Los representantes de clases de equivalencia no válidas no se pueden combinar.
- Los representantes de una clase de equivalencia no válida sólo se puede combinar con otros representantes de clases de equivalencia válidas.
- Los representantes de clases de equivalencia inválidas deben combinarse siempre con los mismos valores de otras clases de equivalencia válidas. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Las clases de equivalencia se pueden identificar también sobre las salidas, valores internos, valores relacionados con el tiempo, y para entradas a través de la interfaz del sistema.

El beneficio de esta técnica es que, de una pequeña cantidad de casos de prueba se puede esperar un valor de cobertura específico.

Ejemplo

Se desean realizar pruebas sobre un programa utilizado por una empresa de transporte para calcular la tarifa de cada boleto según el trayecto y la edad del pasajero. Dicha empresa solo opera viajes entre Córdoba y Buenos Aires ($580) y entre Córdoba y Mendoza ($350). Los menores de 5 años no pagan, y los menores de 12 años pagan la mitad.

Analizando la especificación se tienen como datos de entrada:

- la edad del pasajero (e), para el que se debería admitir sólo valores enteros de dos dígitos;
- el trayecto (T) seleccionado de una lista de dos posibles.

En una pantalla se mostrará el importe a pagar.

Si idealmente se pensara en una prueba exhaustiva, deberían probarse cada uno de los casos entre 0 y 99, lo que implica 100 casos de pruebas más todos aquellos que lleven a confirmar que solo se admiten enteros, no letras ni blancos ni espacio ni más de dos dígitos, entre otros.

Aplicando la técnica se deben definir los rangos de edades posibles, entre 0 y 99, en función del comportamiento que toman en el sistema. De ese rango de edad posible, se tienen las siguientes clases de equivalencia válidas (CE):

1. una que va de 0 a 4;
2. otra que va de 5 a 11;
3. y la última que va de 12 a 99 .

Tabla 1: Equivalencias válidas

$0 \leq e \leq 4$	$5 \leq e \leq 11$	$12 \leq x \leq 99$
CEV1	CEV2	CEV3

Fuente: elaboración propia

Con base en ellas, se definirán las clases de equivalencias inválidas, que generalmente son rangos por encima y por debajo de los rangos de las clases de equivalencia válidas o, para el caso de números enteros, probar letras o caracteres como asteriscos o blancos. El ejemplo de aplicación de la técnica se va a centrar en los rangos inválidos únicamente.

Tabla 2: Equivalencias inválidas

e < 0	0 ≤ e ≤ 99	e > 99
CEI1	CEV1-2-3	CEI2

Fuente: elaboración propia

La técnica indica que una vez identificada las clases de equivalencia, como ya se realizó para este ejemplo, se debe tomar un representante de cada clase, sea válida o inválida.

Tabla 3: Casos aplicables

CEI1	CE1	CE2	CE3	CEI2
-5	3	8	15	120

Fuente: elaboración propia

Esto indica que son 5 los casos aplicables para probar el funcionamiento de este sistema; además, se podrían probar. Se puede apreciar entonces que el mayor beneficio de esta técnica es que se reducen considerablemente los casos de pruebas.

Análisis de valores frontera

Luego de aplicar la técnica de partición de equivalencia en el ejemplo anterior, ¿qué sucedería si en el programa para el primer rango, al preguntar por la edad (e), en vez de preguntar por e ≤ 4 (edad menor o igual a 4), se hubiera preguntado por error si e < 4? No sabríamos el comportamiento que tiene para e = 4, y con las pruebas surgidas de la técnica anterior no es posible detectar el error.

La técnica de análisis de valores fronteras viene a complementar la partición de equivalencia donde, además de seleccionar cualquier elemento como representativo de una clase de equivalencia, se seleccionan los bordes de cada clase o rango. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Ambas técnicas sugieren la selección de representantes. Así como la partición en clases de equivalencia evalúa un representante, es decir, un

valor típico de la clase de equivalencia, el análisis de valores frontera evalúa los valores límite o frontera y su entorno.

Y lo hace bajo el siguiente esquema:

- De cada rango de valores o clase de equivalencia válidas identificadas, toma los valores límites, es decir, el valor mínimo y el valor máximo. Por ejemplo, dado el rango de valores donde:

  0,00 ≤ x ≤ 100,00 (Uninotas, 2017, https://bit.ly/2KGaC4d).

Se tiene que el valor límite inferior o mínimo del rango es 0,00 y el valor límite superior o máximo del rango es 100,00.

- Respecto del valor mínimo, toma el anterior y el posterior. El anterior sería igual al valor mínimo menos un incremento. Por ejemplo, si es un valor entero, sería el valor límite -1. El posterior sería igual al valor mínimo más un incremento. Si es un valor entero, sería el valor límite +1.

  Pero para el caso de 0,00 sería -0,01 (negativo) y 0,01 (positivo).

- Respecto del valor máximo, también toma el anterior y el posterior de la misma manera.

  Para el caso de 100,00 sería 99,99 y 100,01.

  Entonces se tiene: -0,01 / 0,00 / 0,01 / 99,99 / 100,00 / 100,01

  El valor posterior del valor límite inferior o mínimo del rango, es un elemento que puede ser uno de los representativos de la clase de equivalencia. Lo mismo sucede con el anterior del valor límite superior o máximo del rango. (Uninotas, 2017, https://bit.ly/2KGaC4d).

En el ejemplo se podría asumir que 0,01 tiene el mismo comportamiento que 0,00 y que 99,99 tiene el mismo comportamiento que 100,00.

"Los casos de pruebas que se agregan a los de partición de equivalencias por esta técnica serían dos, solamente el -0,01 y el 100,01". (Uninotas, 2017, https://bit.ly/2KGaC4d)

Las aclaraciones que corresponde hacer son:

- hacer lo mismo para los valores límite de las clases de equivalencia no válidas tiene poco sentido;
- para los rangos de valores o clases de equivalencia definidas como un conjunto de valores, como por ejemplo, *soltero, casado, divorciado, viudo*, en general, no es posible definir los valores límites;
- un error de programación que genera un operador de comparación erróneo solo será detectado con los dos valores anterior y posterior del límite.

Diagrama de transición de estados

Las dos técnicas presentadas sólo tienen en cuenta el comportamiento del sistema en términos de datos de entrada y datos de salida.

Para las pruebas sobre objetos de prueba que pueden tomar diferentes estados, se recomienda otro tipo de técnica, donde se tiene en cuenta que el resultado de acciones que ocurrieron en el pasado, causan que el objeto de prueba adquiera un determinado estado.

Cuando se tiene una especificación donde se puede modelar el comportamiento del sistema a través de estados que toma un objeto (y ese objeto es objeto de prueba), se usa esta técnica de diagramas de transición de estado. (Uninotas, 2017, https://bit.ly/2KGaC4d)

> En realidad puede ser tomada como una técnica de modelado que se la extiende al proceso de pruebas para derivar de ahí los casos de prueba que validen el diagrama.
>
> El proceso consiste en tomar el análisis de las especificaciones para modelar el comportamiento por transición de estados, si es que no llega así a pruebas. Para ello, se deben identificar los estados que pueden tomar los objetos, las transiciones entre los estados, los eventos que provocan los cambios de estado (es decir las transiciones) y las otras acciones que puedan resultar de esas transiciones.

A partir de este diagrama se construye un árbol de transición, donde el estado inicial es la raíz del árbol, y para cada estado que pueda ser alcanzado desde el estado inicial se crea un nodo que está conectado a la raíz por una rama y así sucesivamente con todos los estados. El proceso termina cuando cada rama llega a un estado final (hoja) o bien cuando el mismo estado ya es parte del árbol.

Construido esto, restan definir los casos de pruebas, donde por cada camino desde la raíz a una hoja representa un caso de prueba. El criterio de salida para esta técnica es que **todo estado** ha sido alcanzado (Uninotas, 2017, https://bit.ly/2KGaC4d).

Así como la base para la generación de los casos de pruebas es otra, también son diferentes los tipos de defectos a los que apunta.

Los defectos potenciales a detectar a través de estas técnicas son:

- Números incorrectos de estados.
- Transición incorrecta para una entrada dada.
- Salida incorrecta para una entrada dada.
- Conjuntos de estados que se hacen equivalentes inadvertidamente.
- Conjuntos de estados que se dividen para crear duplicados no equivalentes.
- Conjuntos de estados que se han transformado en inalcanzables. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Caso de uso

Para explicar esta técnica, se debe tener en claro qué es un caso de uso y sus diagramas relacionados, que si bien no es tema de esta materia, a los efectos de lo que se necesita para las pruebas, lo explicaremos brevemente.

"Los casos de uso son elementos del Lenguaje Unificado de Modelado (UML[1])" (Uninotas, 2017, https://bit.ly/2KGaC4d). El diagrama de casos de

[1] Unified Modeling Language.

uso es uno de los diferentes tipos de diagramas utilizado por UML, que "describe el comportamiento del sistema, no la secuencia de eventos, y muestra la reacción del sistema desde el punto de vista del usuario" (Uninotas, 2017, https://bit.ly/2KGaC4d).

Cada caso de uso describe la interacción de todos los actores involucrados y conduce a un resultado final por parte del sistema.

"Todo caso de uso tiene precondiciones que deben ser cumplidas con el objeto de ejecutar el caso de uso de forma satisfactoria. También, tiene poscondiciones que describen al sistema tras la ejecución del caso de uso" (Uninotas, 2017, https://bit.ly/2KGaC4d). Cada caso de uso describe una cierta tarea (interacción usuario-sistema) a través de un curso normal y uno o varios cursos alternativos.

El Comité Internacional de Certificaciones de Pruebas de Software (ISTQB) (2011) incorpora a ésta, como una técnica más para únicamente aquellos sistemas que cuenten con modelado de caso de uso.

Esta técnica consiste en tomar como fuente los casos de uso para definir los casos de prueba, donde cada alternativa del caso de uso corresponde, por lo menos, a un caso de prueba separado. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Normalmente, la información aportada por un caso de uso no tiene suficiente detalle para definir casos de prueba de forma directa. Es necesario una serie de datos agregados (datos de entrada, resultados esperados) para construir/desarrollar un caso de prueba.

El proceso que experimentalmente se lleva adelante en esta técnica es el siguiente:

1. "Identificar todos los escenarios presentes en el caso de uso y asignarle un nombre a cada uno" (Uninotas, 2017, https://bit.ly/2KGaC4d). Debe haber un escenario correspondiente al flujo normal del caso de uso, uno con cada uno de los flujos alternativos y otros con cada uno de los flujos que generan condiciones de error.

2. "Identificar aquellos escenarios que resulten redundantes y eliminarlos" (Uninotas, 2017, https://bit.ly/2KGaC4d). Se llama escenario redundante a aquel que válida los mismos aspectos que ya se han validado en otro escenario.

3. "Para cada uno de esos escenarios, se identificarán y documentarán las condiciones de ejecución" (Uninotas, 2017,

https://bit.ly/2KGaC4d). Una condición de ejecución es cada uno de los valores con los cuales se ejecutará el caso de prueba.

4. “Se toma a cada escenario que queda luego de eliminar los redundantes, y ése será por lo menos, un caso de prueba” (Uninotas, 2017, https://bit.ly/2KGaC4d).

5. “Confeccionar una plantilla para registros de los casos de prueba con su lista de chequeo” (Uninotas, 2017, https://bit.ly/2KGaC4d).

6. “Verificar lista de chequeo para validar en cuáles otros escenarios diferentes a los directamente relacionados con el caso de uso” (Uninotas, 2017, https://bit.ly/2KGaC4d) es indispensable probar la aplicación.

Las planillas sugeridas aquí pueden tener la forma que crea conveniente el responsable de este proceso, de manera que se ajusten mejor para el registro, seguimiento y control.

Los casos de prueba derivados por esta técnica son apropiados para las pruebas de aceptación y pruebas de sistema, dado que cada caso de uso describe un escenario de usuario para probar. Sin embargo, es importante recordar que sólo se podrá aplicar si las especificaciones del sistema se encuentran disponibles en UML.

La desventaja fundamental, y que no hay que descuidar, es que existe una nula obtención de casos de prueba adicionales más allá de la información aportada por el caso de uso.

Siempre se recomienda, al momento de la identificación de las condiciones de ejecución, combinar con otras técnicas de prueba basadas en la especificación (caja negra).

Tablas de decisión & gráficos causa-y-efecto

Como se mencionó anteriormente, la partición en clases de equivalencia y el análisis de valores frontera tratan entradas en condiciones aisladas, sin tener en cuenta que una condición de entrada puede tener efectos solo en combinación con otras condiciones de entrada.

> Estas técnicas tienen en cuenta el efecto de dependencias y combinaciones. Si se toma el conjunto completo de las combinaciones (matemáticamente hablando) de todas las clases de equivalencia de entrada produce, normalmente, a un número muy alto de casos de prueba. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Aquí es donde son necesarios los gráficos causa-y-efecto y las tablas de decisión obtenidas a partir de esos gráficos para reducir de forma sistemática a un subconjunto de casos de pruebas.

Estas técnicas no son frecuentemente aplicadas debido a que utilizan un lenguaje formal.

> Metodológicamente se sigue el siguiente proceso:
>
> 1. Se toma la especificación, que normalmente es informal y se la traduce en un gráfico causa-y-efecto de un objeto de prueba a un lenguaje formal.
> 2. Para ello, se identifican los efectos que afectan al objeto de prueba y que se remontan a sus respectivas causas. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Como ejemplo, se tiene: para tal efecto, se dió o causa A o causa B; o, como segundo ejemplo, se tiene: para tal efecto, se dio causa A y causa B.

> 3. Se construye una tabla de decisión de la siguiente manera: 1- se selecciona un efecto, 2- se retrocede en el diagrama para identificar la causa, 3- cada combinación de causas es representada por una columna en la tabla de decisión.

Cada columna de la tabla conduce a, por lo menos, un caso de prueba.

4. Se analiza cada columna de la tabla detectando las combinaciones de causas idénticas que conducen a efectos distintos, para fusionarlas y formar un único caso de prueba.

   Esta técnica es ventajosa por otorgar una vista sistemática de las combinaciones de entradas que no podrían ser identificadas utilizando otros métodos. Pero tiene la desventaja que el establecimiento de un gran número de causas lleva a resultados complejos, incurriendo en muchos errores en la aplicación de este método. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Basadas en estructura y código (White Box)

"El diseño de casos de pruebas de caja blanca o también denominado pruebas estructurales, o pruebas cristal son las pruebas que se derivan del análisis estructural interno de un componente" (Uninotas, 2017, https://bit.ly/2KGaC4d).

La selección de los casos de pruebas caja blanca está basada en un análisis estructural interno de un componente, en la estructura lógica del código.

En comparación con las características de las técnicas de caja negra, para caja blanca se tiene que:

- En caja negra interesa qué hace el sistema y en caja blanca el cómo lo hace.
- Los casos de pruebas derivados están basados en la estructura lógica del código, enfocados en el código, de manera que el probador debe conocer la estructura interna del código.
- La cobertura alcanzada con las pruebas es medida en porcentaje o cantidad de código abarcado y no solo funcional como es el caso de caja negra.
- Puede emplearse en niveles tempranos de pruebas, ni bien estén generados los primeros componentes de código. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Se deben derivar casos de pruebas que garanticen que cada camino en el código se ha ejecutado al menos una vez, que cada decisión lógica se haya evaluado tanto para verdadero como para falso, y que cada bucle se haya ejecutado en sus límites y dentro de los límites.

Es frecuente que las técnicas de caja blanca requieran del apoyo de herramientas tanto para la especificación de casos de pruebas como para la ejecución de la prueba. El uso de herramientas asegura la calidad de las pruebas e incrementa su eficiencia.

Parecería que la prueba con casos de pruebas basados en la estructura llevaría a un cien por ciento correcto, es decir, si se logra que todo el código se ejecute, implica que se encontrarán todos los errores en el código.

Lograr un código libre de errores, ¿implica lograr un sistema que satisfaga las necesidades de usuario?

Por supuesto que no, un código libre de fallas es sólo eso, un código libre de fallas, pero puede ser un sistema que no satisfaga las necesidades del usuario o puede ser un sistema que no abarca todas las funcionalidades solicitadas. Haciendo pruebas enfocadas en el código solamente, no podríamos detectar las funciones no implementadas.

Para cualquiera de las técnicas de caja blanca que se van a explicar, el probador requiere para la planificación y el diseño *test cases* conocimiento de:

- el lenguaje de programación utilizado;
- la base de datos utilizada;
- el sistema operativo utilizado;
- del mismo código a testear.

Complejidad ciclomática

Antes de entrar en la descripción de cada técnica, brindaremos el concepto de complejidad ciclomática propuesto por Tom McCabe (1976).

> La complejidad ciclomática es una métrica cuantitativa de software de la complejidad lógica de un programa. Define el número de caminos independientes en el conjunto básico. Un camino independiente es cualquier camino a través del programa que introduce al menos un nuevo

conjunto de instrucciones o una nueva condición. (Uninotas, 2017, https://bit.ly/2KGaC4d).

Una vez obtenida la complejidad ciclomática, en pruebas obtenemos una idea del límite máximo de casos de pruebas necesarios para ejecutar todas las instrucciones al menos una vez.

“Roger Pressman (2006) presenta tres maneras de calcular esta métrica y en función de cada una, existe una fórmula” (Uninotas, 2017, https://bit.ly/2KGaC4d). Para esta explicación, se lo hará solamente a través de la gráfica de flujo.

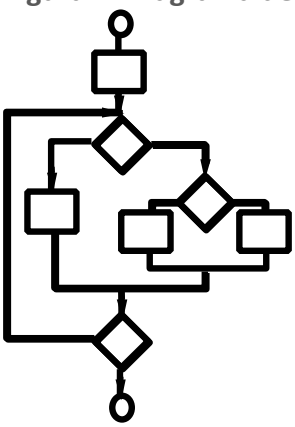
CC = número de decisiones simples + 1

O:

CC = número de áreas cerradas + 1

Dado el siguiente diagrama de flujo, al aplicar la fórmula se tiene:

Figura 1: Diagrama de flujo 1



Fuente: elaboración propia

Al contar las decisiones, se obtiene 3. Si se lo hace contando las áreas cerradas, se obtiene también 3. Por lo que CC = 3 + 1 = 4.

CC = 4 indica que son 4 los casos de pruebas necesarios para probar cada camino, y que cada camino sea ejecutado de una vez.

“Los estudios en la industria de sistemas, han demostrado que cuanto más alto es el valor de la complejidad ciclomática, mayor es la probabilidad de errores” (Uninotas, 2017, https://bit.ly/2KGaC4d).

Las técnicas que se desarrollarán en adelante adoptan el nombre en función del tipo de cobertura que se logra a través de las pruebas.

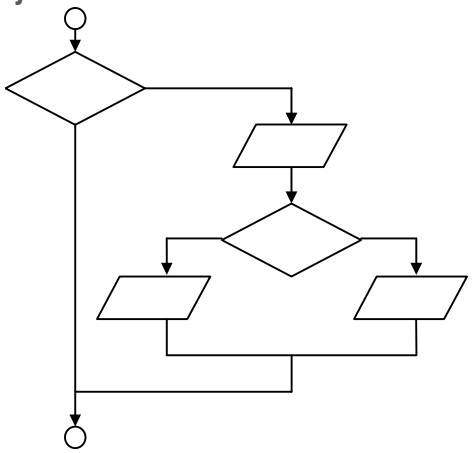
Cobertura de sentencia

El criterio de salida u objetivo de esta técnica "es lograr la cobertura de un porcentaje específico de todas las *sentencias*, es decir, que un porcentaje establecido de sentencias ejecutables haya sido practicado por los casos de prueba derivados" (Uninotas, 2017, https://bit.ly/2KGaC4d). Puede ser aplicada a módulos, clases, elementos de un menú, entre otros. Debe responder a la pregunta ¿qué casos de prueba son necesarios con el objeto de ejecutar todas, o un porcentaje determinado, las sentencias del código existentes?

Dado el siguiente pseudocódigo, se construye el diagrama de flujo que se muestra a la derecha de la página:

Figura 2: Diagrama de flujo 2

```
START
IF v1 > 0 THEN
        DO a1
IF v2 < 5 THEN
                DO a2
        ELSE
                DO a3
        ENDIF
ENDIF
END
```

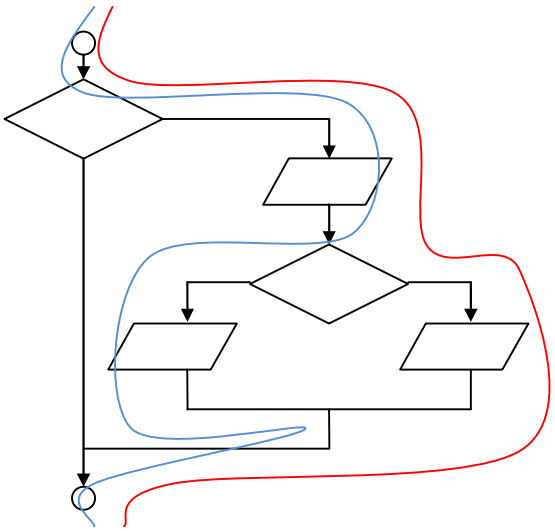


Fuente: elaboración propia

Tanto con el diagrama como con el pseudocódigo se puede ver que este contiene una sentencia IF que contiene a otra sentencia IF. La segunda sentencia IF contiene una instrucción tanto para el THEN como para el ELSE, en cambio, la primera sentencia IF no contiene ELSE.

Esta técnica indica que se debe pensar en los caminos que son necesarios para alcanzar que cada sentencia sea ejecutada al menos una vez.

Figura 3: Diagrama de flujo 3



Fuente: elaboración propia

Así se tiene que son necesarios ***dos*** caminos diferentes a través de este segmento de programa. La primera sentencia IF permite sólo una dirección, la de la derecha, en cambio, la segunda sentencia IF se divide en dos direcciones que hay que comprobar porque ambas tienen sentencias dentro.

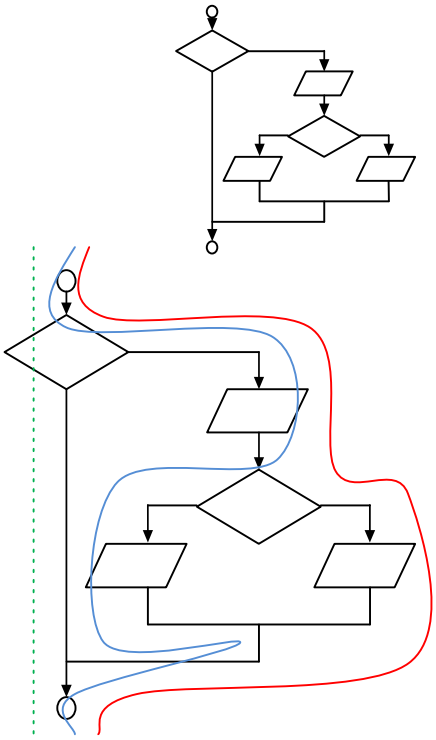
> Dos caminos de prueba serán suficientes para lograr el 100 % de cobertura de sentencia.
>
> Esta técnica detectará el código muerto, es decir, el código constituido por sentencias que nunca se ejecutan, lo que lleva a que no se logre en ese caso el 100 % de cobertura. (Uninotas, 2017, https://bit.ly/2KGaC4d)

Cobertura de decisión

> En lugar de las sentencias, la cobertura de decisión amplía su cobertura y se centra en los caminos del segmento de programa. Todas los caminos deben ser cubiertos al menos una vez, es decir, en un IF, se debe forzar la salida por el THEN y por el ELSE aunque no haya sentencias para ejecutar dentro. (Uninotas, 2017, https://bit.ly/2KGaC4d).

Debe responder a la pregunta ¿qué casos de prueba son necesarios con el objeto de ejecutar todos, o un porcentaje determinado, los caminos del código existentes?

Figura 4: Diagrama de flujo 4



Fuente: elaboración propia.

Aquí es donde entra en uso la CC vista al inicio de las técnicas de caja blanca. Para calcularla, se afirmó que se contaba el número de decisiones o el número de áreas cerradas y se le sumaba 1.

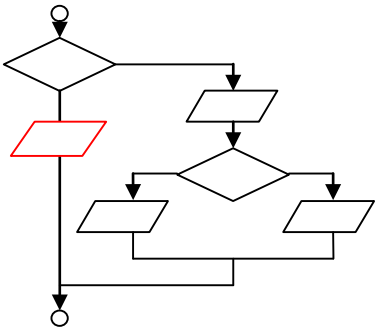
Tomando el mismo ejemplo que el presentado en la cobertura de sentencia, se calcula CC = 2 + 1 = 3.

Retomando el ejemplo anterior, podemos ver que se agregaría un camino más respecto a la cobertura de sentencia, y es el camino que debe recorrer la salida por ELSE del primer IF (camino marcado en verde). En la técnica anterior, no fue necesario ese camino porque no contenía sentencias para ejecutar, pero en esta técnica de decisión sí es necesario. Y si se toma la CC calculada, que indica también 3 caminos.

¿Qué cantidad de caminos son necesarios para alcanzar cobertura de sentencia y cuántos para cobertura de decisión si el ejemplo anterior tuviera una o más sentencias por el camino del ELSE del primer IF?

Nota que en esta situación ambas, cobertura de sentencia y cobertura de decisión, coinciden y es 3.

Figura 5: Diagrama de flujo



Fuente: elaboración propia.

Si bien esta técnica alcanza mayor cobertura que la de sentencia, no es suficiente para probar condiciones complejas, como tampoco es suficiente para probar bucles de forma extensiva, ni considera las dependencias entre los bucles.

Cobertura de condición

Tanto la cobertura de sentencia como la cobertura de decisión son técnicas que se refieren a caminos a través del diagrama de flujo, pero difieren en la cantidad de casos de pruebas para lograr el cien por ciento de cobertura. El punto está en que ambas consideran el resultado final de una condición, a pesar de que la condición resultante puede estar constituida por múltiples condiciones atómicas o simples. Por ejemplo, si se tiene la condición *IF ((a>10) OR (b<100))*, sólo puede ser verdadera o falsa, y el camino del programa para ejecutar depende solamente del resultado final de la condición combinada o múltiple.

Una condición combinada es aquella que está constituida por condiciones simples o atómicas, que se combinan con el uso de operadores lógicos como OR, AND, XOR, entre otros. Una condición simple no contiene operadores lógicos, solo contienen operadores relacionales y el operador NOT (=, >, <, otros).

Muchos fallos debidos a una implementación errónea de las partes de una decisión múltiple pueden no ser detectados, por ello surge la técnica de cobertura de condición donde se tiene en cuenta la complejidad de una condición que esté constituida por múltiples condiciones simples.

Según el nivel de evaluación que se hace sobre la condición combinada es el porcentaje de cobertura que se alcanza. Hay tres tipos:

- cobertura de condición simple;
- cobertura de condición múltiple;
- mínima cobertura de condición.

El ISTQB (2011) plantea tres tipos de cobertura de condición que generan diferentes porcentajes de cobertura.

Cobertura de condición simple

Se basa en que cada condición simple de una sentencia condicional combinada tiene que tomar, al menos una vez, los valores lógicos verdadero y falso.

Si se considera la siguiente condición compuesta o múltiple *(a > 10) OR (b < 100),* los casos de prueba para la cobertura de condición simple podrían ser, por ejemplo:

Tabla 4: Casos aplicables

a = 12 verdadero	b = 120 falso	a > 10 OR b < 100 verdadero
a = 8 falso	b = 50 verdadero	a > 10 OR b < 100 verdadero

Fuente: elaboración propia

Como se puede observar, cada una de las condiciones (a > 10) y (b < 100) tomadas por separado fueron evaluadas para verdadero y para falso. Sin embargo, el resultado combinado es verdadero para ambos casos, ya que con OR se da:

- verdadero OR falso = verdadero
- falso OR verdadero = verdadero

Lo que implica que esta condición no se evaluó para el resultado falso. En un diagrama de flujo, no estaría recorriendo el camino de resultado falso.

Cobertura de condición múltiple

Se seguirá con el ejemplo anterior para comprender la diferencia entre cobertura de condición simple y cobertura de condición múltiple.

Si se considera la siguiente condición compuesta o múltiple *(a > 10) OR (b < 100),* los casos de prueba para la cobertura de condición múltiple deberían ser aquellos que a cada condición tomada por separado la ejecuten por falso y por verdadero, combinando de todas las maneras posibles:

Verdadero – Falso
Verdadero – Verdadero
Falso – Verdadero
Falso – Falso

Tabla 5: Casos aplicables

a = 12 verdadero	b = 120 falso	a > 10 OR b < 100 verdadero
a = 12 verdadero	b = 50 verdadero	a > 10 OR b < 100 verdadero
a = 8 falso	b = 50 verdadero	a > 10 OR b < 100 verdadero
a = 8 falso	b = 120 falso	a > 10 OR b < 100 falso

Fuente: elaboración propia

Como se puede visualizar, con cuatro casos de prueba se puede lograr una cobertura de condición múltiple:

- se han creado todas las combinaciones de los valores verdadero y falso;
- se han logrado todos los posibles resultados de la condición múltiple.

El número de caso de prueba se incrementa de forma potencial, siendo el número de casos de pruebas a realizar. Se calcula de la forma: 2^n, donde *n* es el número de condiciones simples.

Mínima cobertura de condición múltiple

Se trata de una variación del tipo de técnica anterior, donde todas las combinaciones que puedan ser creadas utilizando los resultados lógicos de cada condición simple deben ser parte de las pruebas, solo si el cambio del resultado de una condición simple cambia el resultado de la condición combinada.

Tomando el ejemplo anterior:

Tabla 6: Casos aplicables

a = 12 verdadero	b = 120 falso	a > 10 OR b < 100 verdadero
a = 12 verdadero	b = 50 verdadero	a > 10 OR b < 100 verdadero
a = 8 falso	b = 50 verdadero	a > 10 OR b < 100 verdadero

a = 8 falso	b = 120 falso	a > 10 OR b < 100 falso

Fuente: elaboración propia

Sólo para el caso 2 (verdadero OR verdadero = verdadero) el cambio en cualquier condición simple no generará un cambio en la condición global.

Por más que se provoque que *(a > 10)* resulte falso o que *(b < 100)* resulte falso, no generará que *(a > 10) OR (b < 100)* pase a ser falso. Este es el caso que puede ser omitido y quedan así solo 3 casos de prueba.

Tabla 7: Casos aplicables

A = 12 verdadero	B = 120 falso	a > 10 OR b < 100 verdadero
a = 8 falso	b = 50 verdadero	a > 10 OR b < 100 verdadero
a = 8 falso	b = 120 falso	a > 10 OR b < 100 falso

Fuente: elaboración propia

Luego de haber recorrido los tres tipos de cobertura de condición se puede concluir que la *cobertura de condición simple* es un instrumento débil para probar condiciones múltiples, pues quedan posibles fallos que pueden no ser detectados.

La *cobertura de condición múltiple* es un método mucho mejor, ya que asegura cobertura de sentencia y decisión, sin embargo, tiene como resultado un alto número de casos de prueba: 2^n, y la ejecución de algunas combinaciones no es posible. Por ejemplo ((a > 5) AND (a < 10)) no pueden ser ambas condiciones simples falsas al mismo tiempo.

Finalmente, la *mínima cobertura de condición múltiple* es incluso mejor, debido a que reduce el número de casos de prueba, que puede ser un valor entre $n + 1$ y $2n$.

Cobertura de camino

La cobertura de camino se centra en la ejecución de todos los posibles caminos a través de un programa. Para cobertura de decisión, un solo camino a través de un bucle es suficiente. Para la cobertura de camino hay casos de prueba adicionales:

- Un caso de prueba no entrante al bucle.

- Un caso de prueba adicional para cada ejecución del bucle. (Scrum-QA, 2013, https://bit.ly/2E27iyZ)

- Cada incremento en el contador del bucle añade un nuevo caso de prueba.

"Esto puede conducir a un número muy alto de casos de prueba" (Scrum-QA, 2013, https://bit.ly/2E27iyZ). Cada camino es una vía única desde el inicio al fin del diagrama de flujo de control. Por lo que lograr una cobertura de camino del cien por cien solo se puede en programas muy simples.

¿Qué se obtiene con un 100 % de cobertura de camino?

Pues el 100 % de cobertura de camino incluye 100 % de cobertura de decisión y, a su vez, incluye 100 % de cobertura de sentencia.

Análisis de flujo de datos

Esta técnica se presenta aquí dentro de las de caja blanca, pero puede considerarse dentro del análisis estático con herramientas.

Apunta a detectar anomalías en el flujo de datos con la asistencia de los diagramas de flujo y conjeturas racionales respecto de las secuencias del flujo de datos. Se puede detectar fácilmente la localización exacta de defectos, pero es muy limitado el rango de tipos de defectos.

Una variable cualquiera (por ejemplo, *x*) puede adoptar los siguientes estados a lo largo de la ejecución de un programa:

- indefinida (I): no se le ha asignado ningún valor a la variable x;
- definida (D): se le ha asignado un valor a la variable x, por ejemplo, x = 5;
- referenciada (R): ha sido considerada como referencia, pero el valor de x no cambia, por ejemplo, if (x > 0).

El flujo de datos de una variable puede ser expresado como una secuencia de estados D, I o R, por ejemplo, → IDRDRRI.

¿En qué consiste la técnica? Consiste en analizar las secuencias de estados para una variable y si contiene una subsecuencia que no tiene sentido, se ha identificado una anomalía en el flujo de datos.

Secuencias que no tienen sentido y, por ende, que representan una anomalía podrían ser:

IR: un valor indefinido ha sido utilizado (anomalía).
DI: un valor definido pasa a indefinido antes de ser leído (anomalía).

DD: un valor definido vuelve a ser definido antes de ser leído (anomalía).

Al conocer tanto las técnicas de caja blanca como las técnicas de caja negra, ¿qué conclusiones puedes obtener?

Una de las conclusiones puede ser que no se podría enfocar en un tipo de técnica solamente, ya que tienen objetivos diferentes. Los atributos de las pruebas de caja blanca y de las pruebas de caja negra se pueden combinar para proveer un enfoque que valide la funcionalidad y selectivamente asegure que los aspectos internos son los correctos.

Referencias

Comité Internacional de Certificaciones de Pruebas de Software (ISTQB) (2011). *Fundamentals of Testing*. *Certified Tester Foundation Level Syllabus.* International Software Testing Qualifications Board.

Comité Internacional de Certificaciones de Pruebas de Software (ISTQB) (2011). SSTQB (2008). *Glosario Estándar de términos utilizados en Pruebas de Software.* (Versión 1.*3.ES.0.915)*

McCabe, T. (1976). A Software Complexity Measure. *IEEE Transactions on Software Engineering*, *SE-2*, 308-320.

Scrum-QA (2013) *Téc D P - Caja Blanca - Prueba de Camino y Cobertura (K4).* Recuperado de: http://scrum-qa.blogspot.com/2013/05/tecnicas-de-diseno-de-pruebas-tecnicas.html

Uninotas. (2017). *Clases de equivalencia PRUEBAS.* Recuperado de: https://www.uninotas.net/clases-de-equivalencia-pruebas