**Testing Caja Blanca - Clase Práctica**

El objetivo de esta clase es cubrir los métodos de caja blanca, dentro de los diseños de casos de prueba y estrategia. La clase pasada estuvimos hablando de los métodos de caja negra, cuya particularidad era, que no vamos a conocer los detalles de la implementación. En cambio, nos vamos a basar en el conjunto entrada y el conjunto salida, de esta forma poder armar estos casos de prueba usando algún método como la partición de equivalencias o el análisis de valores límites, pero el objetivo va a ser construir cosas de prueba.

Ahora vamos a ver los **métodos de caja blanca**, en los cuales nosotros disponemos de los detalles de la implementación, es decir, que podemos ver el código y en base a eso poder diseñar nuestros casos de prueba.

**Caja Blanca**

* Se basa en el análisis de la estructura interna del software o un componente de software.
* Se puede garantizar el testing **coverage**

Los métodos de caja blanca nos permite diseñar casos de prueba, maximizando la cantidad de defectos encontrados con la menor cantidad de casos de prueba posibles, disponiendo de los detalles de la implementación. Vamos a ver diferentes coberturas para garantizar que los métodos efectivamente de caja banca nos permite garantizar *(no tiene mucho sentido esta oración).*

**Cobertura**

* Cobertura de enunciados o caminos básicos
* Cobertura de sentencias
* Cobertura de decisión
* Cobertura de condición
* Cobertura de decisión//condición
* Cobertura múltiple
* Etcétera

Definición de Cobertura: forma de poder recorrer los distintos distintos caminos o las estrategias para recorrer distintos caminos que nuestro código nos provee para desarrollar una funcionalidad. Si nos imaginamos al código como un diagrama de flujo o un grafo, cada una de esas ramificaciones que se presentan dentro del código, gráfico, van a representar diferentes caminos que vamos a llegar a recorrer.

1. **Cobertura de enunciados o caminos básicos**

* Propuestos por McCabe
* Permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución

Busca poder garantizar que a todos los caminos independientes que tiene nuestra funcionalidad lo vamos a recorrer al menos una vez, es decir, vamos a pasar por todos los caminos independientes que esta funcionalidad tenga. Si nosotros diseñamos un conjunto de casos de prueba que corresponda a esta cobertura, la de enunciados, podemos garantizar que nuestro código ha sido ejecutado pasando por todos los caminos independientes.

No se si se acuerdan en sintaxis que vieron órdenes de complejidad, complejidad ciclomática, se representaban como log(n), bueno, tiene que ver un poco con esto. La idea es representar un conjunto de caminos en base a una métrica que es la **complejidad ciclomática**, que nos va a dar una idea de cual es la cantidad de caminos independientes que una funcionalidad dada tiene. Entonces, también nos va a dar una idea de cuántos casos de pruebas vamos a tener que hacer para poder recorrerlos a todos. Acuerdense, que vimos que estos casos de prueba van a cubrir un escenario en particular otorgándole valores específicos a las variables que van a permitir que esto se pueda ejecutar en el camino que nosotros estamos buscando.

Para la prueba del camino básico

* Se requiere poder representar la ejecución mediante **grafos de flujo**

Un grafo de nodos interconectados. Esto es un requerimiento que parece trivial, pero a algoritmos o funcionalidades que implementan métodos recursivos, no tendrían esta posibilidad. Entonces, quedarían fuera de esta prueba de cobertura.

* Se calcula la complejidad ciclomática.
* Dado un grafo de flujo se pueden generar casos de prueba. Ahora vamos a ver cómo vamos a generar esos casos de prueba para garantizar recorrer todos estos caminos independientes.

Grafo de flujo de estructuras básicas

|  |  |
| --- | --- |
| **While/Do While**  Puede estar representado mediante los nodos interconectados en forma de bucles |  |
| **If… Else...**  Bifurcación entre nodos con dos caminos  independientes. Si tiene tanto una salida por el lado del sí como por el lado del no. En caso de no tenerla, se une con el siguiente nodo evitando tener uno intermedio |  |
| **Secuencia**  Nodo con nodo en forma directa |  |

Este es el método más sencillo para poder acercarse a la idea de cobertura. En los ejercicios prácticos normalmente no lo cubrimos porque es trivial a la hora de llevarlos adelante, simplemente es una introducción.

*Complejidad ciclomática*

* Es una métrica de software que provee una medición cuantitativa de la complejidad lógica de un programa
* Usada en el contexto de testing, **define el número de caminos independientes en el conjunto básico** y entrega un límite inferior para el número de casos necesarios para ejecutar todas las instrucciones al menos una vez

**¿Cómo se calcula la complejidad ciclomática, que coincide con la cantidad de caminos independientes que hay?**

La fórmula nos dice que el número de aristas del grafo son las flechas que une el grafo, el número de nodos del grafo y la cantidad componentes conexos que los nodos de salida. Entonces hacemos, la cantidad de aristas menos la cantidad de nodos, más dos por la cantidad de nodos de salida, nos da la medida de caminos independientes, o el valor de complejidad ciclomática.

**M = E - N + 2\*P**

Parámetros

M = Complejidad ciclomática

E = Número de aristas del grafo

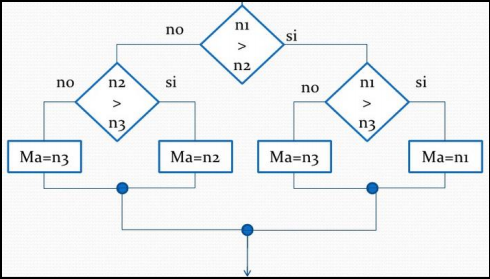
N = Número de nodos del grafo

P = Número de componentes conexos, nodos de salida

N = Número de regiones + 1

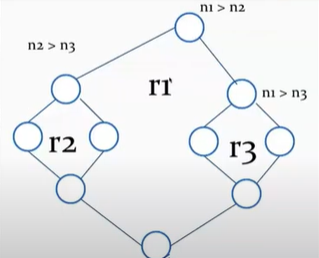
Hay otra forma, una forma visual de representarlo o de calcularlo que nos dice que una vez representada esta funcionalidad como un grafo orientado, lo que vamos a hacer es ver cuantas **regiones cerradas** tenemos dentro de este grafo y sumarle 1 a esas regiones. La forma gráfica la podemos resolver viendo el grafo dibujado y esta fórmula para la complejidad ciclomática se utiliza a nivel de software de soporte para poder brindarnos esa métrica ante una funcionalidad dada.

Ejemplo



Tenemos tres decisiones (if), cada uno de ellos tiene salida por cada uno de los dos lados y finalmente convergen en un punto final. Tenemos un solo componente conexo o un solo nodo de salida.

Si tuviéramos que representar este diagrama de flujo como un grafo, tendríamos



Una gran decisión, que es n1 > n2 y cada una de las decisiones anidadas por cada uno de los lados.

Si nosotros tenemos el diagrama de flujo, salta a la vista cuales son los caminos independientes:

* Toda la izquierda del n2>n3
* Izquierda y el camino de la derecha de n2>n3
* Toda la izquierda de n1>n3
* Toda la derecha de n1>n3

Si nosotros tenemos el grafos, salta a la vista cuales son los caminos independientes:

* Toda la izquierda de r2
* izquierda y el camino de la derecha de r2
* Toda la izquierda de r3
* toda la derecha de r3

Vemos que la cantidad de regiones cerradas es tres, entonces 3 + 1 = 4, es la cantidad de caminos independientes o complejidad ciclomática.

Lo mismo podemos obtener si sumamos la cantidad de aristas y nodos y aplicamos la fórmula.

M = E - N + 2\*P

M = 12 - 10 + 2\*1

M = 4

M = Número de regiones + 1

M = 3 + 1

Al fin y al cabo todos buscan establecer ese número de caminos independientes.

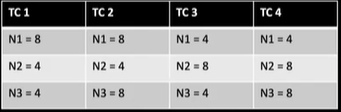
**¿Cómo podríamos diseñar estos casos de prueba? Para poder garantizar que cubrimos todos los caminos independientes, ¿Cuál es la cantidad mínima de casos de prueba que necesitamos desarrollar?** En este caso con solo cuatro podemos garantizar la cobertura de enunciados o de caminos básicos o independientes.

**¿Podemos escribir más de cuatro casos de prueba para garantizar la cobertura de enunciados?** Por más que ya estaría cubierta la cobertura en si, los cuatro caminos independientes ya serían recorridos con estos casos de prueba, no quita que yo pudiera escribir 100 casos de prueba, 1000 o infinitos, simplemente que la cantidad no es mínima.

Al fin y al cabo, las estrategias tanto caja negra o blanca buscan aplicar la economía en cuanto a detección de defectos, es decir, escribir la menor cantidad de casos de prueba que nos permita maximizar la cantidad de defectos encontrados. Entonces, para esta cobertura la menor cantidad de casos de prueba que necesito para garantizarla, es cuatro.

En este caso en particular, si analizan el diagrama de flujo pueden ver que tenemos diferentes operadores, en este caso, operadores matemáticos como n1, n2, n3. Si asignamos diferentes valores, como por ejemplo n1 que sea mayor que n2, ya vamos por el si. Por el otro lado si asignamos un n3 que sea menor que n1, por ejemplo un cero y vamos por el otro lado del si.

De esa forma asignando valores de n1, n2 y n3 yo puedo guiar la ejecución de esta funcionalidad por un camino determinado. Cada uno de esos casos de prueba lo que vamos a hacer es asignar valores a n1, n2 y n3 para llevar a ejecución izquierda, izquierda; izquierda, derecha; derecha, izquierda; derecha, derecha.



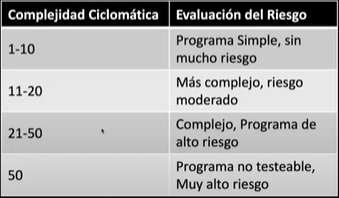
**¿Cada una de estas columnas es un caso de prueba?**

No, porque simplemente es una descripción de cuáles son los valores que nosotros tenemos que asignarle a ciertas variables que están en juego para poder escribir los casos de prueba. Después estos casos de prueba van a tener la representación de artefacto con la plantilla que planteamos la clase pasada. Esa plantilla va a tener un nombre que representa un escenario, precondiciones, pasos para reproducirlo, casos esperados. Entonces, esto no es un caso de prueba, es un conjunto de datos que asignaremos a las variables de entrada o a las variables en juego para poder escribir efectivamente un caso de prueba en particular o cuatro en esta cobertura.

**Pregunta:**¿Estas son todas las alternativas?

No necesariamente, pueden ser variables de entrada o pueden ser precondiciones, depende de cómo esté desarrollada la funcionalidad. Después van a tener que mapear (el cuadro anterior de TC 1, TC 2, TC 3, TC 4) a precondiciones o a variables ingresadas o seleccionadas por el usuario, dependiendo lo que representan en tu funcionabilidad.

Heurística relacionada a la capacidad ciclomática



Una funcionalidad que tenga 50 caminos independientes tiene al menos 50 casos de pruebas para garantizar su cobertura, esto es algo muy grande.

**¿Qué podemos hacer ahí?**

Una forma puede ser recortar testing o establecer prioridades, la otra puede ser modularizar, quizás encapsular una porción de esa funcionalidad y ejecutar pruebas relacionado a eso encapsulado. Después de ejecutar pruebas a la funcionalidad más pequeña también se puede hacer, en la medida de lo posible se puede tratar de optimizar ese código para poder tener la menor cantidad de caminos independientes. Hay muchas alternativas pero el objetivo siempre es el mismo, simplificar para poder tener menos cantidad de caminos independientes.

**Pregunta:**¿Nosotros podemos hacer testing teniendo siempre como entrada una user story o puede ser otra cosa?

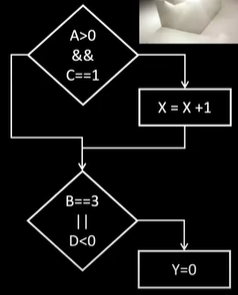
En este caso particular vos vas a tener como entrada una porción de código, porque los métodos de caja blanca se basan en que vos dispongas de los detalles de la implementación, es decir, que dispongas del código para poder hacer el análisis de esto y poder hacer efectivamente un diagrama de flujo o poder hacer un grafo para representar esa funcionalidad.

**Pasos del diseño de pruebas mediante el camino básico**

* Obtener el grafo de flujo
* Obtener la complejidad ciclomática del grafo de flujo
* Definir el conjunto básico de caminos independientes
* Determinar los casos de prueba que permitan la ejecución de cada uno de los caminos anteriores
* Ejecutar cada caso de prueba y comprobar que los resultados son los esperados

Este método no lo vemos en el práctico, lo vemos como una introducción teórica para entender las coberturas o digamos donde vamos a ir haciendo énfasis en la otras coberturas que si las vamos a usar en el práctico.

1. **Cobertura de sentencias**



Para todas las coberturas nos vamos a basar en este ejemplo

IF(A>0 && C==1)

X=X + 1

IF(B==3 || D<0)

Y = 0

END

La representación en pseudocódigo es bastante sencilla, tenemos dos decisiones conectadas en secuencia, es decir, que no depende uno del otro. Además, tenemos asignaciones dentro de variables que van a representar “X” funcionalidad.

El código es bastante simple, representando en un diagrama de flujo, tenemos la primera decisión, vamos a la asignación de variables en el caso positivo y en caso de no se cumpla la decisión no hacemos nada y entramos finalmente en la segunda decisión que tiene una salida por el lado positivos pero nada por el lado negativo.

¿**Cuántas sentencias tenemos en esta funcionalidad?** Tenemos dos, x = x + 1, y = 0. Una sentencia es cualquier instrucción como asignación de variables, invocación de métodos, mostrar un mensaje, lanzar una instrucción. Cualquier instrucción que nosotros demos, ya sea una asignación o una invocación es una sentencia, mientras que no sean estructuras de control, en este caso estas dos estructuras de control son decisiones.

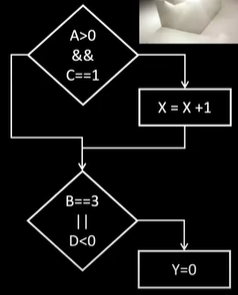
**¿Cuál es el objetivo de la cobertura de sentencias?** Veámoslo en comparación con la cobertura anterior, la cobertura de caminos básicos tenían como objetivo descubrir cuál es la menor cantidad de casos de prueba que me permiten recorrer todos los caminos básicos o pasar por todos los caminos independientes.

Entonces, el objetivo de la cobertura de sentencia es darle cobertura a todas las sentencias (no condiciones). Las condiciones es un término que se reserva para otras cosas, si la cobertura es de sentencias, lo que vamos a buscar cubrir son las sentencias, es decir, buscar cual es la cantidad mínima de casos de prueba que me permiten pasar o ejecutar o evaluar o recorrer todas las sentencias.

**En este ejemplo en particular, ¿Cuál es la cantidad mínima de casos de pruebas que me permiten cubrir todas las sentencias que tengo acá?**

No son excluyentes uno de otro, con que asignes valores a las variables, como son todas diferentes, con un solo caso de prueba deberías poder cubrirlo completo.

Como estas dos decisiones son independientes y además las variables que están en juego no son las mismas, simplemente con elegir una combinación de valores para A, C, B y D que me permita ir por las dos ramas del si, yo con un solo caso de prueba podría cubrir las dos sentencias.



Ejemplo

Caso de prueba

A = 5

C = 1

B = 3

D = 3

¿A igual a 5 es mayor que cero? Si, ¿ C es igual a 1? Si. Como es un “and” y ambos verdaderos vamos la rama de la derecha y cubrimos la primera sentencia.

¿B es igual a 3 ? Si, ¿o D es menor a cero? No, pero como es un “or”, con que el primero sea verdadero salimos por el lado verdadero y ya cubrimos la segunda sentencia. Con solo un caso de prueba hemos cubierto todas las sentencias y de esa forma minimizamos la cantidad de casos de prueba que nos permiten garantizar la cobertura.

No todos los casos van a ser tan sencillos como asignaciones de variables numéricas, sino que pueden representar el estado de un pedido, la selección de una ciudad, la existencia de una patente. Pueden estar relacionados los valores, puede ser que A sea mayor a cero me impide que sea B sea igual a tres, depende de la semántica de esa variable y las reglas de negocio o la funcionalidad o los criterios de aceptación de la funcionalidad que yo estoy evaluando. Importante, leer bien el código y entender el significado de esa variable y entender si esa combinación de valores es una combinación que yo puedo llegar a implementar y no solo se remite a algo teórico.

1. **Cobertura de decisión**

Una decisión es una estructura de control completa, entonces si lo ponemos en el diagrama de flujo que tenemos como ejemplo. Cada estructura de control que nosotros tenemos va a ser cada uno de esos rombos en los cuales podemos tomar o un camino o el otro, según si el resultado de su estructura de control puede dar true o false.

Si vamos al pseudocódigo, esto va a ser cada uno de los paréntesis del if, entonces vemos que tenemos dos if acá, dos expresiones entre paréntesis y cada una decide sobre cosas distintas. Podemos tener estructuras de control de if anidadas, es decir, dada una rama de la otra, por ejemplo, la primera de true podemos ir a una decisión o no.

**¿Qué es lo que va a buscar la cobertura de decisión en testing de caja blanca?**

Queremos ver si cada una de nuestras decisiones funcionan correctamente, para ver esto yo tengo que ver que sí se vaya para la rama del true cuando necesito que sea true y que sí se vaya para la rama de false cuando sea false. Entonces lo que vamos a hacer es buscar la mínima cantidad de casos de prueba para poder probar todas las decisiones que tengo en mi código, ya sea para forzarlo en la rama del true o forzarlo en la rama del false.

**Pregunta:** ¿Independiente si tengo o no sentencias en la rama del false o del verdadero?

Si, es independiente. Lo que interesa ahora no es probar una sentencia, no es probar que nuestras sentencias funcionen bien, sino lo que vamos a buscar probar es que nuestras decisiones funcionen bien. Lo que voy a buscar es poder probar, independiente de si por la rama del false de la primera, por ejemplo, no tengo una sentencia, pero yo quiero saber que cuando vaya por la rama false realmente vaya por esa rama y no ejecute la sentencia que no tiene que ejecutar.

Quiero probar que las decisiones vayan por donde tengan que ir, basados en esa definición, **¿Cuáles creen que son los casos de pruebas que nos van a hacer falta para hacer estas pruebas?**

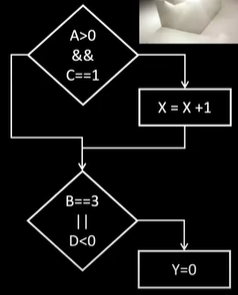
Si decimos una, no hay forma de pasar por verdadero y por falso por la misma decisión, tenemos que ir una vez por true y una vez por false por lo menos.

*Nomenclatura*: ahora estamos evaluando los rombos, que en sí son decisiones y dentro de las decisiones tenemos combinaciones que están unidas por operadores booleanos de condiciones. Entonces A > 0 es una condición, después tenemos el operador booleano que es el “and” y después tenemos otra condición que es C == 1. A la cobertura de decisión no le importa cómo value yo las condiciones que tengo dentro de la decisión, siempre y cuando garantice que esa decisión fue valuada en su valor verdadero o falso.

Con dos me basta, **¿Por qué?**

Porque no están anidados, porque independiente de la primera decisión, voy a pasar si o si por la segunda decisión, sin importar si voy a tener un true o un false en la primera decisión.

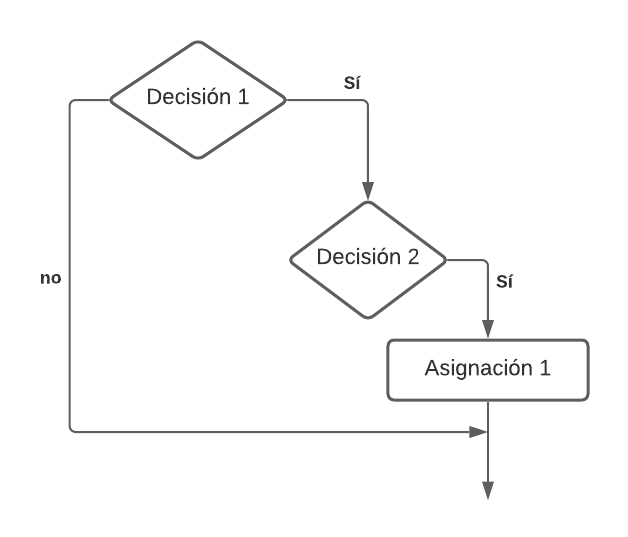
Entonces lo que voy a hacer es pasar primero por true al de arriba y después por true de abajo, todo en el mismo caso de prueba y después pasar por el contrario. También lo puedo cambiar o hacer al revés. Con dos casos de prueba, al no estar anidados, me alcanza.

Ejemplo

|  |  |
| --- | --- |
| **TC1** | **TC2** |
| A = 5 | A = 5 |
| C = 1 | C = 5 |
| B = 3 | B = 5 |
| D = -3 | D = 5 |

Primer caso de prueba, en el cual A es igual a 5 y C es igual a 1, esto hace que la primera decisión de true, entonces hacemos un tic para esa rama y después para la segunda decisión B es igual a 3 con lo cual primero cumplimos con la primera sentencia de esa decisión y D es igual a -3 entonces cumplimos con que esa condición sea menor a cero. Vamos a complir con la parte de true de la segunda decisión. Entonces nos queda probar la rama de false de la primera y la segunda decisión.

En el segundo caso de prueba, a A le vamos a dar un valor igual a 5 para que sea igual a false, y en la segunda decisión vamos a hacer que B sea igual a 5 incumpliendo la primera condición, y segundo incumpliendo que D sea menor a cero, esto nos va a llevar por la rama del false permitiendo probar con todas las posibilidades de nuestra petición con solamente dos casos de prueba.

Esto ocurre porque no están anidados. En cambio, si yo tuviera una decisión en la rama del false de la primera decisión, es decir, una decisión que esté anidada a la primera decisión, entonces voy a tener que tener un caso de prueba extra, porque con solo dos casos de prueba no puedo verificar que la primera decisión me de false y probar la segunda decisión.

Si tenemos una estructura anidada, vamos a necesitar un caso de prueba para verificar la rama verdadera y la rama falsa

**Pregunta:**¿Osea lo que busca es cubrir todos los caminos, independiente si el primero es verdadero y verdadero o verdadero y falso o falso y falso o falso y verdadero?

No importa cómo values las condiciones dentro de la decisión, siempre que garanticen salir al menos una vez por verdadero y una vez por falso para todas las decisiones.

Tenemos cuatro condiciones que son cada una de estas evaluaciones lógicas que están unidas por operadores lógicos dentro de una decisión.

La **cobertura de condición** busca encontrar la menor cantidad de casos de prueba que nos permiten valuar cada una de las condiciones, tanto en su valor verdadero como en su valor falso, independientemente de por dónde salga la decisión.

No vamos a considerar los if que son **cortocircuitados**, es decir, que sabemos que a nivel de operaciones lógicas en un AND si el primer valor se valúa en falso no se valúa la segunda condición o en un OR si la primera condición se valúa en verdadero no se valúa la segunda condición. Para los objetivos del testing de caja blanca de cobertura de condición y las próximas que se ven no le interesa hacer esta salvedad con respecto al cortocircuito de los operadores lógicos, si no que lo que busca hacer es, efectivamente, evaluar cada una de las condiciones que en su valor verdadero o en su valor falso minimizando la cantidad de casos de prueba.

**Entonces, ¿Cuántos casos de prueba necesitamos para poder valuar todas las condiciones en verdadero y en falso?** Hay que recordar que las decisiones no son anidades y que no nos importa el cortocircuito de los operadores lógicos, es decir, que podemos valuar A > 0 en falso y C == 1 en falso al mismo tiempo. Con dos, valuamos todos en true una vez y la segunda todos en false.

Caso de Prueba 1 → Todos en verdadero

A = 5

C = 1

B = 3

D = -1

Caso de Prueba 2 → Todos en falso

A = -1

C = 27

B = “cocodrilo”

D = 5

En dos casos de prueba valuamos todas las condiciones en su valor verdadero y en su valor falso, ya que a este método solo le interesa garantizar que cada una de esas condiciones sale por sus valores lógicos de verdad y falsedad sin importar si la decisión sale por el verdadero o sale por el falso, y tampoco le importa los cortocircuitos de los operadores lógicos.

El **objetivo** es siempre el mismo → minimizar la cantidad de casos de prueba para garantizar cubrir cierto conjunto de operadores o cierto camino o ciertas condiciones.

En este caso particular, no me importa si las decisiones salen por verdadero o falso, solamente evaluamos a nivel de condiciones.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC1** | **Resultado** | **TC2** | **Resultado** |
| A = 0 | Falso | A = 5 | Verdadero |
| C = 1 | Verdadero | C = 5 | Falso |
| B = 3 | Verdadero | B = 5 | Falso |
| D = -3 | Verdadero | D = 5 | Falso |

1. **Cobertura de decisión/condición**

La única variante con las anteriores es que lo que busca esta cobertura es no solamente valuar las decisiones en su valor verdadero y en su valor falso, sino también valuar todas las condiciones en su valor verdadero y en su valor falso.

En la cobertura de condición no importaba por donde salían las decisiones y cuando valuábamos las decisiones no le importaba por donde se valuaban las condiciones. Solo importaba la que estaba bajo análisis en ese momento.

A esta cobertura en particular le interesa garantizar las dos cosas, que las decisiones salgan por verdadero y falso y que las condiciones salgan por verdadero y falso.

**¿Cuántos casos de prueba necesitamos para garantizar que todas las decisiones se valúen en verdadero y falso y que todas las condiciones se valúen en verdadero y falso?** Dos. Si hago que tanto A como C sean verdaderas, al ser un AND sale por el lado verdadero. Si hago que B sea verdadera y que D también, como es un OR, sale por el lado verdadero. Entonces, a todas las condiciones ya las cubrí por el lado verdadero y a todas las decisiones también. Si ahora a A y C le asignamos falso, como es un AND, la ejecución se va para el lado del falso. Ya garantizamos todas las condiciones de arriba en verdadero y falso y la decisión de arriba en verdadero y falso. De nuevo, si valuamos a B y a D en falso, al ser un OR, me da falso. Con ese mismo caso de prueba garantizamos la cobertura de decisión y de ambas condiciones.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TC1** | **Condición** | **Decisión** | **TC2** | **Condición** | **Decisión** |
| A = 5 | Verdadero | Verdadero | A = 0 | Falso | Falso |
| C = 1 | Verdadero | C = 5 | Falso |
| B = 3 | Verdadero | Verdadero | B = 5 | Falso | Falso |
| D = -3 | Verdadero | D = 5 | Falso |

**Pregunta: No me queda claro cual seria la diferencia.** En este caso en particular, por como es la estructura del código que estamos analizando, no hay diferencia. En la cobertura de condición solo buscamos evaluar las condiciones. Si valuamos A verdadero, C falso, B verdadero y D falso, ya tenemos un caso de prueba y evaluamos esos lados de las condiciones. Con verdadero y falso salgo a falso (AND) y con verdadero y falso salgo a verdadero (OR). En otro caso de prueba valuamos falso a A y verdadero a C, y salimos por el falso. En el segundo rombo ponemos falso a B y verdadero a D y sale por el verdadero. Con esta combinación cubrimos todas las condiciones en su verdadero y falso pero en las decisiones solamente las cubrimos en un camino, porque en esta cobertura no me interesan las decisiones, me interesan solo las condiciones. En cambio, en el de decisión/condición, si me interesa hacer las dos cosas al mismo tiempo, entonces ahí sí voy a ser más cuidadoso a la hora de elegir la combinación de diferentes valores para evaluar todo al mismo tiempo. En los ejercicios prácticos suele coincidir la de decisión/condición con la de decisión y condición.

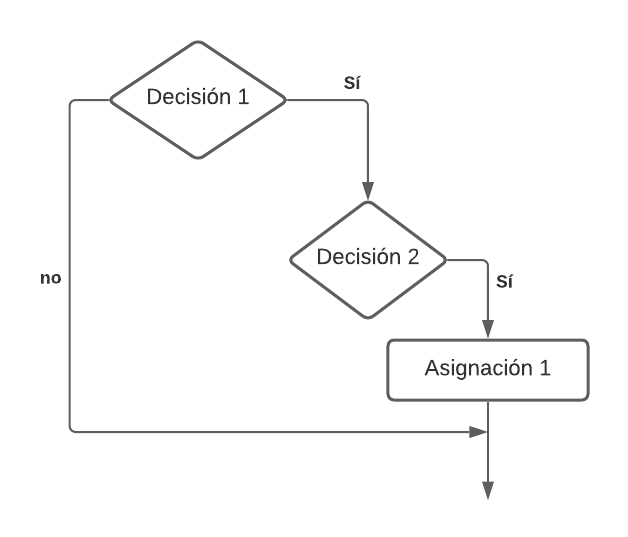
1. **Cobertura de Múltiple**

Es como ponerle el modo verbose a las coberturas. La Cobertura Múltiple busca valuar el combinatorio de todas las condiciones en todos sus valores de verdad posibles. Entonces, lo que hacemos es plantear el combinatorio de los valores de verdad para toda la combinación de condiciones que tenemos disponibles.

**En el ejemplo, ¿Cuántas condiciones son independientes?** Cuatro. El hecho de que tengamos los rombos separados de forma independiente (no están anidados, no dependen uno de otros) nos da una ventaja. Nosotros dentro de las decisiones lo que vamos a buscar con la cobertura múltiple es hacer el combinatorio de todos los valores de verdad posibles para las condiciones. Si esta decisión tiene dos condiciones, ¿Cuanto es el combinatorio de todos los valores de verdad para esas dos condiciones? Cuatro. Tendríamos para el primer rombo:

|  |  |  |  |
| --- | --- | --- | --- |
| **Primer Rombo** | | | |
| Verdadero | Verdadero | Falso | Falso |
| Verdadero | Falso | Verdadero | Falso |
| **Segundo Rombo** | | | |
| Verdadero | Verdadero | Falso | Falso |
| Verdadero | Falso | Verdadero | Falso |

Considerando que las dos decisiones son independientes, ¿Cuál vendría a ser la cantidad de casos de prueba que me permite hacer el combinatorio de todas las condiciones de todas las decisiones? Cuatro. Como las líneas verticales de la tabla. Si estas dos decisiones estuvieran anidadas, ¿Cómo afectaría a la cobertura múltiple? Tendríamos que agregar un caso de prueba más.

**Primera decision (AND)** 

Falso Falso Verdadero Verdadero

Falso Verdadero Falso Verdadero

Como es un AND lo que tengo arriba, ¿Cuál es la única combinación que me permite llegar a siquiera valuar la segunda decisión? Verdadero - Verdadero. Los primeros tres casos de prueba (los pinte de amarillo) no me permiten si quiera empezar a valuar el otro combinatorio. Si tenemos verdadero - verdadero en el primer rombo, recien ahi podemos hacer en el segundo:

Falso Falso Verdadero Verdadero

Falso Verdadero Falso Verdadero

Solamente una combinación para la primera decisión es la que me habilita valuar.

Entonces, son 7. En este caso en particular que las variables son independientes entre sí y no tienen relaciones a nivel de reglas de negocio que condicionen sus valores, simplemente con habilitar la rama de arriba en verdadero - verdadero podemos pasar a la rama de abajo.

**Pregunta: Si el de arriba fuese un OR, ¿serían muchas más?** No necesariamente porque las combinaciones de Verdadero - Falso los vamos a poder reutilizar para poder meter combinaciones de los de abajo. Serían menos, serían 5.

**Objetivo de la cobertura múltiple** → hacer el combinatorio de los valores de verdad de todas las condiciones dentro de las decisiones, siempre teniendo en cuenta todos los operadores lógicos para garantizar que estos casos de prueba sean válidos.

En el caso de que por la semántica del caso de prueba la combinación sea imposible, por más que correspondan con una combinación de la tabla de verdad, se descarta el caso de prueba.

**Elegir un método**

* Cada uno tiene fortalezas y debilidades particulares: un método puede ser bueno para algunas cosas, y no para otras cosas
* El mejor método es no usar un único método. Usar una variedad de técnicas ayudará a un testing efectivo.

Cada método se centra en cubrir una porción en particular o una incumbencia en particular. Lo ideal es no utilizarlos a todos ni utilizar uno solo, hay que elegir una combinación que a nivel de equipo nos permita garantizar la seguridad o calidad de lo que estamos buscando, encontrar la mayor cantidad de defectos.

**Pregunta: ¿Qué pasa cuando es un switch?**

Un Switch es como si fuera una decisión que en lugar de tener un camino verdadero y uno falso, tiene un camino por cada una de las opciones que tiene, por cada uno de los case.

**Instrucciones**

Siempre leer la funcionalidad que está asociada a la User Story que tenemos que desarrollar.

En Datos de los Casos de Prueba asignar los valores específicos de cada una de las variables.

Ejemplo:

1. Ingresa la patente 1234 → Datos específicos para poder garantizar que estoy llevando la ejecución por el lado que yo quiero para ese caso de prueba en particular

Hay precondiciones para el caso que necesitemos agregar alguna precondición para poder garantizar que el valor que elegimos para alguna variable nos va a llevar para ese lado.

Llevar el pseudocódigo a un diagrama de flujo.

**Recomendaciones**

Siempre leer primero la user para saber cuales son los datos que se van a ingresar y cuales no. Si hay restricciones de variables, o valores que se pueden ingresar. Si tenemos precondiciones ya definidas, aprovechemoslas. En caso contrario, podemos definir nosotros las precondiciones.

**Correcciones**

En este caso la user es: *“Ver mapa de taxis con opción de filtrar o buscar”*, el estado = Ocupado, NO lo podés ingresar en tu caso de prueba. Si podes ingresar que la búsqueda va a ser por número de chapa y que la chapa a filtrar va a ser la 3333 para llevarlo a estado Ocupado.

Terminan siendo dos entonces

1. Si se busca por chapa (Si/No)
2. La chapa a filtrar

Se busca por número de chapa que corresponde a Ocupado, después con una chapa que No existe, para llevarlo por el mensaje de error de “Taxi no conectado”, después tenemos el estado libre, solicitado, fuera de servicio y ocupado, para poder ir por cada uno de los cuatro caminos de los estados. Cuando es solicitado u ocupado, muestra los datos del pasajero, y si es ocupado muestra los datos del viaje.

**¿Cuántos casos de prueba tenemos para la cobertura de sentencias?**

5 casos de prueba. Con esta cantidad de casos de prueba cubrimos todas las sentencias que hay.

1. Estado = “Libre”
2. Estado = “Ocupado”
3. Estado = “Solicitado”
4. Estado = “Fuera de servicio”
5. Chapa que no corresponde o que no está disponible. ¿Cómo puedo ingresar una chapa no disponible? Con la inexistente, la 5555. Si queremos agregar otra nosotros, lo podemos hacer tranquilamente.

**Pregunta:** Acá, el único dato que puedo ingresar es el número de chapa? Lo primero que tenes es el número de chapa, pero la otra opcion que podes seleccionar es que efectivamente vas a estar filtrando por chapa de taxi, por número de patente. En la user tenes opciones de filtros por número de chapa. Seleccionas el filtro y después ingresas el valor a filtrar.

**Pregunta:** Tengo que poner numero de chapa = 1111, y después, busqueda de chapa existente? Estado de chapa solicitado? NO. Esto casualmente viene dentro de las precondiciones. Solo con poner chapa=1111 y filtra por chapa, con eso, usando las precondiciones definidas, ya se que es una chapa existente y que le corresponde el estado “Libre”.

Entonces lo correcto sería:

* Chapa = 1111
* Filtra por chapa = Si

**¿Cuántos casos de prueba tenemos para la cobertura de decisión?**

6 casos de prueba, porque se le suma la de “Búsqueda por número de chapa = False”. No busca por número de chapa, busca por otra cosa. Con la anterior de sentencias, ya habíamos cubierto en forma no intencional todas las decisiones que tenemos adentro, pero nos faltaba salir por el lado falso de la primera, que era “Busqueda por numero de chapa = True”. En este caso tendríamos que valuarlo en False.

**Pregunta:** Profe, en este caso hay un condicional múltiple, entonces no sabía si tenía que cubrir todas las condiciones del condicional múltiple, o solamente con una bastaba. Por ejemplo, si mi chapa = 3333, entro a casi todas las estructuras condicionales, pero no a todos los case del condicional múltiple. Tal cual, casualmente en el switch funciona como si fuera una decisión con múltiples salidas, no solamente con true y false, si no con n salidas. Para garantizar la cobertura de decisión en un switch, tengo que salir por cada una de las salidas posibles.

**¿Cuántos casos de prueba tenemos para la cobertura de condición?**

6 casos de prueba. En este caso en particular tenemos solamente una decisión que tiene más de una condición adentro, que es la de Estado=Solicitado o Estado=Ocupado. Pero en este caso, no podemos en base a la semántica de la variable, como es la misma variable, no hay forma de valuarla 2 veces por ejemplo en verdadero al mismo tiempo, o en una condición distinta. Seleccionando el estado que seleccionemos, vamos a estar valuando la misma variable, el mismo condicional, tanto en verdadero o en falso de forma “negada”, en base al valor que tengamos.

Se basa en que tenemos la misma variable en juega para esas mismas 2 condiciones.

Pregunta: Entonces, se puede probar solamente con uno de los dos estados? Si te pones a ver que el estado sea solicitado u ocupado, si tenes que valuar esas condiciones en verdadero y en falso, la única forma que se puede hacer es que:

Caso 1

Estado=Solicitado → Verdadero

Estado=Ocupado → Falso

Caso 2

Estado=Solicitado → Falso

Estado=Ocupado → Verdadero

Caso 3

Estado=Solicitado → Falso

Estado=Ocupado → Falso

En este caso el estado va a ser libre o fuera de servicio.

¿Se puede dar el caso que el estado sea “Solicitado” y “Ocupado”, es decir, que tengamos Verdadero Verdadero? No, porque la semántica de la variable lo impide. Descartamos ese caso. Los otros tres, si utilizamos la cobertura de decisión anterior, ya lo estamos cubriendo, en el momento en que seleccionamos una chapa que corresponde a “Solicitado”, otra que corresponde a “Ocupado” y otra que corresponde a cualquiera de los otros estados. Las otras decisiones tienen solamente una condición, entonces con la cobertura de decisión ya las estamos cubriendo a las de condición al mismo tiempo.

**¿Cuántos casos de prueba tenemos para la cobertura de decisión/condición?**

6 casos de pruebas, que coinciden con los de condición, por el hecho de que en la unica decision que tiene mas de una condición, en la cual podríamos llegar a tener el jueguito de valuarlos de forma diferente para no poder salir por los dos lados de forma alternada, la misma semántica de la variable me lo impide. El estado es uno o es el otro. En este caso, ese OR se puede interpretar se puede interpretar como si fuera un XOR, por el hecho de que la semántica de la variable hace que los valores de esas condiciones sean mutuamente excluyentes. No puede ser solicitado y ocupado al mismo tiempo por definición.

Lo correcto sería así:

Caso de Prueba 1

Chapa=1111

Filtra por chapa=Si

**Pregunta:** Profe escribirlo asi esta bien, o habría que poner “Ingresa numero de chapa…”? Se puede hacer un poco más redactado, pero con esto es suficiente. Lo que no tiene que tener bajo ningún concepto es por ejemplo la asignación de la variable “Búsqueda numero de chapa = True”. Esto no es aceptable a los fines de solución, porque no es un dato que estamos ingresando, sino que es la asignación de la evaluación de esa variable intermedia o el nombre de esa variable, que no es un dato de entrada para la funcionalidad.

**Pregunta:** En el caso de las 6 condiciones, las 2 condiciones que faltarían serian asi:

Caso de prueba 5

Chapa=5555

Filtra por chapa=No

Este caso sería imposible, que ingrese un numero de chapa y que no filtre por chapa. Porque si no filtras por chapa, no podes ingresar un número de chapa. El primero sale por false, entonces al segundo no lo podes valuar porque está anidado. Entonces, vas a tener uno que sea:

Filtra por chapa=No → y solamente esto.

Caso de prueba 6

Chapa=5555

Filtra por chapa=Si

**¿Cuántos casos de prueba tenemos para la cobertura múltiple?**

6 casos de prueba. El único caso en el cual podríamos llegar a tener alguna multiplicación de casos de prueba es en el combinatorio que se produce para el if estado = Solicitado o estado = Ocupado. Pero como vimos en el desarrollo teórico, los casos de prueba que son imposibles como la combinatoria de Verdadero Verdadero, simplemente se descartan. Entonces nos terminan quedando nuevamente, como en la cobertura de condición:

Caso 1

Estado=Solicitado → Verdadero

Estado=Ocupado → Falso

Caso 2

Estado=Solicitado → Falso

Estado=Ocupado → Verdadero

Caso 3

Estado=Solicitado → Falso

Estado=Ocupado → Falso

Ya los estábamos cubriendo con los casos de prueba anteriores.

**Pregunta:** En este caso que todos tienen 6 casos de pruebas, se ponen igual en los cuadrados? Si, tal cual. En la mayoría de los ejercicios prácticos, para varias coberturas podemos reutilizar casos de pruebas, por el hecho de que ya cubrimos ese camino, y de alguna forma valuamos ese conjunto de condiciones o decisiones. En el parcial si nos piden dos coberturas que con el mismo conjunto de casos de pruebas las podemos garantizar a ambas, simplemente copiamos y pegamos, de esta forma ya garantizamos dos coberturas con el mismo conjunto de caso de prueba, que es lo más óptimo que podemos llegar a hacer.

**Ejercicio de un grupo**

Cobertura de **sentencias**

1. Chapa=1111

Filtra por chapa=Si

1. Chapa=2222

Filtra por chapa=Si

1. Chapa=3333

Filtra por chapa=Si

1. Chapa=4444

Filtra por chapa=Si

1. Chapa=5555

Filtra por chapa=Si

**La dificultad está en ver qué es lo que vamos a probar (si las precondiciones y datos concretos ingresables están bien)**

Cobertura de **decisiones**

1. Chapa=1111

Filtra por chapa=Si

1. Chapa=2222

Filtra por chapa=Si

1. Chapa=3333

Filtra por chapa=Si

1. Chapa=4444

Filtra por chapa=Si

1. Chapa=5555

Filtra por chapa=Si

1. **Filtra por chapa=No**

Lo único que quedaba en el anterior por cubrir, era que la primera condición salga por el false.

**Recomendaciones**

Leer siempre la user antes de leer el pseudocódigo, NO saltar derecho al pseudocódigo, porque hay algunas consideraciones que vienen dadas por las user, y que nos pueden ayudar o podemos estar entrando en alguna omisión o error por considerar, por ejemplo una variable de entera cuando no lo era, o no tener en cuenta algunos valores que no puede llegar a tomar la variable.

Una vez que leemos el pseudocódigo, leerlo completo considerando las precondiciones también, porque si están definidas, nos da un conjunto de datos que vamos a poder utilizar para guiar el flujo por donde queremos. Si no están definidas en el enunciado, las vamos a poder definir nosotros, vamos a usar dentro del recuadro para la solución, para definir por ejemplo: “Yo considero que la chapa 1234 corresponde al estado Libre”. Y cuando usemos ese valor ya coincide y podemos garantizar que la ejecución va por cierto camino en particular.

Antes de empezar a hacer las coberturas y el análisis de la cobertura, se sugiere hacer el diagrama de flujo (no es obligación y NO se pide en el parcial, pero sí se recomienda). El diagrama de flujo muchas veces a nivel gráfico ayuda mucho a poder ir dibujando o analizando si estamos olvidando o no algún camino. Si lo hacemos en el parcial, NO hay que entregarlo, porque NO suma (hacerlo a mano si es mas rapido).

Si podemos reutilizar, reutilicemos (Ctrl C, Ctrl V), de esta forma ahorramos tiempo y ahorramos casos de uso.