Técnicas de Diseño de Pruebas

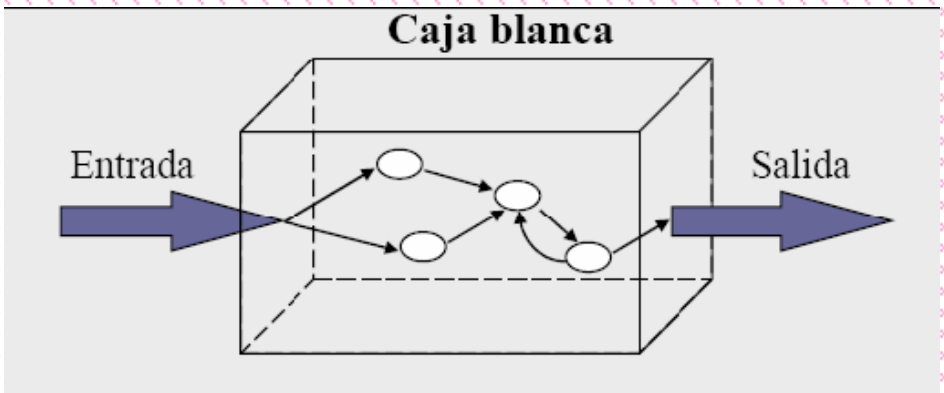
ROCIO SEGOVIA JIMÉNEZ

MSc. En ingenieria

Técnicas de Pruebas

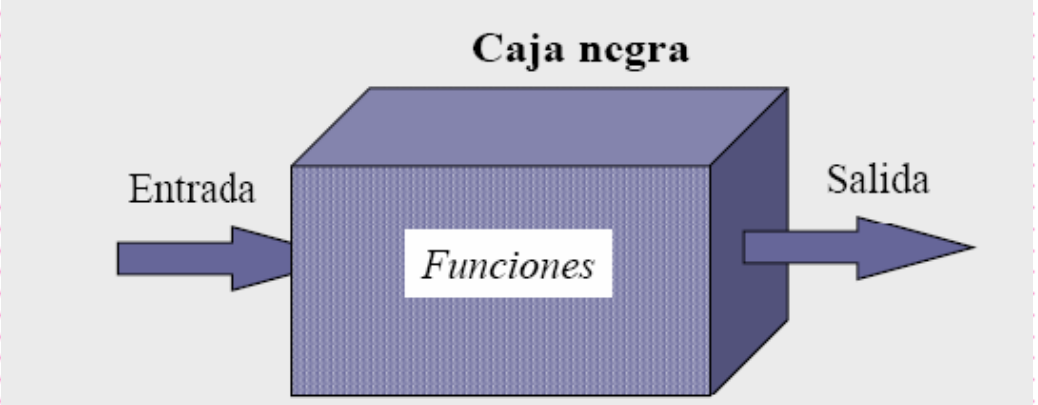
Las técnicas de evaluación dinámica proporcionan distintos criterios para generar casos de prueba que provoquen fallos en los programas. Estas técnicas se agrupan en:

- *Técnicas de caja blanca o estructurales*, que se basan en un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa.



Técnicas de Pruebas

- *Técnicas de caja negra o funcionales*, que realizan pruebas sobre la interfaz del programa a probar, entendiendo por interfaz las entradas y salidas de dicho programa. No es necesario conocer la lógica del programa, únicamente la funcionalidad que debe realizar.

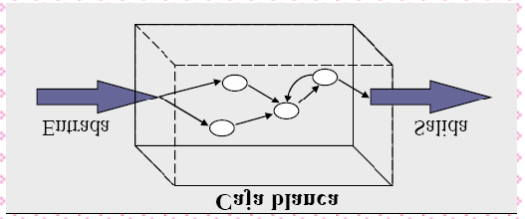


Pruebas de Caja Blanca - Estructurales

A este tipo de técnicas se le conoce también como Técnicas de Caja Transparente o de Cristal.

Este método se centra en cómo diseñar los casos de prueba atendiendo al *comportamiento interno y la estructura del programa*. Se examina así la lógica interna del programa sin considerar los aspectos de rendimiento.

El objetivo de la técnica es diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa, y todas las condiciones tanto en su vertiente verdadera como falsa.



Pruebas de Caja Blanca - Estructurales

Se han definido distintos criterios de cobertura lógica, que permiten decidir qué sentencias o caminos se deben examinar con los casos de prueba.

- *Cobertura de Decisión*
- *Cobertura de Camino Básico*
- Cobertura de Ciclos

Cobertura de Decisión

Se escriben casos de prueba suficientes para que cada decisión en el programa se ejecute una vez con resultado verdadero y otra con el falso.

Miremos el siguiente ejemplo:

```
if (a>0)  { x = x + 1; }
if (b==3) { y = 0; }
```

Cobertura de Decisión

Para aplicar esta técnica necesitaríamos emplear al menos los dos siguientes casos de prueba:

1. Evaluando la parte valida de la condición:

a = 2 y b = 3 (a verdadero, b verdadero)

2. Evaluando la parte invalida de la condición:

a = -2 y b = 3 (a falso, b verdadero)

Con estos dos casos de prueba son evaluadas al menos una vez, las salidas válidas o inválidas.

Cobertura de Caminos

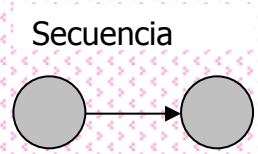
Se escriben casos de prueba suficientes para que se ejecuten todos los posibles caminos de un programa. Entendiendo camino como una secuencia de sentencias encadenadas desde la entrada del programa hasta su salida.

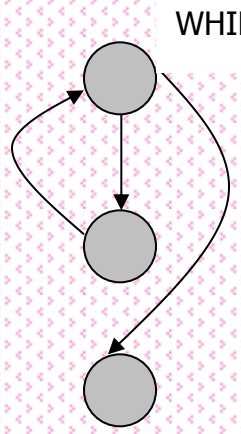
Los pasos a realizar para aplicar esta técnica son:

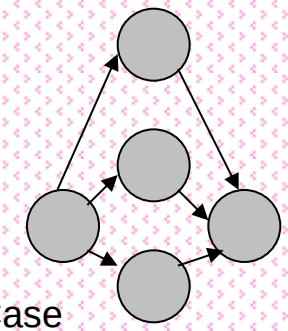
- Representar el programa en un grafo de flujo
- Calcular la complejidad ciclomática
- Determinar el conjunto básico de caminos independientes
- Derivar los casos de prueba.

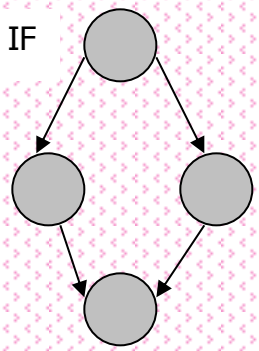
Cobertura de Caminos

- Notación





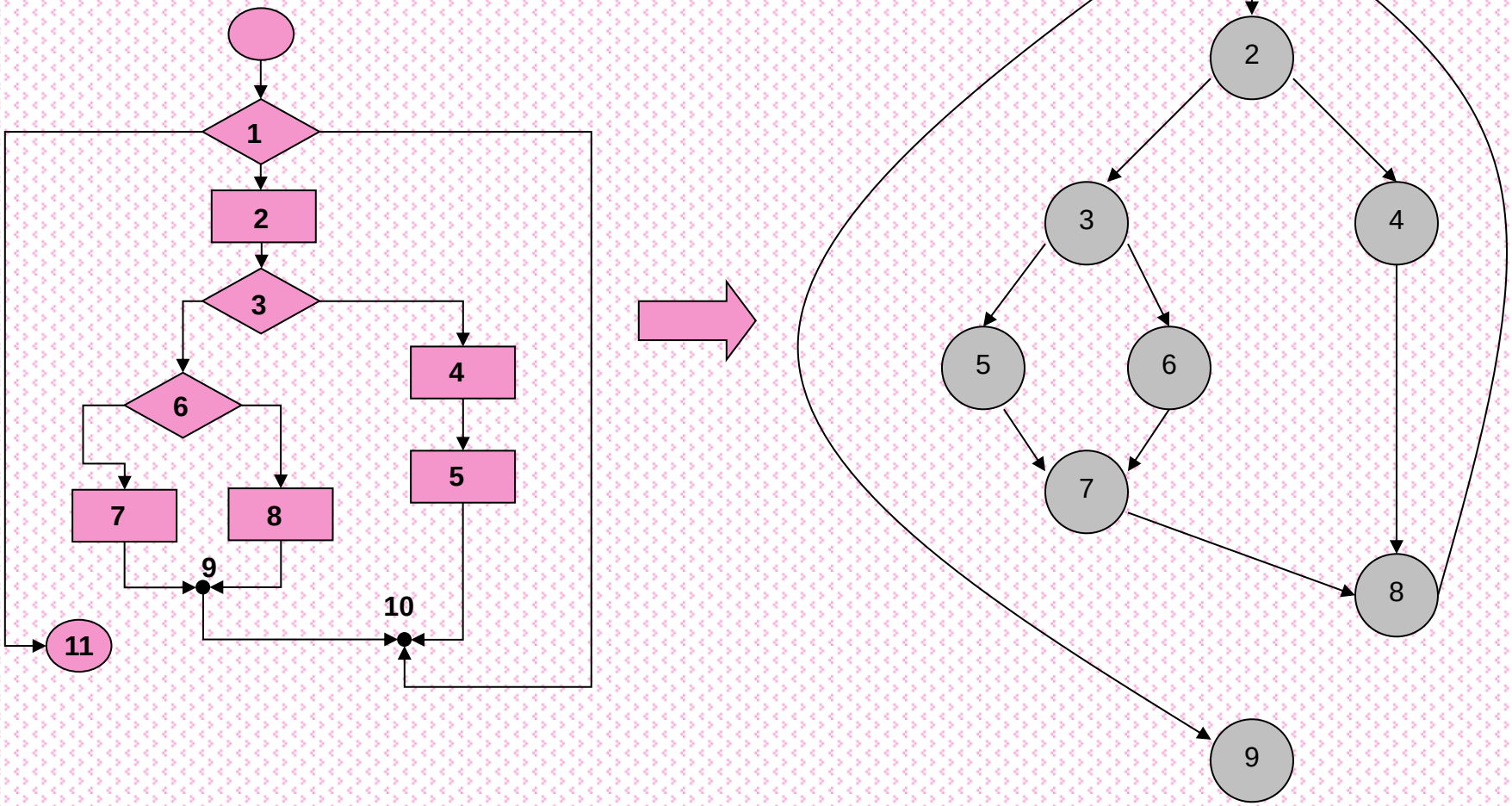




•**Nodos**: Representan cero, una o varias sentencias.
•**Aristas**: líneas que unen dos nodos.
•**Regiones**: áreas delimitadas por aristas y nodos. Cuando se contabilizan las regiones de un programa debe incluirse el área externa como una región más
Nodos Predicado: Cuando en una condición aparecen uno o más operadores lógicos (AND, OR, XOR, ...) se crea un nodo distinto por cada una de las condiciones simples. Cada nodo generado de esta forma se denomina nodo predicado.

Cobertura de Caminos

Paso de diagrama de flujo a grafo de flujo



Cobertura de Caminos

Complejidad Ciclomática

La técnica de camino básico se basa en la medida de complejidad ciclomática que es una métrica de software que provee una medición cuantitativa de la complejidad lógica de un programa.

Usada en el contexto testing, define el número de caminos independientes en el conjunto básico y entrega un limite superior para el número de casos necesarios para ejecutar todas las instrucciones al menos una vez.

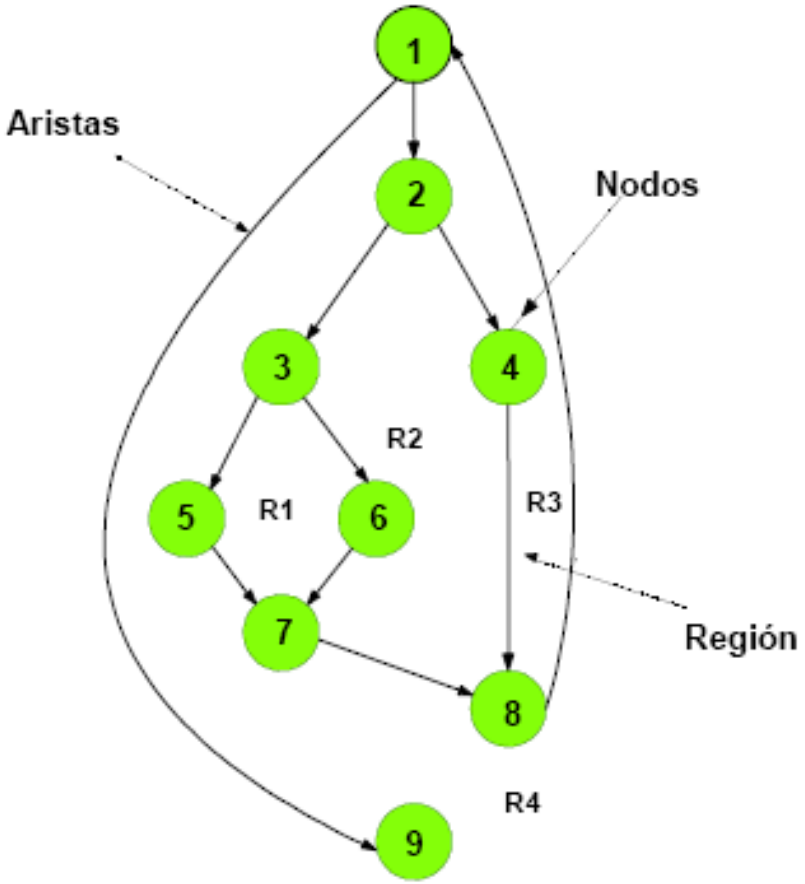
El inconveniente que presenta es que no da idea de la complejidad de los datos.

Cobertura de Caminos

Calculo de la complejidad ciclomática

- V(G) = Número de regiones
- V(G) = Aristas - Nodos + 2
- V(G) = Número de nodos predicado + 1

Cobertura de Caminos



- Considerando el grafo, encontramos el siguiente conjunto de caminos independientes:
  - Camino 1: 1-9
  - Camino 2: 1-2-4-8-1-9
  - Camino 3: 1-2-3-6-7-8-1-9
  - Camino 4: 1-2-3-5-7-8-1-9
- Cada nuevo camino introduce un arco nuevo.
- No se consideran caminos independientes aquellos que resulten de la combinación de otros caminos.
- El conjunto básico no es único.
- Se debe elegir como primer camino aquel que atraviese el mayor número de decisiones en el grafo.

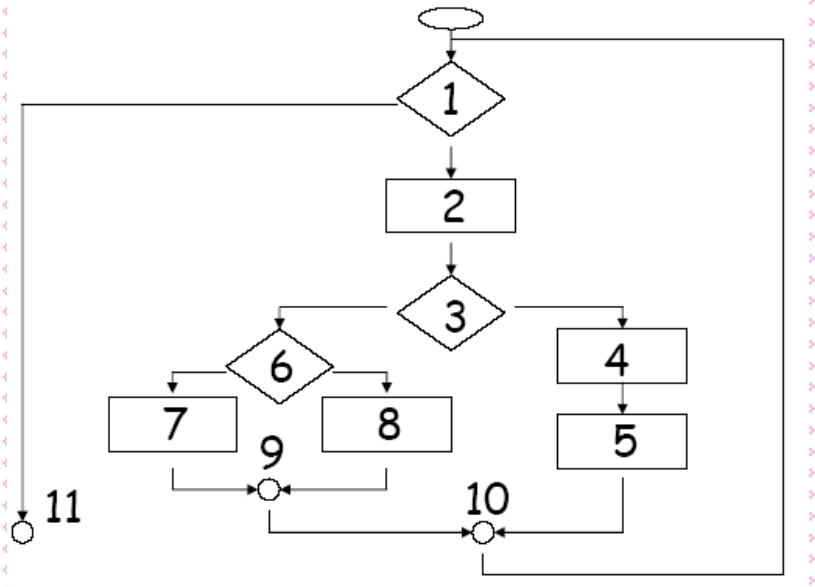
Cobertura de Caminos

Para el caso del grafo anterior, el conjunto básico calculado en todos los casos da 4.

- V(G) = Número de regiones = 4
- V(G) = Aristas – Nodos + 2 =11-9 + 2 = 4
- V(G) = Nodos Predicado + 1 = 3 +1 = 4

Cobertura de Caminos

La Cobertura de ciclos es una técnica de Caja Blanca, en donde el objeto es verificar los ciclos de un programa software.
Estas técnicas se caracterizan por usar grafos para describir su funcionamiento. Estos grafos siempre se componen de: Aristas, nodos y regiones



Cobertura de Ciclos

Como existen diferentes tipos de ciclos hay que tenerlos en cuenta a la hora de analizarlos. Los tipos son:

- Simple
- Anidado
- Concatenado

Además también hay que tener las diferentes sentencias que hay para representar un ciclo:

- While
- Repeat
- For

Cobertura de Ciclos

Para usar esta técnica es importante tener el código del programa que estamos evaluando, buscando los algoritmos que contengan los ciclos.

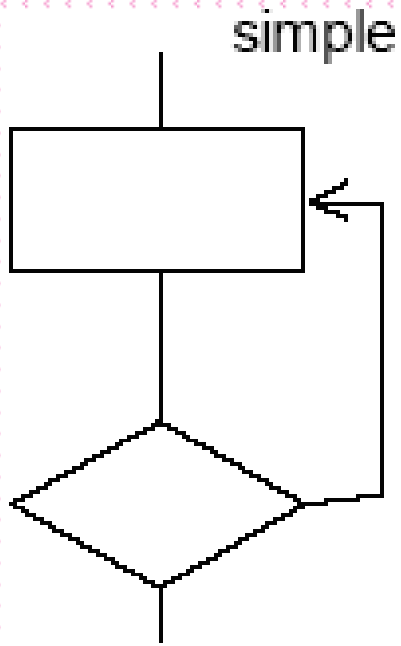
Una vez seleccionados los segmentos de código que contienen ciclos se procede a dibujar el grafo, esto se hace para poder identificar el recorrido lógico del código. Con el grafo y el código se identifica que criterio usar para aplicar pruebas.

A continuación se explican los diferentes tipos de ciclos y sentencias que se usan como criterios para evaluar ciclos.

Cobertura de Ciclos

Ciclos Simples:

Estos ciclos son sencillos, generalmente tienen una condición



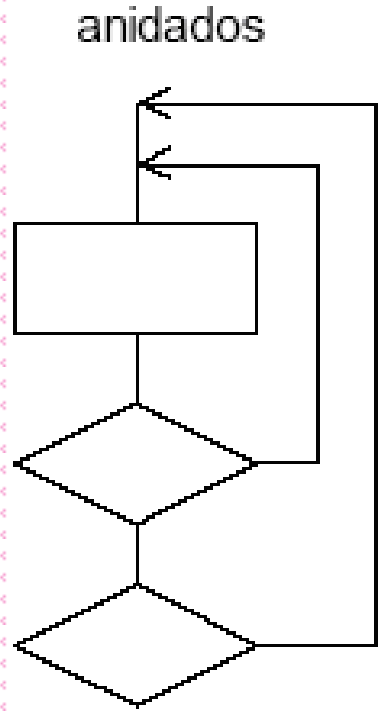
Para probar estos ciclos tenemos unas reglas. Teniendo en cuenta que n es el número de iteraciones del ciclo:

- Pasar por alto el bucle
- Pasar una sola vez
- Pasar 2 veces por el bucle
- Pasar m veces por el bucle, siendo m<n
- Pasar n-1 y n+1 veces

Cobertura de Ciclos

Ciclos Anidados:

Estos ciclos son aquellos que tienen un ciclo en su interior



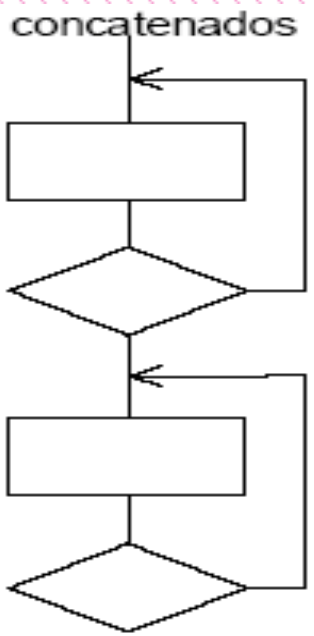
Tenemos las siguientes reglas

- Hacer pruebas con el bucle mas interno y tratarlo como si fuera simple y el externo mantener el numero mínimo de iteraciones
- Pruebas hacia fuera, para los internos mantener valores típicos y externos valores mínimos.

Cobertura de Ciclos

Ciclos Concatenados:

Estos ciclos son aquellos que tienen un ciclo en su interior, pero a diferencia del anterior vuelve no hasta el inicio del Ciclo externo, si no hasta si mismo.



Para estos hay que verificar que forma de concatenación tiene, si es **concatenación independiente** se prueba igual que los bucles simples, pero si es **concatenación no independiente**, se trata como bucles anidados.

Cobertura de Ciclos

Sentencias de ciclo While:

A este tipo de sentencias, por teoría se les deben aplicar como mínimo 3 pruebas:

- De cero ejecuciones
- De 1 ejecución
- De mas de 1 ejecución

Cobertura de Ciclos

Sentencias de ciclo Repeat:

A este tipo de sentencias, por teoría se les deben aplicar como mínimo 2 pruebas:

- De 1 Ejecución
- De mas de 1 Ejecución

Cobertura de Ciclos

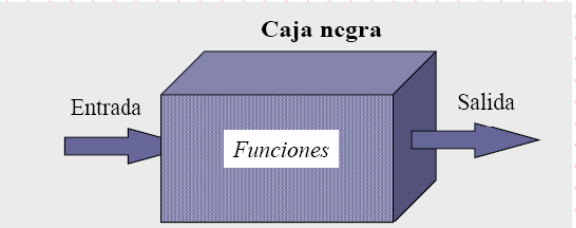
Sentencias de ciclo For:

Con este aparentemente sería sencillo, sólo se tendría que hacer 1 Prueba, pues el ya tiene el numero de veces que se va ejecutar en la cabecera, y las decisiones se pueden revisar con la técnica de cobertura de ramas, pero el For tiene sus trampitas, como que en su interior la variable incremente mas de lo debido, que existan Loop, Goto, Exit, Breaks, que alteraría por completo el comportamiento del ciclo, por lo tanto no podría hablar de 1 prueba, si no de un número incalculable de pruebas.

Pruebas de Caja Negra - Funcionales

También conocidas como Pruebas de Comportamiento, estas pruebas se basan en la *especificación* del programa o componente a ser probado para elaborar los casos de prueba. El componente se ve como una "Caja Negra" cuyo comportamiento sólo puede ser determinado estudiando sus entradas y las salidas obtenidas a partir de ellas.

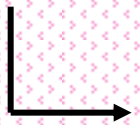
- Particiones de Equivalencia.
- Análisis de Valores Límite.
- Tablas de Decisión
- Arreglos Ortogonales



Particiones de Equivalencia

Definición

Conjunto de *Clases Equivalentes*



Conjunto de datos que definen entradas Válidas y No Válidas al Sistema

Entradas Válidas: Generan un valor esperado

Entradas No Válidas: Generan un valor inesperado

Particiones de Equivalencia

Identificación de las Clases de Equivalencia

Por cada condición de entrada se identifican clases de equivalencia válidas y no válidas.

Sin embargo existen un conjunto de criterios que ayudan a su identificación.

Criterios de Identificación de las Clases de Equivalencia

Condiciones de Entrada	Número de clases de equivalencia válida	Número de clases de equivalencia no válida
1. Rango de valores	**1** clase que contemple los valores del rango	**2** clases fuera del rango, una por encima y otra por debajo de éste
2. Valor específico	**1** clase que contemple dicho valor	**2** clases que representen un valor por encima y otro por debajo
3. Elementos de un conjunto tratados de forma diferente por el programa	**1** clase de equivalencia por cada elemento	**1** clase que represente un elemento fuera del conjunto
4. Condición lógica	**1** clase que cumpla la condición	**1** clase que no cumpla la condición

Particiones de Equivalencia

Partición Equivalente

Divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos o requerimientos de prueba.

Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos inválidas.

Técnica:	Partición equivalente	
T. Entrada:	Rango	
Proceso:	Afiliación usuario a entidad promotora de salud	
Variable:	Edad de afiliación	
Valores:	18 = < x < = 60	
Clases:	18 = < x < = 60	Válido
	x < 17	No válido
	x > 61	No válido

Particiones de Equivalencia

2. Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos inválidas

Técnica:	Partición equivalente	
T. Entrada:	Valor específico	
Proceso:	Afiliación usuario a entidad promotora de salud	
Variable:	Tipo de documento de identidad	
Valores:	Cédula de ciudadanía	
Clases:	Cédula de ciudadanía	Válido
	Tarjeta de identidad	No válido
	Registro Civil	No válido

3. Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una inválida.

Técnica:	Partición equivalente	
T. Entrada:	Miembro de un conjunto	
Proceso:	Afiliación usuario a entidad promotora de salud	
Variable:	Tipo de afiliado	
Valores:	Trabajadores independientes, dependientes	
Clases:	Trabajador independiente	Válido
	Trabajador dependiente	Válido
	Pensionado	No válido

Particiones de Equivalencia

4. Si una condición de entrada es lógica, se define una clase válida y una inválida.

Técnica:	Partición equivalente
T. Entrada:	Variable lógica
Proceso:	Afiliación usuario a entidad promotora de salud
Variable:	Registro en otra EPS
Valores:	Falso
Clases:	Falso Válido
	Verdadero No válido

Análisis de Valor Límite

La técnica se enfoca en la identificación de los casos de prueba asociados con los valores limites del dominio de la función o elementos de entrada al software.

1. Un valor en el limite inferior del rango de entrada — min
2. Un valor por encima del limite inferior — min-
3. Un valor valido dentro del rango de entrada — val
4. Un valor por debajo del limite superior — max-
5. Un valor en el limite superior del rango de entrada — max

Técnica:	Análisis de valor limite
Proceso:	Adición de Beneficiario hijo a Afiliado de EPS
Variable:	Edad de Beneficiario
Valores:	[0 - 18]
Casos:	0 años
	1 año
	5 años
	17 años
	18 años

Análisis de Valor Límite

Ventajas de la técnica:

La técnica reduce el número de casos de pruebas que deben ser creados y ejecutados.

Esta técnica permite elegir un subconjunto de las pruebas que son eficiente y eficaces en encontrar no conformidades.

Desventajas de la técnica:

No prueba todas las entradas posibles.

No prueba las dependencias entre las combinaciones de entrada.

El tester no puede identificar que porcentaje del sistema ha sido probado.

Tablas de Decisión

- Son utilizadas para registrar reglas del negocio complejas basadas en un conjunto de condiciones, las cuales deben ser implementadas en el sistema.

- Muy riguroso en análisis lógico

Tablas de Decisión

	Regla 1	Regla2	….	Regla N
Condiciones				
cond..1				
cond..2				
……				
cond.. P				
Acciones				
Acción 1				
Acción 2				
…..				
Acción M				

Condiciones de entrada

Acciones que se deben tomar, dependiendo de las combinaciones de las condiciones

Tablas de Decisión

- ***Condiciones***

  Valores de entrada

- ***Acciones***

  - Dependen de los valores que puedan tomar las condiciones y no del orden en que sean evaluadas

  - Se asume que todos los valores están disponibles simultáneamente.

- ***Reglas***

  Definen una combinación única de condiciones que resultan en la ejecución de las acciones asociadas con esa regla

Tablas de Decisión

Condiciones

- Un tester debe verificar que estén definidas todas las combinaciones.
- Especifica las entradas

Reglas

- Cada regla genera una acción.
- Cada regla puede especificar una acción única o puede ser compartida por varias acciones
- Cada regla es un caso de prueba

Tablas de Decisión

Tips

Si el sistema a probar tiene unas reglas del negocio complejas y los analistas y diseñadores del negocio no tienen documentadas las reglas de esta manera, el tester debe reunir esta información y representarla en forma de tablas de decisión

Razón

Representa el sistema en una forma compacta y completa y los casos de pruebas pueden ser creados directamente de la tabla de decisión

Crear almenos un caso de prueba por cada regla.

Si las condiciones son binarias, un solo caso de prueba por cada combinación puede ser suficiente.

Si las condiciones tienen un rango de valores, considere al menos un dato de prueba por encima y otro por debajo del rango de valores

Se mezcla las pruebas de valor límite con tablas de decisión

Tablas de Decisión

Caso 1.

Una compañía aseguradora de autos hace descuentos a los conductores que sean casados y/o buenos estudiantes.

Condiciones:

- Casado?? (si, no)
- Buen estudiante??? (si, no)

	Regla 1	**Regla 2**	**Regla 3**	**Regla 4**
Condiciones				
Casado??	Si	No	Si	No
Buen estudiante???	Si	Si	No	No

Tablas de Decisión

Acciones

- Descuento
- Las tablas de decisión deben especificar mas de una acción para las reglas
- Especifica el resultado esperado

	Regla 1	**Regla 2**	**Regla 3**	**Regla 4**
Condiciones				
Casado??	Si	No	Si	No
Buen estudiante???	Si	Si	No	No
Acciones				
Descuento ($)	60	25	50	0

Tablas de Decisión

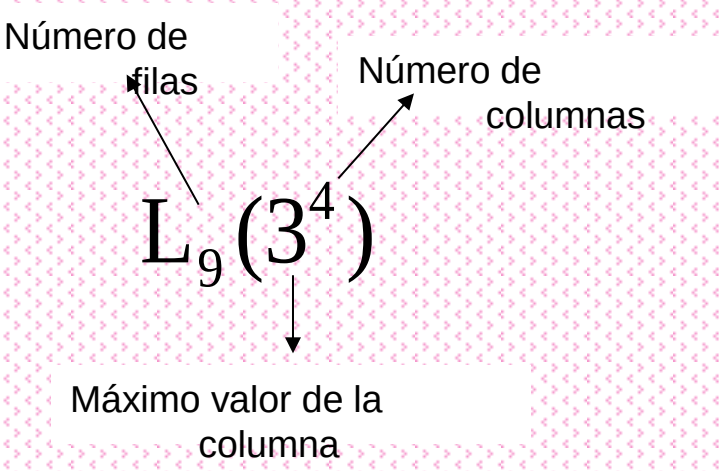
Casos de prueba

ID. Caso de Prueba	Casado?	Estudiante?	Resultado Esperado
CP1	Si	Si	Desc. 60
CP2	No	Si	Desc. 25
CP3	Si	No	Desc. 50
CP4	No	No	0

Arreglos Ortogonales

¿Qué son arreglos Ortogonales?

Un arreglo ortogonal es un arreglo en 2 dimensiones, con la siguiente propiedad: Si se seleccionan 2 columnas cualquiera en el arreglo, todas las combinaciones de pares aparecerán en esas columnas.



Arreglos Ortogonales

	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Posibles combinaciones (3 x 3 x 3 x 3) = 81.

Los arreglos ortogonales solo garantizan que todas las combinaciones de pares existen en el arreglo.

Arreglos Ortogonales

¿Cuándo usar la técnica?

- En situaciones donde se tienen muchos casos de prueba para diseñar y ejecutar.
- No se tienen los suficientes recursos.

La técnica ayuda a seleccionar un “buen subconjunto” de casos de prueba.

Arreglos Ortogonales

Situación 1.

Un sitio Web debe operar correctamente con diferentes browsers , usando diferentes plug – ins, corriendo sobre diferentes sistemas operativos en el cliente, utilizando diferentes servidores de aplicaciones y diferentes sistemas operativos en el servidor.

Browsers: IE 5.0, 5.5 y 6.0. Nestcape 6.0, 6.1. Mozilla 1.1. Opera 7,x

Plug-ins: RealPlayer, MediaPlayer, ninguno.

Sistemas operativos cliente: Windows 95, 98, ME, NT, 2000 y XP.

Servidor de Aplicaciones: IIS, Apache y WebLogic

Sistemas operativos servidor: Windows NT, 2000 y Linux.

Combinaciones: 8 x 3 x 6 x 3 x 3 = 1296.

Arreglos Ortogonales

Situación 2.

Un banco tiene un nuevo sistema de información listo para probar. El banco tiene diferentes clases de clientes, diferentes clases de cuentas y opera en diferentes estados del país, cada uno con diferentes regulaciones.

Tipos de cliente: 4

Tipos de cuenta: 5

Estados: 6

Combinaciones: 4 x 5 x 6 = 120

Arreglos Ortogonales

Aplicabilidad.

Esta técnica es aplicable en los siguientes niveles de prueba:

- Unidad.
- Integración.
- Sistema.
- Aceptación.

Arreglos Ortogonales

Proceso para usar la técnica.

- Identificar las variables.
- Determinar el número de opciones de cada variable.
- Encontrar un arreglo ortogonal el cual tenga el tamaño requerido.
- Ej: $L_9(3^4)$

- Mapear el problema dentro del arreglo ortogonal.
- Construir los casos de prueba.

Arreglos Ortogonales

Ejercicio.

- Un sitio Web debe operar correctamente con diferentes browsers, usando diferentes plug – ins, corriendo sobre diferentes sistemas operativos en el servidor, utilizando diferentes servidores de aplicaciones.

  - **Browsers:** Nestcape 6.2. , IE 6.0, Opera 4.0.
  - **Plug-ins:** Ninguno, RealPlayer, MediaPlayer.
  - **Servidor de Aplicaciones:** IIS, Apache, Nestcape Enterprise
  - **Sistemas operativos servidor:** Windows 2000, Windows NT, Linux.

Arreglos Ortogonales

- Identificar las variables.
  - Browsers
  - Plug-ins
  - Servidor de Aplicaciones
  - Sistemas operativos

- Determinar el número de opciones de cada variable.
  - Browsers : 3
  - Plug-ins: 3
  - Servidor de Aplicaciones: 3
  - Sistemas operativos: 3

  - Combinaciones: 3 x 3 x 3 x 3 = 81.

Arreglos Ortogonales

- Encontrar un arreglo ortogonal el cual tenga el tamaño requerido.

- Columnas (Variables) : 4.

- Máximo valor por columna (número de valores que puede tomar cada variable): 3 (1,2 ó 3)

$$3^4 \rightarrow L_9(3^4)$$

	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Arreglos Ortogonales

- Mapear el problema dentro del arreglo ortogonal.

- **Browsers**
- 1 -> Nestcape 6.2.
- 2 -> IE 6.0
- 3 -> Opera 4.0.

- **Plug-ins**
- 1 -> Ninguno
- 2 -> RealPlayer
- 3 -> MediaPlayer

- **Servidor de Aplicaciones**
- 1 -> IIS
- 2 -> Apache
- 3 -> Nestcape Enterprise

S. O

1 -> Windows 2000

2 -> Windows NT

3 -> Linux

Arreglos Ortogonales

4. Mapear el problema dentro del arreglo ortogonal.

	Browser	Plug - in	Servidor	S. O
1	Nestcape 6,2	Ninguno	IIS	Windows 2000
2	Nestcape 6,2	RealPlayer	Apache	Windows NT
3	Nestcape 6,2	MediaPlayer	Nestcape Entreprise	Linux
4	IE 6,0	Ninguno	Apache	Linux
5	IE 6,0	RealPlayer	Nestcape Entreprise	Windows 2000
6	IE 6,0	MediaPlayer	IIS	Windows NT
7	Opera 4,0	Ninguno	Nestcape Entreprise	Windows NT
8	Opera 4,0	RealPlayer	IIS	Linux
9	Opera 4,0	MediaPlayer	Apache	Windows 2000

5. Construir los casos de prueba.

Acceder al Sitio Web utilizando como navegador IE 6.0, utilizando Media Player Como Plug-in, en un servidor con Sistema Operativo Windows NT y servidor de Aplicaciones IIS.

Arreglos Ortogonales

Notas

- La técnica de arreglos ortogonales reduce significativamente el número de pruebas que deben diseñarse y ejecutarse.
- Pueden existir combinaciones dadas por el arreglo ortogonal que no son válidas.
- En arreglos ortogonales no todas las columnas deben tener el mismo rango de valores (1…2, 1…3, 1…5, etc.). Algunos arreglos ortogonales son mixtos.

Ej: $L_{18}(2^1 3^7)$

Arreglos Ortogonales

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	1	1	2	2	2	2	2	2
3	1	1	3	3	3	3	3	3
4	1	2	1	1	2	2	3	3
5	1	2	2	2	3	3	1	1
6	1	2	3	3	1	1	2	2
7	1	3	1	2	1	3	2	3
8	1	3	2	3	2	1	3	1
9	1	3	3	1	3	2	1	2
10	2	1	1	3	3	2	2	1
11	2	1	2	1	1	3	3	2
12	2	1	3	2	2	1	1	3
13	2	2	1	2	3	1	3	2
14	2	2	2	3	1	2	1	3
15	2	2	3	1	2	3	2	1
16	2	3	1	3	2	3	1	2
17	2	3	2	1	3	1	2	3
18	2	3	3	2	1	2	3	1

Arreglos Ortogonales

- Un tester no tiene que crear arreglos ortogonales, todo lo que debe hacer es encontrar uno del tamaño adecuado y luego realizar el mapeo del problema sobre el arreglo. Libros, sitios web y herramientas automatizadas ayudarán a hacerlo.

- Los arreglos ortogonales no pueden ser construidos para cualquier parámetro de tamaño arbitrario. Estos tienen tamaños fijos.

- Si el arreglo del tamaño perfecto no existe, seleccione uno que sea un poco más grande y aplique estas 2 reglas:

  - Columnas extras. Si el arreglo seleccionado tiene más columnas que las necesitadas para un problema, simplemente se borran. El arreglo permanecerá ortogonal.
  - Valores extra para una variable. No borrar las filas que contienen esas variables. El arreglo puede dejar de ser ortogonal. Cada fila en el arreglo existe para dar por lo menos una combinación de par que no aparece en ninguna otra parte.