PRUEBA DE SOFTWARE

Mg. Marcela Daniele

Universidad Nacional de Río Cuarto
Facultad de Ciencias Exactas, Físico-Químicas y Naturales
Departamento de Computación

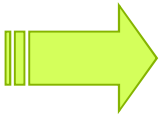
Pruebas de Software – Validación & Verificación

- Conceptos
- Consideraciones
- Objetivos
- Prueba de Unidades
  - Pruebas de Caja Blanca o Estructural
  - Pruebas de Caja Negra o Funcional

Pruebas de Software – Validación & Verificación

Validación

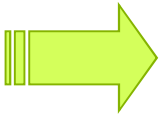
Consiste en comprobar que el sistema y sus componentes cumplen con las especificaciones del problema.



¿Estamos resolviendo el problema correcto?

Verificación

Consiste en evaluar el sistema y sus componentes durante el proceso de desarrollo, y determinar si satisface los requisitos de las especificaciones.



¿Estamos resolviendo correctamente el problema?

Pruebas de Software – Concepto de Prueba

Prueba es un tipo de V&V. Consiste en determinar cómo operan las componentes de un sistema en situaciones representativas y verificar si su comportamiento es el esperado.

- **Caso de Prueba:** entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular.
- **Falla:** no hace lo requerido o hace algo que no debería.

  Especificaciones mal definidas o incompletas, algún requerimiento que no puede implementarse, fallas en el diseño o en el código.
- **Error:** diferencia entre un valor calculado u observado y el valor correcto esperado.

  Defecto en alguna especificación. Cualquier resultado incorrecto. Una acción humana que conduce a un resultado incorrecto.

Pruebas de Software – Consideraciones

- Es muy difícil probar todas las condiciones de operación posible, y encontrar casos de prueba que muestren la evidencia de que el comportamiento deseado será visto en todos los casos restantes.
- Un proceso de prueba será exitoso cuando encuentre "errores".
- Cada caso de prueba define un resultado de salida esperado.
- El programador debe evitar probar sus propios programas.
- La Prueba es una actividad critica de la IS y debería realizarse tan sistemáticamente como fuera posible.
- Al generar casos de prueba, se deben incluir tanto datos de entrada válidos y esperados como no válidos e inesperados.
- La planificación de las Prueba debe iniciarse desde la primera etapa.
- Las pruebas deben Probar si el software no hace lo que debe hacer y Probar si el software hace lo que no debe hacer.

Pruebas de Software – Objetivos

- La Prueba es el proceso de ejecución de un programa con la intención de **descubrir algún error**.
- Se considera que el Caso de Prueba es "muy bueno" si posee una alta probabilidad de **descubrir al menos un nuevo error**.
- La Prueba debe servir para **Localizar errores** y no solo detectar su presencia.
- La Prueba debe ser **repetible**, aplicar más de una vez la misma entrada en la misma pieza y verificar que produce el mismo resultado.

Pruebas de Software – Facilidad de Prueba

La facilidad de prueba del software es la facilidad con la que se puede probar un programa. Existen métricas para ello.

Características que llevan a que un software sea más fácil de probar

- **Operatividad:** un buen funcionamiento.
- **Observabilidad:** salidas claras. Código accesible. Se detectan los errores fácilmente.
- **Controlabilidad:** consistente. Permite automatizar las pruebas.
- **Capacidad de Descomposición:** modularizado.
- **Simplicidad:** funcional, estructural y de código.
- **Estabilidad:** pocos cambios y controlados, recuperación correcta e inmediata ante fallos.
- **Facilidad de Comprensión:** diseño claro, comunicación adecuada, documentación accesible, detallada y organizada.

Testing in the Small (Prueba de Unidades)

El diseño de la pruebas debe considerar
la mayor probabilidad de encontrar el mayor número de errores con la mínima cantidad de esfuerzo y tiempo posible.

¿Es posible la prueba exhaustiva?

La prueba de unidades consiste en probar módulos individuales.

- **Prueba de Caja Blanca o Estructural**: usa la estructura interna del programa para derivar los casos de prueba.
- **Prueba de Caja Negra o Funcional:** los casos de prueba derivan solo de la especificación. Solo interesa la funcionalidad.

Testing in the Small (Prueba de Unidades)

- **Prueba de Caja Blanca o Estructural**

  - Criterio de Cobertura de Sentencia

  - Criterio de Cobertura de Arco

  - Criterio de Cobertura de Condición

  - Criterio de Cobertura de Camino

Caja Blanca - Criterio de Cobertura de Sentencia

Ejecuta el programa y se asegura de que todas sus sentencias son ejecutadas al menos un vez.

Asegura que se puede descubrir si las partes del programa contienen algún error, ejecutando cada sentencia al menos una vez.

Definición

Seleccionar un conjunto de casos de prueba T tal que, ejecutando el programa P por cada d en T, cada sentencia elemental de P es ejecutada al menos una vez.

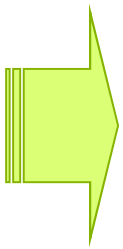
Debilidades

- Ejecutar cada sentencia y ver como se comporta no garantiza que el programa no contenga errores.
- No está clara la definición de sentencia, cual es la sentencia más elemental.

Caja Blanca - Criterio de Cobertura de Sentencia

Ejemplos

```
read (x)
read (y)
If x > 0 then
      write ('x es positivo')
      x:=-2
else
      write ('x es negativo')
endif
If y > 0 then
      write ('y es positivo')
else
      write ('y es negativo')
endif
```



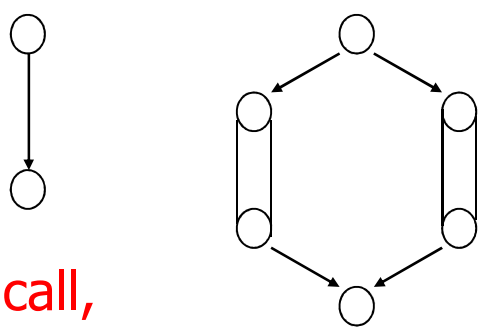
Un conjunto mínimo de Casos de Prueba que no encuentra error

Ej: T={<x=-5, y=10>,
<x=5, y=-10>}

Caja Blanca - Criterio de Cobertura de Arco

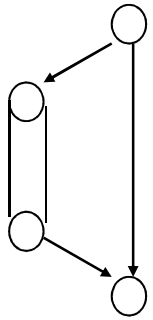
Describe el programa con un grafo del flujo de control del programa, y asegura que se recorren todos los arcos en dicho grafo.

Construcción del grafo del flujo de control de un programa según:

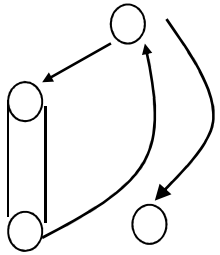


I/O, call, asignación

If-then-else



If-then



while loop

sentencias secuenciales

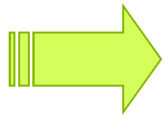
Definición

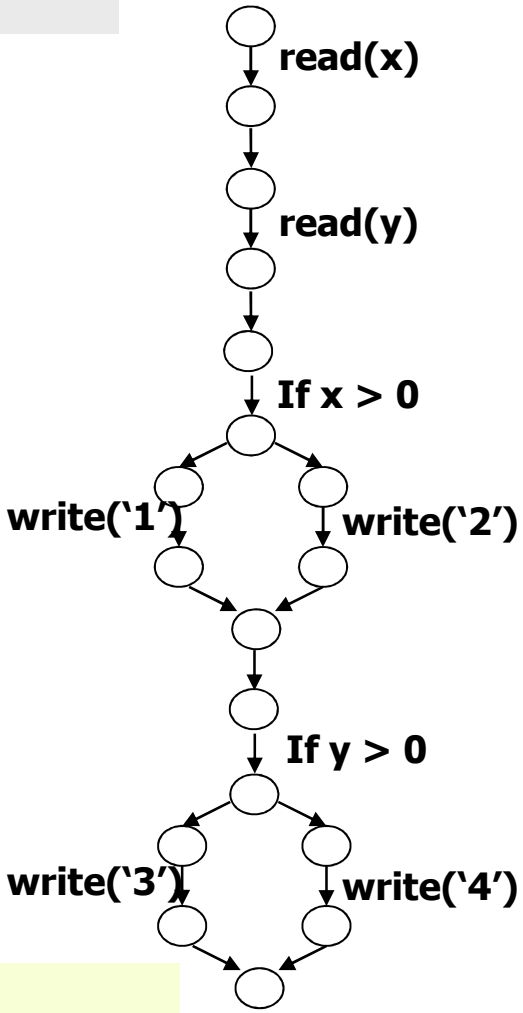
Se selecciona un conjunto de prueba T tal que, ejecutando el programa P por cada d en T, cada arco del flujo de control de P es atravesado al menos una vez.

Caja Blanca - Criterio de Cobertura de Arco

Ejemplo

```
read (x)
read (y)
If x > 0 then
        write (‘1’)
else
        write (‘2’)
endif
If y > 0 then
        write (‘3’)
else
        write (‘4’)
endif
```





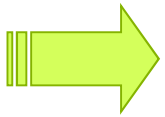
Conjunto mínimo de Casos de Prueba

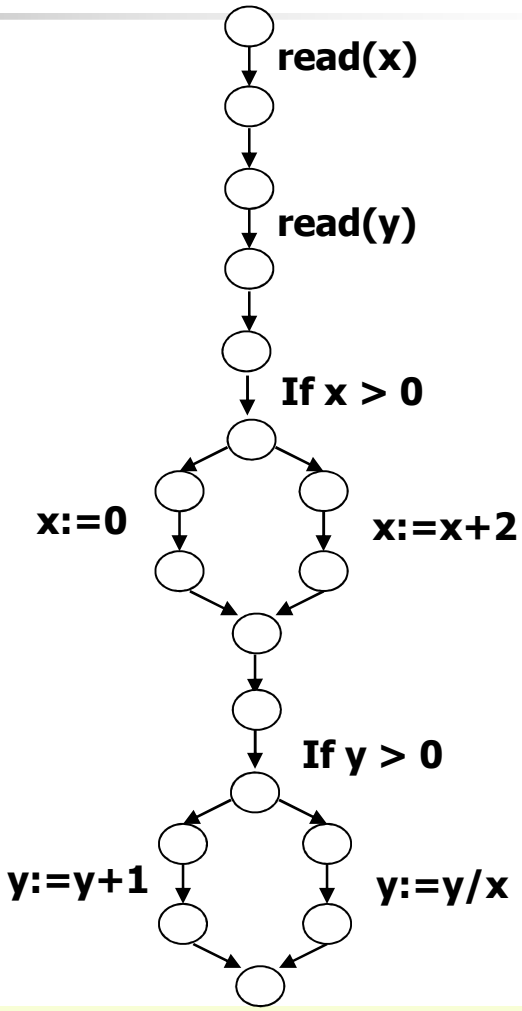
Ej: T = {<x = 5, y = 10>, <x = -5, y = -10>}

Caja Blanca - Criterio de Cobertura de Arco

Ejemplo

```
read (x)
read (y)
If x > 0 then
        x:=0
else
        x:=x+2
endif
If y > 0 then
        y:=y+1
else
        y:=y/x
endif
```





Ejemplos de conjunto mínimo de Casos de Prueba. Ambos satisfacen el criterio

T1 = {<x = 5, y = 10>, <x = -5, y = -10>} no encuentra el error

T2 = {<x = 5, y = -10>, <x = -5, y = 10>} **encuentra el error !!!**

Caja Blanca - Criterio de Cobertura de Condición

Divide las expresiones booleanas complejas en sus componentes e intenta cubrir todos los valores posibles de cada uno de ellos.

Definición

Se selecciona un conjunto de prueba T tal que, ejecutando el programa P por cada d en T, cada arco del flujo de control de P es atravesado al menos una vez y todos los posibles valores de los constituyentes de las condiciones compuestas son probadas al menos una vez.

Ejemplo: x y z

< x = V , z = V >

< x = F , z = ? >

< x = V , z = F >

Caja Blanca - Criterio de Cobertura de Camino

Asegura que todos los caminos del grafo de flujo de control del programa son atravesados al menos una vez.

Se selecciona un conjunto de prueba T tal que, ejecutando el programa P por cada d en T, todos los caminos principales o independientes desde el nodo inicial al final del flujo de control de P son atravesados.

Complejidad Ciclomática

- Es una métrica que proporciona una medida cuantitativa de la complejidad lógica de un programa.
- Define el número de caminos independientes de un programa .
- Define un límite superior de CP para conseguir una cobertura de arcos completa y un límite inferior de los posibles caminos a través del grafo de flujo de control.

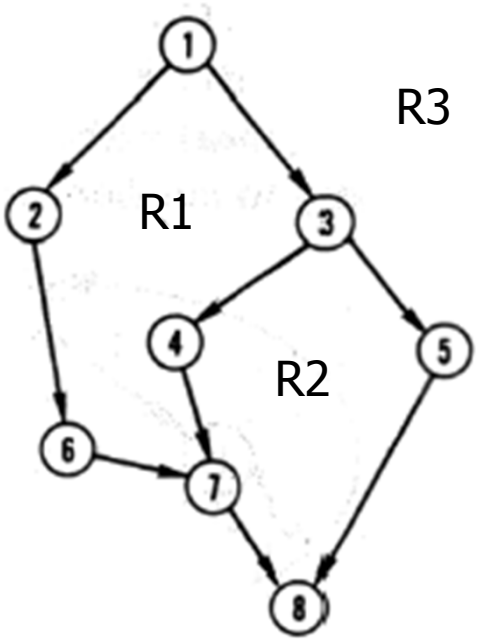
Formas posibles de calcular la **Complejidad Ciclomática**:

1. Contando el número de regiones del grafo de flujo.
2. A través de aristas y nodos
3. Contando los nodos predicado

Caja Blanca - Criterio de Cobertura de Camino

Formas posibles de calcular la **Complejidad Ciclomática**:

1. Contando el número de regiones del grafo de flujo.



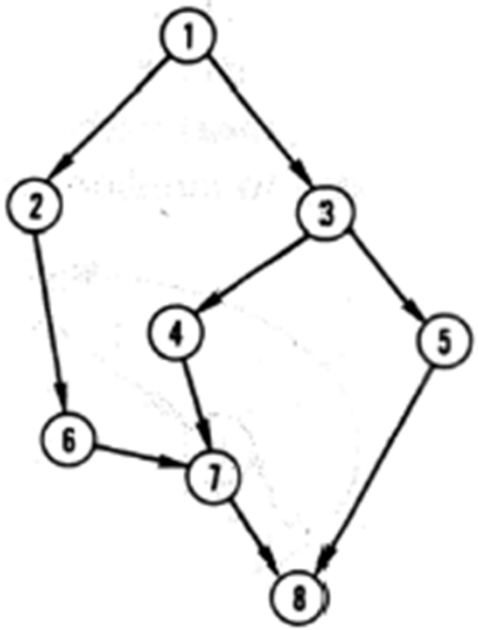
V (G) = 3

Caja Blanca - Criterio de Cobertura de Camino

2. Aplicando la siguiente fórmula, en base a las aristas y nodos:

$$V(G) = A - N + 2$$

donde A es el número de aristas del grafo de flujo y N es el número de nodos del mismo.



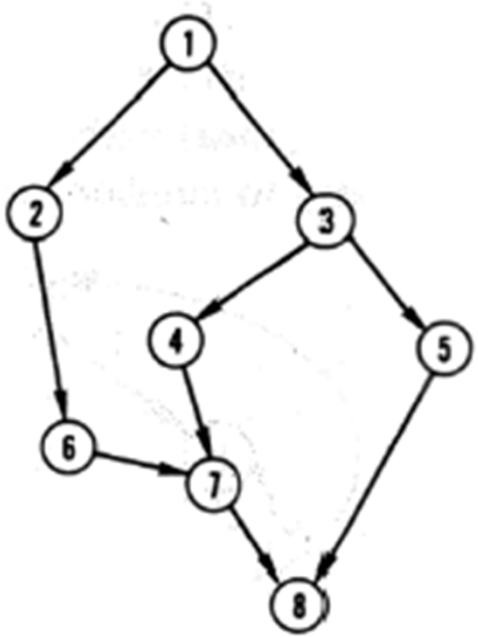
$$V(G) = A - N + 2$$
$$V(G) = 9 - 8 + 2$$
$$V(G) = 3$$

Caja Blanca - Criterio de Cobertura de Camino

3. Contando los nodos predicado:

$$V(G)=P+ 1$$

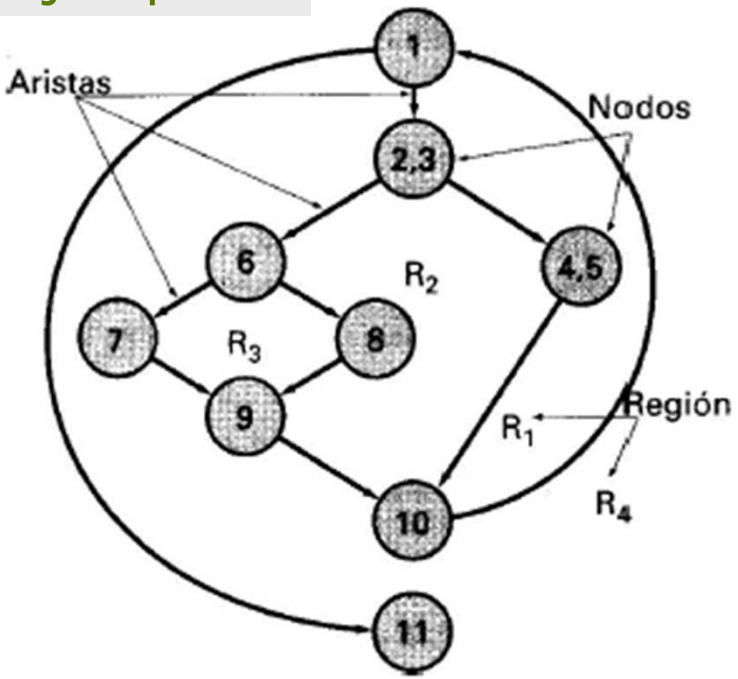
donde P es el número de nodos predicado contenidos en el grafo de flujo G.



$$V(G) = P + 1$$
$$V(G) = 2 + 1$$
$$V(G) = 3$$

Caja Blanca - Criterio de Cobertura de Camino

Ejemplo



Grafo de Flujo

Nodos (N): una o más sentencias.

Aristas (A): flujos de control.

Regiones: áreas delimitadas por aristas y nodos.

$$V(G) = A - N + 2$$

$$11 - 9 + 2 = 4$$

Un camino independiente es cualquier camino del programa que introduce al menos un nuevo conjunto de sentencias o una nueva arista.

camino 1: 1-11
camino 2: 1-2-3-4-5-10-1-11
camino 3: 1-2-3-6-8-9-10-1-11
camino 4: 1-2-3-6-7-9-10-1-11

Ej. dos sentencias if-then-else secuenciales generan 4 CP.

Una sentencia while genera 3 CP: no entra al loop, un nro. máximo de veces, y un nro. promedio de veces.

Testing in the Small (Prueba de Unidades)

- **Prueba de Caja Negra o Funcional**

  - Análisis del Valor Límite

  - Prueba de Clases de Equivalencia

  - Pruebas basadas en Tablas de Decisión

Caja Negra - Análisis del Valor Límite

Define un rango de valores posibles para una variable y realiza la prueba focalizando en los valores de los extremos del rango.

- Esta técnica trabaja con variables independientes
  (no es apropiada cuando hay dependencia entre variables)

- Se aplica con variables que se refieren a cantidades físicas: temperatura, velocidad del aire, presión, ángulo, etc.

Dada una función F (programa) y dos valores de entrada x1 y x2, las variables tendrán valores límites:

$a \leq x1 \leq b$ $c \leq x2 \leq d$

Caja Negra - Análisis del Valor Límite

- El análisis de valor límite se focaliza en los límites del espacio de entrada para identificar los Casos de Prueba.
- Esta técnica asegura que los errores tienden a ocurrir cerca de los valores extremos de una variable de entrada.
- Define los casos de prueba con el min, min+1, nom, max, max-1 para cada variable.
- Fija cada variable en el valor nominal y se mueve por los valores de la otra variable.

Caja Negra - Análisis del Valor Límite

Ejemplo: dados los lados de un triángulo, definir qué tipo de triángulo es.

Para las variables a, b y c, se define el rango entre 1 y 200.

min=1 min+1=2 nom=100 max=200 max-1=199

Caso	a	b	c	Salida Esperada
1	100	100	1	Isósceles
2	100	100	2	Isósceles
3	100	100	100	Equilátero
4	100	100	199	Isósceles
5	100	100	200	No es un Triáng.
6	100	1	100	Isósceles
7	100	2	100	Isósceles
8	100	100	100	Equilátero
9	100	199	100	Isósceles
10	100	200	100	No es un Triáng.
11	1	100	100	Isósceles
12	2	100	100	Isósceles
13	100	100	100	Equilátero
14	199	100	100	Isósceles
15	200	100	100	No es un Triáng.

Caja Negra - Análisis del Valor Límite

Variantes de la Técnica de Análisis de Valor Límite

- **Prueba de Robustez**: extensión de la técnica de análisis de valor límite. Agrega los valores min-1 y el máx+1 para cada variable. El objetivo es determinar qué pasa cuando aparece un valor que excede el máximo permitido y debería direccionar a una excepción.

Ejemplo: Triángulo

Para las variables a, b y c, se define el rango entre 1 y 200.

min=1 min-1= 0 min+1=2 nom=100 max=200 max-1=199 max+1=201

Caso	a	b	c	Salida Esperada
1	100	100	0	No es un triángulo
2	100	100	1	Isósceles
3	100	100	2	Isósceles
4	100	100	100	Equilátero
5	100	100	200	No es un Triáng.
6	100	100	199	Isósceles
7	100	100	201	No es un triángulo
...	...	...	...	...

Caja Negra - Análisis del Valor Límite

Variantes de la Técnica de Análisis de Valor Límite

- **Prueba del Peor Caso**: generalización de la técnica de análisis de valor límite. Los casos de prueba se definen realizando el producto cartesiano con los x valores posibles para cada una de las n variables (x^n). Esta técnica requiere de mucho esfuerzo.

Ejemplo: Triángulo

Para las tres variables a, b y c, con el mismo rango. Cada variable asume 5 valores posibles. Los CP son $5^3 = 125$.

min=1 min+1=2 nom=100 max=200 max-1=199

Caso	a	b	c	Salida Esperada
1	100	100	1	Isósceles
2	100	100	2	Isósceles
3	100	100	100	Equilátero
4	100	100	199	Isósceles
...	...	...	...	...
...	...	...	...	...
125	200	200	200	Equilátero

Caja Negra - Prueba de Clases de Equivalencia

Permite particionar un conjunto, donde dicha partición tiene una colección de subconjuntos mutuamente disjuntos y su unión forma el conjunto entero.

Clases de Equivalencia: forman particiones en un conjunto.

- El conjunto entero es una forma de completitud, y la disyunción asegura la no redundancia.
- Los subconjuntos o clases de equivalencia son determinados por una relación de equivalencia.
- Métodos de prueba con Clase de Equivalencia: débil y fuerte.

Caja Negra - Prueba de Clases de Equivalencia

Prueba con Clase de Equivalencia Débil

Un valor para cada clase de equivalencia es un caso de prueba.

Supongamos:

$A = A1 \cup A2 \cup A3$

$B = B1 \cup B2 \cup B3 \cup B4$

$C = C1 \cup C2$

Caso	a	b	c
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c1
4	a1	b4	c2

Caja Negra - Prueba de Clases de Equivalencia

Prueba con Clase de Equivalencia Fuerte

Define el producto cartesiano entre las clases de equivalencia definidas. Ej: A x B x C = 3 x 4 x 2 = 24 casos de prueba.

El producto cartesiano garantiza la propiedad de completitud, porque cubre todas las clases de equivalencia posibles, y se obtiene un caso de prueba de cada posible combinación de entradas.

Caso	a	b	c
1	a1	b1	c1
2	a1	b1	c2
3	a1	b2	c1
4	a1	b2	c2
5	a1	b3	c1
6	a1	b3	c2
.....	.....	.....	.....
22	a3	b3	c2
23	a3	b4	c1
24	a3	b4	c2

Caja Negra - Pruebas basadas en Tablas de Decisión

Se utilizan para representar y analizar la complejidad de las relaciones lógicas. Describen situaciones en las cuales un número de combinaciones de acciones son realizadas bajo la variación de un conjunto de condiciones.

- La propiedad de completitud de las tablas de decisión garantiza una prueba completa.
- Las tablas de decisión con todas entradas binarias se llaman Tablas de Decisión de Entrada Limitada.
- Si las condiciones permiten tener valores distintos, las tablas resultantes se llaman Tablas de Decisión de Entrada Extendida.
- No hay un orden particular para las condiciones, ni para la ocurrencia de las acciones seleccionadas.

Caja Negra - Pruebas basadas en Tablas de Decisión

Para identificar los casos de prueba con Tablas de Decisión se identifican condiciones como entrada y acciones como salidas. Entonces las reglas son los casos de prueba.

Ejemplo: El problema del Triángulo

c1: a,b,c son parte de un triángulo?	N							Y							
c2: a = b ?	-			Y						N					
c3: a = c ?	-	Y				N				Y				N	
c4: b = c ?	-	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
a1: no es un triángulo	X														
a2: Escaleno															X
a3: Isósceles							X				X		X		
a4: Equilátero		X													
a5: Imposible			X	X					X						

Bibliografía

- Software Testing. A Craftsman´s Approach. Paul Jorgensen. 1995.

  Capítulo 5: Boundary Value Testing

  Capítulo 6: Equivalence Class Testing

  Capítulo 7: Decision Table Based Testing

- Software Engineering. A Practitioner´s Approach, Eighth Edition. Roger S. Pressman. 2015.

  Capítulo 22: Software Testing Strategies

  Capítulo del 23 al 26: Métodos y técnicas de testing que implementan las estrategias

- Fundamentals of Software Engineering. C. Ghezzi, M. Jazayeri, D. Mandrioli. 1991.

  Capítulo 6: Verification. 6.1 6.2. 6.3.