

Módulo 9: Testing Funcional



Diseño y Desarrollo de Software *(1er. Cuat. 2019)*


Profesora titular de la cátedra:
Marcela Capobianco

Profesores interinos:
Sebastian Gottifredi y
Gerardo I. Simari

**Licenciatura en Ciencias de
la Computación – UNS**

Licencia



- Copyright ©2019 Marcela Capobianco.
 - Se asegura la libertad para copiar, distribuir y modificar este documento de acuerdo a los términos de la GNU Free Documentation License, Version 1.2 o cualquiera posterior publicada por la Free Software Foundation, sin secciones invariantes ni textos de cubierta delantera o trasera.
 - Una copia de esta licencia está siempre disponible en la página <http://www.gnu.org/copyleft/fdl.html>
- 

Resumen



- Las **técnicas** de prueba intentan establecer una **guía sistemática** para desarrollar pruebas que:
 - comprueben la **lógica** de los **componentes**
 - **verifiquen** los dominios de **entrada y salida** del programa
- El SW se prueba desde dos perspectivas:
 - pruebas de **caja negra** (requerimientos del software)
 - pruebas de **caja blanca** (lógica interna)



Pruebas de *caja negra* (Testing Funcional)

- Se centran en los **requerimientos funcionales** del software.
- El programa es considerado como una *función* de sus **entradas**.
- Se intentan encontrar **defectos**:
 - **Funciones** incorrectas o ausentes
 - Defectos de **interfaz**
 - Defectos en **estructuras de datos**
 - Problemas de **inicialización y terminación**
 - Problemas de **rendimiento**

Clasificación de técnicas funcionales/caja negra



- Testing de **valores límite**
- Testing basado en **tablas de decisiones**
- Testing de **clases de equivalencia**





Análisis de Valores Límite

Análisis de valores límite

Supongamos que una función $F(x_1, x_2)$ es implementada como un programa tal que:

- x_1 y x_2 tienen *valores límites*:
 - $a \leq x_1 \leq b$ y
 - $c \leq x_2 \leq d$.
- Debemos asegurarnos que la función se aplica dentro de los rangos validos
 - Esto algunas veces lo determina automáticamente el tipo de dato y es controlado en compilación
 - Esta técnica nos interesa cuando no es ese caso!

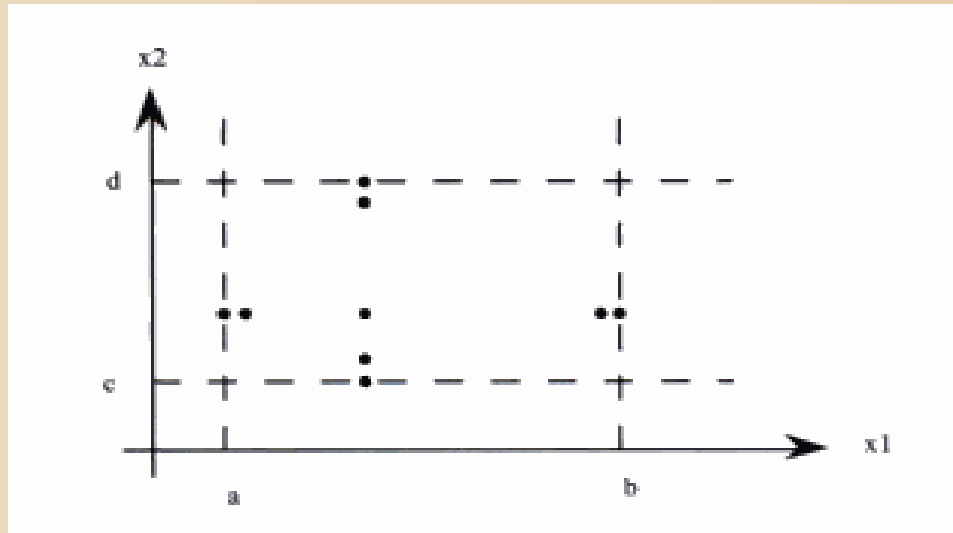
Análisis de valores límite

- Se focaliza en los **límites** del espacio de **entrada** para **identificar casos de prueba**.
- La idea básica es testear en los siguientes valores:
 - ***min, min+***: Valor mínimo y el próximo
 - ***nom***: Valor nominal
 - ***max, max-***: Valor máximo y el anterior
- ***Suposición crítica***: Rara vez una falla resulta de dos o más defectos *simultáneos*.
 - Los casos de test para funciones con **más** de una **variable** de entrada **mantienen** todas las variables **menos una** en el ***valor nominal***.

Análisis de valores límite

Ejemplo para función F de dos variables:

$\{(X1_{nom}, X2_{min}), (X1_{nom}, X2_{min+}), (X1_{nom}, X2_{nom}),$
 $(X1_{nom}, X2_{max-}), (X1_{nom}, X2_{max}), (X1_{min}, X2_{nom}),$
 $(X1_{min+}, X2_{nom}), (X1_{max-}, X2_{nom}), (X1_{max}, X2_{nom})\}$




Análisis de valores límite

- Claramente, se puede generalizar a funciones de cualquier cantidad de variables.
 - Para n variables obtenemos $4n+1$ casos de test.
- No tiene sentido para variables booleanas o tipos que no permiten una eenumeración de más de 4 valores.
- No es muy util cuando hay mucha dependencia entre las variables (por ejemplo en *NextDate*).

Testing de robustez



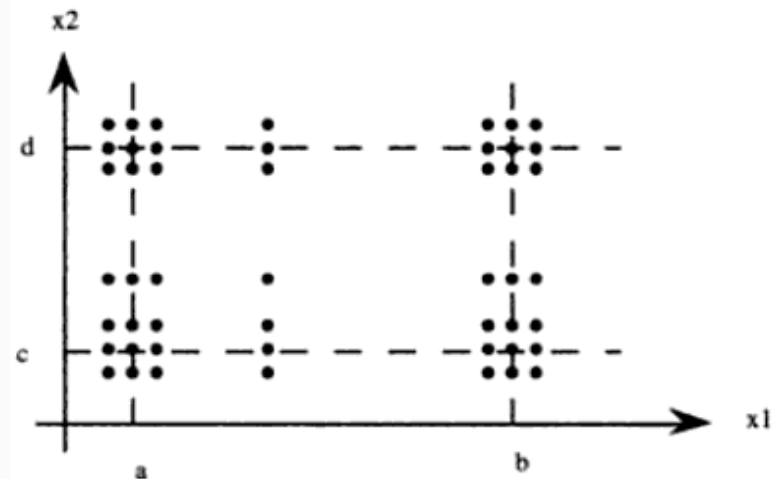
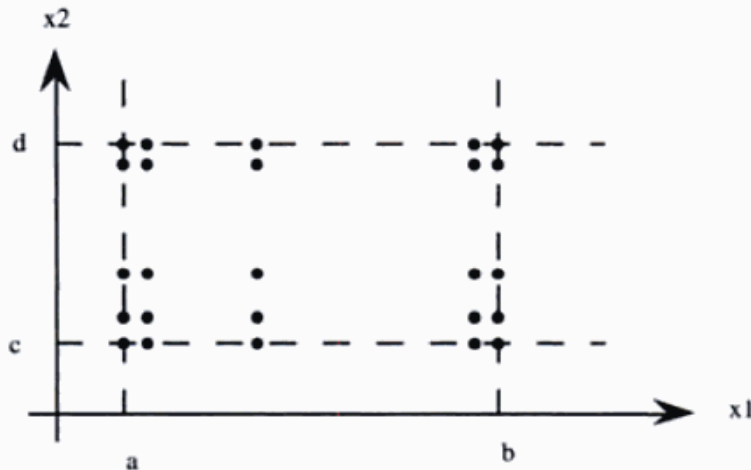
- Es una **extensión simple** del análisis de valores límite donde también se prueban dos valores más:
 - *max+*
 - *min-*
 - Fuerza a fijar la **atención** en el manejo de **excepciones y mensajes de error**
 - Por ejemplo, ¿Qué pasa si un ascensor excede su capacidad de carga?
- 

Testing del peor caso

- El testeo del peor caso *no tiene en cuenta* la suposición crítica
- Queremos ver qué pasa cuando **más de una variable** tiene un **valor extremo**.
- Para **dos o mas** variables tomamos el *producto cartesiano* de $\{min, min+, nom, max-, max\}$.

Testing del peor caso

- Ahora la función del número de casos es exponencial: 5^n para n variables.
- Para ser todavía más cautelosos, podemos *combinarlo* con testing de robustez.



Testing de valores especiales



- Es la más usada y la más intuitiva.
- La persona que testea debe usar su conocimiento del dominio (*ad hoc*).
- Es un testimonio del “arte del testing”.



Recomendaciones para testing de valores límite



- Es el más **rudimentario**, supone que las variables son **independientes**.
 - Esto lleva a problemas en algunos casos (“31 de febrero de 1905”).
- Pueden ser **normales** o **robustos**, defecto simple vs. multiple.
- Dado que la **cantidad de casos varía mucho**, se debe distinguir y usar en forma juiciosa el que corresponda.





Tablas de Decisión



Tablas de decisión

- Es un método clásico y útil para analizar *relaciones lógicas* más **complejas** entre los **datos**.
- Son ideales para situaciones en las que tenemos esquemas *patrón-acción*.
- Tienen dos partes:
 - **Condiciones:** Cada columna en esta parte será una regla.
 - **Acciones:** Denotan qué debe ocurrir si se dan las diferentes condiciones. Puede ser útil tener una acción especial correspondiente a casos *imposibles*.

Tablas de decisión: Notación

Table 7.1 Portions of a Decision Table

<i>Stub</i>	<i>Rule 1</i>	<i>Rule 2</i>	<i>Rules 3, 4</i>	<i>Rule 5</i>	<i>Rule 6</i>	<i>Rules 7, 8</i>
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	—	T	F	—
a1	X	X		X		
a2	X				X	
a3		X		X		
a4			X			X

Tablas de decisión

- La parte de **condiciones**, en general, es una **tabla de verdad** aumentada con valores “*don't care*” (no importa el valor), denotados con “-”.
- Esto lleva a que consideremos sistemáticamente **todas las combinaciones** para ejercitar las **condiciones a fondo**.
- Para generar **casos de test**, interpretamos a las **condiciones** como **entradas** y a las **acciones** como **salidas**.
 - *Las reglas son los casos de testeo.*

Ejemplo para el problema del triángulo

Table 7.2 Decision Table for Triangle Problem

c1: a, b, c form a triangle?	F	T	T	T	T	T	T	T	T
c2: a = b?	—	T	T	T	T	F	F	F	F
c3: a = c?	—	T	T	F	F	T	T	F	F
c4: b = c?	—	T	F	T	F	T	F	T	F
a1: Not a triangle	X								
a2: Scalene									X
a3: Isosceles					X		X	X	
a4: Equilateral		X							
a5: Impossible			X	X		X			

Con distintas condiciones

Table 7.3 Refined Decision Table for Triangle Problem

c1: $a < b + c$?	F	T	T	T	T	T	T	T	T	T	T
c2: $b < a + c$?	—	F	T	T	T	T	T	T	T	T	T
c3: $c < a + b$?	—	—	F	T	T	T	T	T	T	T	T
c4: $a = b$?	—	—	—	T	T	T	T	F	F	F	F
c5: $a = c$?	—	—	—	T	T	F	F	T	T	F	F
c6: $b = c$?	—	—	—	T	F	T	F	T	F	T	F
a1: Not a triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

Uso de *don't care*



- Para tablas del tipo sin “don't care”, se cumple que para n condiciones hay 2^n reglas.
- Cada regla sin *don't cares* cuenta como una.
- Pero cada entrada *don't care* duplica el contador de la regla.



Contando las reglas

Table 7.5 Decision Table for Table 7.3 with Rule Counts

c1: $a < b + c$?	F	T	T	T	T	T	T	T	T	T	T
c2: $b < a + c$?	—	F	T	T	T	T	T	T	T	T	T
c3: $c < a + b$?	—	—	F	T	T	T	T	T	T	T	T
c4: $a = b$?	—	—	—	T	T	T	T	F	F	F	F
c5: $a = c$?	—	—	—	T	T	F	F	T	T	F	F
c6: $b = c$?	—	—	—	T	F	T	F	T	F	T	F
Rule count	32	16	8	1	1	1	1	1	1	1	1
a1: Not a triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

Tablas con Problemas: Redundancia



Table 7.9 A Redundant Decision Table

<i>Conditions</i>	1-4	5	6	7	8	9
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	X
a2	—	X	X	X	—	—
a3	X	—	X	X	X	X




Tablas con Problemas: Inconcistencia

Table 7.10 An Inconsistent Decision Table

<i>Conditions</i>	1-4	5	6	7	8	9
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	—
a2	—	X	X	X	—	X
a3	X	—	X	X	X	—

Tablas con Condiciones de Valor Multiple



- Permiten crear tablas donde una **condicion** puede tomar **mas de dos valores**.
 - En general son utilizadas para **datos** que pueden tomar un **valor** en dentro de un **rango reducido enumerable**
 - Nos evita la multiplicada de condiciones V/F
 - Por ejemplo, en NextDate:
 - La condicion Mes podria tomar el valor
 - M1: Mes de 28 días
 - M2: Mes de 30 días
 - M3: Mes de 31 días
- 

Casos de test para *NextDate*

- La función *NextDate* es buena para ilustrar dependencias entre datos de entrada.
- El formato de tabla de decisión nos permite explicitar los casos imposibles.
- Las acciones que tenemos se van a usar son:
 - Incrementar Día
 - Incrementar Mes
 - Incrementar Año
 - Resetear Día
 - Resetear Mes
 - Fecha Imposible

NextDate: Tabla de Decisión



- Condiciones sobre Día:
 - D1: {día entre 1 y 27}
 - D2: día = 28
 - D3: día = 29
 - D4: día = 30
 - D5: día = 31
- Condiciones sobre Mes:
 - M1: mes tiene 30 días
 - M2: mes tiene 31 días excepto diciembre
 - M3: mes = diciembre
 - M4: mes = febrero
- Condiciones sobre Año:
 - Y1: año es bisiesto
 - Y2: año no es bisiesto

Acciones:
Incrementar Día
Incrementar Mes
Incrementar Año
Resetear Día
Resetear Mes
Fecha Imposible

NextDate: Tabla de Decisión



- Cond. Día:
 - D1: {día entre 1 y 27}
 - D2: día = 28
 - D3: día = 29
 - D4: día = 30
 - D5: día = 31
- Cond. Mes:
 - M1: mes de 30 días
 - M2: mes de 31 días sin dic.
 - M3: mes = diciembre
 - M4: mes = febrero
- Cond. Año:
 - Y1: año es bisiesto
 - Y2: año no es bisiesto

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
c1: Month in	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2	M3	M3	M3	M3	M3	M4	M4	M4	M4	M4	M4	M4
c2: Day in	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5	D1	D2	D2	D3	D3	D4	D5
c3: Year in	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Y1	Y2	Y1	Y2	-	-
Actions																						
a1: Impossible					X															X	X	X
a2: Increment day	X	X	X			X	X	X	X		X	X	X	X		X	X					
a3: Reset day				X						X					X			X	X			
a4: Increment month				X						X								X	X			
a5: Reset month															X							
a6: Increment year															X							

NextDate: Tabla de Decisión



- Cond. Día:
 - D1: {día entre 1 y 27}
 - D2: día = 28
 - D3: día = 29
 - D4: día = 30
 - D5: día = 31
- Cond. Mes:
 - M1: mes de 30 días
 - M2: mes de 31 días sin dic.
 - M3: mes = diciembre
 - M4: mes = febrero
- Cond. Año:
 - Y1: año es bisiesto
 - Y2: año no es bisiesto

Podemos Agrupar mas!!!

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
c1: Month in	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2	M3	M3	M3	M3	M3	M4	M4	M4	M4	M4	M4	M4	
c2: Day in	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5	D1	D2	D2	D3	D3	D4	D5	
c3: Year in	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Y1	Y2	Y1	Y2	-	-	
Actions																							
a1: Impossible					X																X	X	X
a2: Increment day	X	X	X			X	X	X	X		X	X	X	X		X	X						
a3: Reset day				X						X					X			X	X				
a4: Increment month				X						X								X	X				
a5: Reset month															X								
a6: Increment year															X								

NextDate: Tabla de Decisión

Table 7.16 Decision Table Test Cases for NextDate

<i>Case ID</i>	<i>Month</i>	<i>Day</i>	<i>Year</i>	<i>Expected Output</i>
1-3	4	15	2001	4/16/2001
4	4	30	2001	5/1/2001
5	4	31	2001	Invalid input date
6-9	1	15	2001	1/16/2001
10	1	31	2001	2/1/2001
11-14	12	15	2001	12/16/2001
15	12	31	2001	1/1/2002
16	2	15	2001	2/16/2001
17	2	28	2004	2/29/2004
18	2	28	2001	3/1/2001
19	2	29	2004	3/1/2004
20	2	29	2001	Invalid input date
21, 22	2	30	2001	Invalid input date



Clases de Equivalencia



Testeo de clases de equivalencia

- Recordemos de **Álgebra**:

Las **clases de equivalencia** de un conjunto forman una *partición* del mismo: conjuntos **disjuntos** tal que su **unión** es el conjunto **original**.

- Se logra el conjunto entero: *Completitud*.
- Los conjuntos son disjuntos: *Se evita la redundancia*.
- Los elementos de un conjunto tienen algo en común.

Testeo de clases de equivalencia

- **Idea:** Identificar los casos de test usando un elemento de cada clase de equivalencia.
- **Clave:** Elegir bien la forma de particionar en clases.
- ¿Por que?
 - *Ejemplo:* Para probar triángulos equiláteros elijo un valor como (5, 5, 5); no agregaría nada probar con otros de la forma (X, X, X).

Testeo de clases de equivalencia

- La intuición respecto a lo que vimos de testing estructural: los casos de test dentro de una **misma clase** seguirían el *mismo camino*.
- Distinguiremos dos formas:
 - **Fuerte**
 - **Débil**

Testeo de clases de equivalencia

- Supongamos que tenemos una función P de 3 variables a , b y c con dominios A , B y C .
- Elegimos una **relación de equivalencia** apropiada que induce la siguiente **partición**:
 - $A = A1 \cup A2 \cup A3$
 - $B = B1 \cup B2 \cup B3 \cup B4$
 - $C = C1 \cup C2$
- Elegimos valores: $a1$ pertenece a $A1$, $b1$ pertenece a $B1$ y $c1$ pertenece a $C1$, etc.

Equivalencia débil

Usamos **un valor de cada clase de equivalencia** en un caso de test:

$$T1 = (a1, b1, c1) \quad T2 = (a2, b2, c2)$$

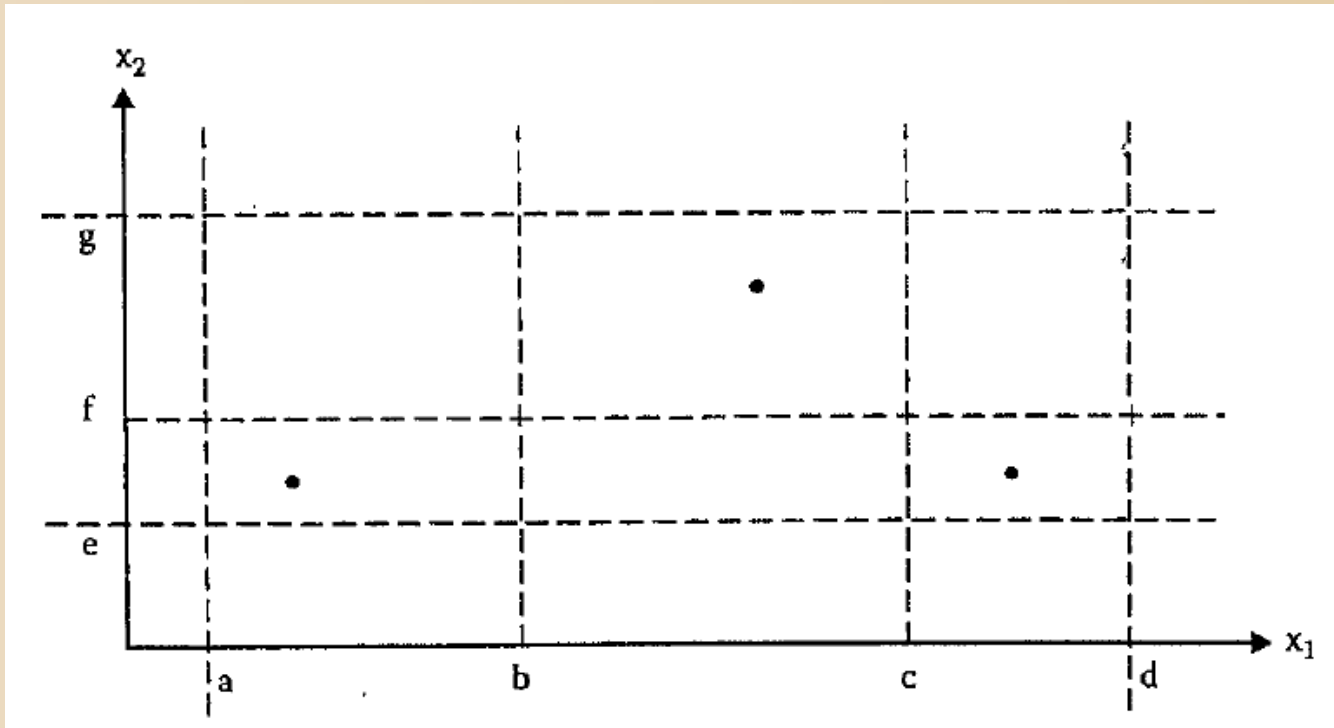
$$T3 = (a3, b3, c1) \quad T4 = (a1, b4, c2)$$

- Siempre tendremos el ***mismo número*** de casos de test que de clases en la partición con ***más subconjuntos***.

Equivalencia débil *normal*

Los datos enteros x_1 y x_2 , los partimos en:

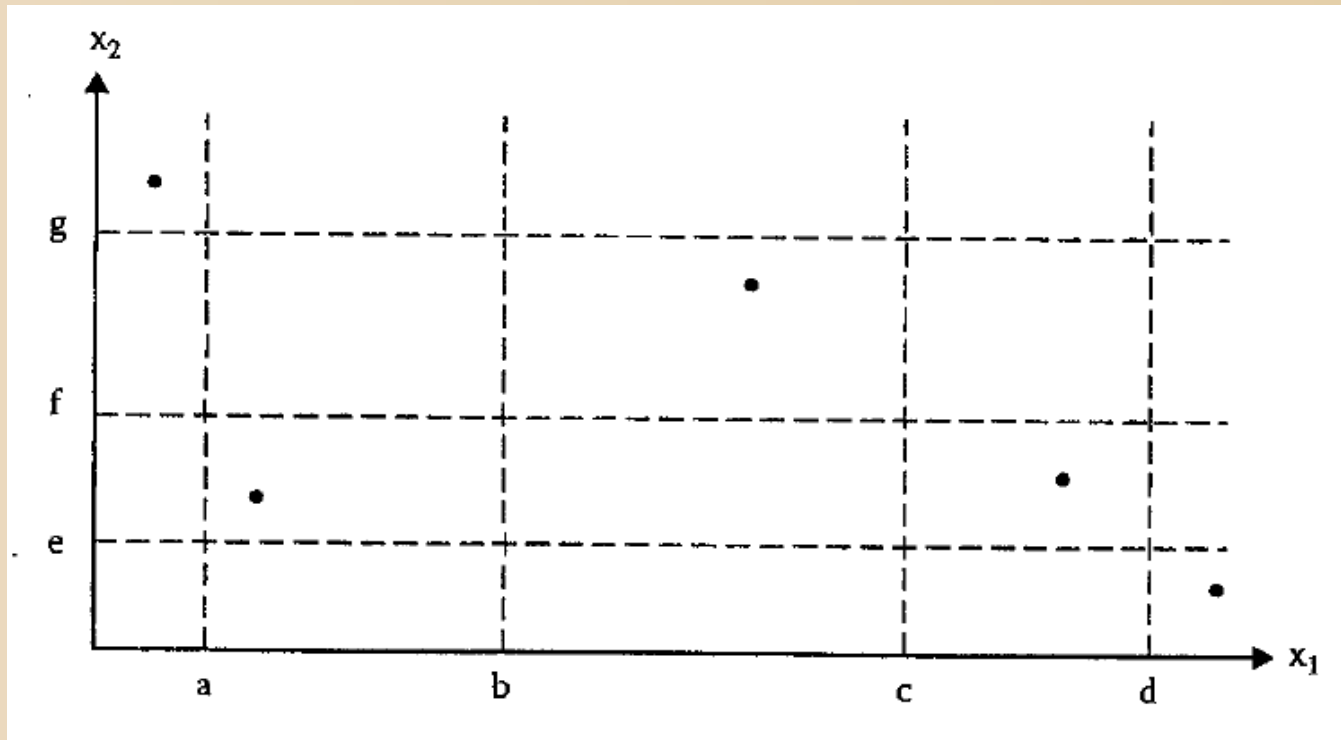
- x_1 en tres clases ($a < x_1 < b$, $b < x_1 < c$, $c < x_1 < d$)
- x_2 en dos clases ($e < x_2 < f$, $f < x_2 < g$)



Equivalencia débil *robusta*

Los datos enteros x_1 y x_2 , los partimos en:

- x_1 en tres clases ($a < x_1 < b$, $b < x_1 < c$, $c < x_1 < d$)
- x_2 en dos clases ($e < x_2 < f$, $f < x_2 < g$)



Equivalencia fuerte

- Se basa en realizar el *producto cartesiano* de los **subconjuntos de particiones**.
- En el caso anterior, $A \times B \times C$ tendrá $3 * 4 * 2 = 24$ elementos. Se usan *todas* las combinaciones:

$$T1 = (a1, b1, c1), T2 = (a1, b1, c2),$$

$$T3 = (a1, b2, c1), T4 = (a1, b2, c2),$$

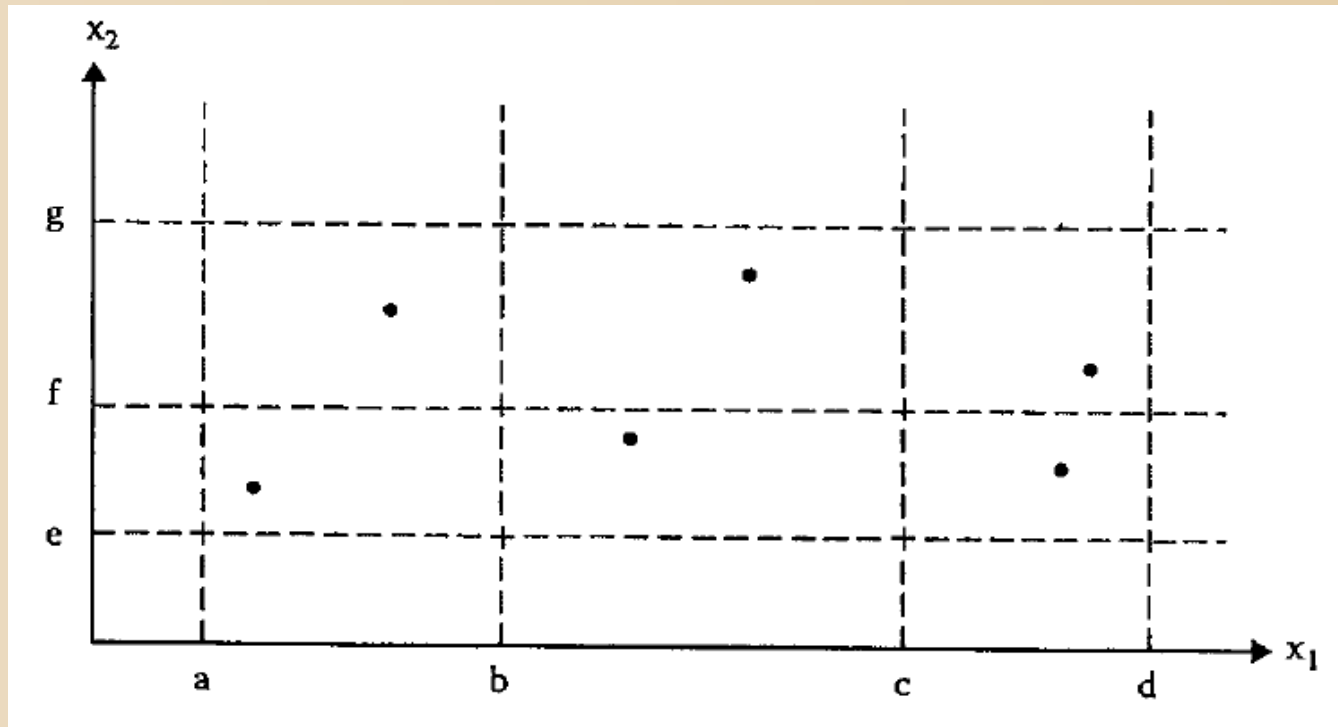
...

- **Parecido** a construir una **tabla de decisiones** que vimos antes.

Equivalencia fuerte *normal*

Los datos enteros x_1 y x_2 , los partimos en:

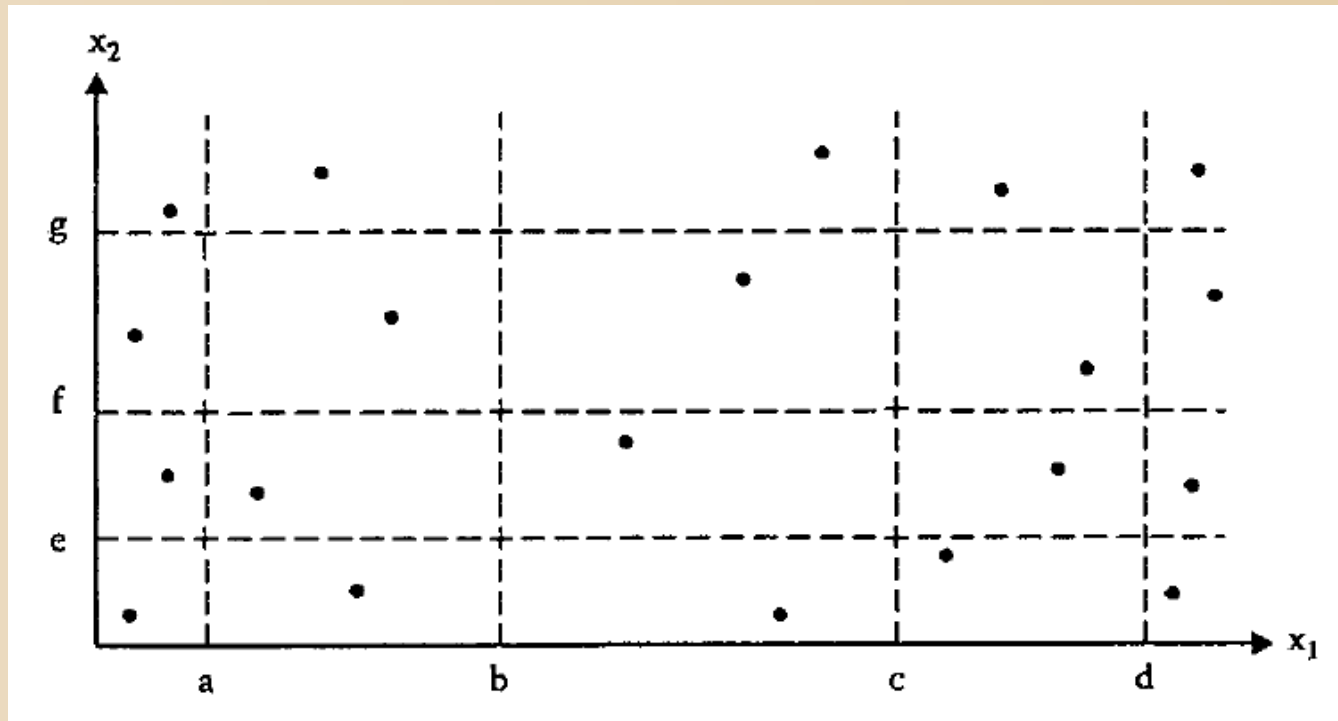
- x_1 en tres clases ($a < x_1 < b$, $b < x_1 < c$, $c < x_1 < d$)
- x_2 en dos clases ($e < x_2 < f$, $f < x_2 < g$)



Equivalencia fuerte *robusta*

Los datos enteros x_1 y x_2 , los partimos en:

- x_1 en tres clases ($a < x_1 < b$, $b < x_1 < c$, $c < x_1 < d$)
- x_2 en dos clases ($e < x_2 < f$, $f < x_2 < g$)



Clases de equivalencia

Considerando los Invalidos



- Los **valores invalidos** tambien los podemos considerar como **clases de equivalencia**.
 - Esto tiene sentido, en general, si son valores permitidos por el lenguaje pero no por nuestro sistema.
- Si las condiciones de **error** son importantes, se puede **dividir** la clase inválida en **varias clases**.



Clases de equivalencia Fuerte(Variante)



- Si consideramos las **CE invalidas** y seguimos la **suposición critica**
- Nos lleva a considerar una **variante** el modo **Fuerte**
 - Para las **CE validas** seguimos la **estrategia fuerte**
y
 - Para por **cada CE invalida** armamos un caso donde solo **un valor** pertenece a una clase **invalida**



Observaciones sobre testeo de clases de equivalencia



- Es **útil** cuando la función/funcionalidad es **compleja**.
- La versión **fuerte** supone cierto grado de **independencia**, si no pueden aparecer **entradas inválidas** (ejemplo *NextDate*).
 - Si hay alta dependencia entre los datos de entrada es mejor otro método.
- Se pueden necesitar varios intentos antes de la **relación correcta**. En otros casos puede ser evidente.



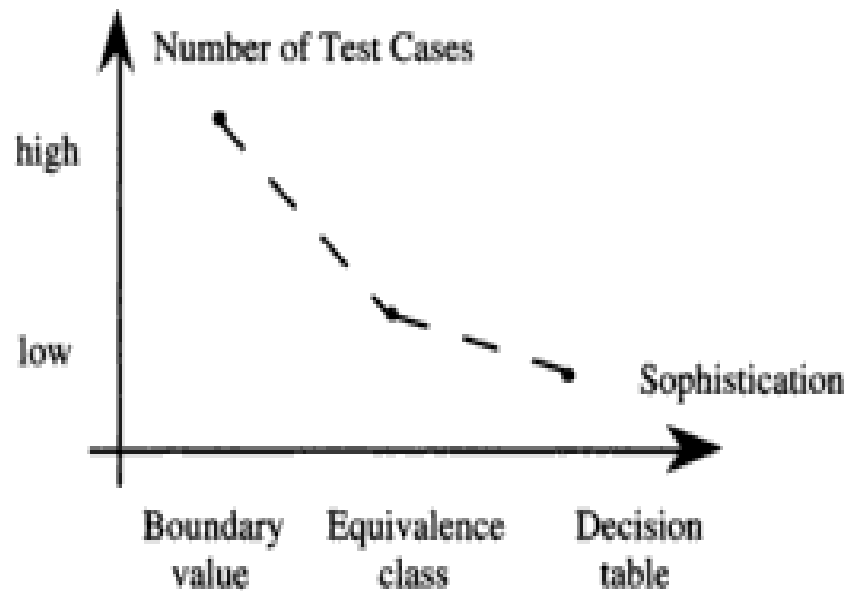
Conclusiones sobre testing funcional



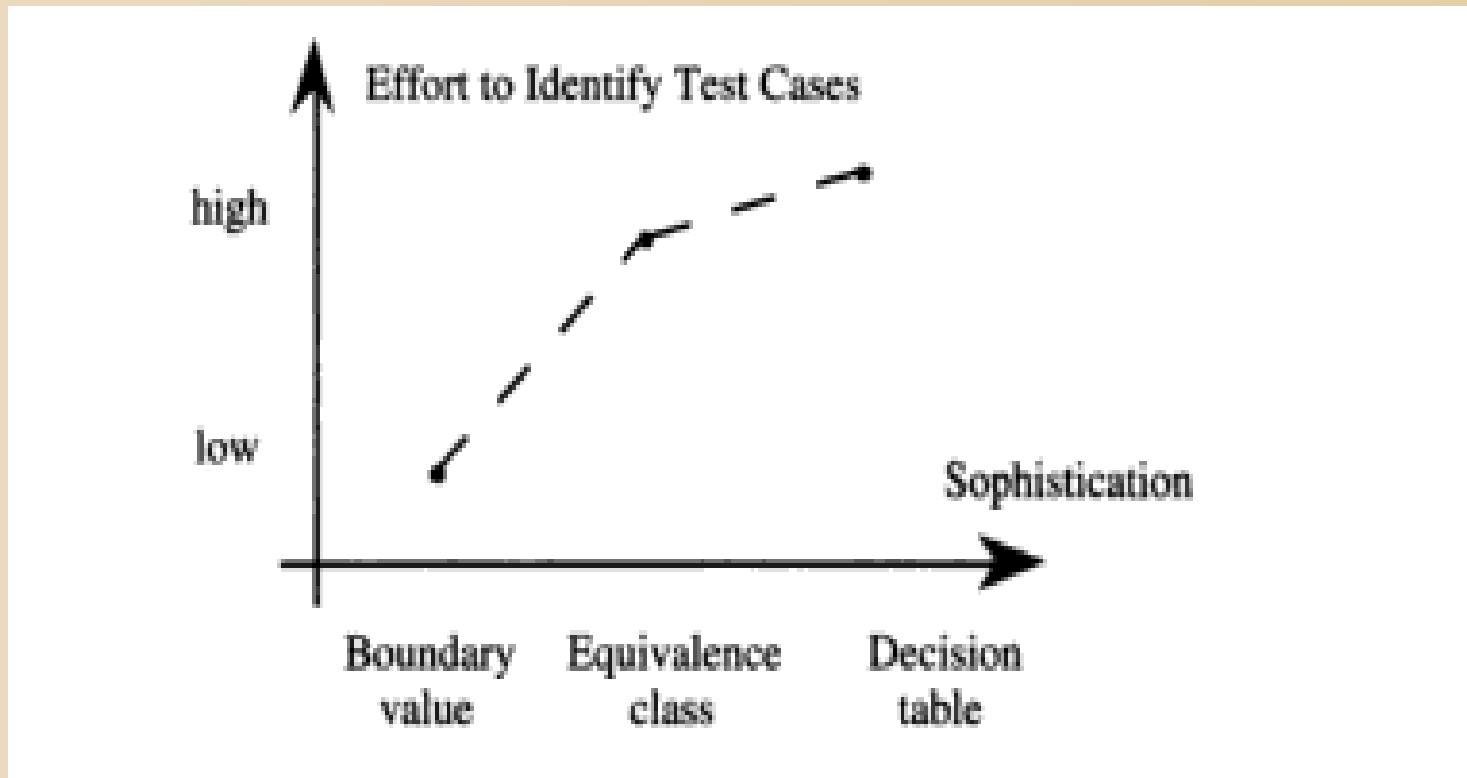
- Comparamos los métodos en términos de **esfuerzo**, **eficiencia** del proceso de testing y **efectividad**.
- Los métodos varían en cuanto al **número** de casos de test generados y al **esfuerzo** para obtener estos casos.
- Veamos algunos gráficos...



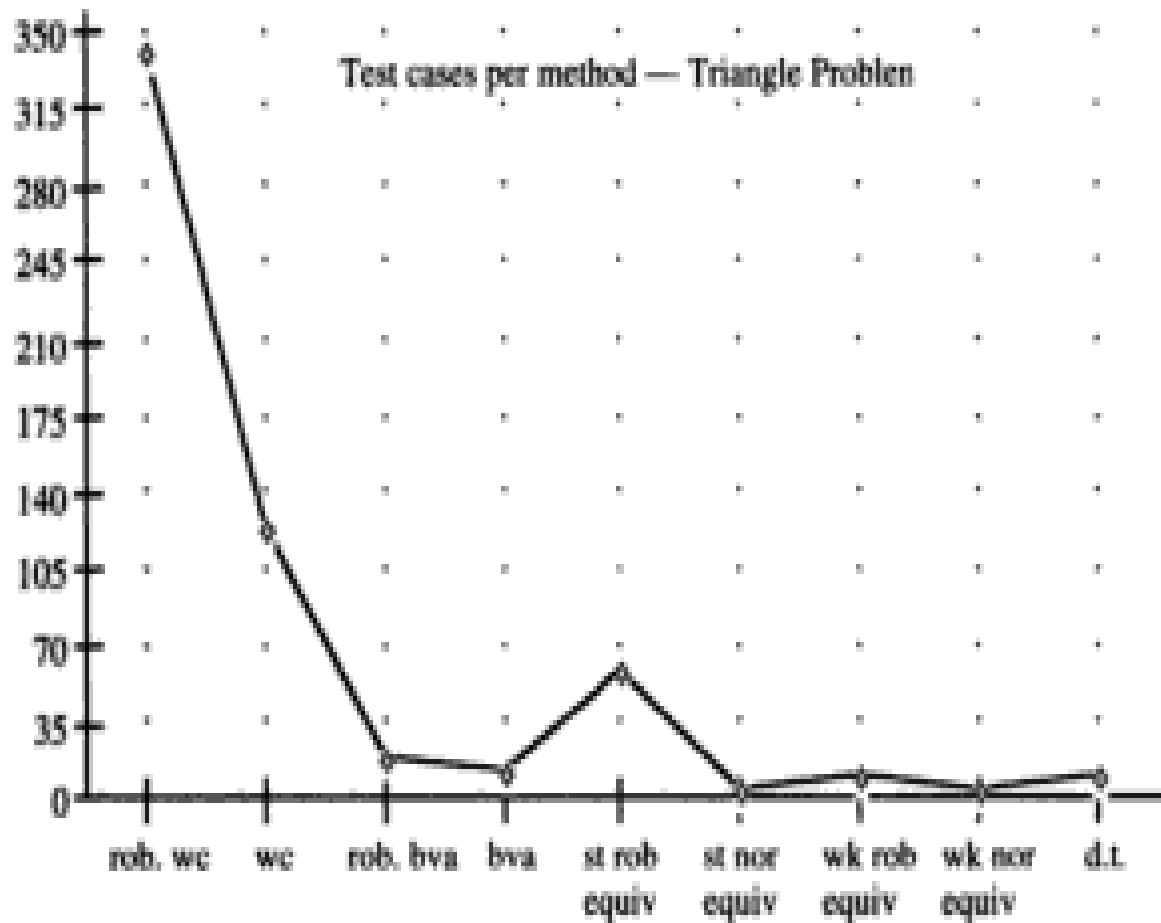
Casos de test por método



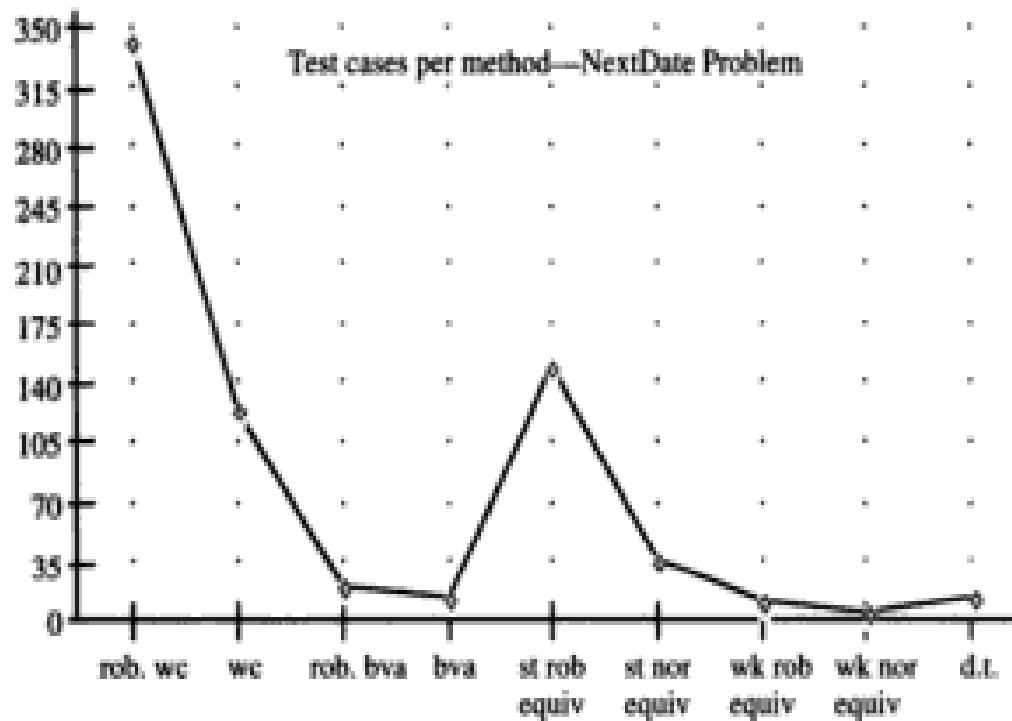
Esfuerzo para identificar los casos por método



Casos de test para el *Problema del Triángulo*



Casos de test para *NextDate*



Atributos importantes para elegir qué método usar



- Ver si las variables representan cantidades **discretas** o no
- Ver si existen *dependencias* entre las variables.
- Si se asumen **fallas simples** o *múltiples*.
- Si es importante el manejo de *excepciones*.



Bibliografía



P. Jorgensen: “*Software Testing: A Craftsman’s Approach*”,
4th Ed. Auerbach Publications, 2013. Capítulos 1, 2 y 5-7.

