Testing Software

Pruebas y mantenimiento de software

Jueves de 8:00 a 11:00

en CReCE

Prof. José Antonio Cervantes Álvarez

antonio.varez@academicos.udg.mx

Unidad IV. Pruebas dinámicas

4.1 Técnicas de prueba de caja negra.

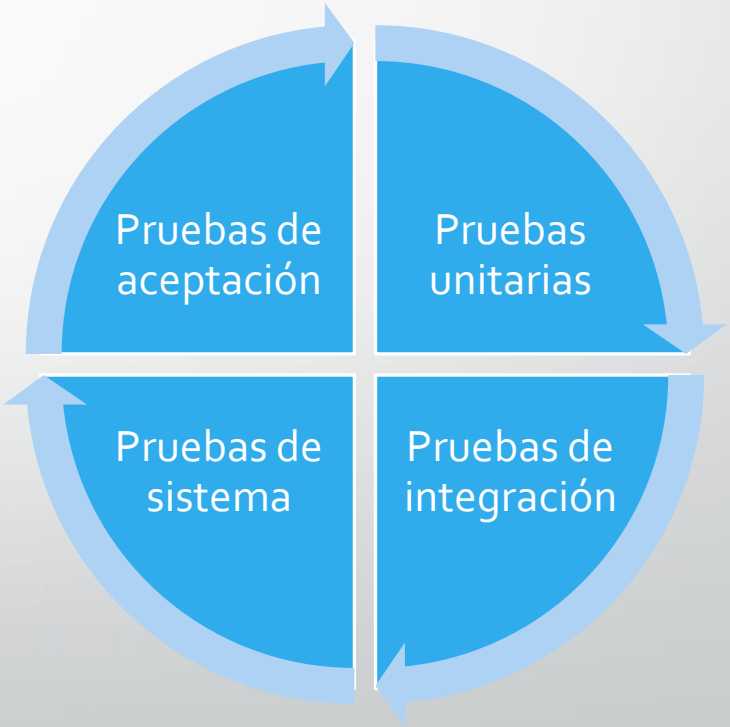
- 4.1.1 Partición de equivalencia.
- 4.1.2 Análisis del valor límite.
- 4.1.3 Pruebas de transición de estado.
- 4.1.4 Pruebas basadas en técnicas de lógica.
- 4.1.5 Pruebas basadas en caso de uso.

4.2 Técnicas de prueba de caja blanca.

- 4.2.1 Pruebas de ruta básica.
- 4.2.2 Pruebas de ciclos o bucles.
- 4.2.3 Pruebas de condiciones.

4.3 Pruebas intuitivas y basadas en la experiencia.

- 4.3.1 Predicción de error.
- 4.3.2 Pruebas exploratorias.



Objetivo

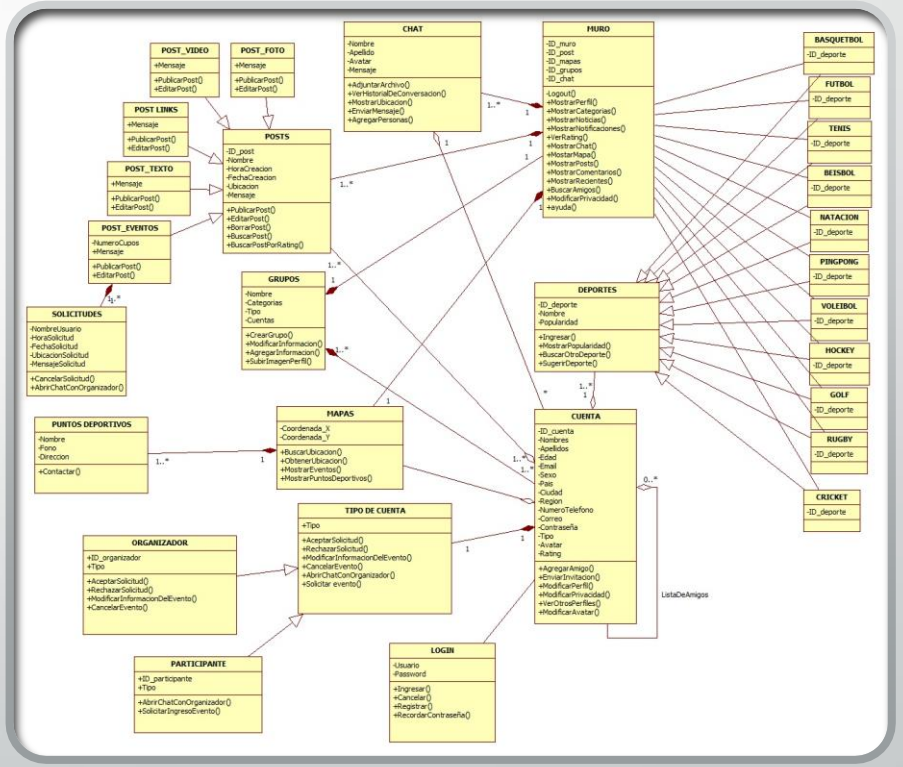
- En esta unidad, los estudiantes aprenderán diferentes técnicas para probar software a través de la ejecución de objetos de prueba en una computadora. Conocerá las técnicas de diseño para el desarrollo de pruebas de caja negra y caja blanca así como las características y diferencias que existen entre estos tipos de pruebas.

Técnicas de diseño de pruebas dinámicas

- Usualmente, la prueba de software es vista como la ejecución del objeto de prueba en una computadora. Donde el objeto de prueba (programa) es alimentada con datos de entrada y ejecutado.

- Para ejecutar el objeto de prueba, el programa debe ser ejecutable. Sin embargo, **en las pruebas de bajo nivel** (pruebas unitarias y de integración), **el objeto de prueba no puede ser ejecutado de manera independiente.** Se requiere embeberlo en una cama de pruebas para obtener un programa ejecutable.

- El objeto de prueba usualmente llamará diferentes partes del programa a través de interfaces predefinidas. Estas partes del programa son representadas por marcadores denominados **stubs**.

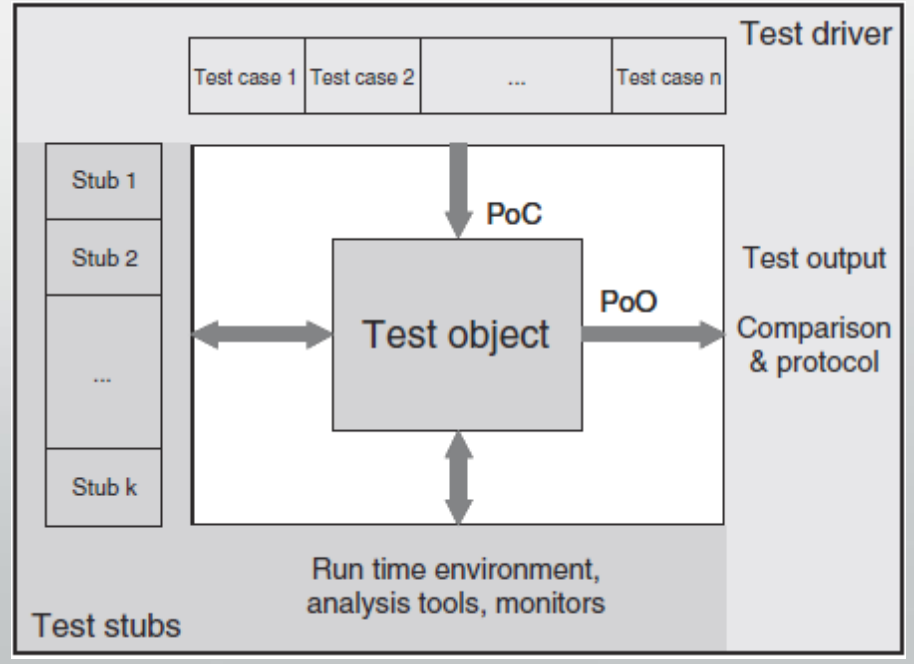


Ejemplo de un proyecto que deberá ser probado utilizando pruebas de bajo (unitarias e integración) y alto nivel (sistema y aceptación)

Cama de pruebas

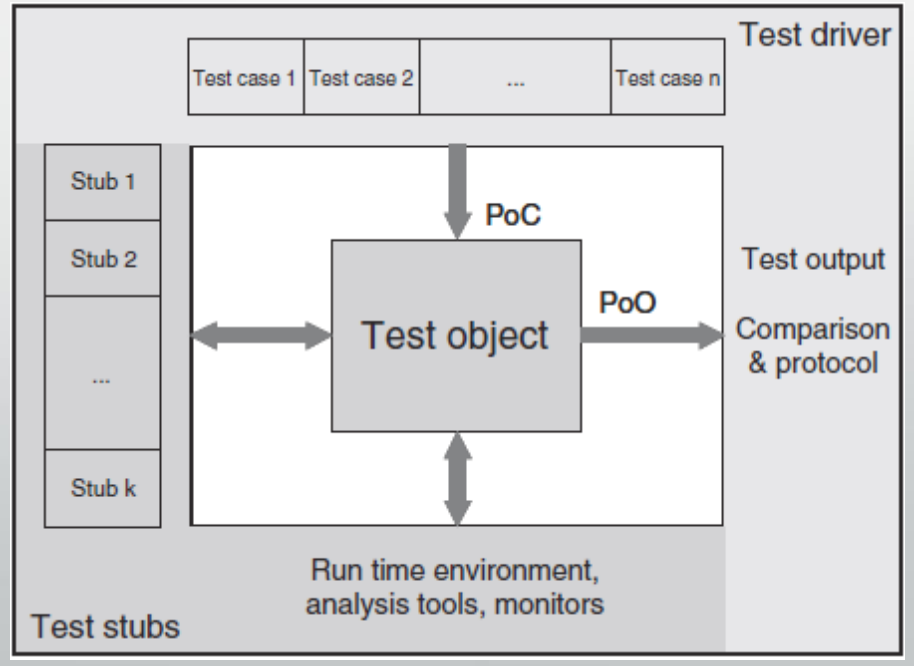
- Los **stubs** sustituyen a las partes del programa que aún no han sido implementadas y por lo tanto no pueden ser usadas. Es decir, deben ser simuladas para la ejecución de una prueba en particular del objeto de prueba.

- Los **stubs** simulan las entradas y salidas (es decir, el comportamiento) de una parte del programa que debe ser llamada por el objeto de prueba.



Cama de pruebas

- La cama de pruebas debe suministrar con entradas al objeto de prueba.
- El **test driver** es el encargado de simular una parte del programa que se supone llama al objeto de prueba.
- El **test driver** y el **stub** se combinan para **formar la cama de pruebas**. Juntos constituyen un programa ejecutable junto con el objeto de prueba.



Técnicas de diseño de pruebas

- El objetivo de las pruebas es que a un costo "relativamente bajo" (como sea posible) revisar/encontrar el mayor número de posibles fallas.
  - Para lograr este objetivo se requiere de un enfoque sistemático para el diseño de las pruebas.
  - **Pruebas no estructuradas basadas** únicamente en el instinto del ingeniero de software **no garantizan que se prueben todas las situaciones** posibles e incluso todas las admitidas por el objeto de pruebas.

Técnicas de diseño de pruebas

- Los pasos para ejecutar las pruebas son:
  - Determinar las condiciones y precondiciones para la prueba y el objetivo a alcanzar.
  - Especificar los casos de prueba individuales.
  - Determinar cómo ejecutar las pruebas (generalmente encadenando varios casos de prueba).

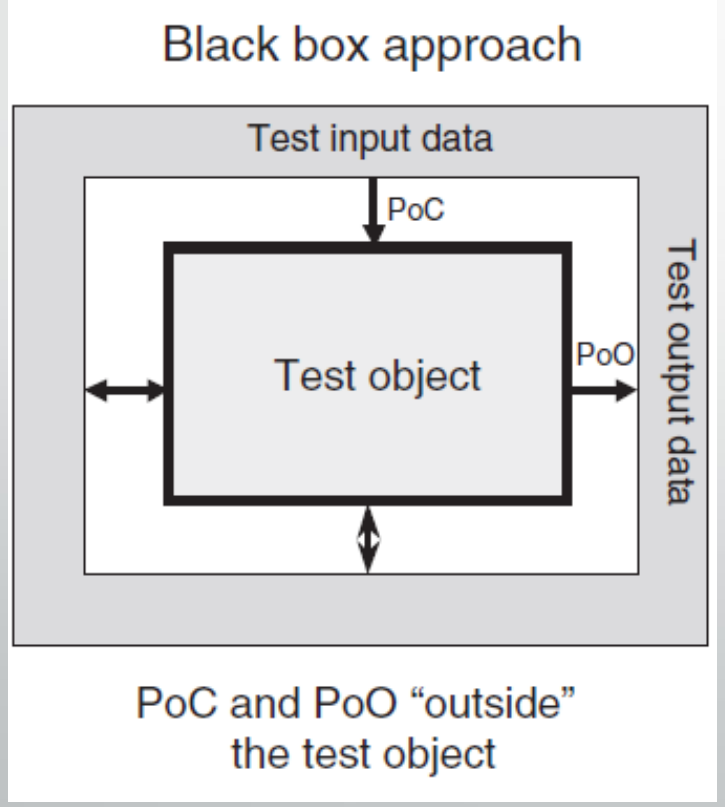
- Esta actividad del diseño y ejecución de pruebas puede ser muy informal o formal. El grado de formalidad depende de varios factores tales como:
  - La naturaleza del software (si se trata de un sistema crítico de seguridad o un prototipo comercial)
  - La madurez de los procesos de desarrollo y prueba.
  - Restricciones de tiempo.
  - El conocimiento y habilidades de los participantes.
  - Etc.

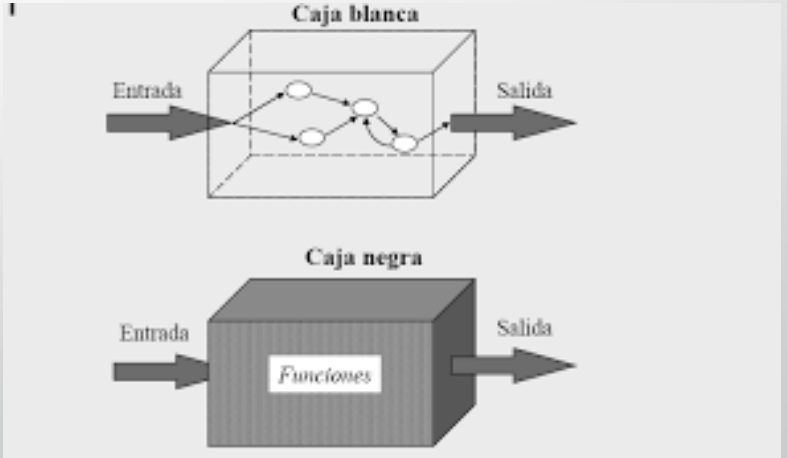
Condiciones, precondiciones y objetivos de las pruebas

- La base para el diseño de la pruebas es analizar y determinar qué debe ser probado.
  - Ejemplo, determinar si una transacción en particular es ejecutada correctamente.
- El objetivo de la prueba debe ser definido.
  - Ejemplo, demostrar que se cumplen los requerimientos.
- Los riesgos de fallo deben ser tomados en consideración.
- El ingeniero de pruebas debe identificar las precondiciones y condiciones necesarias para la prueba, tales como qué datos debería tener la base de datos.

Prueba de caja negra

- En las pruebas de caja negra, el objeto de prueba es visto como una caja negra.
- Las **pruebas de caja negra** también son denominadas **pruebas funcionales** porque el principal objetivo es observar el comportamiento de las entradas y salidas del objeto bajo prueba.





Prueba de caja negra

- **Los casos de prueba son derivados a partir de la especificación del objeto de prueba.**
- No es necesaria tener información de las estructuras internas del objeto de pruebas.
- El comportamiento del objeto de pruebas es observado desde afuera.
- **Los casos de prueba son diseñados usando las especificaciones o requerimientos del objeto de prueba.**
- Las **pruebas de caja negra** son utilizadas en el desarrollo de pruebas de alto nivel tales como **pruebas de sistema y de aceptación.**

Técnicas de prueba de caja negra

- Durante el diseño de la prueba, un subconjunto razonable de todos los posibles casos de prueba puede ser seleccionado.

- Existen diferentes métodos para el diseño de las pruebas.
  - Partición de clases de equivalencia (o partición de equivalencia).
  - Análisis del valor límite.
  - Pruebas de transición de estado.
  - Pruebas basadas en técnicas de lógica (grafo causa-efecto y tablas de decisión)
  - Pruebas basadas en casos de uso.

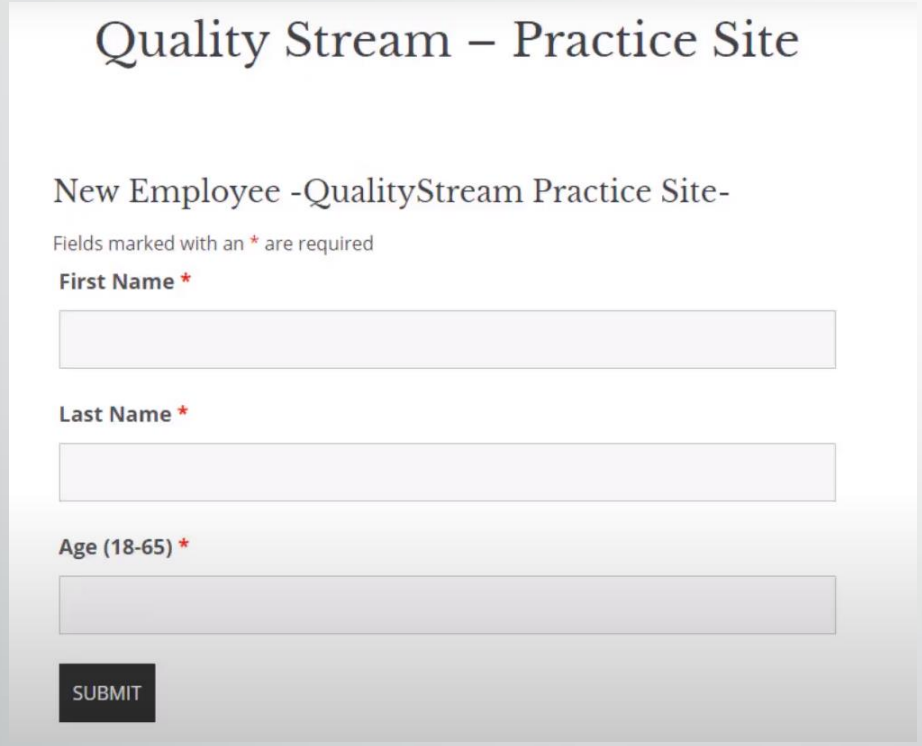
Partición de equivalencia

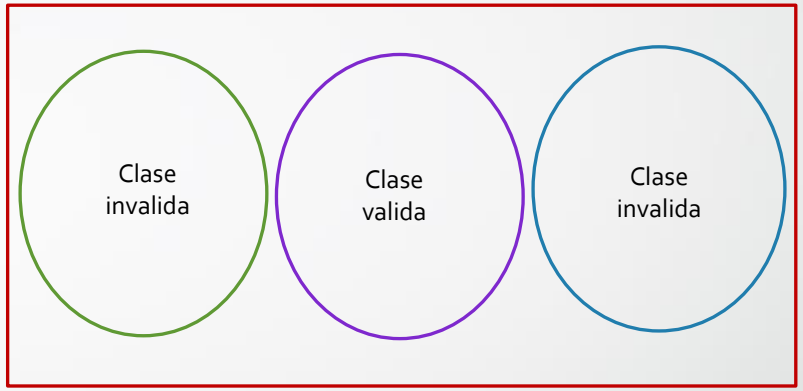
- El dominio de los posibles datos de entrada para cada elemento de entrada es dividido en clases de equivalencia.

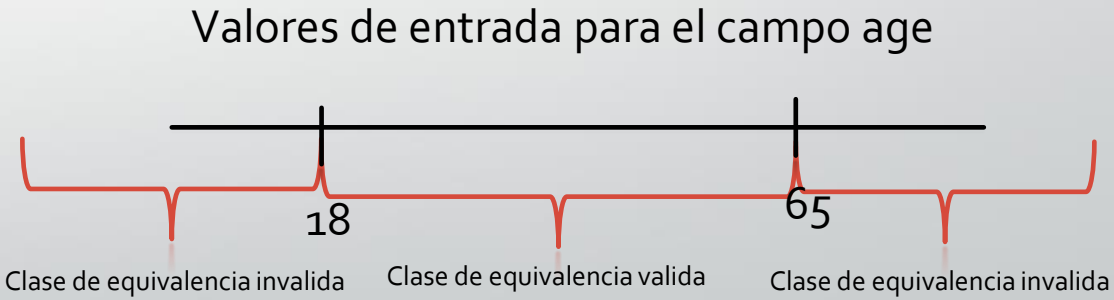
- **Una clase de equivalencia es un conjunto de valores** (datos de entrada) que el ingeniero de pruebas asume son procesadas de la misma forma por el objeto de prueba.

- La prueba de un valor de la clase de equivalencia se considera suficiente porque se asume que el objeto de prueba mostraría el mismo comportamiento con cualquier otro valor de entrada de la misma clase.

Partición de equivalencia



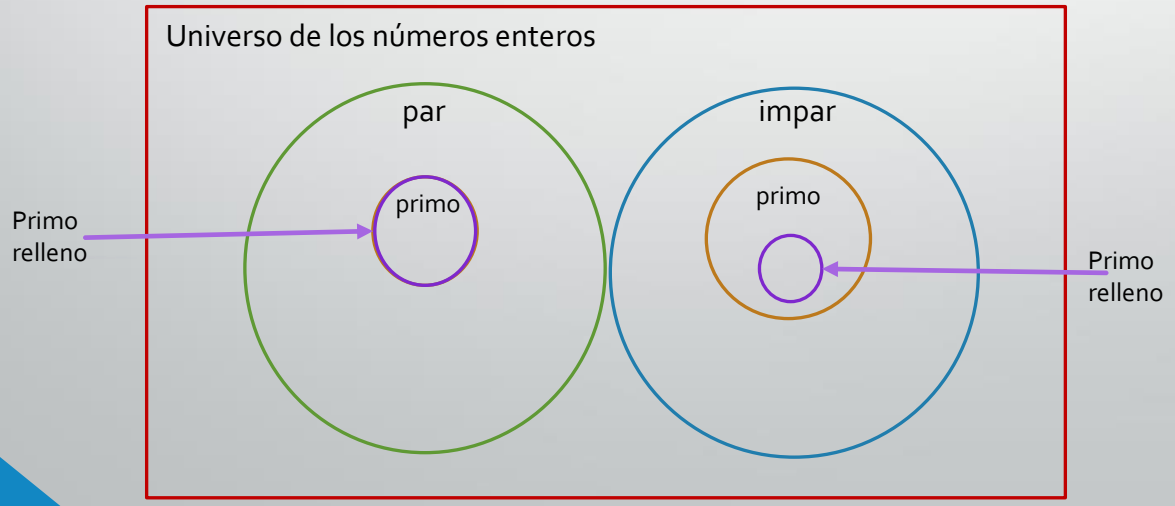


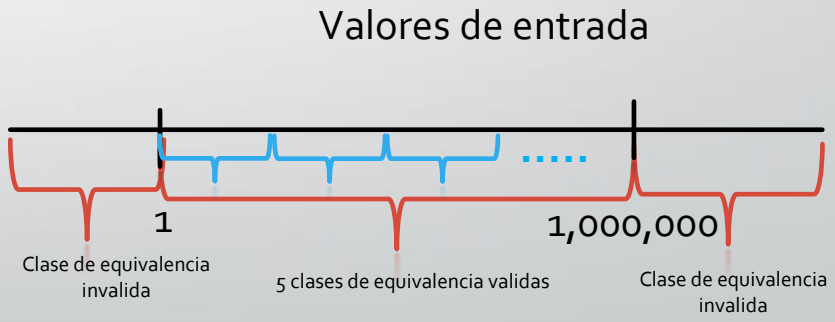


Ejercicio

- Supongamos que un estudiante de licenciatura realiza un programa para determinar si un número entero dado por el usuario es un número par o impar, además deberá indicar si el número también es primo, primo relleno o ninguno de los anteriores. El rango de valores que deberá aceptar el programa es de 1 a 1,000,000.

- **¿Cuántas clases de equivalencia existirían para este programa?**





Ejercicio

- Usando el software de ventas, el distribuidor de automóviles es capaz de definir reglas de descuentos para sus agentes de ventas:
  - Vehículos con un precio **menor a** 15,000 dls no tienen descuento.
  - Vehículos con un precio **mayor o igual a** 15,000 **y hasta** 20,000 dls tendrán un descuento del 5%.
  - Vehículos con un precio **mayor a** 20,000 **pero menor a** 25,000 tendrán un descuento de 7%.
  - Vehículos con **un precio de** 25,000 **o superior** tendrán un descuento del 8.5%.

Solución del ejercicio

- Existen 4 diferentes clases de equivalencia con entradas válidas (denominadas *valid equivalence classes, or vEC*, en la tabla)

Parameter	Equivalence classes	Representative
Sales price	vEC1: $0 \leq x < 15000$	14500
	vEC2: $15000 \leq x \leq 20000$	16500
	vEC3: $20000 < x < 25000$	24750
	vEC4: $x \geq 25000$	31800

- Los valores de entrada $14,500, $16,500, $24,750 y $31,800 son representativos de las 4 clases de equivalencia. Por lo tanto se asume que la ejecución de pruebas con otros valores tales como $13,400, $17,000, $22,300, y $28,900 no aportan más información y por lo tanto no permiten encontrar nuevas fallas.

Solución del ejercicio

- Las entradas incorrectas o invalidas también deben ser probadas. Estos valores de entrada forman parte de la clase de equivalencia de entradas incorrectas (valores inválidos).

Parameter	Equivalence classes	Representative
Sales price	iEC1: x < 0 negative, i.e., wrong sales price iEC2: x > 1000000 unrealistically high sales price[a]	-4000 1500800

invalid equivalence classes (iEC)

Conclusiones del ejercicio

- La partición de los datos de entrada en clases de equivalencia y la selección de los elementos representativos de cada clase debe ser realizada de forma cuidadosa.
- La probabilidad de identificar fallas depende en gran medida de las particiones realizadas y los casos de prueba ejecutados.
- Los mejores valores de prueba son aquellos que se encuentra en los límites de las clases de equivalencia.
  - Es común la presencia de inexactitudes al momento de redactar los requerimientos debido a la naturaleza de nuestro lenguaje. En ocasiones no establecer de manera precisa los límites de las clases de equivalencia.
  - **Ejemplo:** en la frase coloquial … menos de 15,000 … en un requerimiento puede significar que el valor 15,000 esta dentro (x <= 15,000) o fuera (x < 15,000)

Ventajas de utilizar partición de equivalencias

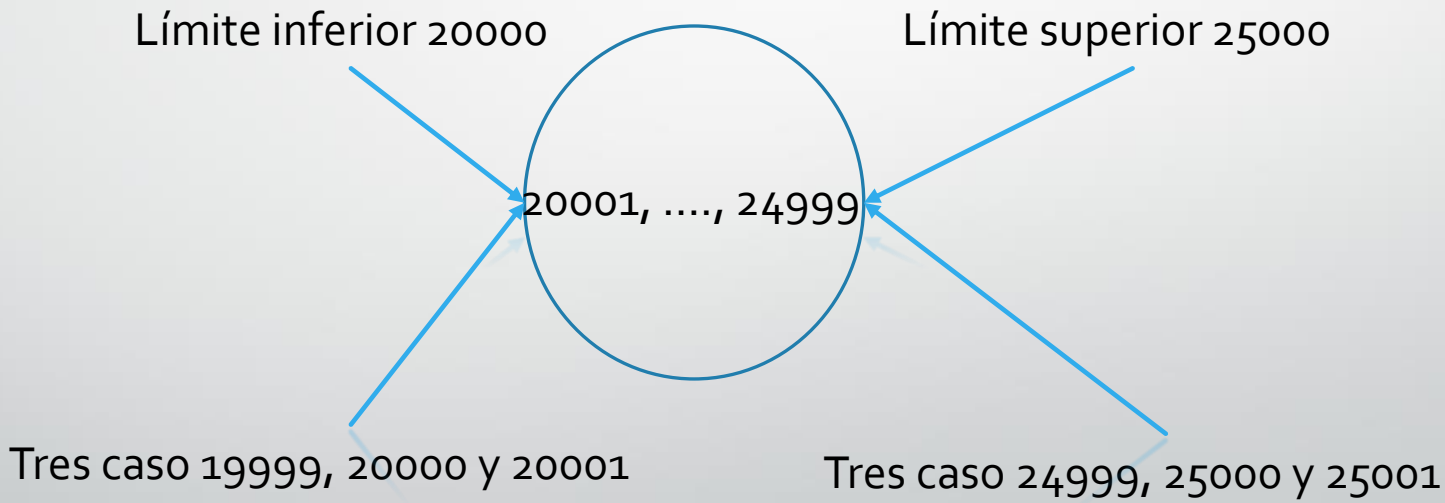
- La partición de equivalencias es una técnica sistemática que permite reducir el número de casos de prueba innecesarios.
  - Los casos de prueba innecesarios son aquellos que utilizan datos de la misma clase y por lo tanto el comportamiento (o resultado) del objeto de prueba es similar.
- Este método puede ser utilizado en cualquier nivel de pruebas.
  - Probar valores de entrada y salida de métodos y funciones.
  - Probar valores internos y estados.
  - Valores que dependen de eventos (por ejemplo antes o después de un evento).
  - Parámetros de una interfaz.
- Esta técnica puede ser muy poderosa/relevante al momento de combinarla con la técnica de análisis de valores límite.

Análisis del valor límite

- Las fallas a menudo aparecen en los límites de las clases de equivalencia.
- Esto sucede porque comúnmente los límites no son definidos claramente o son sobreentendidos.
- El objetivo de esta técnica es identificar los límites de las clases de equivalencia.
- En cada clase se debe identificar el valor exacto del limite tanto interno como externo para ser probados.
- Para valores de punto flotante deben ser definidos los rangos de tolerancia.
- Tres casos de prueba deben ser considerados para cada límite. Sin embargo, el valor límite superior de una clase puede ser el valor límite inferior de otra clase adyacente.

Análisis del valor límite

- Tres casos de prueba deben ser considerados para cada límite. Si el valor límite superior de una clase es el valor límite inferior de otra clase adyacente, entonces los casos de prueba respectivos también coinciden.



Ejemplo

- Supongamos que un código debe contener la instrucción: ***if(x<25000)***

- ¿Qué casos de prueba podrían encontrar un error en la implementación de esta estructura condicional?
  - 24999 y 25000.
  - 24999, 25000 y 25001.

Los valores de verdad para 24999, 25000 y 25001 son: verdadero, falso y falso. Aparentemente el 25001 no parece agregar un valor adicional, porque el 25000 ya ha generado el valor falso.

Ejemplo

- La siguiente tabla muestra las diferentes condiciones y los valores de verdad de los correspondientes valores limite.

Si en lugar de colocar < se coloca <> sólo el tercer valor podría mostrar el defecto.

Implemented condition	24999	25000
X < 25000 (correct)	True	False
X ≤ 25000	True	**True**
X <> 25000	True	False
X > 25000	**False**	False
X ≥ 25000	**False**	**True**
X == 25000	**False**	**True**

- Se debe establecer cuando una prueba con dos valores se considera suficiente y cuando es beneficioso probar el límite con tres valores.

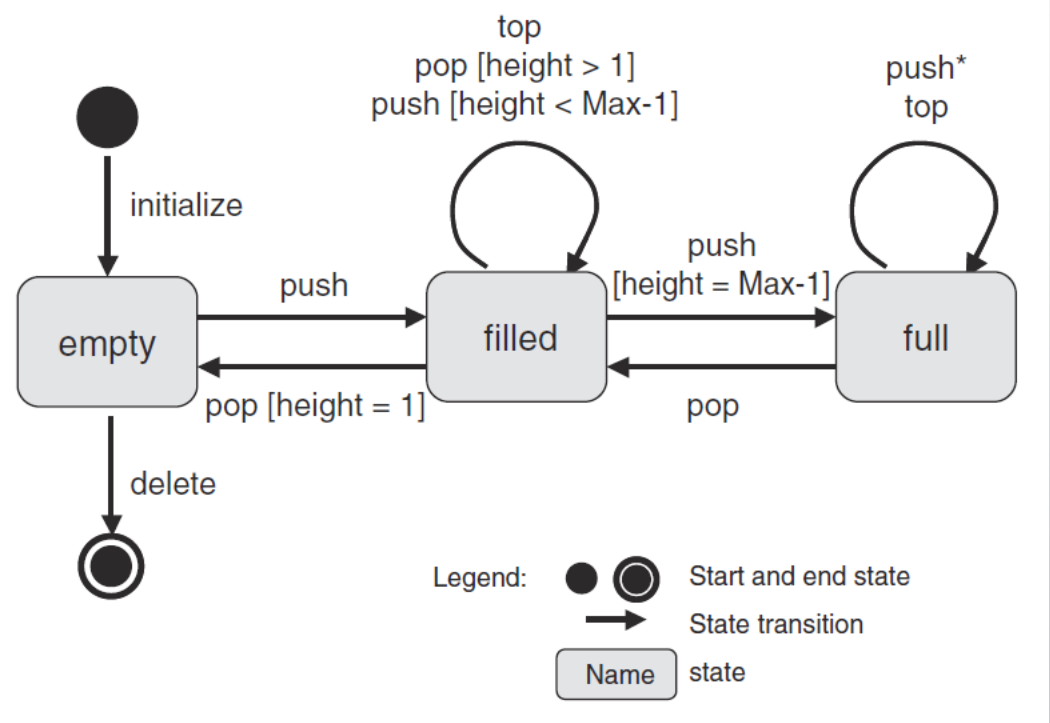
Pruebas de transición de estado

- En muchos sistemas, no solo las entradas actuales al sistema influyen en su comportamiento. Un sistema puede presentar diferentes comportamientos según su estado actual o eventos previos.

- El sistema u objeto de prueba comienza en un estado inicial y pueden generarse diferentes estados (o acciones) del objeto de prueba. Este aspecto del sistema puede ser modelado mediante:
  - Diagrama de transición de estados.
  - Tablas de transición de estado.
  - Máquinas de estado finito.

Pruebas de transición de estado

- El diagrama de transición de estados permite al tester visualizar los estados, transiciones, entradas de datos o eventos que las desencadenan y las acciones (comportamiento del software) que pueden resultar.



Pruebas de transición de estado

- Una tabla de transición de estados, muestra las relaciones entre los estados y las entradas de datos. Esto puede ayudar a identificar posibles transiciones invalidas.

Estado	Initialize	Delete	Push (height<Max-1)	Push (height==Max-1)	Pop (height>1)	Pop (height == 1)	Push	Pop
S_0	S_1							
S_1		S_4					S_2	
S_2			S_2	S_3	S_2	S_1		
S_3								S_2
S_4								

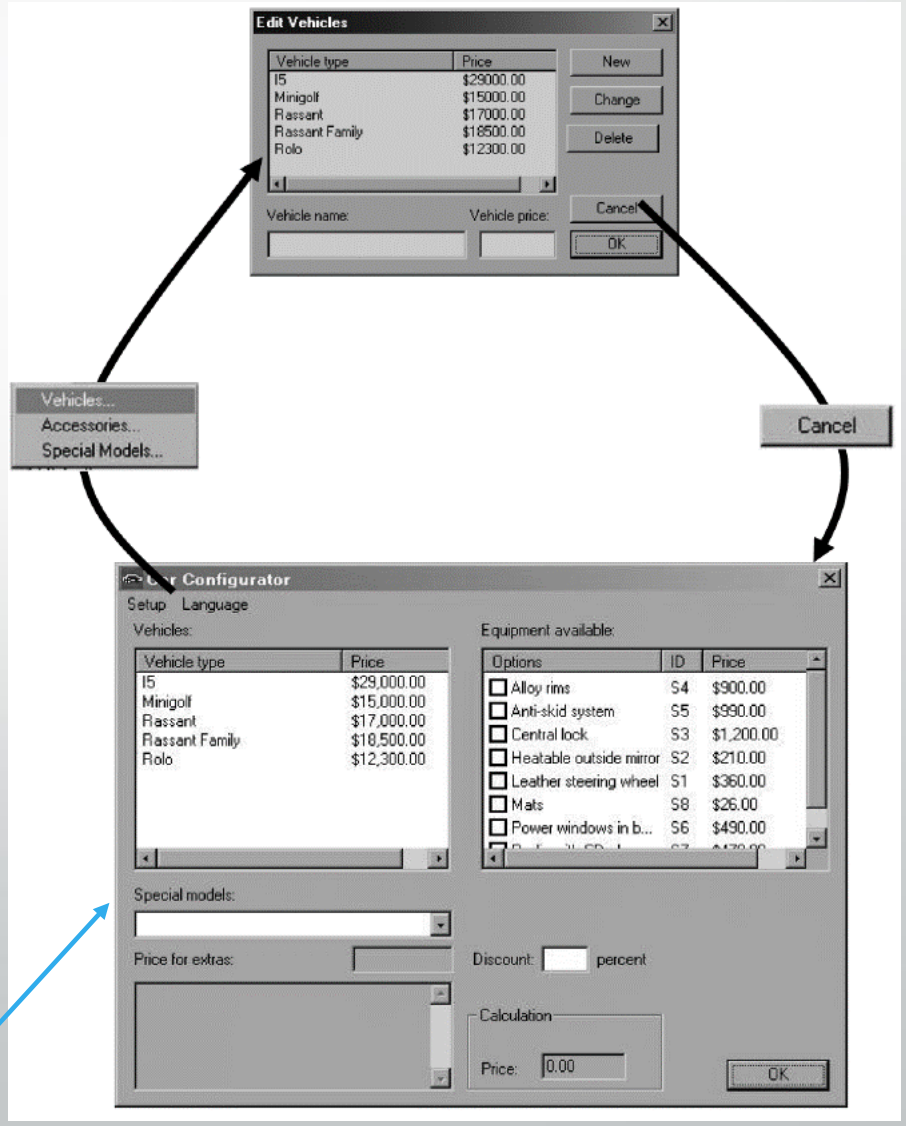
S_0 → estado inicial
S_4 → estado final

Pruebas de transición de estado

- **Ejemplo: Prueba de la interfaz gráfica del DreamCar.**

- Para crear casos de prueba se requiere de la siguiente información:
  - Estado inicial del objeto de prueba (componente o sistema)
  - Las entradas al objeto de prueba.
  - El resultado o comportamiento esperado.
  - El estado final esperado.

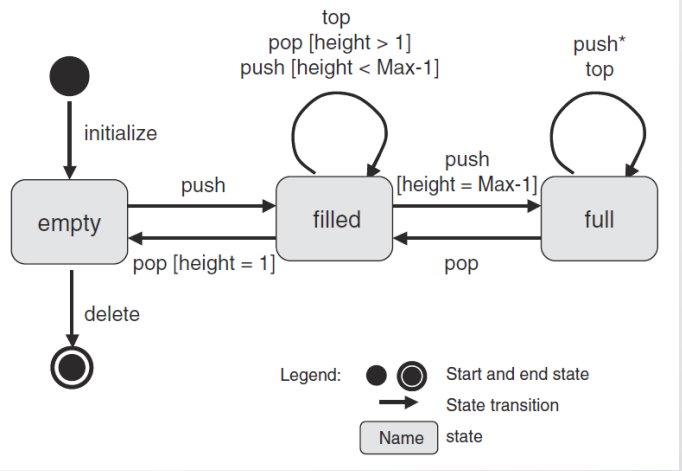
- Adicionalmente, para cada transición de estado, se deben definir los siguientes aspectos:
  - El estado previo a la transición.
  - El evento inicial que activa la transición.
  - El comportamiento esperado generado por la transición.
  - El siguiente estado esperado.



estado inicial y final (So)

Pruebas de transición de estado

- Definición de los criterios de salida de la prueba.
  - Cada estado debe ser alcanzado al menos una vez.
  - Cada transición debe ser ejecutada al menos una vez.
  - Verificar cada transición que infringe la especificación.



Estado	Initialize	Delete	Push (height<Max-1)	Push (height==Max-1)	Pop (height>1)	Pop (height == 1)	Push	Pop
S0	**S1**	invalid	invalid	invalid	invalid	invalid	invalid	invalid
S1	invalid	**S4**	invalid	invalid	invalid	invalid	**S2**	invalid
S2	invalid	invalid	**S2**	S3	**S2**	**S1**	invalid	invalid
S3	invalid	invalid	invalid	invalid	invalid	invalid	invalid	**S2**
S4	invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid

Pruebas basadas en técnicas de lógica

Grafo de causa-efecto. Es una técnica que usa las dependencias para identificar los casos de prueba.

La lógica relacionada entre las causas y sus efectos en un componente o sistema son presentadas en un grafo denominado grafo de causa-efecto.

- Cada causa es descrita como una condición que consiste de valores de entrada.
- Las condiciones son conectadas con operadores lógicos (AND, OR, NOT).
- La condición y su causa pueden ser TRUE o FALSE.

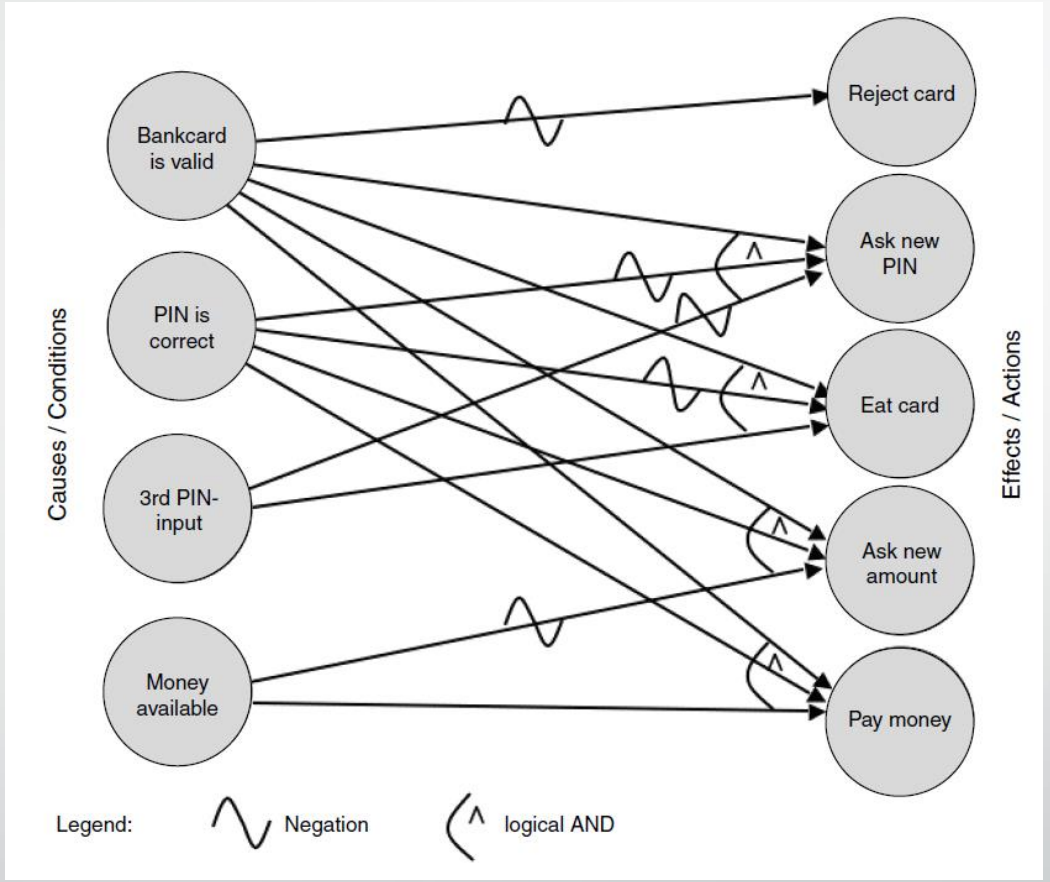
Pruebas basadas en técnicas de lógica

- El siguiente ejemplo considera la acción de retirar dinero de un cajero automático (ATM) para ejemplificar un grafo causa-efecto.

- Para retirar dinero de un ATM, se deben cumplir las siguientes condiciones:
  - La tarjeta de debito o crédito debe ser válida.
  - El PIN es correcto.
  - El número máximo de entradas del PIN es 3.
  - Existe dinero suficiente en el ATM y en la cuenta.

- Las siguientes acciones son posibles en el ATM:
  - Tarjeta rechazada.
  - Introducir nuevamente el PIN.
  - Tragarse (retener) la tarjeta.
  - Preguntar por otra cantidad (monto).
  - Entregar el monto solicitado.

Pruebas basadas en técnicas de lógica



Nota: No se presenta una descripción completa del ATM, sólo se modela una parte para ilustrar esta técnica.

Pruebas basadas en técnicas de lógica

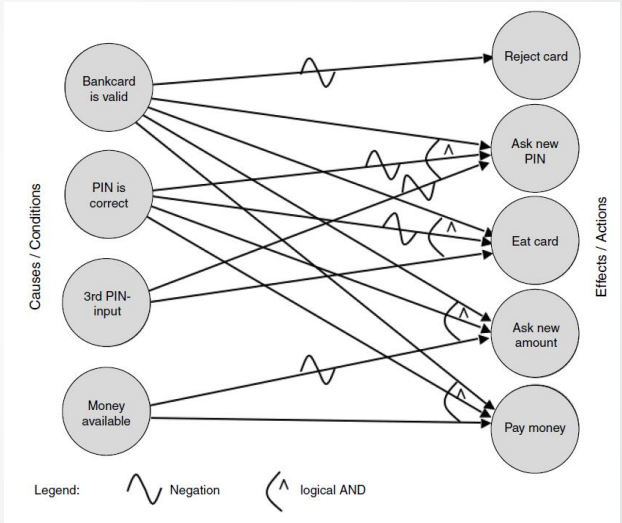
El grafo debe ser transformado en una **tabla de decisiones** a partir de la cual los casos de prueba pueden ser obtenidos.

Pasos para transformar un grafo a tabla:

- Seleccionar un efecto.
- Buscar en el grafo la combinación de causas que generan este efecto y las combinaciones que no generan este efecto.
- Agregar una columna en la tabla por cada uno de estas combinaciones de causa-efecto. Incluir los estados de causa de los efectos restantes.
- Verificar si las entradas de la tabla de decisiones ocurre varias veces y, si lo hacen, eliminarlas.

Pruebas basadas en técnicas de lógica

- Una tabla de decisiones esta forma por dos partes:
  1. En la mitad superior de la tabla se colocan las entradas (causas)
  2. En la mitad inferior de la tabla se colocan las salidas (efectos)



- Cada columna define la situación de la prueba.

- Dado que existen 4 condiciones en el grafo, en teoría se tienen 16 (2^4) posibles combinaciones. Sin embargo, no todas son tomadas en consideración. Por ejemplo, si la tarjeta es invalida, las demás condiciones son irrelevantes, porque el ATM debe rechazar la tarjeta.

- Una tabla de decisiones optimizada no contiene todas las posibles combinaciones.

Pruebas basadas en técnicas de lógica

- Cada columna de la tabla se interpreta como un caso de prueba.
- A partir de la tabla, es posible identificar las condiciones de entrada y las acciones esperadas.
- El caso de prueba 5 (TC5) muestra las siguientes condiciones: El dinero es entregado sólo si la tarjeta es valida, el PIN es correcto y existe dinero suficiente tanto en el ATM como en la cuenta.

Decision table		TC1	TC2	TC3	TC4	TC5
Conditions	Bank card valid?	N	Y	Y	Y	Y
	PIN correct?	-	N	N	Y	Y
	Third PIN attempt?	-	N	Y	-	-
	Money available?	-	-	-	N	Y
Actions	Reject card	Y	N	N	N	N
	Ask for new PIN	N	Y	N	N	N
	“Eat” card	N	N	Y	N	N
	Ask for new amount	N	N	N	Y	N
	Pay cash	N	N	N	N	Y

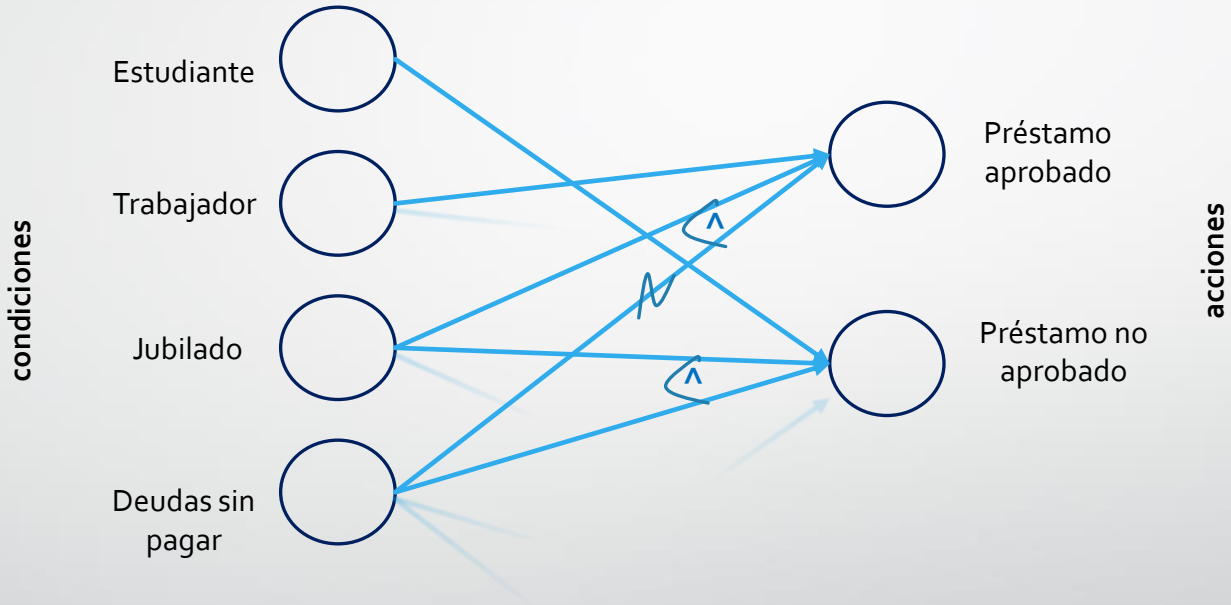
Pruebas basadas en técnicas de lógica

Ejercicio: Supongamos que tenemos un proceso bancario que se encarga de la aprobación de préstamos.

Reglas de negocio a considerar:

- Si la persona que solicita el préstamo es un **estudiante**, **el préstamo no será aprobado** aunque éste **no tenga deudas sin pagar.**
- Si la persona que solicita el préstamo es un **empleado**, **el préstamo será aprobado** aunque éste **sí tenga deudas sin pagar.**
- Si la persona que solicita el préstamo es un **jubilado**, **el préstamo será aprobado** si éste **no tiene deudas sin pagar.**
- Si la persona que solicita el préstamo es un **jubilado**, **el préstamo no será aprobado** si éste **tiene deudas sin pagar.**

Pruebas basadas en técnicas de lógica



Pruebas basadas en técnicas de lógica

Condiciones

Tabla de decisiones	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}
Estudiante	V	V	V	V	V	V	V	V	F	F	F	F	F	F	F	F
Trabajador	V	V	V	V	F	F	F	F	V	V	V	V	F	F	F	F
Jubilado	V	V	F	F	V	V	F	F	V	V	F	F	V	V	F	F
Deudas sin paga	V	F	V	F	V	F	V	F	V	F	V	F	V	F	V	F

- Posibles combinaciones = 2^4
- C_n → Combinación *n*.
- No todas las combinaciones son necesarias por lo que debemos optimizar la tabla.

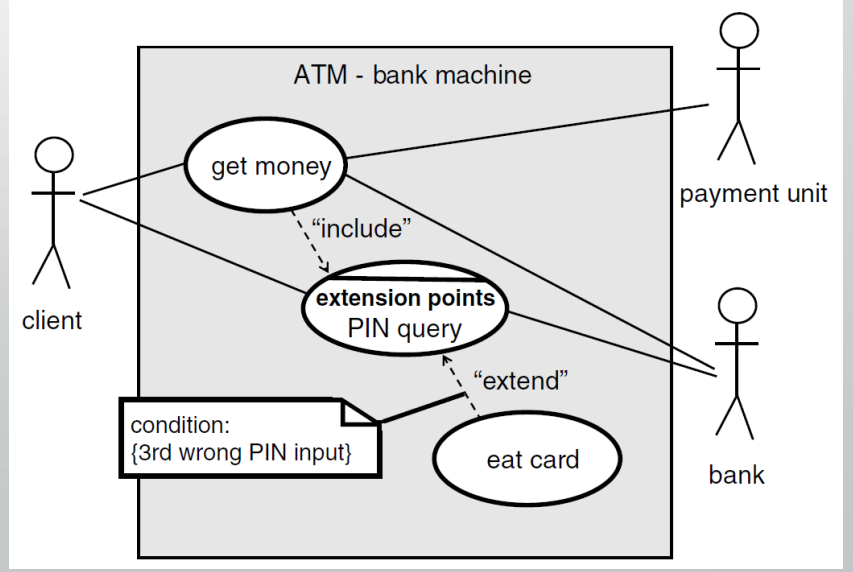
Pruebas basadas en técnicas de lógica

Casos de prueba → TC_1 TC_2 TC_3 TC_4 TC_5 TC_6

Tabla de decisiones		R_7	R_8	R_{11}	R_{12}	R_{13}	R_{14}
Condiciones	Estudiante	V	V	F	F	F	F
	Trabajador	F	F	V	V	F	F
	Jubilado	F	F	F	F	V	V
	Deudas sin paga	V	F	V	F	V	F
Acciones	Préstamo aprobado	No	No	Si	Si	No	Si
	Préstamo no aprobado	Si	Si	No	No	Si	No

Pruebas basadas en casos de uso

- Los casos de uso modelados comúnmente con UML describen las interacciones entre actores (que pueden ser usuarios o sistemas)
- A partir de los casos de uso se pueden derivar casos de prueba.
- Los casos de uso son útiles para definir las pruebas de aceptación, en las que puede participar el cliente.

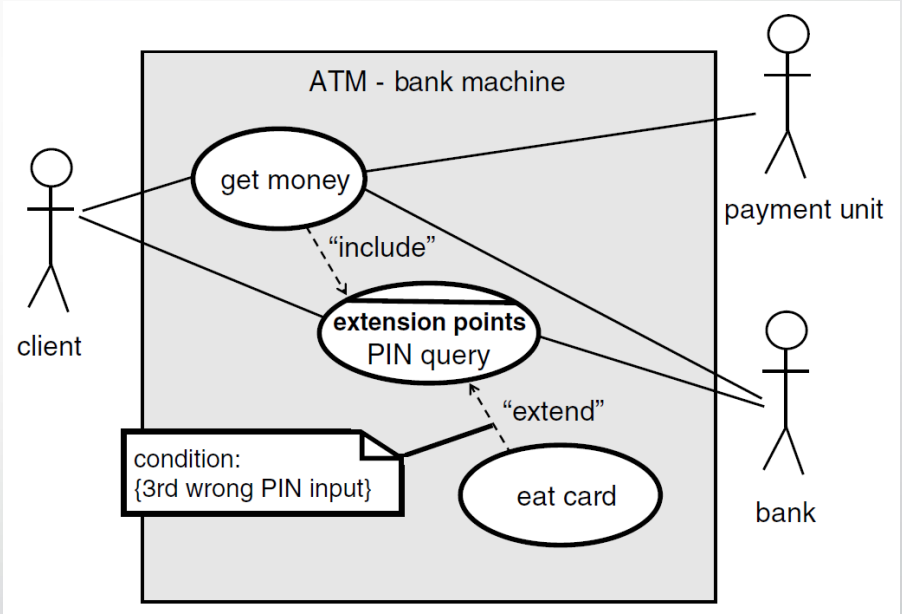


Pruebas basadas en casos de uso

- Para cada caso de uso, existen ciertas **precondiciones** que deben ser cubiertas para permitir la ejecución de dicho caso de uso. Una precondición para retirar dinero del ATM, es por ejemplo, que la tarjeta del banco sea válida.

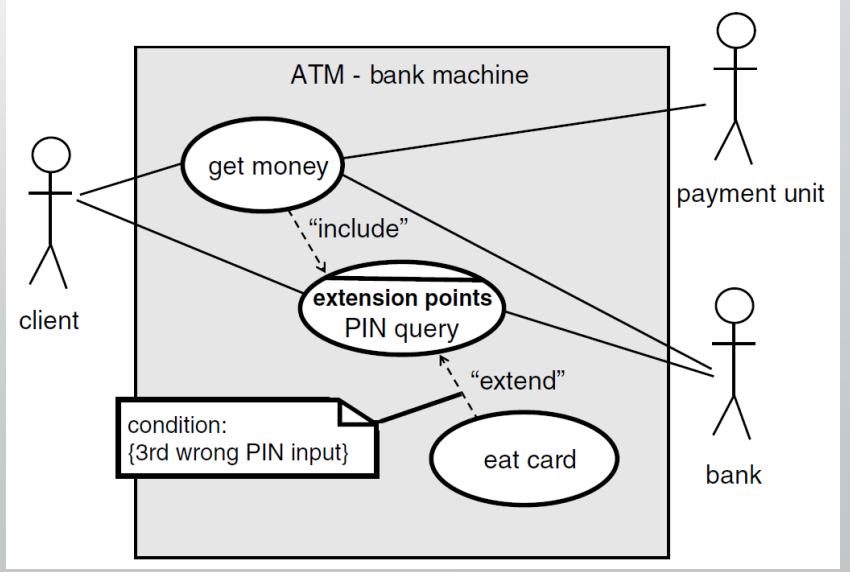
- Después de la ejecución de un caso de uso, existen postcondiciones. Por ejemplo, después de introducir correctamente el PIN, es posible retirar el dinero. Sin embargo, primero se debe ingresar el monto a retirar y éste debe estar disponible.

- Las precondiciones y postcondiciones son aplicables para seguir el flujo de los casos de uso de un diagrama.



Pruebas basadas en casos de uso

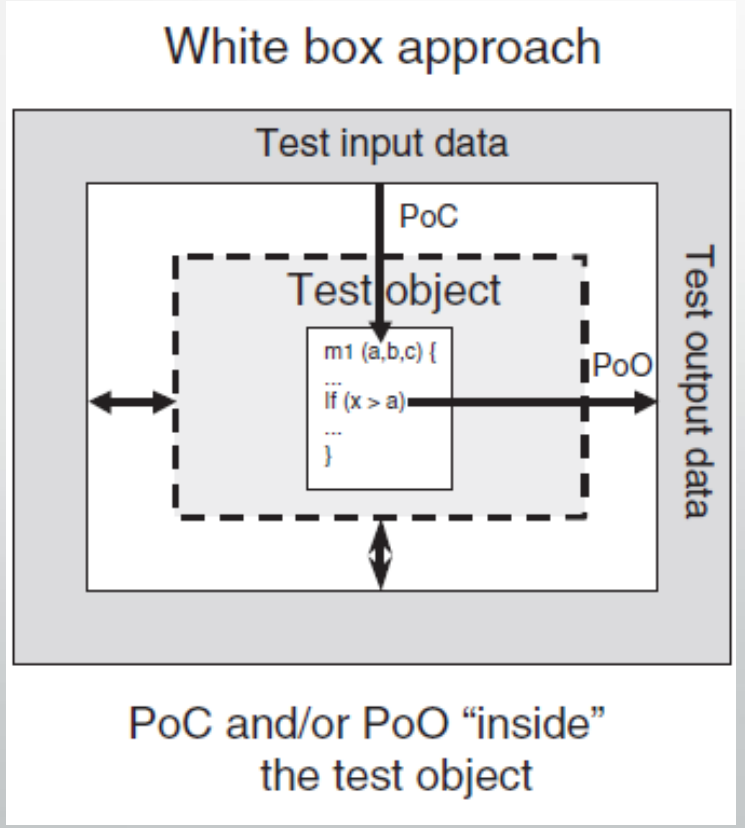
- Los siguientes datos son necesarios para la creación de casos de prueba a partir de diagramas de casos de uso.
  - Identificar las precondiciones.
  - Identificar condiciones adicionales.
  - Identificar el resultado esperado del caso de uso.
  - Identificar las postcondiciones.



Prueba de caja blanca

- **En las pruebas de caja blanca, el código del programa es usado para el diseño de la prueba.**

- Las pruebas de caja blanca también son denominadas pruebas estructurales porque consideran la estructura del objeto de prueba.



Técnicas de prueba de caja blanca

- Durante la ejecución de las pruebas el flujo interno del objeto de prueba es analizado.
- En situaciones especiales, es posible realizar una intervención directa en el flujo de ejecución del objeto de prueba, por ejemplo, para ejecutar pruebas negativas o cuando la interfaz del componente no es capaz de inicializar la falla para probar un caso negativo.
- **Los casos de prueba son diseñados observando el código fuente del objeto de prueba.**
- El principal objetivo de utilizar pruebas de caja blanca es para alcanzar una cobertura especificada.
  - Por ejemplo, el 80% de todas las sentencias del objeto de pruebas deben ser ejecutadas al menos una vez.
  - Casos de pruebas adicionales pueden ser sistemáticamente derivados con el propósito de incrementar el grado de cobertura.
- Las pruebas de caja blanca comúnmente son pruebas de bajo nivel tales como pruebas unitarias y de integración.