Unidad 7 - Conociendo Las Técnicas De Prueba

Técnicas de prueba.

Razones fundamentales de porque estudiar las técnicas de prueba de software:

- Aseguramiento de la calidad: Las pruebas permiten identificar y corregir defectos antes de que lleguen a los usuarios finales, garantizando un producto de alta calidad.
- Cumplimiento de Requisitos: Las pruebas son la herramienta principal para verificar que el software cumple con los requisitos especificados. Al aplicar técnicas de pruebas adecuadas, se puede garantizar que el software funcione de acuerdo con las expectativas del cliente y que se ajuste a los estándares y regulaciones pertinentes.
- Optimización de Recursos: El conocimiento de diversas técnicas de pruebas permite a los profesionales del testing seleccionar la estrategia de pruebas más eficiente y efectiva para cada proyecto. ahorra tiempo y recursos al enfocarse en áreas críticas del software y minimizar pruebas innecesarias.
- Detección Temprana de Defectos: Al aplicar técnicas de pruebas durante todo el ciclo de vida del desarrollo del software, es posible detectar y solucionar problemas en las etapas iniciales del proceso. Esto reduce significativamente los costos y los esfuerzos necesarios para corregir defectos en etapas posteriores del proyecto.
- Mejora Continua: El aprendizaje y la aplicación de diversas técnicas de pruebas permiten a los profesionales del testing mejorar constantemente sus habilidades y enfoques.

Esta unidad les proporcionará las herramientas necesarias para seleccionar las técnicas de pruebas más adecuadas según el contexto y los objetivos del proyecto de software. Además, aprenderán a planificar y ejecutar las pruebas de manera eficiente, maximizando la detección de defectos y minimizando el tiempo y los recursos invertidos.

Análisis estático: disciplina esencial que nos permite examinar el código fuente y detectar posibles errores y vulnerabilidades sin necesidad de ejecutar el programa.

¿Qué es el Análisis Estático? Proceso de examinar el código fuente de un programa en busca de posibles errores, vulnerabilidades, mala calidad de código y problemas de estilo, sin necesidad de ejecutar el programa. Se logra mediante el uso de herramientas automatizadas que inspeccionan el código y aplican reglas y

estándares predefinidos.

Importancia del Análisis Estático

1. Detección temprana de errores: Permite encontrar y corregir errores antes de que el software se ejecute, lo que reduce costos y problemas posteriores.
2. Mejora de la calidad del código: Identifica problemas de estilo, redundancias y malas prácticas que pueden afectar la legibilidad y el mantenimiento del código.
3. Seguridad: Ayuda a identificar vulnerabilidades de seguridad antes de que sean explotadas por atacantes.
4. Cumplimiento de estándares: Garantiza que el código cumpla con estándares y convenciones definidos por el equipo o la industria.

Video 01

Las técnicas de pruebas se dividen en dos:

Caja blanca : Son utilizadas por el equipo de desarrollador. Uno de los objetivos que tienes es la validación del flujo de la prueba, es decir validar que líneas de código son ejecutadas cuando nuestra prueba está en proceso. EJ= Un sistema que recibe datos de entrada y luego de salida el objetivo es ver que camino tomar nuestra prueba desde el inicio hasta que sale.
Caja Negra: Son utilizadas por el equipo de QA. El objetivo de validar es el resultado de las pruebas, será transparente porque parte del código viaja en las prueba, lo que importa aquí es validar que el resultado de las prueba sea el esperado.

Video 02

Categorías de técnicas de prueba:

- **Propósito:** Ayudar a identificar las condiciones de prueba los casos de prueba y los datos de prueba.

La elección de las técnicas de prueba a utilizar depende de varios factores, que incluyen:

- Complejidad de componentes o sistemas
- Estándares regulatorios
- Requisitos contractuales o del cliente
- Niveles y tipos de riesgo
- Documentación disponible
- Conocimientos y habilidades del tester
- Herramientas disponibles
- Tiempo y presupuesto
- Modelo de ciclo de vida de desarrollo de software
- Los tipos de defectos esperados en el componente o sistema.

Algunas técnicas son más aplicables a determinadas situaciones y niveles de prueba; otros son aplicables a todos los niveles de prueba.

Al crear casos de prueba, los tester generalmente usan una combinación de técnicas de prueba para lograr los mejores resultados del esfuerzo de prueba.

El uso de técnicas de prueba en el análisis de prueba, el diseño de prueba y las actividades de implementación de prueba puede variar desde muy informal (poca o ninguna documentación) hasta muy formal.

El nivel apropiado de formalidad depende del contexto de las pruebas, incluida la madurez de los procesos de prueba y desarrollo, las limitaciones de tiempo, los requisitos de seguridad o reglamentarios, el conocimiento y las habilidades de las personas involucradas y el modelo de ciclo de vida de desarrollo de software que se sigue.

Video 03

Técnicas de diseño de pruebas:

Particiones Equivalentes (Equivalence Partitioning)

- Es una técnica de prueba basada en especificación.
- Puede ser aplicado a cualquier nivel de pruebas.
- Se basa en dividir condiciones de prueba en grupos equivalentes o iguales.
- También se conoce como Clases Equivalentes (Equivalent Classes)
- Se prueba una condición de cada partición o clase.

- Especificación: Es decir, requerimientos del cliente.
- niveles de prueba: unitarias, integración, sistemas y aceptación.
- Todos los escenarios que estén en un grupo deberían de dar el mismo resultado de prueba.
- Si en cada clase tengo 10 casos de prueba, no es necesario que se prueben los 10 si todos van a dar el mismo resultado.

En resumen: Es tratar de ser más eficiente de con menos pruebas cubrir más escenarios, en lugar de repetir 20 casos de un mismo escenario.

Análisis de valor frontera (Boundary value analysis)

- Se basa en probar los límites de las particiones.
- Tiene límites válidos e inválidos.
- Complementa la técnica de particiones equivalentes.

- Un estudiante puede tener una calificación entre 0 y 100.
  - Para hacer análisis de valor límite usamos el min y max valor posible.
  - Adicional el valor anterior y siguiente a cada límite.

EJ =Se realizan la prueba de los numeros lateras e interiores de los limites.

	Mínimo			Máximo	
-1	0	1	99	100	101

Límite abierto (Open Boundary)

- Es cuando un límite no tiene un valor máximo o mínimo o no ha sido definido.
- Son más difíciles de probar.
- Un enfoque puede ser investigar otras áreas del sistema con que se relaciona.

Decision Table Testing (Pruebas de tablas de decision)

- Es una técnica de prueba basada en especificación.
- Está más enfocada en lógica y reglas de negocio.
- Es una Buena forma de lidiar con combinaciones.
- Proporcionar una forma sistemática de establecer reglas de negocio complejas

Como usar tablas de decision?

- Identificar un sistema o subsistema que reaccione a una combinación de entradas.
- El Sistema no debe contener demasiadas combinaciones de lo contraro seria inmanejable.
- Colocar los aspectos a combinar en una tabla.

- Diferentes variables de entradas el resultado va a ir variando.
- Seria muy dificil de manejar y diseñar

Como usar tablas de decision?

- Una aplicación de préstamos en la que se recibe como entrada el monto a pagar mensual y la cantidad de años del préstamo. Las dos condiciones serían el pagaré y la finalización del préstamo.

Como usar tablas de decision?

- Identificamos las combinaciones de V y F.
  - Con 2 condiciones, que ambas pueden ser verdaderas o falsas tendríamos 4 combinaciones. (2 ^ 2).

Condiciones	Regla 1	Regla 2	Regla 3	Regla 4
Monto a pagar	V	V	F	F
Finalización del préstamo	V	F	V	F

- Lo siguiente es identificar las salidas.Podemos introducir uno o ambos campos. A cada combinación se le conoce como una regla.

Condiciones	Regla 1	Regla 2	Regla 3	Regla 4
Monto a pagar	V	V	F	F
Finalización del préstamo	V	F	V	F
Acciones/Salidas				
Procesar monto	V	V		
Procesar término	V		V	

Puntos claves: Tablas de decisión

- Esta técnica descubre omisiones o ambiguedades en las especificaciones.
- Usualmente algunas combinaciones son omitidas en las especificaciones.
- El paso final de esta técnica es escribir casos de prueba contra cada una de las reglas de la tabla.

Segundo ejemplo:

Ejercicio: Tablas de decisión

- Si eres un nuevo cliente y quieres solicitar una tarjeta de crédito, obtienes un 15% de descuento en tus compras de hoy. Si eres un cliente existente y tienes una tarjeta de lealtad obtienes un 10% de descuento. Si tienes un cupon obtienes 20% de descuento (pero no puede ser usado con el nuevo cliente).

- Construir la tabla de decision para este caso.

Condiciones	Regla1	Regla2	Regla3	Regla4	Regla5	Regla6	Regla7	Regla8
Nuevo Cliente (15%)	V	V	V	V	F	F	F	F
Tarjeta Lealtad (10%)	V	V	F	F	V	V	F	F
Cupón (20%)	V	F	V	F	V	F	V	F
Resultado								
% Descuento	X	X	20	15	30	10	20	0

Transition testing (Pruebas de transición de estados)

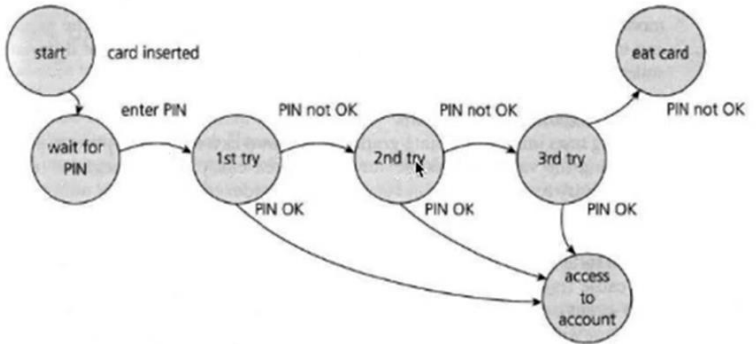
- Se aplica para sistemas que tienen un número finito de diferentes estados.
- Cualquier Sistema en que obtienes resultados diferentes con las mismas entradas dependiendo de que pasó antes es un Sistema de estados finito.
- Puede ser representado en un diagram de estados.
- El modelo puede ser tan abstracto o tan detallado como necesites.

Partes básicas de un modelo de transición de estados.

- Los estados en que puede estar el software (Abierto/cerrado o con fondos/sin fondos, etc)
- Transición de un estado a otro (No todas las transiciones son permitidas)
- Los eventos que causan las transiciones (cerrar un archivo, retirar o depositar dinero, etc).
- Las acciones que resultan de una transición (Mensaje de error o completar un proceso).

Ejemplo: Pruebas de transición de estados

- Introducir el PIN de una cuenta de banco.



Ejemplo: Pruebas de transición de estados

- Realizando los casos de prueba del ejemplo anterior:
  - Cada estado representa una condición de prueba.
  - El diagrama nos da información de como diseñar casos útiles.
  - Se puede medir la cobertura en base a las transiciones.
  - Pruebas de transición de estados es una técnica de diseño de pruebas basado en especificación.

Use Case Testing (Pruebas de casos de uso)

- Es una técnica que nos ayuda a identificar casos que cubren el Sistema de inicio a fin.
- Un caso de uso es una descripción de un uso particular del Sistema por un actor.
- Los actores son generalmente personas (usuarios) pero pueden también ser sistemas.
- Los casos de uso se definen en términos del autor, de lo que puede y no puede hacer el mismo y no en términos de Sistema.

Use Case Testing (Pruebas de casos de uso)

- Generalmente usan un lenguaje de negocio y no técnico.
- Sirven de base para creación de casos de prueba de aceptación.
- Cada caso debe especificar cualquier pre-condición que se deba cumplir para ejecutar dicho caso.
- Cada caso debe especificar post-condiciones o resultados y descripciones de estados finales luego de ser ejecutado.

Ejemplo: Use Case Testing (Pruebas de casos de uso)

	Step	Description
Main Success Scenario **A: Actor** **S: System**	1	A: Inserts card
	2	S: Validates card and asks for PIN
	3	A: Enters PIN
	4	S: Validates PIN
	5	S: Allows access to account
Extensions	2a	Card not valid S: Display message and reject card
	4a	PIN not valid S: Display message and ask for re-try (twice)
	4b	PIN invalid 3 times S: Eat card and exit

Experience based techniques (Técnicas basadas en experiencia)

- El conocimiento y experiencia de las personas es de suma importancia para aplicar esta técnica.
- La experiencia de personas técnicas y de negocio es requerida.
- Pruebas basadas en la experiencia son pruebas basadas en la estructura del Sistema.
- Este tipo de prueba se da más para sistemas de bajo riesgo.

Error guessing (Predicción de errores)

- Son pruebas realizadas por los Testers.
- El éxito de esta técnica va a depender de las habilidades del tester.
- Generalmente se utiliza al final (o luego de aplicar técnicas más formales) para aprovechar el conocimiento adquirido en las pruebas.
- Pruebas típicas incluyen: División entre 0, tipos de datos incorrectos, campos vacíos.
- Un enfoque estructurado es hacer una lista de posibles defectos y luego escribir casos que apunten a encontrar dichos defectos.

El conocimiento de estas técnicas es esencial ya que son la base sobre la cual se construye la estrategia de aseguramiento de la calidad. Brindan herramientas para identificar defectos, evaluar la funcionalidad y la seguridad del software, y garantizar que cumpla con los requisitos y estándares necesarios. Razones fundamentales por las cuales estos temas son cruciales:

- **Mejora de la calidad del software**: Las técnicas de pruebas permiten identificar y corregir defectos y errores en el software antes de su implementación o lanzamiento. Esto contribuye significativamente a mejorar la calidad y confiabilidad del software, evitando posibles problemas y fallos en el futuro.
- **Cumplimiento de requisitos y estándares:** Las técnicas de pruebas son esenciales para verificar que el software cumple con los requisitos específicos del cliente o del proyecto, así como con los estándares de calidad establecidos en la industria.
- **Optimización de recursos:** Al aplicar técnicas de pruebas de manera efectiva, se pueden identificar y resolver problemas de manera temprana, lo que ahorra tiempo y recursos en comparación con la detección de problemas en etapas más avanzadas del desarrollo.
- **Aseguramiento de la seguridad:** En el contexto actual de la ciberseguridad, las técnicas de pruebas de seguridad son esenciales para identificar vulnerabilidades y brechas de seguridad en el software. Esto es crítico para proteger los datos y la privacidad de los usuarios.
- **Mejora de la experiencia del usuario:** Las pruebas de usabilidad y pruebas de aceptación del usuario son técnicas que permiten evaluar la experiencia del usuario final. Esto asegura que el software sea intuitivo, eficiente y cumpla con las expectativas de los usuarios.
- **Reducción de costos a largo plazo**: Detectar y solucionar problemas de software en etapas tempranas del desarrollo es más económico que hacerlo después de que el software está en producción. Las técnicas de pruebas ayudan a evitar costosos retrabajos y reparaciones posteriores.

En resumen, el estudio de las técnicas de pruebas de software es esencial para garantizar que el software cumpla con los estándares de calidad, sea seguro, eficiente y satisfaga las necesidades de los usuarios. Contribuye en gran medida a la creación de software de alta calidad y al éxito en el desarrollo de proyectos de software en diversos sectores de la industria.

Las Compuertas Lógicas, tanto las básicas como las avanzadas. Las Compuertas Lógicas Básicas, como AND, OR y NOT, serán exploradas en detalle, comprendiendo su funcionamiento, tablas de verdad y su aplicación en la realización de operaciones lógicas básicas. Además, se analizarán las Compuertas Lógicas Avanzadas, como NAND, NOR y XOR, las cuales brindarán funcionalidades adicionales y permitirán implementar operaciones lógicas más complejas.

Técnicas de caja negra: Esta técnica se utiliza para evaluar la funcionalidad de un

software sin necesidad de conocer su estructura interna.

¿Qué es la Técnica de Caja Negra? También conocida como prueba funcional, se basa en la idea de que un evaluador debe examinar un software desde el exterior, sin acceso a su código fuente ni conocimiento detallado de su diseño. En otras palabras, el probador trata el software como una "caja negra" en la que solo se conocen las entradas y las salidas, pero no los procesos internos. Esto simula la forma en que los usuarios interactúan con la aplicación, ya que rara vez comprenden su estructura interna.

Metodología de Prueba de Caja Negra

Las pruebas de caja negra se centran en verificar que el software cumple con las especificaciones y los requisitos funcionales. Los pasos clave en la metodología de prueba de caja negra incluyen:

- **Identificación de Requisitos:** El primer paso implica comprender los requisitos funcionales del software. Estas son las funciones y comportamientos que el software debe ofrecer.
- **Diseño de Casos de Prueba:** Con base en los requisitos, se crean casos de prueba que incluyen entradas, acciones del usuario y resultados esperados. Estos casos de prueba se diseñan para cubrir todas las posibles combinaciones de entrada y situaciones de uso.
- **Ejecución de Pruebas:** Los casos de prueba se ejecutan en el software, utilizando diversas combinaciones de entrada. El probador observa las salidas y compara los resultados con los esperados.
- **Registro de Resultados:** Se documentan los resultados de las pruebas, incluyendo cualquier discrepancia entre las salidas observadas y las esperadas.
- **Informe de Defectos:** Si se encuentra alguna discrepancia o defecto, se documenta y se informa al equipo de desarrollo para su corrección.

Importancia de la Técnica de Caja Negra

- **Simula la Experiencia del Usuario:** Al evaluar el software desde una perspectiva de caja negra, se simula la experiencia real del usuario, lo que garantiza que el software funcione como se espera en situaciones del mundo real.
- **Enfoque en Requisitos:** Se centra en la verificación de que el software cumple con los requisitos funcionales establecidos, lo que garantiza que la aplicación haga lo que se supone que debe hacer.
- **Independencia del Código Fuente:** No requiere acceso al código fuente, lo que permite que los probadores y evaluadores trabajen de manera independiente al equipo de desarrollo.
- **Identificación Temprana de Problemas:** Puede ayudar a identificar problemas en las primeras etapas del desarrollo, lo que reduce los costos y los esfuerzos de corrección.

En resumen, las técnicas de caja negra desempeñan un papel fundamental en la evaluación y el aseguramiento de la calidad del software. Al enfocarse en la

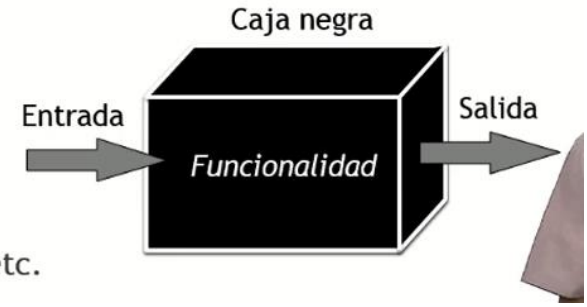
funcionalidad y la experiencia del usuario, estas pruebas garantizan que el software cumpla con sus objetivos sin necesidad de profundizar en su estructura interna.

VIDEO 01

Pruebas de caja negra

Las pruebas de **caja negra** se basan en la comprobación de la **funcionalidad esperada** en base a su interfaz (**parámetros** de **E/S**) del componente software

- ☐ probar la **funcionalidad** del módulo, **sin disponer** del código fuente
- ☐ intentar encontrar errores de **estructuras** de datos, **inicialización** y **finalización**, **rendimiento**, interfaces de **E/S**, etc.



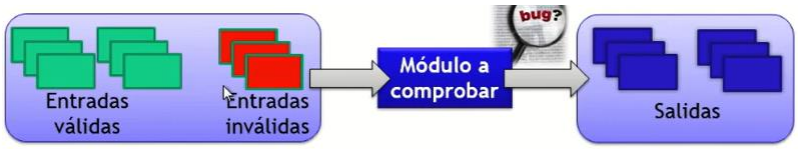
Técnica de partición equivalente:

Consiste en derivar **casos de prueba** mediante la división del dominio de entrada en clases de equivalencia, evaluando su comportamiento para un valor cualquiera representativo de dicha clase

"Una clase de equivalencia representa al conjunto de estados válidos o inválidos para todas las condiciones de entrada que satisfagan dicha clase"

Suponemos *razonablemente* que el **resultado** de éxito/fallo será el **mismo** para cualquier valor de esa clase

La idea de esto es tratar de establecer una serie de clases y extraer de estas una serie de valores que sean suficientemente representativos, si analizamos estos valores, el resultado sera el mismo para cualquier valor de esa clase



Hay que tener claro que no estamos realizando la prueba con todos los valores, ya que sería muy caro de calcular

Condiciones de entrada (restricciones de contenido o formato) de datos válidos y de datos inválidos

Reduciendo el conjunto de casos de prueba a uno más pequeño (**arriesgado**)

Es decir, se busca descubrir clases de errores **para cada clase** de equivalencia **identificada**

Ejemplos:

Heurísticas de identificación de clases de equivalencia

Tipo de entrada	Nº clases válidas	Nº clases inválidas
Rango de valores Ej. [20..30]	1: valor en rango (25)	2: valor por debajo y otro por encima del rango (15 y 40)
Conjunto finito de valores Ej. {2,4,6,8}	1: valor en el conjunto (4)	2: valor fuera del conjunto, por debajo y por encima (1 y 10)
Condición booleana (T/F) Ej. "debe ser una letra"	1: valor evaluado a cierto ('J')	1: valor evaluado a falso ('#')
Conjunto de valores admitidos Ej. {opción1, opción2, opción3}	3 en este ejemplo: tantos como valores admitidos (opción1, opción2, opción3)	1: valor no admitido (opción4)
Clases menores Ej. "2 rangos: 0<valor<5; 10<=valor<20"	*Se divide en distintas subclases que se manejan exactamente igual que en los puntos de arriba (una para "0<valor<5" y otra para "10<=valor<20")*	

Una vez que identificamos las clases, construimos los casos de prueba:

Diseño de casos de prueba Entrada → Proceso → Salida

Entrada: condición/parámetro con su **tipo de entrada**

Proceso:
1. **Para cada condición** de entrada, identificar **heurísticamente** sus **clases de equivalencia**
2. **Etiquetar** unívocamente cada clase de equivalencia
3. **Mientras** existan clases de equivalencia válidas **por cubrir**, diseñar un caso de prueba que cubra **tantas clases** válidas pendientes como sea posible
4. **Repetir** el paso 3 para las clases de equivalencia **inválidas**, cubriendo **una y solo** una clase inválida cada vez

Salida: conjunto de **casos de prueba**, **cubriendo todas** las clases de equivalencia (válidas e inválidas)

Si queremos extender los casos de prueba que se generaron con esta técnica podemos utilizar otra técnica:

Análisis de los Valores Límite (AVL)

Complemento a la técnica de partición equivalente que busca explotar los **errores que se pueden producir** en los **extremos tanto de entrada como de salida** (ej. límites de rangos, estructuras, bucles, clases equivalentes, etc.)

En lugar de diseñar una clase válida con un **elemento representativo** de la partición equivalente, se **escogen** los **valores en** los **límites de la clase**

Tipo de entrada	N° clases válidas	N° clases inválidas
Rango de valores Ej. [20..30]	**4**: valor en los **límites del rango** (20, 21, 29 y 30)	2: valor **justo** por debajo y **justo** por encima del rango (19 y 31)
Conjunto finito de valores Ej. {2,4,6,8}	**4**: valores **mínimo** y **máximo** en el conjunto (2, 4, 6 y 8)	2: valor fuera del conjunto, **justo** por debajo y por encima (1 y 9)

Esta técnica hace hincapié de los valores que están en los límites, en los límites operacionales. Las clases validas e invalidas se calculan muy similar a lo que vimos.

Conclusión

Las pruebas de **caja negra** se diseñan en base a las **entradas/salidas** de los componentes a analizar

Obtener la **totalidad de casos** de prueba es muy **costoso** o imposible

La técnica de **partición equivalente** ofrece un método **razonablemente representativo** para cubrir todas las clases válidas e inválidas

Se complementa con **AVL**, que genera casos de prueba también para los valores de **entrada/salida** en sus **valores extremos** (mínimo y máximo)

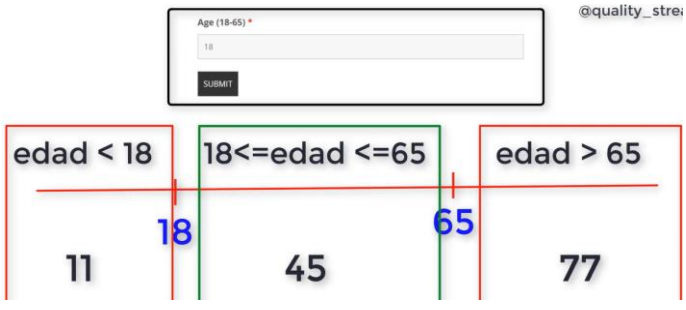
VIDEO 002

Profundización en la partición equivalente

En el 2018 las 5 pruebas más utilizadas fueron, pruebas de casos de uso, testing exploratorio, análisis de valores límites basadas en listas de chequeos , predicción de errores, partición de equivalencias.

Supongamos que tenemos un formulario para el ingreso de nombre, apellido, dni y la edad entre 18 y 65. El tester va a probar el campo edad con valores 18, 45 y 58. Analizamos estos valores de prueba = están probando los 3 valores lo mismo (Rangos válidos dentro del margen).

La técnica de partición de equivalencia está basada en la idea de dividir los datos en particiones, es decir cualquier partición puede subdividirse en sus particiones y es importante saber que cada valor debe pertenecer a una y sólo una partición y que la técnica de partición de equivalencia puede aplicarse a todos los niveles de prueba.



La hipótesis de esta prueba es que si una partición pasa el resto de los valores tendrá el mismo comportamiento, de igual forma si uno de ellos falla el resto de los elementos de esa partición hará que nuestro test falle. Es importante probar los valores no válidos de forma individual para evitar la enmascarados de fallos. ¿Cómo puede ocurrir ? esta enmascarados de fallos ,probando varios valores

NO VÁLIDOS al mismo tiempo, ocurren fallos y puede ser que uno de estos fallos hacen que el resto no sean visibles y pasen sin ser detectados.

En resumen, la partición de equivalencia es:

- **Definir la particiones**
- **Identificar aquellas particiones válidas e inválidas.**

Pregunta: 25

Puntos: 01

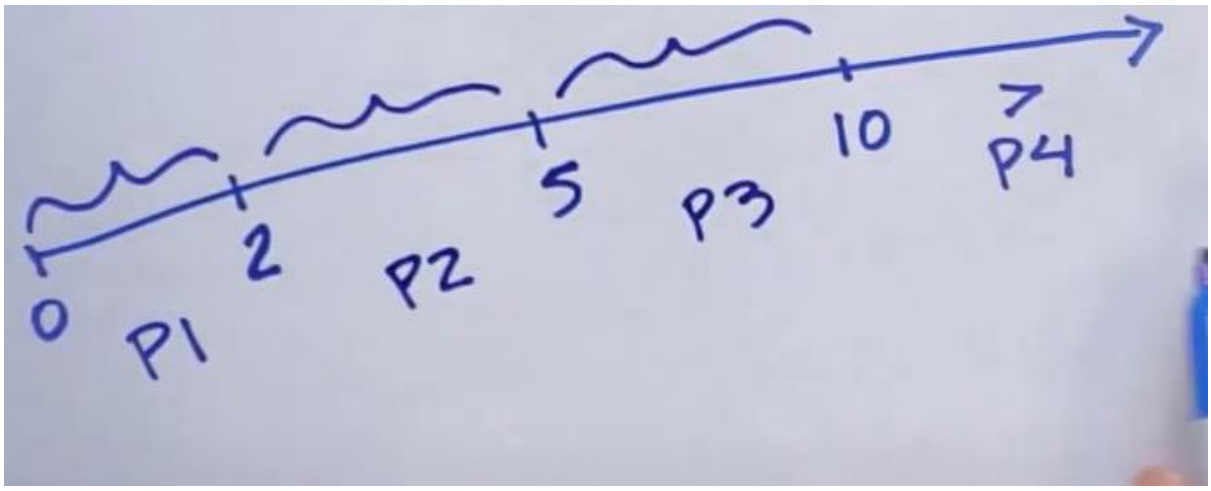
Se debe calcular la bonificación de un empleado. No puede ser negativa, pero se puede reducir a cero. La bonificación se basa en la duración del empleo:

- menos de o igual a 2 años,
- más de 2 años pero menos de 5 años,
- 5 a 10 años inclusive o más de 10 años.

¿Cuál es el número mínimo de casos de prueba necesario para cubrir todas las particiones de equivalencia válidas para calcular la bonificación?

a) 3.
b) 5.
c) 2.
d) 4.

Seleccionar **UNA** opción.



Pregunta: 21

Puntos: 01

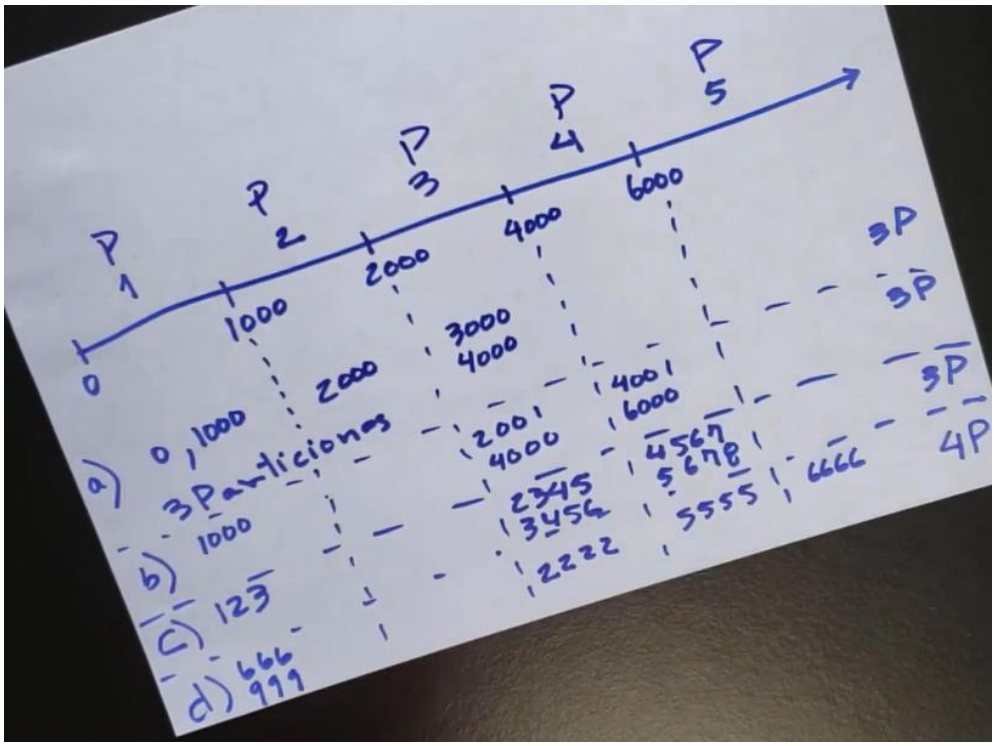
Una aplicación de entrenamiento físico mide el número de pasos que se caminan cada día y proporciona información para animar al usuario a mantenerse en forma.

La retroalimentación para las diferentes cantidades de pasos debe ser:

Hasta 1000 pasos - ¡Lleva una vida sedentaria!
Más de 1000 pasos, hasta 2000 - ¡Lleva una vida poco activa!
Más de 2000 pasos, hasta 4000 - ¡Se acerca al objetivo!
Más de 4000 pasos, hasta 6000 - ¡No está mal!
Más de 6000 pasos - ¡Así se hace!

¿Cuál de los siguientes conjuntos de entradas de prueba lograría la cobertura de partición de equivalencia más alta?

a)	(0,	1000,	2000,	3000,	4000)
b)	(1000,	2001,	4000,	4001,	6000)
c)	(123,	2345,	3456,	4567,	5678)
d)	(666,	999,	2222,	5555,	6666)



Ejercicios de partición equivalente

VIDEO 03

Pregunta: 22

Puntos: 01

Un registrador de radiación diaria para plantas genera una puntuación de radiación solar basada en una combinación del número de horas a la que una planta está expuesta al sol (menos de 3 horas, de 3 a 6 horas o más de 6 horas) y la intensidad media de la luz solar (muy baja, baja, media, alta).

Dados los siguientes casos de prueba:

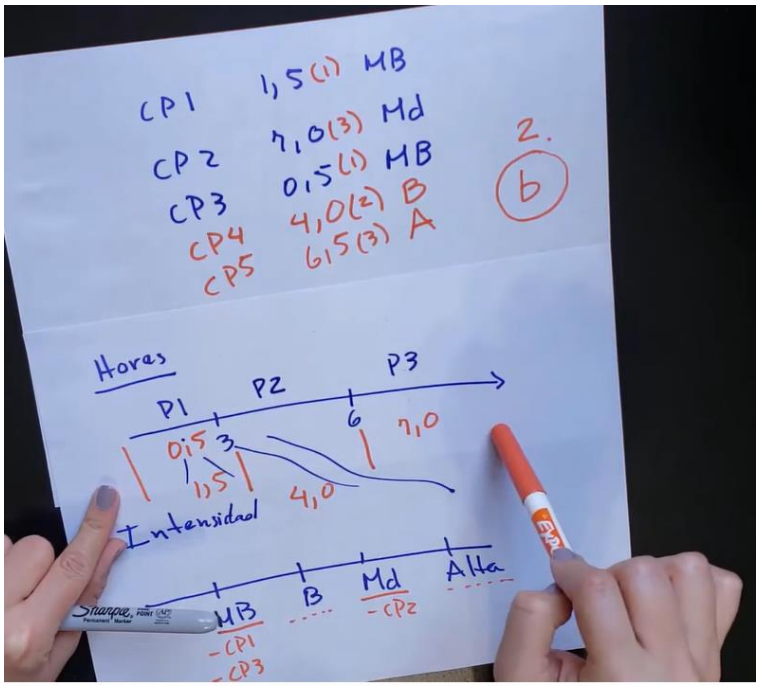
	HORAS	INTENSIDAD	PUNTUACIÓN
CP 1	1,5	muy baja	10
CP 2	7,0	media	60
CP 3	0,5	muy baja	10

¿Cuál es el número mínimo de casos de prueba adicionales que se necesitan para garantizar la cobertura completa de todas las particiones de equivalencia de **ENTRADA** válidas?

a) 1
b) 2
c) 3
d) 4

Seleccionar **UNA** opción.

La teoria dice que hay que encontrar 1 valor por cada partición



Pregunta # 29 (1 punto) Está probando un sistema de comercio electrónico que vende suministros de cocina como especias, harina y otros artículos a granel.
Las unidades en las que se venden los artículos son gramos (para especias y otros artículos caros) o kilogramos (para harina y otros artículos baratos). Independientemente de las unidades, la cantidad de pedido válida más pequeña es 0.5 unidades (por ejemplo, medio gramo de vainas de cardamomo) y la cantidad de pedido válida más grande es 25.0 unidades (por ejemplo, 25 kilogramos de azúcar).
La precisión del campo de unidades es de 0.1 unidades.

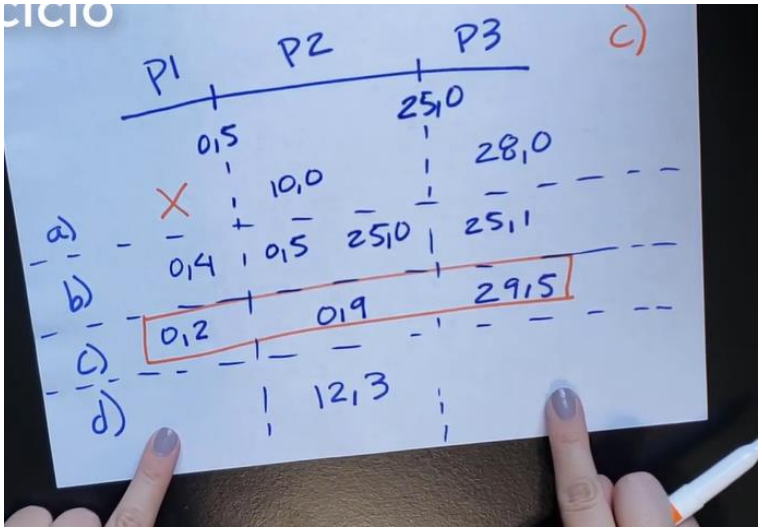
¿Cuál de los siguientes es un conjunto MÍNIMO de valores de entrada que cubren las particiones de equivalencia para este campo?

a) 10.0, 28.0
b) 0.4, 0.5, 25.0, 25.1
c) 0.2, 0.9, 29.5
d) 12.3

Seleccione UNA opción.

Examen C

@quality_s



Técnica Análisis de valores límites **VIDEO 04**

Análisis de valores límites es una extensión de la técnica de Partición de Equivalencia

Solo puede usarse cuando la partición está ordenada y consiste en datos numéricos o secuenciales.

Los valores mínimo y máximo (o los valores primeros y últimos) de una partición son sus valores límites (Beizer 1990).

Es importante aplicarla ya que hay probabilidad de que existe el comportamiento erroneo en los limites de las particiones

En el ejemplo anterior del formulario con la edad (18 a 65) Los valores limites son: 17-18 y 65-66 aca usamos la técnica de analisis de valor límites que usamos 2 valores límites.

Hay una de 3 límites acá se verifican los valores antes y después del valor límite,

es decir (17-18-19 / 64-65-66).

La técnica de análisis de valores límites se puede aplicar en todos los niveles de prueba y se usa generalmente cuando requieren de números ej= rango de números, fechas, horas, etc.
La cobertura de los limites de una partición se calcula = (Nro de valores límites probados / nro total de valores de prueba límites identificados) * 100.

Pregunta: 26

Examen A

Puntos: 01

Un sistema de control e información de velocidad tiene las siguientes características:

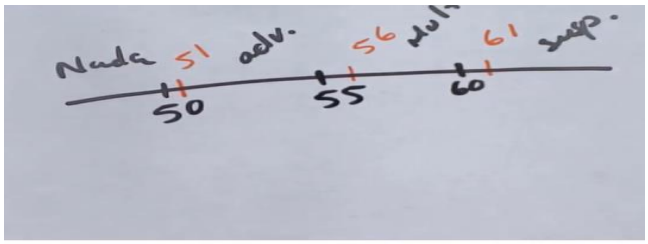
- Si usted conduce a 50 km/h o menos, no ocurrirá nada.
- Si usted conduce a más de 50 km/h, pero a 55 km/h o menos, será advertido.
- Si usted conduce a una velocidad superior a 55 km/h pero no superior a 60 km/h, se le impondrá una multa.
- Si usted conduce a más de 60 km/h, su licencia de conducir será suspendida.

La velocidad, en km/h, está disponible para el sistema como un valor entero.

¿Cuál sería el conjunto de valores (km/h) más probable identificado mediante la aplicación del análisis de valores frontera, en el que sólo son relevantes los valores frontera en las fronteras de las clases de equivalencia?

a) 0, 49, 50, 54, 59, 60.
b) 50, 55, 60.
c) 49, 50, 54, 55, 60, 62.
d) 50, 51, 55, 56, 60, 61.

Seleccionar **UNA** opción.

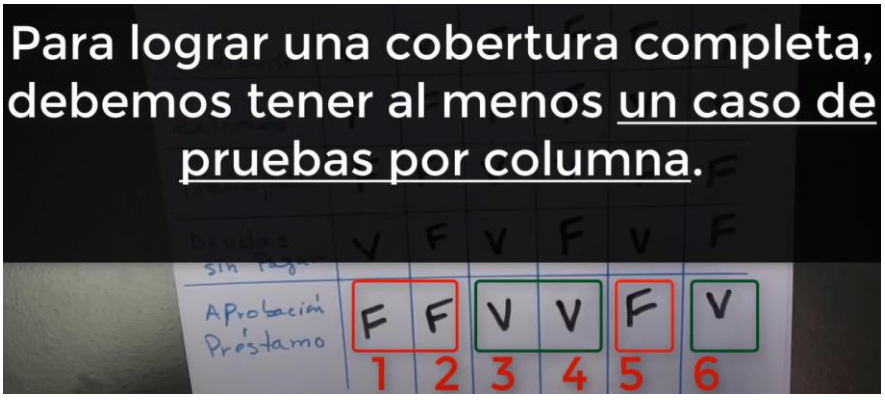


a) 0, 49, 50, 54, 59, 60.
b) 50, 55, 60.
c) 49, 50, 54, 55, 60, 62.
d) 50, 51, 55, 56, 60, 61.

Técnica de Tablas de decisión **VIDEO 05**

Es un tipo de técnica de caja negra y es muy útil cuando tenemos reglas de negocio complejas y cuando tenemos diferentes combinaciones de las condiciones del sistema producen resultados diferentes. EJ= **Sistema bancario que se encarga de la aprobación de préstamos.**

- Si la persona que solicita el préstamos es un estudiante no será aprobado el préstamo aunque éste no tenga deudas sin pagar.
- Si la persona que solicita el préstamos es un empleado/trabajador el préstamo será aprobado aunque este sí tenga deudas sin pagar.
- Si la persona que solicita el préstamo es un Retirado el préstamo será aprobado si este no presenta deudas sin pagar.
- Si la persona que solicita el préstamo es un Retirado el préstamo NO será aprobado si este presenta deudas sin pagar.



En este caso tenemos 6 casos de pruebas para lograr una cobertura completa.

si para alguna combinación no se conoce cual es el resultado esperado

Fortalezas de la técnica de la Tabla de Decisiones

- Ayuda a identificar todas las combinaciones importantes de las condiciones

- Ayuda a encontrar cualquier brecha en los requisitos

entonces pueden aclarar estos temas.

esta técnica puede aplicarse a todas las situaciones en la que el sistema dependa de una combinación de condiciones y puede aplicarse en todos los niveles de prueba.

Pregunta: 27

Puntos: 01

Los empleados de una empresa reciben bonificaciones si trabajan más de un año en la empresa y alcanzan un objetivo que se acuerda individualmente con anterioridad.

Estos hechos se pueden reflejar en una tabla de decisión:

ID Prueba		CP1	CP2	CP3	CP4
Condición1	¿Antigüedad en el empleo de más de 1 año?	SI	NO	NO	SI
Condición2	¿Objetivo acordado?	NO	NO	SI	SI
Condición3	¿Objetivo logrado?	NO	NO	SI	SI
Acción	Pago de bonificación	NO	NO	NO	NO

¿Qué caso de prueba para un escenario real falta en la tabla de decisión anterior?

a) Condición1 = SÍ, Condición2 = NO, Condición3 = SÍ, Acción = NO
b) Condición1 = SÍ, Condición2 = SÍ, Condición3 = NO, Acción = SÍ
c) Condición1 = NO, Condición2 = NO, Condición3 = SÍ, Acción = NO
d) Condición1 = NO, Condición2 = SÍ, Condición3 = NO, Acción = NO

Seleccionar **UNA** opción.

ID Prueba		CP1	CP2	CP3	CP4	d
Condición1	¿Antigüedad en el empleo de más de 1 año?	SI	NO	NO	SI	NO
Condición2	¿Objetivo acordado?	NO	NO	SI	SI	SI
Condición3	¿Objetivo logrado?	NO	NO	SI	SI	NO
Acción	Pago de bonificación	NO	NO	NO	NO	NO

Qué caso de prueba para un escenario real falta en la tabla de decisión anterior?

a) Condición1 = SÍ, Condición2 = NO, Condición3 = SÍ, Acción = NO
b) Condición1 = SÍ, Condición2 = SÍ, Condición3 = NO, Acción = SÍ
c) Condición1 = NO, Condición2 = NO, Condición3 = SÍ, Acción = NO
d) Condición1 = NO, Condición2 = SÍ, Condición3 = NO, Acción = NO → Escenario real

eleccionar **UNA** opción.

Técnica Transición de estados **VIDEO 06**

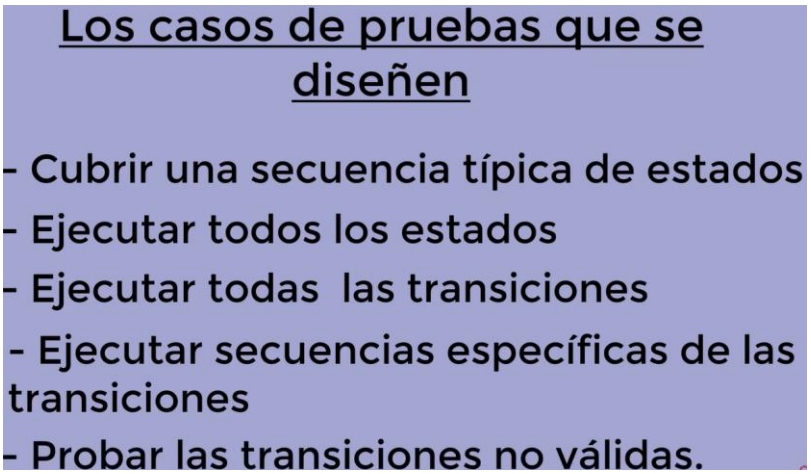
Se utiliza en aplicaciones basadas en menus, también es utilizada en industria de software embebido. Es adecuada para modelar escenarios de negocio que tengan estados específicos y tambien para probar como es que

ocurre la navegación en una pantalla.

Aplicación de la técnica podemos tener diagramas de transición de estado y/o tablas de transición de estado

El diagrama nos muestra posibles estados del software. El software ingresa, sale y transita entre los diferentes estados.Una transición se inicia mediante la ejecución de un evento. Y los eventos se ejecutarán por la acción de un usuario. Esto inicia la transición del software de A hacia B.

Tablas de transición de estado, tenemos las transiciones válidas, inválidas, eventos y acciones resultantes.

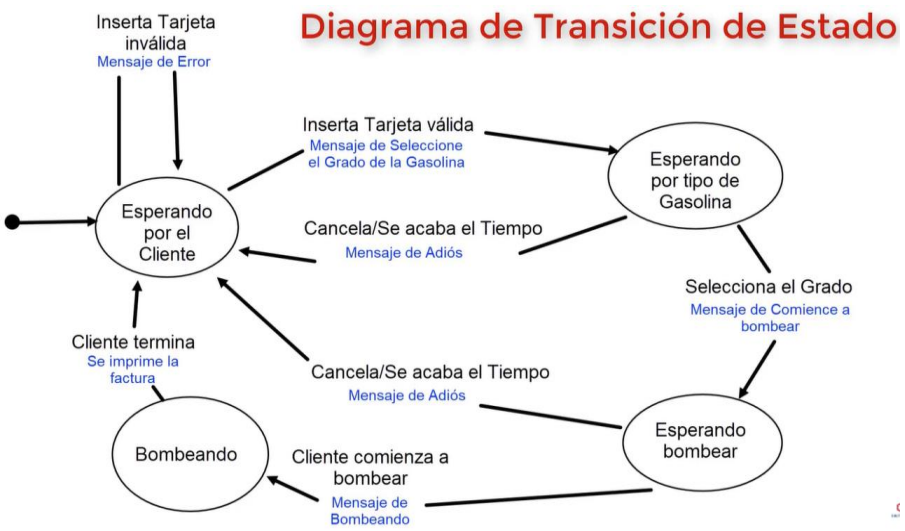


Cobertura de esta técnica:

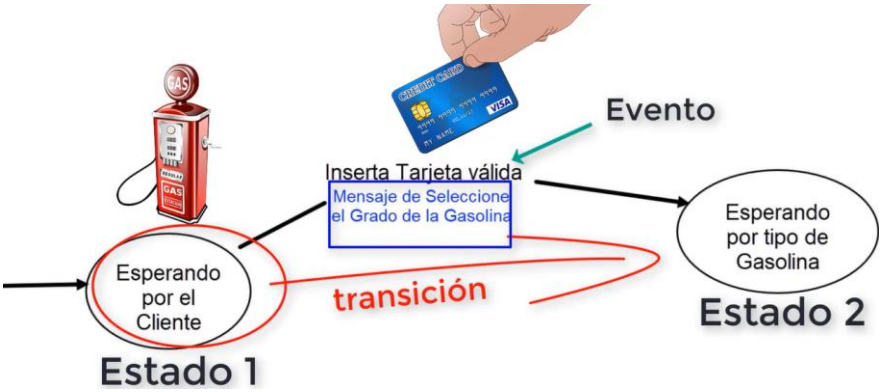
Estados identificados * 100 / Total de estados

Transiciones probadas * 100 / Total de transiciones

EJ= De una estación de gasolina



Tenemos 4 estados por donde el software puede pasar.



Estados= esperando por el cliente
Evento= Inserta tarjeta valida
Acción= Mensaje de seleccione el grado de la gasolina

Casos de uso

Técnica de casos de uso

- Un caso de uso es una descripción de uso particular del sistema por un actor.
- Ayuda a identificar casos que cubren el sistema de inicio a fin.
- Los actores son generalmente personas (usuarios) pero también pueden ser otros sistemas o sub-sistemas.
- Los casos de uso se definen en términos del autor, de lo que puede y no puede hacer.

Técnica de casos de uso

- Generalmente usan un lenguaje de negocio y no técnico.
- Sirven de base para creación de casos de prueba de aceptación.
- Cada caso debe especificar cualquier pre-condición que se deba cumplir para ejecutar dicho caso.
- Cada caso debe especificar cualquier post-condición o resultados y descripciones de estados finales luego de ser ejecutado.

Ejemplo

	Step	Description
Main Success Scenario **A: Actor** **S: System**	1	A: Inserts card
	2	S: Validates card and asks for PIN
	3	A: Enters PIN
	4	S: Validates PIN
	5	S: Allows access to account
Extensions	2a	Card not valid S: Display message and reject card
	4a	PIN not valid S: Display message and ask for re-try (twice)
	4b	PIN invalid 3 times S: Eat card and exit

Los casos de uso son herramientas esenciales para comprender cómo los usuarios interactúan con un sistema de software y garantizar que cumpla con sus requisitos

Paso 1: Identificar Actores Principales

La creación de casos de uso es identificar los actores principales involucrados en el sistema. Los actores son personas, otros sistemas o entidades que interactúan con el software. Por ejemplo, en un sistema de compras en línea, los actores pueden ser usuarios, administradores y proveedores.

Paso 2: Definir Características Clave

Debemos definir las características clave que el sistema debe proporcionar para satisfacer las necesidades de esos actores. Estas características

pueden incluir la búsqueda de productos, la realización de pedidos, la gestión de cuentas de usuario, entre otras.

Paso 3: Crear Diagramas de Casos de Uso

Es una excelente forma de visualizar las interacciones entre los actores y las características del sistema. Utilizamos cajas para representar los casos de uso y líneas para mostrar cómo los actores interactúan con ellos. Por ejemplo, podríamos tener un diagrama que muestra cómo un usuario realiza un pedido en un sistema de compras en línea.

Paso 4: Especificar Detalles de los Casos de Uso

Esto implica escribir una descripción detallada de cómo se realiza cada acción y cuáles son las entradas y salidas esperadas. Por ejemplo, para el caso de uso "Realizar Pedido", describiríamos el proceso paso a paso, incluyendo la selección de productos, la introducción de la información de envío y el proceso de pago.

Paso 5: Diseñar Casos de Prueba

Una vez que tengamos definidos los casos de uso y sus detalles, podemos diseñar casos de prueba basados en ellos. Cada caso de prueba representa una forma de probar una función específica del sistema. Por ejemplo, podríamos tener un caso de prueba que simula el proceso de realizar un pedido con un producto agotado para asegurarnos de que el sistema maneje esta situación correctamente.

Ejemplo Práctico: Sistema de Compras en Línea

Supongamos que estamos probando un sistema de compras en línea. Hemos identificado a los actores principales (usuarios, administradores y proveedores) y hemos definido características clave como la búsqueda de productos, la realización de pedidos y la gestión de cuentas.

Hemos creado un diagrama de casos de uso que muestra cómo los usuarios pueden buscar productos, agregarlos al carrito de compras y realizar pedidos. Luego, hemos especificado los detalles del caso de uso "Realizar Pedido", detallando cómo un usuario completa esta acción, incluyendo la validación de la información de envío y pago.

Finalmente, hemos diseñado casos de prueba que abordan diferentes escenarios, como pedidos exitosos, pedidos con productos agotados y pedidos con información de pago incorrecta.

Dominar el proceso paso a paso para casos de uso es fundamental para realizar pruebas efectivas de software. Con una comprensión clara de los actores, las características, los diagramas de casos de uso, los detalles de los casos de uso y los casos de prueba, podemos asegurarnos de que el software cumpla con los requisitos y funcione de manera óptima.