Introducción a los ciclos de prueba software

Caso práctico

Tenemos que desarrollar una aplicación de gestión de cuentas bancarias que es similar a la que se pidió en el hito 1.

Los datos de la aplicación se alojan en la base de datos relacional *MySQL* de nombre *BANCO*. Las principales tablas de la base de datos *BANCO* son:

- *CLIENTE* con los campos DNI (clave primaria), NOMBRE y SALDO.
- *MOVIMIENTO* con los campos DNI e IMPORTE.

Un cliente tiene entre 0 y n movimientos asociados.

Se accederá a la aplicación desde el siguiente menú de consola:

```
1. Crear nueva cuenta.
2. Consultar saldo.
3. Ver movimientos.
4. Realizar depósito.
5. Realizar retiro.
6. Eliminar cuenta.
7. Salir.
```

La aplicación se programará en lenguaje JAVA en los paquetes *es.banco* y *es.banco.app*.

Terminología

Vamos a ver la terminología asociada a los ciclos de prueba software.

Ciclo de prueba

Un ciclo de prueba es una ejecución completa del conjunto de casos de prueba diseñado para evaluar si el software cumple con sus requisitos. Cada ciclo se realiza en una versión específica del software y permite detectar cumplimientos y fallos a medida que se hacen cambios y mejoras.

Lo habitual en el mundo del desarrollo software es realizar al menos 3 ciclos de prueba, pues se da por hecho que en el primer ciclo aparecerán errores que habrá que corregir antes de realizar el segundo ciclo de prueba.

Tipos de prueba asociados a un ciclo de prueba

Hay diferentes tipologías de prueba en el ciclo de vida de un desarrollo de software. Lo habitual es que un ciclo de prueba se asocie a una tipología de prueba. Es decir, todos los casos de prueba son de la misma tipología de prueba.

La terminología varía entre metodologías de prueba. Algunas de las tipologías de prueba más habituales son:

- **Prueba de humo:** prueba inicial para verificar que las funcionalidades básicas y críticas son correctas antes de proceder con pruebas más exhaustivas. Por ejemplo, probar que hay conectividad con la base de datos.
- **Pruebas funcionales**: verifican que las funcionalidades cumplan con los requisitos establecidos.
- **Pruebas de integración**: evalúan cómo interactúan diferentes componentes del sistema entre sí. Por ejemplo, probar que los componentes desarrollados por personas diferentes interactúan correctamente.
- **Pruebas de rendimiento**: evalúan el desempeño del sistema bajo condiciones específicas de rendimiento.

Por tanto, y a modo de ejemplo, no deberíamos diseñar un ciclo de prueba que tuviera casos de prueba funcionales y casos de prueba de rendimiento. Deberíamos diseñar un ciclo de pruebas funcionales y otro ciclo de pruebas de rendimiento.

Caso de prueba

Un caso de prueba es un conjunto específico de condiciones y pasos que permiten verificar el correcto funcionamiento de un componente del sistema, y que da como resultado que el sistema cumple (“aprobado”, OK) o incumple (“fallo”, NOK).

Tipos de prueba asociados a un caso de prueba

Un caso de prueba se asocia a uno de estas tipologías:

- **Pruebas de progresión**: se utilizan para validar una nueva funcionalidad o un cambio sobre una funcionalidad. Es decir, se trata de la primera vez que probamos algo. El primer ciclo de prueba debería estar formado por pruebas de progresión únicamente..
- **Pruebas de confirmación**: validan que un error detectado se ha corregido. Típicamente la prueba de confirmación de un ciclo de prueba se hace para comprobar que se han corregido los fallos detectados en el ciclo de prueba anterior.
- **Pruebas de regresión**: garantizan que el sistema sigue funcionando correctamente después de un cambio o corrección. Es decir, queremos asegurarnos que un caso de prueba que no fallaba en el ciclo anterior, sigue sin fallar en este ciclo. Es frecuente que al “arreglar” algo “estropeemos” otra cosa.

Pruebas positivas y pruebas negativas

Las **pruebas positivas** y **pruebas negativas** son enfoques complementarios en la verificación de software que ayudan a garantizar que una aplicación no solo funciona correctamente en escenarios normales (esperados), sino que también maneja adecuadamente situaciones de error o de entradas incorrectas.

Pruebas positivas

Las pruebas positivas buscan confirmar que el sistema se comporta correctamente cuando se usan entradas y condiciones válidas, es decir, escenarios donde se espera que la funcionalidad se ejecute sin problemas.

Son pruebas orientadas a comprobar que “el sistema hace lo que tiene que hacer”.

Las pruebas positivas tienen como objetivo verificar que el sistema cumple con los requisitos funcionales en condiciones normales de operación.

En la app del banco, serían pruebas positivas:

- Crear una cuenta con un nombre y un DNI válidos, y verificar que la cuenta se crea correctamente.
- Depositar una cantidad positiva (por ejemplo, 100 €) en una cuenta y confirmar que el saldo aumenta en consecuencia.

Pruebas negativas

Las pruebas negativas buscan confirmar que el sistema responde adecuadamente a entradas o condiciones inesperadas o inválidas. Evalúan la robustez del software, comprobando que se eviten, manejen o reporten adecuadamente los errores en situaciones anómalas.

Son pruebas orientadas a comprobar que “el sistema no hace lo que no tiene que hacer”.

Las pruebas negativas tienen como objetivo verificar que el sistema no falle de manera incontrolada (por ejemplo excepciones no capturadas) y que proporcione mensajes de error o respuestas adecuadas ante situaciones inesperadas.

En la app del banco, serían pruebas negativas:

- Intentar crear una cuenta con un DNI no válido (por ejemplo, con caracteres en lugar de números) y verificar que el sistema rechace la entrada con un mensaje de error.
- Intentar retirar una cantidad mayor al saldo disponible y confirmar que la operación se bloquea, proporcionando un mensaje de error sobre la falta de fondos.

Pruebas positivas y negativas en ciclos de pruebas

Ambas pruebas son esenciales y complementarias en un mismo ciclo de pruebas de software:

- **Pruebas positivas**: Validan que las funcionalidades básicas y el flujo principal de la aplicación funcionan como se espera en condiciones normales.
- **Pruebas negativas**: Aseguran que la aplicación sea resistente a los errores y a los datos no válidos, previniendo comportamientos indeseados y potenciales fallos.

En un ciclo de pruebas se recomienda empezar por las pruebas negativas.

Partes de un caso de prueba

Un caso de prueba incluye las siguientes partes:

- **ID**: Identificador único.
    - Ej.: *CP01*
- **Tipo de Prueba**: progresión, confirmación o regresión.
    - Ej.: *progresión*
- **Componente**: componente software que se prueba.
    - Ej.: Clase *es.banco.GestorApp*
- **Objetivo**: breve descripción de lo que queremos comprobar con este caso de prueba.
    - Ej.:*El sistema da de alta correctamente a un nuevo cliente*
- **Precondiciones**: estado del sistema antes de la ejecución de la prueba. Es decir, qué debe ser cierto antes de la prueba para que pueda realizar la prueba.
    - Ej.: *La tabla CLIENTE no contiene un tupla con DNI = "12345678A".*
    - Ej.: *La query SELECT COUNT(*) FROM CLIENTE WHERE DNI= "12345678A" devuelve 0.*
- **Datos de prueba**: datos específicos y concretos que son necesarios para ejecutar la prueba. Pueden ser valores que debo introducir por pantalla, valores que debe tener una tabla, un archivo,...
    - Ej.: Introducir *Nombre = "Javier" y DNI = "12345678A"*.
- **Resultado esperado**: resultado que se espera si la prueba es correcta. Por ejemplo, qué valor debe salir por pantalla.
    - Ej.: Sale por pantalla *"El nuevo cliente se ha insertado correctamente"*.
- **Comentario**: observaciones adicionales.
- **Postcondiciones**: Estado del sistema tras la prueba. Es decir, que debe ser cierto después de ejecutar el caso de prueba.
    - Ej.: *La query SELECT * FROM CLIENTE WHERE DNI= "12345678A" devuelve 12345678A Javier.*
- **Estado**:
    - Pending: pendiente de ejecutar.
    - OK: aprobado.
    - NOK: fallido**.**
    - Blocked: bloqueado, por ejemplo hasta que se corrija un caso de prueba anterior.
- **Evidencia**: capturas de pantallas o logs que respaldan los resultados obtenidos.

Ejemplo aplicado al caso práctico

Paso 1: ciclo de prueba 0

Este ciclo de prueba contiene pruebas de humo para comprobar la conectividad con la base de datos *BANCO* y las tablas *CLIENTE* y *MOVIMIENTO*.

El objetivo de este ciclo inicial es asegurar que la aplicación puede conectarse a la base de datos *BANCO* y que las tablas necesarias *CLIENTE* y *MOVIMIENTO* existen y están configuradas correctamente:

Ciclo de prueba	Humo v1
Tipo de pruebas	pruebas de humo
Componentes software afectados	BBDD BANCO, tablas CLIENTE y MOVIMIENTO v1.0

ID	Estado	Tipo	Componente	Objetivo	Precondiciones	Datos de prueba	Resultado esperado	Comentarios	Postcondiciones
CPH-10	Pending	Progresión	BANCO	Conectividad con el gestor de BBDD	MySQL instalado	Accede al workbench con user=”root” y password=”campusfp”.	Se accede a la instancia *BANCO*		
	Estado	OK	Evidencia	Captura de pantalla					
…		…		…	…	…	…		
	Estado	Pending	Evidencia						

Paso 2: diseño de los casos de prueba

Diseñamos casos de prueba funcionales para comprobar el ciclo de vida de un cliente.
Estos casos de prueba se redactan antes de empezar a codificar.

Casos de prueba	Funcional v0
Tipo de pruebas	pruebas funcionales
Componentes software afectados	paquetes es.banco y es banco.app v0.0

ID	Tipo	Componente	Objetivo	Precondiciones	Datos de prueba (Valores a introducir en la consola)	Resultado esperado (mensaje en la consola)	Comentarios	Postcondiciones
CPF-10	Progresión	es.banco.GestorCuenta	Comprobar que se crea correctamente un nuevo cliente	CPH-10 ejecutado correctamente SELECT COUNT(*) FROM CLIENTE WHERE DNI=”12345678A” devuelve 0	Opción de menú: 1 Nombre: Javier. DNI: 12345678A.	El cliente se ha creado correctamente		SELECT * FROM CLIENTE WHERE DNI=”12345678A” devuelve Javier 12345678A 0€
CPF-20	Progresión	es.banco.GestorCuenta	Comprobar que el saldo inicial es 0€	CPF-10 ejecutado correctamente	Opción de menú: 2	El saldo de la cuenta es 0€		SELECT SALDO FROM CLIENTE WHERE DNI=”12345678A” devuelve 0€
CPF-30	Progresión	es.banco.GestorCuenta	Comprobar que no hay movimientos	CPF-20 ejecutado correctamente	Opción de menú: 3	No hay movimientos en la cuenta		SELECT COUNT(*) FROM MOVIMIENTO WHERE DNI=”12345678A” devuelve 0
CPF-40	Progresión	es.banco.GestorCuenta	Comprobar que se realiza un depósito correctamente.	CPF-30 ejecutado correctamente	Opción de menú: 4 Importe: 100€.	El depósito se ha creado correctamente		SELECT DEPOSITO FROM CLIENTE WHERE DNI=”12345678A” devuelve 100€ SELECT COUNT(*) FROM MOVIMIENTO WHERE DNI=”12345678A” devuelve 1
…	…	…	…	…	…	…		…

Paso 3: ejecutamos el ciclo de prueba 1

Codificamos la aplicación, realizamos pruebas unitarias y pruebas de integración si fueran necesarias.
Es posible que durante la codificación de la aplicación hayamos visto la necesidad de añadir, eliminar o modificar los casos de prueba funcional originales.
Durante la codificación hemos podido ir ejecutando también algunos de los casos de prueba para ir realizando validaciones parciales.
Una vez que consideremos que la aplicación está estable generamos la versión 1.0 del código fuente.
Procedemos a ejecutar el ciclo 1 de pruebas funcionales.

Ciclo de prueba	Funcional v1
Tipo de pruebas	pruebas funcionales
Componentes software afectados	paquetes es.banco y es banco.app v1.0

ID	Tipo	Componente	Objetivo	Precondiciones	Datos de prueba (Valores a introducir en la consola)	Resultado esperado (mensaje en la consola)	Comentarios	Postcondiciones
CPF-10	Progresión	es.banco.GestorCuenta	Comprobar que se crea correctamente un nuevo cliente	CPH-10 ejecutado correctamente SELECT COUNT(*) FROM CLIENTE WHERE DNI=”12345678A” devuelve 0	Opción de menú: 1 Nombre: Javier. DNI: 12345678A.	El cliente se ha creado correctamente		SELECT * FROM CLIENTE WHERE DNI=”12345678A” devuelve Javier 12345678A 0€
	Estado	OK	Evidencia	Captura de pantalla de la app. Captura de pantalla de la consola SQL				
CPF-20	Progresión	es.banco.GestorCuenta	Comprobar que el saldo inicial es 0€	CPF-10 ejecutado correctamente	Opción de menú: 2	El saldo de la cuenta es 0€		SELECT SALDO FROM CLIENTE WHERE DNI=”12345678A” devuelve 0€
	Estado	OK	Evidencia	Captura de pantalla de la app. Captura de pantalla de la consola SQL				
CPF-30	Progresión	es.banco.GestorCuenta	Comprobar que no hay movimientos	CPF-20 ejecutado correctamente	Opción de menú: 3	No hay movimientos en la cuenta		SELECT COUNT(*) FROM MOVIMIENTO WHERE DNI=”12345678A” devuelve 0
	Estado	OK	Evidencia	Captura de pantalla de la app. Captura de pantalla de la consola SQL				
CPF-40	Progresión	es.banco.GestorCuenta	Comprobar que se realiza un depósito correctamente.	CPF-30 ejecutado correctamente	Opción de menú: 4 Importe: 100€.	El depósito se ha creado correctamente	No se ha creado una nueva tupla de MOVIMIENTO. SELECT COUNT(*) FROM MOVIMIENTO WHERE DNI=”12345678A” devuelve 0 en vez de 1. Se interrumpe el ciclo de prueba.	SELECT DEPOSITO FROM CLIENTE WHERE DNI=”12345678A” devuelve 100€ SELECT COUNT(*) FROM MOVIMIENTO WHERE DNI=”12345678A” devuelve 1
	Estado	NOK	Evidencia	Captura de pantalla de la app. Captura de pantalla de la consola SQL				
…	Progresión	…	…	…	…	…		…
	Estado	Blocked	Evidencia					

En este ejemplo vemos que el caso de prueba CPF-40 ha fallado y que decidimos interrumpir la ejecución del ciclo de prueba.

Paso 4: ejecutamos el ciclo de prueba 2

Corregimos la incidencia detectada al ejecutar el caso de prueba CPF-40, realizamos pruebas unitarias y pruebas de integración si fueran necesarias.
La corrección realizada afecta a los archivos fuente implicados en los casos de prueba anteriores.
Es posible que durante la corrección de la incidencia hayamos visto la necesidad de añadir, eliminar o modificar casos de prueba funcionales del ciclo anterior.
Durante la codificación hemos podido ir ejecutando también algunos de los casos de prueba para ir realizando validaciones parciales.
Una vez que consideremos que la aplicación está de nuevo estable generamos la versión 1.1 del código fuente.
Procedemos a ejecutar el ciclo de prueba 2 con una nueva versión de los casos de prueba funcionales. Volvemos a ejecutar los casos de prueba que no dieron error antes (prueba de regresión) pues pueden haberse visto afectados por la corrección de la incidencia.

Ciclo de prueba	Funcional v1.1
Tipo de pruebas	pruebas funcionales
Componentes software afectados	paquetes es.banco y es banco.app v1.1

ID	Tipo	Componente	Objetivo	Precondiciones	Datos de prueba (Valores a introducir en la consola)	Resultado esperado (mensaje en la consola)	Comentarios	Postcondiciones
CPF-10	Regresión	es.banco.GestorCuenta	Comprobar que se crea correctamente un nuevo cliente	CPH-10 ejecutado correctamente SELECT COUNT(*) FROM CLIENTE WHERE DNI=”12345678A” devuelve 0	Opción de menú: 1 Nombre: Javier. DNI: 12345678A.	El cliente se ha creado correctamente		SELECT * FROM CLIENTE WHERE DNI=”12345678A” devuelve Javier 12345678A 0€
	Estado	OK	Evidencia	Captura de pantalla de la app. Captura de pantalla de la consola SQL				
CPF-20	Regresión	es.banco.GestorCuenta	Comprobar que el saldo inicial es 0€	CPF-10 ejecutado correctamente	Opción de menú: 2	El saldo de la cuenta es 0€	La aplicación lanza una excepción. Se detiene la ejecución del ciclo de pruebas	SELECT SALDO FROM CLIENTE WHERE DNI=”12345678A” devuelve 0€
	Estado	NOK	Evidencia	Captura de pantalla de la app. Captura de pantalla de la consola SQL				
CPF-30 BLOQUEADA hasta que se corrija CPF-20	Regresión	es.banco.GestorCuenta	Comprobar que no hay movimientos	CPF-20 ejecutado correctamente	Opción de menú: 3	No hay movimientos en la cuenta		SELECT COUNT(*) FROM MOVIMIENTO WHERE DNI=”12345678A” devuelve 0
	Estado	Blocked	Evidencia					
CPF-30 BLOQUEADA hasta que se corrija CPF-20	Confirmación	es.banco.GestorCuenta	Comprobar que se realiza un depósito correctamente.	CPF-30 ejecutado correctamente	Opción de menú: 4 Importe: 100€.	El depósito se ha creado correctamente	No se ha creado una nueva tupla de MOVIMIENTO. SELECT COUNT(*) FROM MOVIMIENTO WHERE DNI=”12345678A” devuelve 0 en vez de 1. Se interrumpe el ciclo de prueba.	SELECT DEPOSITO FROM CLIENTE WHERE DNI=”12345678A” devuelve 100€ SELECT COUNT(*) FROM MOVIMIENTO WHERE DNI=”12345678A” devuelve 1
	Estado	Blocked	Evidencia					
…	Progresión	…	…	…	…	…		…
	Estado	Blocked	Evidencia					

En este ejemplo vemos que el caso de prueba CPF-40 ha fallado y que decidimos interrumpir la ejecución del ciclo de prueba.

Paso 5: ejecutamos el ciclo de prueba 3

Corregimos la incidencia detectada en la ejecución del caso de prueba CPF-20, realizamos pruebas unitarias y pruebas de integración si fueran necesarias.
La corrección realizada afecta a los archivos fuente implicados en los casos de prueba anteriores.
Es posible que durante la corrección de la incidencia hayamos visto la necesidad de añadir, eliminar o modificar casos de prueba funcionales del ciclo anterior.
Durante la codificación hemos podido ir ejecutando también algunos de los casos de prueba para ir realizando validaciones parciales.
Una vez que consideremos que la aplicación está de nuevo estable generamos la versión 1.2 del código fuente.
Procedemos a ejecutar el ciclo de prueba 3 con una nueva versión de los casos de prueba funcionales. Volvemos a ejecutar los casos de prueba que no dieron error antes (prueba de regresión) pues pueden haberse visto afectados por la corrección de la incidencia.

Ciclo de prueba	Funcional v1.2
Tipo de pruebas	pruebas funcionales
Componentes software afectados	paquetes es.banco y es banco.app v1.2

ID	Tipo	Componente	Objetivo	Precondiciones	Datos de prueba (Valores a introducir en la consola)	Resultado esperado (mensaje en la consola)	Comentarios	Postcondiciones
CPF-10	Regresión	es.banco.GestorCuenta	Comprobar que se crea correctamente un nuevo cliente	CPH-10 ejecutado correctamente SELECT COUNT(*) FROM CLIENTE WHERE DNI=”12345678A” devuelve 0	Opción de menú: 1 Nombre: Javier. DNI: 12345678A.	El cliente se ha creado correctamente		SELECT * FROM CLIENTE WHERE DNI=”12345678A” devuelve Javier 12345678A 0€
	Estado	OK	Evidencia	Captura de pantalla de la app. Captura de pantalla de la consola SQL				
CPF-20	Confirmación	es.banco.GestorCuenta	Comprobar que el saldo inicial es 0€	CPF-10 ejecutado correctamente	Opción de menú: 2	El saldo de la cuenta es 0€	Incidencia corregida	SELECT SALDO FROM CLIENTE WHERE DNI=”12345678A” devuelve 0€
	Estado	OK	Evidencia	Captura de pantalla de la app. Captura de pantalla de la consola SQL				
CPF-30	Regresión	es.banco.GestorCuenta	Comprobar que no hay movimientos	CPF-20 ejecutado correctamente	Opción de menú: 3	No hay movimientos en la cuenta		SELECT COUNT(*) FROM MOVIMIENTO WHERE DNI=”12345678A” devuelve 0
	Estado	OK	Evidencia	Captura de pantalla de la app. Captura de pantalla de la consola SQL				
CPF-40	Confirmación	es.banco.GestorCuenta	Comprobar que se realiza un depósito correctamente.	CPF-30 ejecutado correctamente	Opción de menú: 4 Importe: 100€.	El depósito se ha creado correctamente	Incidencia corregida	SELECT DEPOSITO FROM CLIENTE WHERE DNI=”12345678A” devuelve 100€ SELECT COUNT(*) FROM MOVIMIENTO WHERE DNI=”12345678A” devuelve 1
	Estado	OK	Evidencia	Captura de pantalla de la app. Captura de pantalla de la consola SQL				
…	Progresión	…	…	…	…	…		…
	Evidencia						Resultado	

En este ejemplo todos los casos de prueba se ejecutan correctamente. Damos por finalizadas las pruebas funcionales.