Pruebas del software: uso de un framework de testing

Trabajo Práctico 1 - 28 de Agosto del 2024
Ingeniería de Software 1 (PU) - Ingeniería de Software 2 (LI / II)

Este trabajo práctico está diseñado para guiarte desde las pruebas unitarias simples y de integración hasta escenarios de prueba más complejos, aumentando gradualmente la complejidad. Puedes usar un lenguaje de programación y framework a elección

Problema: “Sistema de Gestión de Productos en una Tienda”

Vas a desarrollar y probar un sistema básico de gestión de productos en una tienda, utilizando dos clases principales: `Producto` y `Tienda` dadas a continuación. La clase `Tienda` gestionará un inventario de productos, permitiendo agregar, eliminar, buscar productos, aplicar descuentos, y calcular el precio total de un carrito de compras.

Organización del Proyecto:

- **Producto** (`producto.py`): Clase que representa un producto en la tienda.
- **Tienda** (`tienda.py`): Clase que maneja el inventario y las operaciones de la tienda.
- **Tests** (`test_tienda.py/test_producto.py`): Archivo donde se implementarán las pruebas utilizando el framework.

Código base en python:

```python
class Producto:
    def __init__(self, nombre, precio, categoria):
        self.nombre = nombre
        self.precio = precio
        self.categoria = categoria
```

```python
from producto import Producto

class Tienda:
    def __init__(self):
        self.inventario = []
    def agregar_producto(self, producto):
        self.inventario.append(producto)
    def buscar_producto(self, nombre):
        for producto in self.inventario:
            if producto.nombre == nombre:
                return producto
        return None
    def eliminar_producto(self, nombre):
        for producto in self.inventario:
            if producto.nombre == nombre:
                self.inventario.remove(producto)
                return True
        return False
```

1) Configuración Inicial y Pruebas Básicas

Objetivo: Aprender a configurar un entorno de pruebas con el framework elegido y realizar pruebas básicas.

1. Usa la implementación base de la clase `Producto` con atributos como `nombre`, `precio`, y `categoria`.
2. Usa la implementación base la clase `Tienda` con métodos para:
    a. Agregar un producto al inventario.
    b. Buscar un producto por su nombre.
    c. Eliminar un producto del inventario.

Tarea práctica:

- Escribe pruebas básicas para estas clases y métodos utilizando el framework elegido.
- Asegúrate de cubrir casos simples, como agregar un producto y verificar que esté en la tienda, buscar un producto existente y no existente, y eliminar un producto.

Preguntas:

- ¿Puedes identificar pruebas de unidad y de integración en la práctica que realizaste?

2) Pruebas con Excepciones

Objetivo: Ampliar las pruebas para manejar y verificar el manejo adecuado de excepciones.

1. Modifica el método `buscar_producto` para que lance una excepción si no se encuentra un producto.
2. Modifica el método `eliminar_producto` para que lance una excepción si se intenta eliminar un producto que no existe.
3. Implementa un método `actualizar_precio` en la clase `Producto` que permita cambiar el precio de un producto y lance una excepción si el nuevo precio es negativo.

Tarea:

- Escribe pruebas que verifiquen que se lanzan las excepciones correctas al intentar eliminar un producto inexistente o actualizar un precio a un valor negativo.
- Utiliza el framework para comprobar que las excepciones se lanzan correctamente.

Preguntas:

- Podría haber escrito las pruebas primero antes de modificar el código de la aplicación? ¿Cómo sería el proceso de escribir primero los tests?

3) Uso de Mocks para aislar unidades bajo prueba

Objetivo: Introducir el uso de mocks para aislar unidades de prueba y evitar dependencias innecesarias.

1. Implementa un método `aplicar_descuento` en la clase `Tienda` que acepte un nombre de producto y un porcentaje de descuento y que modifique el precio del producto.
2. Utiliza un “mock” para simular el comportamiento del producto.

Tarea:

- Escribe pruebas utilizando mocks para verificar que el método `aplicar_descuento` calcula correctamente el nuevo precio sin usar un objeto real.

- Asegúrate de verificar que la función llama correctamente al método `actualizar_precio` del producto.
- [Opcional] Utilizar mocks para los tests de agregar, buscar y eliminar productos.

Preguntas:
- En lo que va del trabajo práctico, ¿puedes identificar 'Controladores' y 'Resguardos'?
- ¿Qué es un mock? ¿Hay otros nombres para los objetos/funciones simulados?

4) Uso de Fixtures para Reutilización de Datos

Objetivo: Aprender a usar “fixtures” para preparar y limpiar el entorno de pruebas.

1. Crea un fixture que inicialice una tienda con algunos productos de ejemplo antes de cada prueba.
2. Modifica las pruebas existentes para usar este fixture en lugar de crear los productos manualmente en cada prueba.

Tarea:
- Implementa el fixture y utilízalo en las pruebas de `agregar_producto` y `buscar_producto`.
- Verifica que las pruebas se ejecutan correctamente utilizando los productos predefinidos.

Preguntas:
- ¿Qué ventajas ve en el uso de fixtures? ¿Qué enfoque estaría aplicando?
- Explique los conceptos de Setup y Teardown en testing.

5) Pruebas de Integración y Cobertura Completa

Objetivo: Realizar pruebas más complejas que verifiquen la integración de varias funciones y un flujo completo del sistema.

1. Implementa un método `calcular_total_carrito` en la clase `Tienda` que reciba una lista de nombres de productos (el carrito) y devuelva el precio total.
2. Integra todas las funciones anteriores para permitir agregar productos a un carrito y calcular el total.

Tarea:
- Escribe pruebas de integración que verifiquen el flujo completo para calcular el total del carrito después de aplicar los descuentos.
- Utiliza un fixture para inicializar una tienda con varios productos.
- Verifica que el precio total calculado sea correcto.

Preguntas:
- ¿Puede describir una situación de desarrollo para este caso en donde se plantee pruebas de integración ascendente? Describa la situación.