Profundiza más

Recurso de Profundización

Clase 16

Autor: Damián Nicolalde Rodríguez

Objetivo del Documento: Comprender y aplicar los principales tipos de pruebas de software —unitarias, de integración, de manejo de errores y de usabilidad— a través de ejemplos prácticos en Python, con el propósito de fortalecer la calidad, confiabilidad y experiencia de usuario en el desarrollo de aplicaciones.

1. Pruebas Unitarias

¿Qué son?
Verifican que cada función o módulo individual del programa funcione correctamente de forma aislada.

Ventajas:

- Detectan errores de forma temprana.
- Son fáciles de automatizar.
- Aumentan la confianza al hacer cambios en el código.

Ejemplo en Python:

```python
# Función a probar
def suma(a, b):
    return a + b

# Prueba unitaria simple
def test_suma():
    assert suma(2, 3) == 5
    assert suma(-1, 1) == 0
    assert suma(0, 0) == 0

test_suma()
print("☑ Pruebas unitarias superadas")
```

Resultado: Si todas las pruebas son correctas, el programa imprime que fueron superadas.

2. Pruebas de Integración

¿Qué son?
Evalúan cómo interactúan varias partes del sistema entre sí.

Ejemplo en Python:

```
def guardar_resultado(resultado):
    with open("resultados.txt", "a", encoding="utf-8") as f:
        f.write(str(resultado) + "\n")

def leer_resultados():
    with open("resultados.txt", "r", encoding="utf-8") as f:
        return f.readlines()

# Prueba de integración
def test_integracion():
    guardar_resultado(10)
    resultados = leer_resultados()
    assert resultados[-1].strip() == "10"

test_integracion()
print("☑ Prueba de integración superada")
```

3. Pruebas de Manejo de Errores (Robustez)

¿Qué son?
Evalúan la capacidad del programa para responder ante errores sin colapsar.

Ejemplo en Python:

```
def dividir(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: No se puede dividir por cero"

# Prueba de robustez
def test_division():
```

```
    assert dividir(10, 0) == "Error: No se puede dividir por cero"
    assert dividir(10, 2) == 5.0

test_division()
print("☑ Manejo de errores correcto")
```

4. Pruebas de Usabilidad

¿Qué son?
Evalúan qué tan fácil e intuitivo es el programa para el usuario.

Ejemplo práctico:
No se automatizan fácilmente, pero se puede incluir feedback en el código.

```
def menu():
    print("1. Sumar\n2. Restar\n3. Salir")
    opcion = input("Seleccione una opción: ")
    if opcion not in ["1", "2", "3"]:
        print("⚠ Opción inválida. Intente nuevamente.")
    return opcion
```

Consejo: Validar con usuarios reales si comprenden las opciones del menú y si los mensajes de error son útiles.

Recomendaciones

- Implementa pruebas desde el inicio del proyecto.
- Usa comentarios para explicar el objetivo de cada prueba.
- Simula situaciones reales y errores comunes.
- Documenta los resultados de tus pruebas para futuras mejoras.