LABORATORIO UNIDAD II - ISO410 – V & V

🎯 Objetivo

Aplicar, de forma individual, los conceptos clave de la Unidad II mediante ejercicios breves y prácticos: verificación vs validación, inspección y walkthrough, pruebas (unitarias/integración/aceptación), un toque mínimo de métodos formales y **una prueba end-to-end con Playwright**.

✅ Ejercicio 1 — Fundamentos (verificación vs validación)

Qué hacer (paso a paso):

1. En tu respuesta del foro, crea una **tabla** de 2 columnas con 3 ejemplos por lado:
    - **Verificación** (¿estamos construyendo el producto correctamente?): p.ej., revisión de requisitos, análisis estático, pruebas unitarias.
    - **Validación** (¿estamos construyendo el producto correcto?): p.ej., UAT con usuarios, pruebas de usabilidad, revisión con negocio.
2. Añade 1–2 líneas de **reflexión**: ¿qué diferencia te pareció más útil?

Evidencia: la tabla incluida en tu post.

✅ Ejercicio 2 — Inspección breve + Walkthrough

Crea src/calculadora.py con este código intencionalmente imperfecto:

2A) Inspección (checklist mínima)

```
# src/calculadora.py

def dividir(a, b):
    # BUG: sin validación de tipos y b==0
    return a / b

def porcentaje(valor, p):
    # BUG: sin rangos; redondeo pobre
    return int(valor * p / 100)
```

1. En el foro (o en un .md que adjuntes), genera una **checklist** con 5 ítems:

- Nombres claros • Manejo de errores • Validación de entradas • Casos borde • Comentarios/docstrings

2. Revisa el archivo y lista **3 hallazgos** (ID, descripción, severidad, recomendación).

Evidencia: checklist + 3 hallazgos (texto en el post).

2B) Walkthrough (recorrido guiado)

1. En 5–7 líneas, explica cómo **recorrerías** dividir y porcentaje ante un compañero (entradas válidas/ inválidas que mostrarías y 1 mejora que propondrías).
2. Agrega 1–2 líneas de **reflexión**.

Evidencia: tu guion breve en el post.

✅ Ejercicio 3 — Pruebas unitarias e integración (pytest)

3A) Pruebas unitarias

Crea src/math_utils.py:

```
# src/math_utils.py

def suma(a, b): return a + b


def division(a, b):
    if b == 0:
        raise ZeroDivisionError("b no puede ser 0")
    return a / b


def factorial(n):
    if n < 0:
        raise ValueError("n >= 0")
    r = 1
    for i in range(2, n+1):
        r *= i
    return r
```

Crea tests/test_math_utils.py:

```
import pytest
from src.math_utils import suma, division, factorial


def test_suma():
    assert suma(2, 3) == 5


def test_division_ok():
    assert division(10, 2) == 5


def test_division_cero():
    with pytest.raises(ZeroDivisionError):
        division(1, 0)


@pytest.mark.parametrize("n, esperado", [(0,1), (1,1), (5,120)])
def test_factorial(n, esperado):
    assert factorial(n) == esperado
```

Ejecuta:

```
pytest -q
```

Evidencia: captura de terminal con tests pasando + 1 línea de reflexión.

3B) Prueba de integración (módulos colaborando)

Crea src/repositorio.py:

```
class RepoUsuarios:
    def __init__(self):
        self._data = {"ana": "activa", "luis": "inactiva"}
    def estado(self, usuario):
        return self._data.get(usuario)
```

Crea src/auth.py:

```
class AuthService:
    def __init__(self, repo):
        self.repo = repo
    def puede_login(self, usuario, password_ok):
        return self.repo.estado(usuario) == "activa" and password_ok
```

Crea tests/test_integracion.py:

```
from src.repositorio import RepoUsuarios
from src.auth import AuthService


def test_login_integra_repo_y_servicio():
    repo = RepoUsuarios()
    auth = AuthService(repo)
    assert auth.puede_login("ana", True) is True
    assert auth.puede_login("ana", False) is False
    assert auth.puede_login("luis", True) is False
    assert auth.puede_login("otro", True) is False
```

Ejecuta pytest -q.

Evidencia: captura + 1 línea (qué contrato entre módulos validaste).

✅ **Ejercicio 4 — Aceptación (UAT rápida, manual)**

1. Redacta una **User Story**: “Como usuario, quiero registrarme con un email válido para poder iniciar sesión”.
2. Define **3 criterios de aceptación** (ej.: email con @, password ≥ 8, mensaje de error claro).
3. Escribe 1 **caso de prueba** (pasos + resultado esperado).
4. Ejecuta el caso en una app de ejemplo (o describe simulación si no hay app real).

Evidencia: tabla del caso (Aprobado/Falló) + 1 línea de reflexión.

✅ Ejercicio 5 — Mini métodos formales (muy ligero)

1. Define proposiciones:
    - A = “cuenta activa”
    - P = “password válida”
    - L = “login permitido”
2. Especificación formal: $L \leftrightarrow (A \land P)$
3. Crea la **tabla de verdad** (4 filas para $A,P \in \{V,F\}$) y evalúa L.

Evidencia: foto o tabla escrita + 1 línea de reflexión.

✅ Ejercicio 6 — Prueba E2E con Playwright (Python)

Usaremos **pytest-playwright** con el fixture page.

6A) HTML local mínimo (calculadora)

Crea web/calculadora.html:

```
<!doctype html>
<html>
<body>
  <h1 id="titulo">Calc</h1>
  <input id="a"><input id="b">
  <button id="sumar" onclick="res.value = (+a.value) + (+b.value)">Sumar</button>
```

```
    <input id="res" readonly>
</body>
</html>
```

6B) Test Playwright

Crea tests/test_playwright_calc.py:

```
from pathlib import Path

# 'page' es provisto por pytest-playwright
def test_suma_local(page):
    url = Path("web/calculadora.html").resolve().as_uri()
    page.goto(url)
    page.fill("#a", "2")
    page.fill("#b", "3")
    page.click("#sumar")
    assert page.input_value("#res") == "5"
```

Ejecuta:

```
# en modo rápido (headless)
pytest -q --browser=chromium

# opcional: ver el navegador
pytest -q --browser=chromium --headed -k test_suma_local
```

Evidencia: captura de terminal (y si usaste --headed, una captura de la ventana con el resultado “5”).

Opcional (muy útil): Usa el grabador de Playwright para generar pasos automáticamente:
playwright codegen --target python -o steps.py https://example.com
(Si tu instalación no admite --target, igual puedes copiar los pasos que genera y adaptarlos.)

📊 **Rúbrica general**

Criterio	*Logrado*	*Medio*	*No logrado*
Aplicación técnica	Ejecuta cada ejercicio correctamente (tests pasan; Playwright funcional).	Parcial o con errores menores.	No ejecuta o falla sin corrección.
Evidencias y claridad	Entrega ordenada con capturas/código mínimos y legibles.	Evidencias incompletas o poco claras.	Falta de evidencias o confusas.
Análisis / Reflexión	Aporta ideas útiles y mejoras concretas.	Comentarios superficiales.	Sin reflexión o irrelevante.
Buenas prácticas	Código simple, nombres claros, casos borde básicos.	Mezcla de buenas y malas prácticas.	Malas prácticas evidentes.