# **¿Qué es *testing* (en una frase)?**

Testing es el conjunto de actividades para comprobar que el software hace lo que debe (y no rompe cuando cambia). Se organiza por **niveles** (unitario, integración, funcional, sistema, E2E) y por **enfoques** (cómo diseñás las pruebas).

# **Caja negra (Black-box)**

**Idea:** pruebas sin mirar el código interno; solo importan entradas, salidas y comportamiento observable.

* **Qué se prueba:** requisitos, especificaciones, casos de uso, interfaces públicas (APIs, UI).
* **Técnicas típicas:** partición de equivalencia, análisis de valores límite, pruebas de regresión, pruebas funcionales.
* **Ventajas:** detecta fallos desde la perspectiva del usuario; no exige conocer la implementación.
* **Desventajas:** puede no cubrir caminos internos; encontrar la causa raíz puede ser más difícil.
* **Ejemplo:** enviar peticiones HTTP a un endpoint y validar códigos y payloads.

# **Caja blanca (White-box)**

**Idea:** pruebas con conocimiento del código: cubrís rutas, condiciones, bucles y estructuras internas.

* **Qué se prueba:** lógica interna, condiciones, manejo de excepciones, cobertura de líneas/ramas.
* **Técnicas típicas:** pruebas unitarias, coverage (lines/branches), análisis de caminos, pruebas de mutación.
* **Ventajas:** detecta errores lógicos, te permite asegurar cobertura de ramas críticas.
* **Desventajas:** puede ser frágil a refactors (porque dependen de la implementación); no garantiza que cumpla requisitos de negocio por sí sola.
* **Ejemplo:** probar métodos privados/puentear comportamiento con mocks, validar que una condición if sea cubiertA.

# **Testeo por código (Coding tests)**

Esto normalmente se refiere a **escribir pruebas automáticas** en código (unit tests, integration tests, etc.). Herramientas: PHPUnit, Pest, Codeception.

* **Unit tests:** prueban unidades pequeñas (clases/métodos). Rápidos, aislados (uso de mocks).
* **Integration tests:** verifican que varios componentes funcionen juntos (DB real o in-memory, filesystem).
* **Functional / E2E:** simulan flujo completo (navegador con Selenium/Playwright o peticiones HTTP completas).
* **Fixtures / Setup-Teardown:** preparar y limpiar el contexto de la prueba.
* **Mocks/Stubs/Fakes:** reemplazan dependencias externas (DB, API) para aislar el código bajo prueba.

# **El *Test Pyramid* (pirámide de testing)**

* Base amplia: **Unit tests** (mucha cantidad, rápidos).
* Medio: **Integration tests** (menos, más costosos).
* Cima: **E2E / UI** (pocos, lentos, fragiles).  
   La idea: escribir primero unit tests y luego solo los E2E que realmente hacen falta.

# **Cobertura y calidad**

* **Code coverage:** porcentaje de líneas/branches ejecutadas por tests. Buen indicador, no un objetivo final (coverage 100% no garantiza ausencia de bugs).
* **Mutation testing:** introduce cambios pequeños (“mutantes”) para ver si tus tests los detectan — mide la calidad real de los tests.

**ID del Caso:** TC-001  
 **Nombre:** Validar la operación de suma con dos enteros positivos  
 **Módulo:** Calculadora  
 **Prioridad:** Alta  
 **Tipo de Prueba:** Unitario – Caja blanca (usa el método directamente)  
 **Precondiciones:**

* El objeto Calculadora debe estar instanciado.
* Los parámetros de entrada deben ser números enteros válidos.

**Datos de Entrada:**

* a = 2
* b = 3

**Pasos para Ejecutar:**

1. Crear una instancia de la clase Calculadora.
2. Invocar el método sumar(a, b) pasando los valores definidos.
3. Capturar el resultado devuelto.

**Resultado Esperado:**

* El método debe retornar el valor 5.
* No debe lanzar ninguna excepción.

**Resultado Real:**

* (Se completa al ejecutar el test).

**Estado:**

* **Pendiente** / **Aprobado** / **Fallido** (se completa después de correr la prueba).

Principales Assertions en PHPUnit

## **1. Comparación de valores**

| **Método** | **Uso** | **Ejemplo** |
| --- | --- | --- |
| assertEquals($esperado, $actual) | Compara valores (==) | $this->assertEquals(5, $calc->sumar(2,3)); |
| assertSame($esperado, $actual) | Compara valores y **tipo** (===) | $this->assertSame(5, 5); |
| assertNotEquals($noEsperado, $actual) | Verifica que no sean iguales | $this->assertNotEquals(10, $resultado); |
| assertNotSame($noEsperado, $actual) | Verifica que no sean idénticos | $this->assertNotSame('5', 5); |

## **2. Booleanos**

| **Método** | **Uso** | **Ejemplo** |
| --- | --- | --- |
| assertTrue($condición) | Debe ser true | $this->assertTrue($usuario->activo); |
| assertFalse($condición) | Debe ser false | $this->assertFalse($usuario->suspendido); |

## **3. Nulos y vacíos**

| **Método** | **Uso** | **Ejemplo** |
| --- | --- | --- |
| assertNull($valor) | Verifica que sea null | $this->assertNull($obj->getDato()); |
| assertNotNull($valor) | Verifica que NO sea null | $this->assertNotNull($db); |
| assertEmpty($valor) | Verifica que esté vacío ("", [], null, 0) | $this->assertEmpty([]); |
| assertNotEmpty($valor) | Verifica que NO esté vacío | $this->assertNotEmpty($resultado); |

## **4. Tipos de datos**

| **Método** | **Uso** | **Ejemplo** |
| --- | --- | --- |
| assertIsArray($valor) | Verifica que sea un array | $this->assertIsArray($response); |
| assertIsString($valor) | Verifica que sea string | $this->assertIsString($nombre); |
| assertIsInt($valor) | Verifica que sea int | $this->assertIsInt($edad); |
| assertIsFloat($valor) | Verifica que sea float | $this->assertIsFloat($precio); |
| assertInstanceOf(Clase::class, $obj) | Verifica que sea instancia de la clase | $this->assertInstanceOf(User::class, $usuario); |
| assertNotInstanceOf(Clase::class, $obj) | Verifica que NO sea instancia | $this->assertNotInstanceOf(Exception::class, $respuesta); |

## **5. Arrays y colecciones**

| **Método** | **Uso** | **Ejemplo** |
| --- | --- | --- |
| assertCount($esperado, $array) | Verifica cantidad de elementos | $this->assertCount(3, $resultado); |
| assertContains($valor, $array) | Verifica que el valor esté en el array | $this->assertContains('admin', $roles); |
| assertNotContains($valor, $array) | Verifica que el valor NO esté | $this->assertNotContains('banned', $tags); |
| assertArrayHasKey($key, $array) | Verifica que exista la clave | $this->assertArrayHasKey('email', $usuario); |
| assertArrayNotHasKey($key, $array) | Verifica que NO exista la clave | $this->assertArrayNotHasKey('password', $publicData); |

## **6. Strings**

| **Método** | **Uso** | **Ejemplo** |
| --- | --- | --- |
| assertStringContainsString($needle, $haystack) | Busca substring | $this->assertStringContainsString('OK', $respuesta); |
| assertStringNotContainsString($needle, $haystack) | Verifica que NO esté | $this->assertStringNotContainsString('Error', $log); |
| assertMatchesRegularExpression($regex, $string) | Valida regex | $this->assertMatchesRegularExpression('/^user\_\d+$/', $username); |

## **7. Excepciones y errores**

| **Método** | **Uso** | **Ejemplo** |
| --- | --- | --- |
| expectException(Clase::class) | Espera una excepción | $this->expectException(InvalidArgumentException::class); |
| expectExceptionMessage('texto') | Verifica mensaje de excepción | $this->expectExceptionMessage('No se puede dividir entre cero'); |
| expectExceptionCode($codigo) | Verifica código de excepción | $this->expectExceptionCode(400); |

## **8. Varios**

| **Método** | **Uso** | **Ejemplo** |
| --- | --- | --- |
| assertGreaterThan($min, $valor) | Mayor que | $this->assertGreaterThan(0, $total); |
| assertLessThan($max, $valor) | Menor que | $this->assertLessThan(100, $porcentaje); |
| markTestSkipped('razón') | Salta test | $this->markTestSkipped('Funcionalidad no implementada aún'); |