UNIDAD 5: PRUEBAS Y DEPURACIÓN DE SOFTWARE
SESIÓN 17: TÉCNICAS DE PRUEBA: CAJA NEGRA Y CAJA BLANCA

INTRODUCCIÓN REALISTA: POR QUÉ PROBAR SOFTWARE ES OBLIGATORIO

En programación, muchas veces ocurre esto:
"El programa funciona en mi ordenador, así que está bien."
En el mundo profesional, esta frase **es una bandera roja**.

¿Por qué?
Porque el software:

- lo usan miles de personas
- en condiciones distintas
- con datos que tú no controlas
- en dispositivos que no conoces

Probar software no busca demostrar que funciona

- **busca demostrar que puede fallar y detectarlo antes**.

DOS FORMAS DE ENTENDER UN PROGRAMA

Todo software se puede observar desde **dos perspectivas complementarias**:

Perspectiva externa (usuario)

- No sabe cómo está hecho el programa
- Solo ve pantallas, botones, resultados
- Quiere que funcione "como se espera"

Perspectiva interna (desarrollador)

- Conoce el código
- Entiende variables, bucles, condiciones
- Le importa la estructura y la lógica

Estas dos perspectivas **no son equivalentes** y **ninguna es suficiente por sí sola**
De aquí nacen:

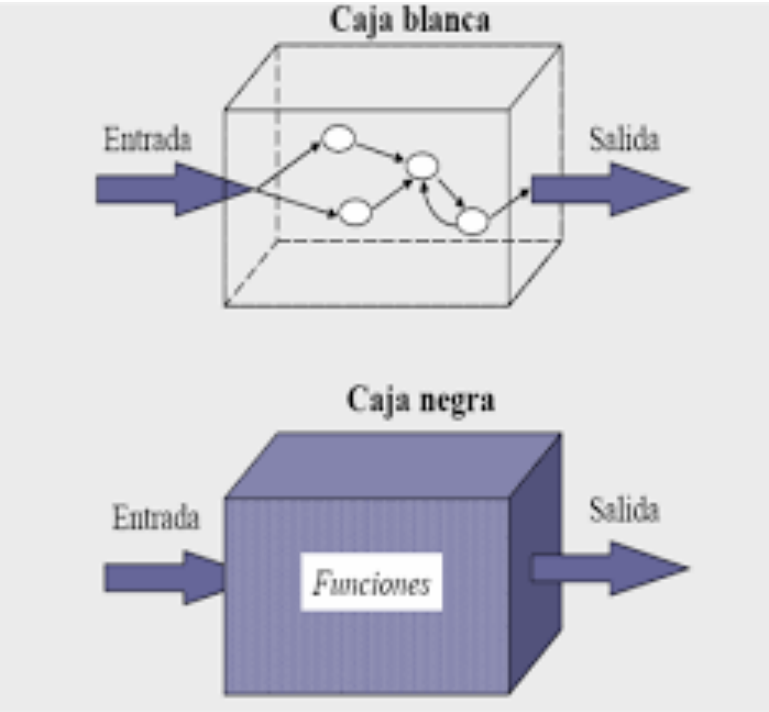
- **Caja negra**
- **Caja blanca**

¿QUÉ ES UNA "CAJA"?

En testing, "caja" significa:

Qué parte del sistema puedo ver y analizar

- Caja cerrada → no veo el interior → **caja negra**
- Caja transparente → veo el interior → **caja blanca**
-



Describe **el acceso que tiene el tester**.

PRUEBAS DE CAJA NEGRA (BLACK BOX TESTING)

Las **pruebas de caja negra** evalúan el sistema:

- sin conocer el código
- sin analizar algoritmos
- sin ver estructuras internas

El tester solo trabaja con:

- **entradas**
- **salidas**
- **comportamiento observable**

El sistema es una “caja cerrada”.

¿El sistema hace lo que el usuario espera que haga?

No pregunta:

- cómo lo hace
- si el código es elegante
- si es eficiente internamente

Solo importa: el resultado visible

Imagina una **máquina expendedora**:

- Metes una moneda
- Pulsas un botón
- Sale (o no) el producto

Tú no sabes:

- cómo funciona el motor
- qué sensores tiene
- cómo está programada

Pero sí sabes:

- si te ha dado el producto correcto

Eso es **caja negra**.

Quién realiza pruebas de caja negra

En proyectos reales:

- testers funcionales
- QA
- usuarios finales
- clientes
- incluso desarrolladores (cambiando de rol)

No hace falta saber programar para hacer caja negra, pero **sí hace falta entender el negocio y el uso real**.

Ejemplo

Formulario web de registro:

Campos:

- email
- contraseña
- botón “Registrarse”

Pruebas de caja negra:

- ¿Acepta un email válido?

- ¿Rechaza un email mal formado?
- ¿Muestra mensaje de error?
- ¿Redirige correctamente tras registrarse?

No miras:
- el controlador
- la base de datos
- la validación interna

Solo: comportamiento visible

TÉCNICAS DE CAJA NEGRA

Partición de equivalencia
No es necesario probar **todas las entradas posibles**.
Se agrupan entradas que deberían comportarse igual.

Ejemplo

Campo edad para registrarse:
- edad < 18 → NO válido
- edad ≥ 18 → VÁLIDO

Clases de equivalencia:
- grupo 1: menores de edad
- grupo 2: mayores de edad

Con probar:
- 17
- 18

Es suficiente.
Esto ahorra tiempo y es **profesional**.

Análisis de valores límite
Muchísimos errores ocurren **en los límites**.

Ejemplo
Contraseña:
- mínimo 8 caracteres

Valores a probar:
- 7
- 8
- 9

Los límites son **zonas peligrosas**.

Casos de uso
Se prueban **escenarios reales de usuario**, no funciones aisladas.

Ejemplo web real

Usuario:
1. Inicia sesión
2. Añade producto
3. Finaliza compra
4. Recibe confirmación

Si algo falla en el paso 3:
- no importa que las funciones individuales estén bien
- el sistema **falla para el usuario**

CAJA NEGRA EN DESARROLLO WEB

APIs REST

Cuando pruebas una API con Postman:
- envías una petición
- recibes una respuesta
- validas JSON, códigos HTTP

Eso es caja negra

No sabes:
- si el backend es Java, PHP o Node
- cómo está implementado

Solo importa: que la API cumpla el contrato

PRUEBAS DE CAJA BLANCA (WHITE BOX TESTING)

Las **pruebas de caja blanca** analizan:

- el código fuente
- la estructura lógica
- las rutas de ejecución

Aquí el tester:

- **sí conoce el código**
- **sí analiza el interior**

¿El programa está bien construido internamente?

No basta con que funcione:

- debe ser correcto
- mantenible
- testeable
- predecible

Analogía

Imagina un coche:

- Caja negra: el coche arranca y anda
- Caja blanca: revisas el motor, frenos, cableado

Un coche puede:

- andar hoy
- y romperse mañana

La caja blanca **previene problemas futuros**.

Quién realiza pruebas de caja blanca

- desarrolladores
- QA técnico
- equipos de arquitectura

Tú mismo con tests unitarios.

EJEMPLO TÉCNICO

Función Java:

```
public String validarUsuario(int edad, boolean suscrito) {
   if (edad < 18) {
      return "No permitido";
   } else if (suscrito) {
      return "Usuario premium";
   } else {
      return "Usuario estándar";
   }
}
```

Caja blanca analiza:

- ¿se ejecuta cada return?
- ¿qué pasa con edad = 18?
- ¿qué rutas existen?
- ¿hay condiciones redundantes?

TÉCNICAS DE CAJA BLANCA

Cobertura de sentencias

¿Cada línea de código se ha ejecutado al menos una vez?
Si una línea:

- nunca se ejecuta: es código muerto o mal diseñado

Cobertura de decisiones

Cada condición (if, while, for) debe evaluarse:

- una vez como true
- una vez como false

Si no:

- hay rutas sin probar

Cobertura de caminos

Se prueban **todas las rutas posibles** del flujo lógico.
Cuantos más if:

- más caminos
- más riesgo
- más necesidad de tests

RELACIÓN CON PRUEBAS UNITARIAS

La caja blanca se materializa en:

- JUnit
- tests unitarios
- cobertura de código

Cada test:

- ejecuta código real
- valida una ruta concreta
- documenta el comportamiento esperado

COMPARACIÓN

Aspecto	Caja Negra	Caja Blanca
Visión	Externa	Interna
Acceso al código	No	Sí
Enfoque	Funcional	Estructural
Quién la usa	Usuarios / QA	Desarrolladores
Detecta	Errores visibles	Errores lógicos
Garantiza	Validación	Verificación

Validar ≠ verificar
Ambas son necesarias

RESUMEN

- La **caja negra** valida el comportamiento externo.
- La **caja blanca** verifica la lógica interna.
- No existe software profesional sin testing.
- Cuanto antes se prueba, menos cuesta el error.