TESTING-US

VALIDÁ TU APRENDIZAJE

TEST
BUG
QA

CUADERNILLO DE ESTUDIO · NIVEL BÁSICO

Testing y QA

probar con método

Principios del testing, niveles y tipos de prueba, técnicas para diseñar casos, reporte de defectos y automatización, con ejemplos simples.

1 Fundamentos

2 Niveles y tipos

3 Técnicas

4 Casos y defectos

5 Automatización

+ Ejercicios con respuestas

Testing-US · Validá tu aprendizaje
testing-us.com

Edición octubre 2026

Antes de empezar

Probar software no es "ver si anda": es buscar, con método, la información que permite decidir si un producto está listo y qué riesgos quedan. Este cuadernillo explica los fundamentos del testing y del control de calidad con ejemplos simples: cómo se organizan las pruebas, cómo diseñar casos que encuentren defectos, cómo reportarlos para que se puedan corregir y qué lugar ocupa la automatización.

CÓMO APROVECHARLO

- Para cada técnica, inventá un ejemplo propio con un formulario que uses seguido.
- Practicá escribiendo un reporte de defecto completo de algún error que hayas visto en una app.
- Hacé los ejercicios del final de cada tema antes de mirar las respuestas.
- Cuando termines, rendí la evaluación **Testing de software y calidad (QA)** en testing-us.com.

TIP Antes de ejecutar una prueba, escribí qué esperás que pase. Si no, cualquier resultado parece correcto.

Contenido

1 **Fundamentos del testing y la calidad**
Verificación y validación, error, defecto y falla, los siete principios.

2 **Niveles y tipos de prueba**
Unitario, integración, sistema y aceptación; regresión, humo y exploratorias.

3 **Técnicas para diseñar casos de prueba**
Particiones, valores límite, tablas de decisión, estados y cobertura.

4 **Casos de prueba, planificación y defectos**
Caso de prueba, plan, reporte, severidad, prioridad y ciclo de vida.

5 **Automatización y calidad en el proceso**
Pruebas estáticas, pirámide, integración continua, TDD y BDD.

TEMA 1

1 Fundamentos del testing y la calidad

LO QUE VAS A APRENDER

- Distinguir verificación y validación.
- Distinguir error, defecto y falla.
- Conocer los siete principios del testing.
- Entender el rol de quien prueba.

Para qué se prueba

- **Probar (testing)** es evaluar un producto para encontrar defectos, comprobar que cumple lo pedido y dar información para **decidir** (por ejemplo, si se puede publicar).
- **Verificación:** ¿lo construimos **bien**? Se compara contra la especificación.
- **Validación:** ¿construimos **lo correcto**? Se compara contra las necesidades reales del usuario.
- **QA (aseguramiento de la calidad)** mira el **proceso**: busca prevenir defectos (estándares, revisiones, buenas prácticas). **El testing** es parte del **control de calidad**, que mira el **producto**.

Error, defecto y falla

- **Error (equivocación):** una acción humana equivocada, por ejemplo entender mal un requisito.
- **Defecto (bug):** la imperfección que queda en el código, en un documento o en un diseño por ese error.
- **Falla:** lo que se observa cuando el defecto se ejecuta y el sistema hace algo distinto de lo esperado.
- Un defecto puede estar en el código y **no producir una falla** hasta que se ejecute en ciertas condiciones.

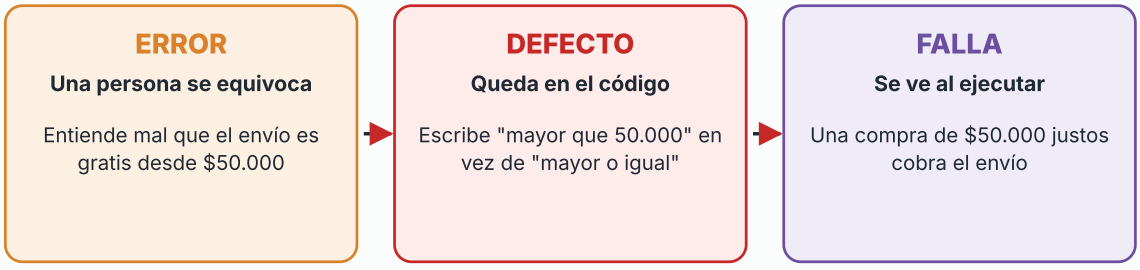


Figura 1.1 Del error humano a la falla que ve el usuario.

Los siete principios del testing

- 1. El testing muestra la **presencia** de defectos, no puede demostrar su **ausencia**.
- 2. Probar **todo** (todas las combinaciones de datos y caminos) es **imposible**: se elige qué probar según el riesgo.
- 3. **Probar temprano** ahorra tiempo y dinero: un defecto encontrado en los requisitos cuesta mucho menos que en producción.
- 4. Los defectos **se agrupan**: unos pocos módulos suelen concentrar la mayoría.
- 5. Las pruebas **se desgastan** (paradoja del pesticida): si se repiten siempre igual, dejan de encontrar defectos nuevos.
- 6. El testing **depende del contexto**: no se prueba igual una app de juegos que un sistema de un hospital.

- 7. La **falacia de la ausencia de defectos**: un sistema sin defectos conocidos puede igual no servir si no cubre lo que el usuario necesita.

TIP Probar no demuestra que no haya defectos: da información para decidir con menos riesgo.

El rol de quien prueba

- Piensa en lo que **puede salir mal**, documenta lo que encuentra con **objetividad** y comunica los problemas sin culpar a personas.
- Participa desde el principio: revisar requisitos e historias de usuario también es probar.
- La **independencia** ayuda: otra persona ve errores que el autor no ve, aunque quien programa también prueba su código.

Para practicar

1. Un sistema cumple la especificación pero no sirve a sus usuarios. ¿Falló la verificación o la validación?
2. Inventá un ejemplo de error, defecto y falla.
3. ¿Qué principio explica que conviene renovar los casos de prueba cada tanto?
4. ¿Por qué no se puede probar todo?

LAS RESPUESTAS ESTÁN AL FINAL DEL CUADERNILLO.

TEMA 2

2 Niveles y tipos de prueba

LO QUE VAS A APRENDER

- Reconocer los cuatro niveles de prueba.
- Distinguir pruebas funcionales y no funcionales.
- Diferenciar confirmación y regresión.
- Saber para qué sirven las pruebas exploratorias.

Niveles de prueba

- **Unitario (de componente):** prueba una pieza chica (una función, una clase) aislada. Las suelen escribir los propios programadores; son **rápidas** y se ejecutan muchas veces al día.
- **De integración:** prueba que los componentes o sistemas **se comuniquen bien** entre sí (por ejemplo, la app con la base de datos o con un servicio externo).
- El programa ISTQB distingue la **integración de componentes** (piezas de un mismo sistema) y la **integración de sistemas** (con otros sistemas o servicios externos).
- **De sistema:** prueba el sistema **completo** contra sus requisitos, en un entorno parecido al real.
- **De aceptación:** confirma que el sistema sirve para el usuario o el cliente y que se puede poner en uso. Incluye la aceptación del usuario (**UAT**), las pruebas **alfa** (usuarios en el lugar del desarrollo) y **beta** (usuarios en su propio entorno).

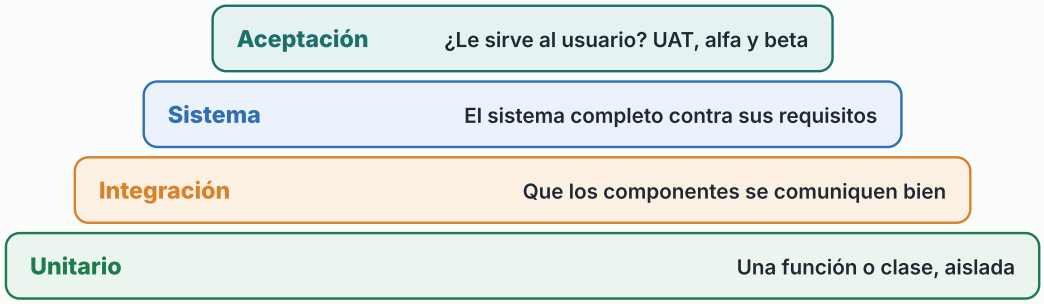


Figura 2.1 Los cuatro niveles, de lo más chico a lo más amplio.

Pruebas funcionales y no funcionales

- **Funcionales:** qué hace el sistema (por ejemplo, que calcule bien un descuento).
- **No funcionales:** cómo lo hace: **rendimiento**, **seguridad**, **usabilidad**, **accesibilidad**, **compatibilidad**, **confiabilidad**.
- **Rendimiento:** de **carga** (con el volumen de usuarios esperado) y de **estrés** (más allá del límite, para ver cómo falla).

Caja negra y caja blanca

- **Caja negra:** se diseña a partir de la especificación, sin mirar el código.
- **Caja blanca:** se diseña a partir de la estructura interna (el código, los caminos).

Pruebas relacionadas con cambios

- **Prueba de confirmación (re-test):** después de corregir un defecto, se vuelve a ejecutar la prueba que falló para confirmar que quedó resuelto.
- **Prueba de regresión:** se vuelven a probar partes que ya andaban para detectar si un cambio **rompió algo** que antes funcionaba. Es la candidata natural a automatizar.
- **Prueba de humo (smoke test):** un conjunto corto que verifica que lo esencial funciona antes de seguir probando en detalle.

OJO Confirmación: ¿se arregló lo que fallaba? Regresión: ¿el cambio rompió algo que andaba?

Pruebas exploratorias

- Se diseñan, ejecutan y aprenden **al mismo tiempo**, guiadas por la experiencia y por un objetivo (una **misión**), en sesiones con tiempo acotado.
- Complementan a las pruebas planificadas: encuentran defectos que nadie anticipó.

Para practicar

1. La app y el sistema de facturación no se entienden entre sí. ¿Qué nivel lo detecta?
2. Clasificá en funcional o no funcional: tiempo de respuesta, cálculo del IVA, contraste de colores.
3. ¿Qué diferencia hay entre una prueba alfa y una beta?
4. Se corrigió un defecto. ¿Qué dos tipos de prueba conviene hacer?

LAS RESPUESTAS ESTÁN AL FINAL DEL CUADERNILLO.

TEMA 3

3 Técnicas para diseñar casos de prueba

LO QUE VAS A APRENDER

- Armar particiones de equivalencia.
- Elegir valores límite.
- Contar las reglas de una tabla de decisión.
- Distinguir cobertura de sentencias y de ramas.

Particiones de equivalencia

- Se dividen los datos en **grupos que el sistema debería tratar igual**; alcanza con probar **un valor de cada grupo**.
- Ejemplo: un formulario acepta edades de **18 a 65**. Hay tres particiones: **menores de 18** (inválida), **de 18 a 65** (válida) y **mayores de 65** (inválida). Con 3 valores (por ejemplo 10, 30 y 70) se cubren las tres.
- Se cubren tanto las particiones **válidas** como las **inválidas**.

Análisis de valores límite

- Muchos defectos aparecen en los **bordes** (un "<" en lugar de un "<=").
- En la versión de **dos valores**, se prueba cada límite y el valor vecino de afuera. Para edades enteras de 18 a 65: **17, 18, 65 y 66**.
- En la versión de **tres valores** se suma también el vecino de adentro: 17, 18, 19, 64, 65 y 66.
- Solo se aplica a datos **ordenados** (números, fechas, cantidades).

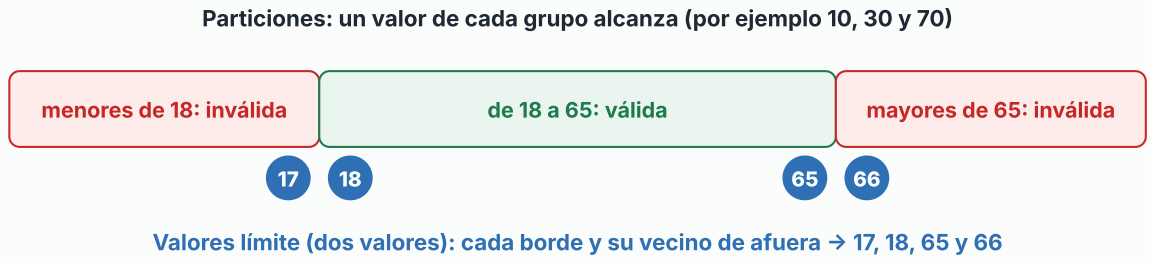


Figura 3.1 Particiones y valores límite para un campo de edad de 18 a 65.

Tablas de decisión

- Sirven cuando el resultado depende de **combinaciones de condiciones**.
- Cada columna es una **regla**: una combinación de condiciones y la acción que corresponde.
- Con condiciones de sí/no, la tabla completa tiene **2 elevado a la cantidad de condiciones**: 2 condiciones dan 4 reglas; 3 condiciones dan 8.

Transición de estados

- Se usa cuando el sistema se comporta distinto según su **estado** (por ejemplo, una cuenta: activa, bloqueada, cerrada).
- Se prueban los **cambios de estado válidos** (bloquear tras 3 contraseñas erróneas) y que los **inválidos** no ocurran.

Caja blanca: cobertura

- **Cobertura de sentencias:** porcentaje de **sentencias** (instrucciones ejecutables) que ejecutaron las pruebas.
- **Cobertura de ramas (decisiones):** porcentaje de salidas de cada decisión (verdadero y falso de un "si") ejecutadas. Cubrir todas las ramas implica cubrir todas las sentencias, pero no al revés.
- 100% de cobertura **no garantiza** que no haya defectos.

OJO Un 100% de cobertura dice qué se ejecutó, no que el sistema esté libre de defectos.

Técnicas basadas en la experiencia

- **Conjetura de errores:** pensar qué errores son típicos (campos vacíos, ceros, caracteres especiales, doble clic) y probarlos.
- **Listas de verificación (checklists)** y **pruebas exploratorias**.

Para practicar

1. Un campo acepta de 1 a 100 unidades. ¿Cuáles son las particiones?
2. Para ese campo, ¿qué valores elige el análisis de valores límite de dos valores?
3. Un envío gratis depende de 3 condiciones de sí/no. ¿Cuántas reglas tiene la tabla completa?
4. ¿Qué técnica usarías para probar un pedido que pasa por "pendiente", "pagado" y "enviado"?

LAS RESPUESTAS ESTÁN AL FINAL DEL CUADERNILLO.

TEMA 4

4 Casos de prueba, planificación y defectos

LO QUE VAS A APRENDER

- Escribir un caso de prueba completo.
- Distinguir criterios de entrada y de salida.
- Escribir un buen reporte de defecto.
- Distinguir severidad y prioridad.

El caso de prueba

- Un buen caso de prueba tiene: **identificador**, **objetivo**, **precondiciones**, **datos**, **pasos**, **resultado esperado** y, al ejecutarlo, el **resultado obtenido**.
- El resultado esperado se define **antes** de ejecutar, a partir de los requisitos: si no, cualquier cosa parece correcta.
- Un caso debe poder repetirlo **otra persona** y obtener lo mismo.

Planificar las pruebas

- El **plan de pruebas** dice qué se va a probar y qué no (**alcance**), con qué enfoque, quién, en qué entorno y con qué cronograma.
- **Criterios de entrada:** qué tiene que estar listo para empezar (por ejemplo, entorno armado y versión instalada).
- **Criterios de salida:** cuándo se considera terminado (por ejemplo, todos los casos críticos ejecutados y ningún defecto crítico abierto).
- **Testing basado en riesgo:** se prueba primero y con más profundidad lo que tiene más **probabilidad** de fallar y más **impacto** si falla.
- La **trazabilidad** vincula requisitos con casos y defectos: permite saber qué requisitos quedaron sin probar.

El reporte de defectos

- Un buen reporte tiene: **título claro**, **pasos para reproducirlo**, **resultado esperado** y **resultado obtenido**, **entorno** (versión, navegador, dispositivo), **severidad**, **prioridad** y **evidencia** (capturas, registros).
- Se describe **un defecto por reporte**, con hechos y sin juicios sobre las personas.

TIP Un buen reporte permite que otra persona reproduzca el problema sin preguntarte nada.

Severidad y prioridad

- **Severidad:** cuánto **impacto** tiene el defecto en el sistema (por ejemplo, el sistema se cae: severidad crítica).
- **Prioridad:** qué tan **urgente** es corregirlo según el negocio.
- No siempre coinciden: un error de ortografía en el logo de la portada tiene severidad **baja** pero puede tener prioridad **alta**.

Ciclo de vida de un defecto

- Estados habituales: **nuevo** → **asignado** → **corregido** → **en verificación (re-test)** → **cerrado**.
- Si al verificar sigue fallando, se **reabre**. Si no es un defecto (funciona como se pidió) puede quedar **rechazado**; si ya estaba reportado, **duplicado**.



Figura 4.1 Estados habituales de un defecto.

Para practicar

1. Mejorá este título de reporte: "no anda el login".
2. ¿Qué severidad y prioridad le darías a un error de ortografía en el botón principal de la portada?
3. Nombrá cuatro partes de un caso de prueba.
4. Un defecto corregido sigue fallando en el re-test. ¿Qué estado toma?

LAS RESPUESTAS ESTÁN AL FINAL DEL CUADERNILLO.

TEMA 5

5 Automatización y calidad en el proceso

LO QUE VAS A APRENDER

- Saber qué son las pruebas estáticas.
- Elegir qué conviene automatizar.
- Explicar la pirámide de pruebas.
- Reconocer TDD, BDD e integración continua.

Pruebas estáticas

- Se revisa el producto **sin ejecutarlo**: revisión de requisitos, revisión de código entre pares, **análisis estático** con herramientas (detectan variables sin usar, código inalcanzable, vulnerabilidades conocidas).
- Encuentran defectos **temprano**, cuando son más baratos de corregir.

Cuándo automatizar

- Conviene automatizar lo que se repite mucho: **regresión**, pruebas de humo, cálculos con muchos datos.
- La automatización **no reemplaza** el criterio humano: las pruebas exploratorias y de usabilidad siguen siendo manuales en gran parte.
- Las pruebas automatizadas son **código**: hay que mantenerlas cuando cambia el sistema.
- Una prueba **inestable (flaky)** a veces pasa y a veces falla sin cambios en el código: hay que corregirla, porque quita confianza en todo el conjunto.

TIP Automatizar una prueba mal diseñada solo hace que falle más rápido: primero diseñá bien el caso, después automatizalo.

La pirámide de pruebas

- **Base ancha:** muchas pruebas **unitarias**, rápidas y baratas.
- **Medio:** pruebas de **integración** o de servicios (API).
- **Punta:** pocas pruebas de **interfaz de punta a punta**, más lentas y frágiles.

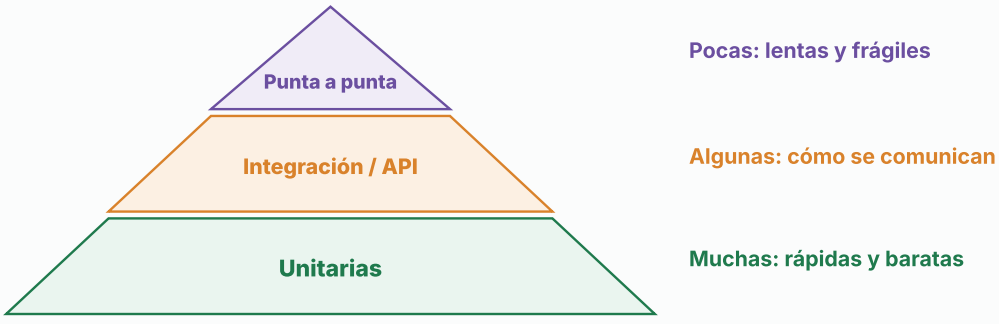


Figura 5.1 Muchas pruebas unitarias, algunas de integración y pocas de punta a punta.

Integración continua

- En la **integración continua (CI)**, cada cambio que se sube al repositorio dispara automáticamente la compilación y las pruebas.
- Si una prueba falla, el equipo lo sabe **en minutos** y corrige antes de seguir sumando cambios.

Testing en equipos ágiles

- Se prueba **dentro de cada iteración**, no al final del proyecto.
- Los **criterios de aceptación** de cada historia de usuario dicen qué condiciones tiene que cumplir para que se la acepte; la **definición de terminado** del equipo suele incluir que tenga pruebas.
- **TDD (desarrollo guiado por pruebas):** primero se escribe una prueba que falla, después el código mínimo que la hace pasar, y luego se mejora el código sin cambiar lo que hace (**refactorización**).
- **BDD:** los comportamientos se escriben en lenguaje cercano al negocio con la forma **Dado / Cuando / Entonces** (Given / When / Then).

Para practicar

1. ¿Qué conviene automatizar primero: la regresión o una prueba exploratoria nueva?
2. ¿Qué pasa en la integración continua cuando alguien sube un cambio?
3. Ordená los pasos de TDD.
4. Escribí un escenario BDD para iniciar sesión con contraseña correcta.

LAS RESPUESTAS ESTÁN AL FINAL DEL CUADERNILLO.

PARA CONTROLAR

Respuestas de los ejercicios

Tema 1

1. Falló la validación: se construyó bien, pero no lo correcto.
2. Por ejemplo: alguien entiende mal un requisito (error), escribe una condición equivocada (defecto) y el total sale mal en pantalla (falla).
3. Las pruebas se desgastan (paradoja del pesticida).
4. Porque las combinaciones de datos y caminos son prácticamente infinitas: se elige según el riesgo.

Tema 2

1. La prueba de integración.
2. Tiempo de respuesta: no funcional. Cálculo del IVA: funcional. Contraste de colores: no funcional (accesibilidad).
3. La alfa se hace con usuarios en el lugar del desarrollo; la beta, con usuarios en su propio entorno.
4. Confirmación (re-test) de lo corregido y regresión de lo que ya andaba.

Tema 3

1. Menos de 1 (inválida), de 1 a 100 (válida) y más de 100 (inválida).
2. 0, 1, 100 y 101.
3. 2 × 2 × 2 = 8 reglas.
4. Transición de estados.

Tema 4

1. Por ejemplo: "Con contraseña correcta, el inicio de sesión muestra 'usuario inexistente' en el celular".
2. Severidad baja (no afecta el funcionamiento) y prioridad alta (se ve en la portada).
3. Por ejemplo: identificador, precondiciones, pasos, datos y resultado esperado.
4. Reabierto.

Tema 5

1. La regresión, porque se repite en cada versión.
2. Se compila y se ejecutan las pruebas automáticamente; si algo falla, el equipo se entera enseguida.
3. Escribir una prueba que falla, escribir el código mínimo que la hace pasar y refactorizar.
4. Dado que tengo una cuenta activa, cuando ingreso mi usuario y mi contraseña correcta, entonces veo mi panel.

Validá lo que aprendiste

Rendí la evaluación **Testing de software y calidad (QA)** en **testing-us.com**. Es gratis: en cada intento te tocan 10 preguntas al azar, dos de cada tema de este cuadernillo. Tenés 20 minutos y aprobás con 70%.

Si aprobás, recibís un certificado con código de verificación que podés sumar a tu currículum y compartir con quien te contrate.

MIS NOTAS

© 2026 Testing-US. Material de estudio propio. Se permite imprimirlo y compartirlo sin modificaciones y sin fines comerciales.

Referencias: ISTQB Certified Tester Foundation Level (programa v4.0.1, 2024) y glosario de testing de ISTQB, usados como referencia de vocabulario; el contenido y los ejemplos son propios. Consultados en octubre de 2026. Este cuadernillo no reemplaza la certificación ISTQB.

El certificado de Testing-US acredita la aprobación de una evaluación de conocimientos. No es un título oficial ni habilita para el ejercicio profesional.