

# TESTING I

## MÓDULO I.

### CLASE I

#### Qué es el Testing

Probar si funciona todo de acuerdo a lo esperado. Un tester parte siempre de la suposición de que el programa tiene errores. El testing brinda calidad a los desarrollos. La calidad, es la satisfacción del cliente ante un producto o servicio. El testing brinda confianza. Realizar pruebas frecuentes es vital en el proceso de desarrollo. El testing sirve para:

 **Encontrar defectos y remediarlos**

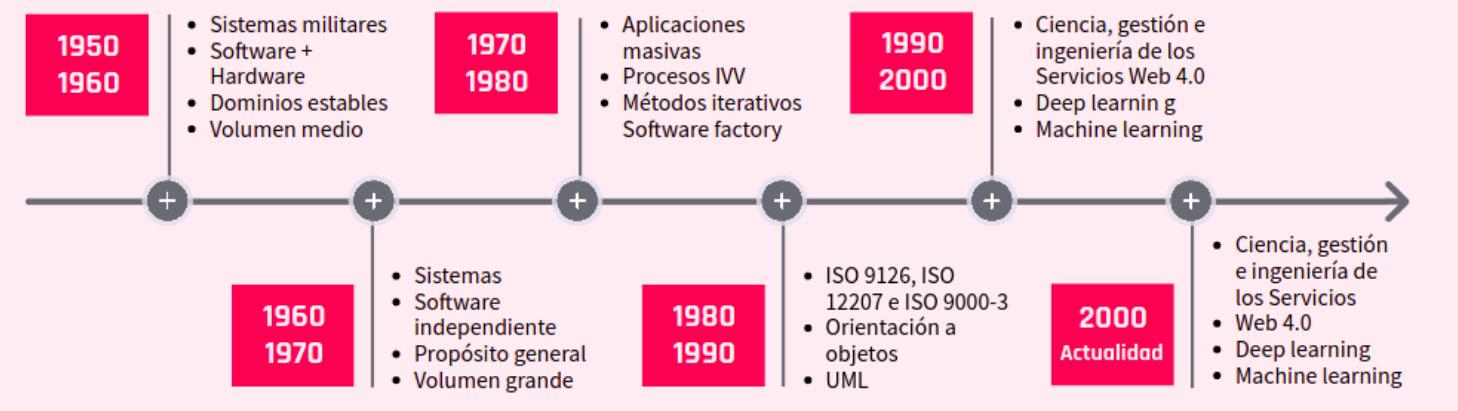
 **Asegurar el buen funcionamiento del producto**

 **Lograr un mayor grado de calidad**

#### ¿Qué es Testing?

- Probar un software es un proceso que incluye muchas actividades diferentes:
  - Ejecución de prueba y comprobación de resultados.
  - Planificar las pruebas.
  - Analizar, diseñar e implementar las pruebas.
  - Informar el avance y resultado de la ejecución de pruebas.
  - Evaluar la calidad de un objeto de prueba.

## Evolución histórica en la calidad del software



- 1950-1960: Después de la Segunda Guerra Mundial, los importantes avances en el desarrollo de software tuvieron lugar en EE. UU. y se generaron en el ámbito de la industria militar. En aquella etapa de la evolución del software, las aplicaciones eran desarrolladas para un hardware dedicado, sistemas que contaban al software como una de sus partes. La calidad asociada a estos sistemas se lograba con pruebas exhaustivas una vez terminado de construir.
- 1960-1980: Con los avances en el hardware y la aparición de lenguajes de alto nivel, se estableció una nueva tendencia en el desarrollo: se comenzaron a producir sistemas no militares e independientes del hardware. Los avances estuvieron

orientados a producir sistemas de propósito general. A partir de los inconvenientes de sobrepresupuesto y tiempo adicional necesarios en la terminación del proyecto de desarrollo del sistema operativo de IBM, se generó una alerta en el sentido de la necesidad de contar con métodos de desarrollo que garantizaran la calidad de los productos de software.

- 1980-1990: Esta alerta generó el convencimiento de la necesidad y los primeros esfuerzos en la creación de una nueva disciplina llamada ingeniería de software. Mientras esto sucedía, la tecnología seguía avanzando y se contaba con plataformas de bajo volumen y costo que ofrecían la posibilidad de desarrollar software como una oportunidad de negocio a gran escala. Luego de su paso por Japón, vuelve a EE.UU. W. Demming e introduce el siguiente concepto “la calidad de un producto está directamente relacionada al proceso utilizado para crearlo”, de esta manera, las empresas estadounidenses comienzan a adoptar la estrategia conocida como Gestión de la calidad total.
- 1990-2000: En la década de 1990, el crecimiento de los sistemas se acentuó, con protagonistas como Microsoft –ya convertida en líder mundial– Netscape y Oracle, entre otros. Además, se consolidaron las metodologías de desarrollo de tipo iterativas, las cuales van suplantando a las conocidas como cascada. Aparecieron algunas metodologías llamadas ágiles y el concepto de integración continua y también se sigue trabajando en IVV (Independent Verification Validation). Estas formas de trabajo tienen una fuerte influencia en la calidad del software.
- 2000-2010: El escenario establecido para el desarrollo de software está determinado por un hardware cada vez más poderoso, software de última generación, modelos de desarrollo y metodologías ágiles. Esta velocidad creciente impuesta por el mercado de productos de software tiene un impacto importantísimo en la calidad de los productos y servicios ofrecidos. Es de notar que estos cambios en la evolución de esta industria hicieron que la preparación de los desarrolladores de hoy día sea muy distinta a la que tenían aquellos programadores de sistemas integrados a un hardware dedicado y con requerimientos estables de los años cincuenta.
- 2010-actualidad: En esta época se afianza la integración entre la ingeniería del software y la ingeniería de sistemas destacándose el papel de los requisitos no funcionales y, sobre todo, de la seguridad; la importancia de la “ciencia, gestión e ingeniería de los servicios” que requiere un enfoque interdisciplinario –informática, marketing, gestión empresarial, derecho, entre otros– a la hora de abordar el diseño de los servicios; la necesidad de adaptar los métodos de desarrollo de software para trabajar en un “mundo abierto” –teniendo en cuenta la inteligencia ambiental, las aplicaciones conscientes del contexto, y la computación pervasiva–; los sistemas de sistemas intensivos en software (SISOS) con decenas de millones de líneas de código, decenas de interfaces externas, proveedores “competitivos”, jerarquías complejas, entre otros. También estamos viendo ya la implantación de la ingeniería del software continua, y su correspondiente tecnología y “filosofía” DevOps, que logran reducir el tiempo entre que se compromete un cambio en el sistema y se implementa en producción; lo que requiere un cambio cultural para aceptar la responsabilidad compartida –entre desarrollo y operación– de entregar software de alta calidad al usuario final.

# 7 Principios del testing



1. No puede probar que no hay defectos. Reduce la probabilidad de que queden defectos no descubiertos en el software, pero, incluso si no se encuentran, el proceso de prueba no es una demostración de corrección.
2. No es posible probar todo —todas las combinaciones de entradas y precondiciones—, excepto en casos triviales. En lugar de intentar realizar pruebas exhaustivas se deberían utilizar el análisis de riesgos, las técnicas de prueba y las prioridades para centrar los esfuerzos de prueba.
3. Para detectar defectos de forma temprana, las actividades de testing, tanto estáticas como dinámicas, deben iniciarse lo antes posible en el ciclo de vida de desarrollo de software para ayudar a reducir o eliminar cambios costosos.
4. En general, un pequeño número de módulos contiene la mayoría de los defectos descubiertos durante la prueba previa al lanzamiento o es responsable de la mayoría de los fallos operativos.
5. Si las mismas pruebas se repiten una y otra vez, eventualmente estas pruebas ya no encontrarán ningún defecto nuevo. Para detectarlo, es posible que sea necesario cambiar las pruebas y los datos de prueba existentes.
6. Por ejemplo, el software de control industrial de seguridad crítica se prueba de forma diferente a una aplicación móvil de comercio electrónico.
7. El éxito de un sistema no solo depende de encontrar errores y corregirlos hasta que desaparezcan ya que puede no haber errores, pero sí otros problemas. Existen otras variables a tener en cuenta al momento de medir el éxito.

## El rol del tester

### Aspecto psicológico del Testing

El testing es el proceso de ejecución de un programa con la intención de encontrar errores.

Los seres humanos tienden a ser sumamente orientados a objetivos y el establecimiento de la meta adecuada tiene un efecto psicológico importante. Si nuestro objetivo es demostrar que un programa no tiene errores, entonces, subconscientemente estaremos dirigidos a esa meta, es decir, tendemos a seleccionar los datos de prueba que tienen una baja probabilidad de causar que el programa falle. Por otro lado, si nuestro objetivo es demostrar que un programa tiene errores, nuestros datos de prueba tendrán una mayor probabilidad de encontrarlos.

Más allá del desarrollador o el tester, las tareas de prueba pueden ser realizadas por personas que desempeñan un rol de prueba específico u otro rol —por ejemplo, clientes—.

### Prueba independiente

La forma en que se implementa la independencia de la prueba varía dependiendo del modelo de ciclo de vida de desarrollo de software. Por ejemplo, en el desarrollo ágil, los probadores pueden formar parte de un equipo de desarrollo. En algunas organizaciones que utilizan métodos ágiles, estos probadores también pueden ser considerados parte de un equipo de prueba independiente más grande. Además, en dichas organizaciones, los propietarios de producto pueden realizar la prueba de aceptación para validar las historias de usuario al final de cada iteración.

### Beneficios potenciales de la independencia de la prueba



Es probable que los probadores independientes reconozcan diferentes tipos de fallos en comparación con los desarrolladores debido a sus diferentes contextos, perspectivas técnicas y sesgos.



Un probador independiente puede verificar, cuestionar o refutar las suposiciones hechas por los implicados durante la especificación e implementación del sistema.

### Desventajas de la independencia de la prueba



Los desarrolladores pueden perder el sentido de la responsabilidad con respecto a la calidad.



Los probadores independientes pueden ser vistos como un cuello de botella o ser culpados por los retrasos en el lanzamiento o liberación.



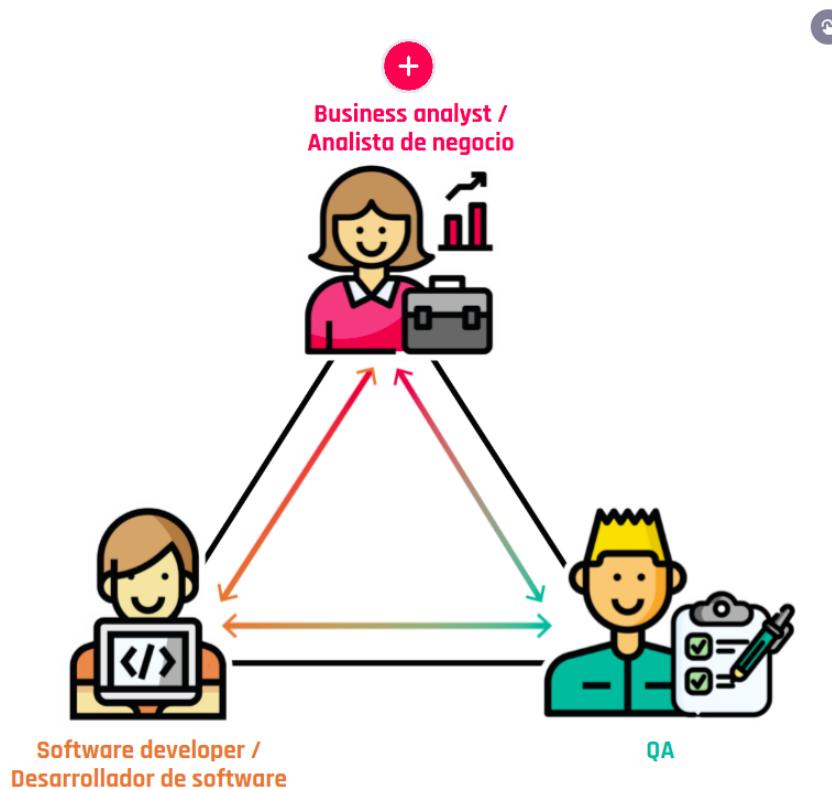
Los probadores independientes pueden carecer de información importante —por ejemplo, sobre el objeto de prueba—.

## Mesa de 3 patas

Si bien cada actor tiene un rol definido, es necesario un trabajo en comunión entre los 3 actores. Es decir, es necesario que trabajen como equipo. Por eso, utilizamos la analogía con una mesa de 3 patas, pues si falta alguna de ellas, la mesa no podría estar de pie.

En algunas empresas de software pequeñas o “start up” es posible que una misma persona tenga más de un rol.

Además, algo importante dentro de las metodologías de desarrollo ágiles, es la reunión de los 3 amigos. Es una sesión en la que participan estos tres roles y cada uno de ellos da su punto de vista respecto al software que está bajo desarrollo. Aquí, más que nunca se pone en manifiesto el funcionamiento de la mesa.



**Business analyst:** Se encarga de detectar los factores clave del negocio y es el intermediario entre el departamento de sistemas y el cliente final.

**Software developer:** Su función es diseñar, producir, programar o mantener componentes o subconjuntos de software conforme a especificaciones funcionales y técnicas para ser integrados en aplicaciones.

**QA:** La principal función es probar los sistemas informáticos para que funcionen correctamente de acuerdo a los requerimientos del cliente, documentar los errores

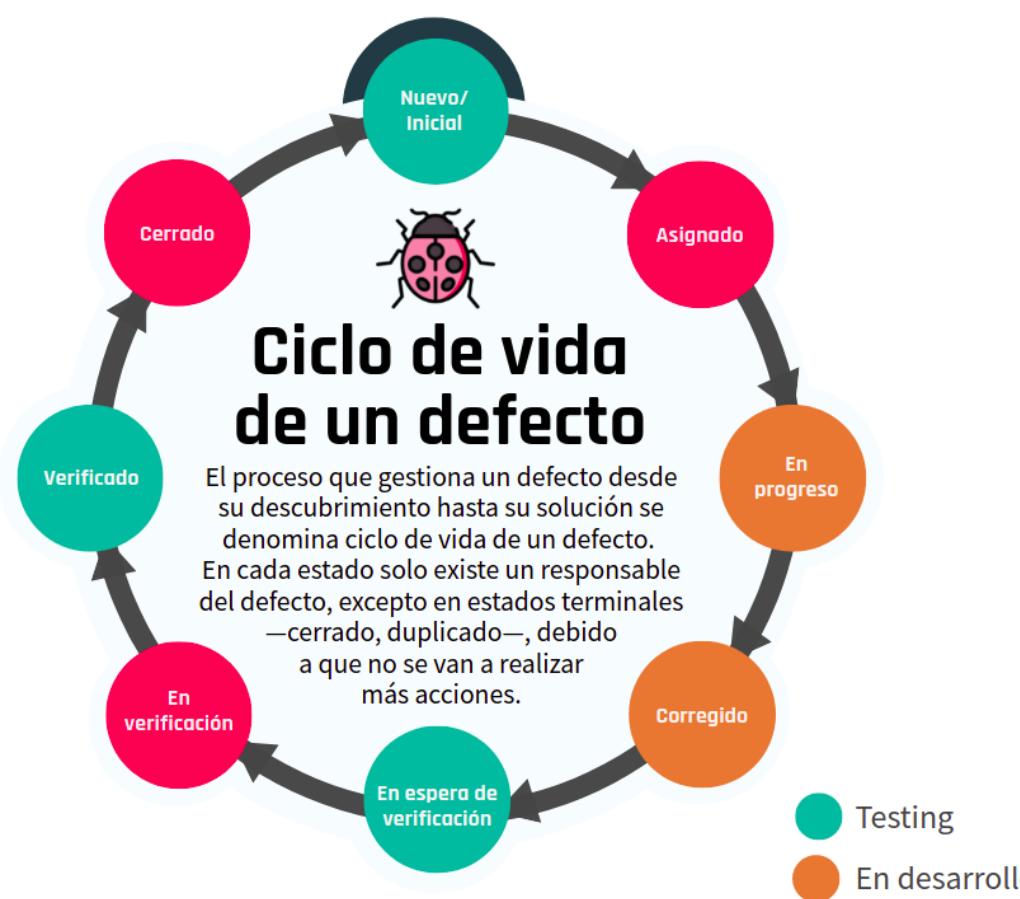
encontrados y desarrollar procedimientos de prueba para hacer un seguimiento de los problemas de los productos de forma más eficaz y eficiente.

### Qué es un defecto

Error: se produce por una equivocación de una persona. El error genera un **defecto en el software** que desencadena en un fallo del sistema al ejecutarse..



### Ciclo de vida de un defecto



**Nuevo/Inicial:** Se recopila la información y se registra el defecto.

**Asignado:** Si es un defecto válido y debe solucionarse se asigna al equipo de desarrollo, sino se puede rechazar o diferir (bug triage).

**Duplicado:** Si el defecto se repite o existe otro con una misma causa raíz.

**Devuelto o rechazado:** Se solicita más información o el receptor rechaza el defecto.

**Diferido:** el defecto no es prioritario y se solucionará en una próxima versión.

**En progreso:** Se analiza y trabaja en la solución

**Corregido:** Se realizan los cambios de código para solucionar el defecto.

**En espera de verificación:** En espera de que sea asignado a un probador. El desarrollador está a la expectativa del resultado de la verificación

**En verificación:** El probador ejecuta una prueba de confirmación.

**Reabierto:** debe contener la siguiente descripción “La prueba de confirmación indica que el defecto no se ha solucionado.”

**Verificado:** Se obtiene el resultado esperado en la prueba de confirmación.

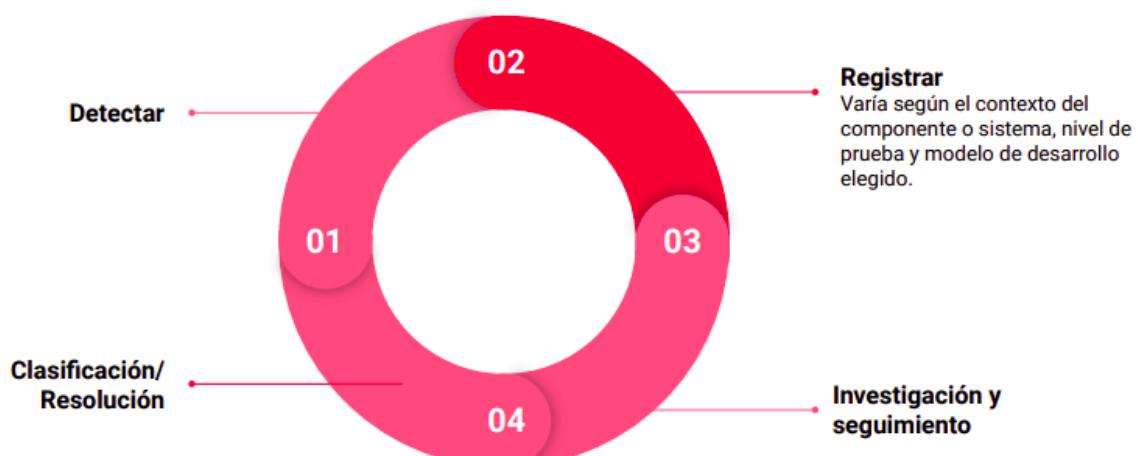
**Cerrado:** El defecto fue corregido y se encuentra disponible para el usuario final.

---

## Gestión de defectos

### 1. Proceso general

## Proceso de gestión de defectos



# Objetivos



Brindar información sobre cualquier evento adverso que haya ocurrido, para poder identificar efectos específicos, aislar el problema con una prueba de reproducción mínima y corregir los defectos potenciales.



Proporcionar a los jefes de prueba un medio para hacer un seguimiento de la calidad del producto de trabajo y del impacto en la prueba.



Dar ideas para la mejora de los procesos de desarrollo y prueba.

## 2. Escribir un informe de defectos

Si el defecto se reporta eficientemente, las probabilidades de que sea solucionado rápidamente es mayor. Entonces, la solución de un defecto dependerá de la eficiencia con que se reporte.

## ¿Qué condiciones debemos tener en cuenta?



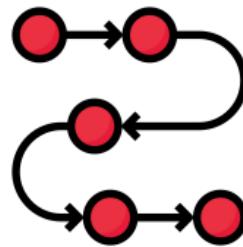
### Los bugs deben tener identificadores únicos.

Si bien muchas herramientas de bug tracking asignan automáticamente un ID único a los bugs, muchas veces se reportan fallas por medio de e-mails, saltando la registración en la herramienta.



### Una falla debe ser reproducible para reportarla.

Si el defecto no es reproducible, no es un defecto. Para defectos que ocurren en forma aislada, podemos realizarnos una nota personal para investigar luego y determinar qué condiciones deben ser dadas para que el mismo se produzca.



### **Ser específico.**

No se debe escribir suposiciones o ideas sobre lo que está ocurriendo u otra cosa que no sea la información relevante para poder reproducir el defecto.

### **Reportar cada paso realizado para reproducirlo.**

Toda la información que podamos darle al desarrollador para que pueda reproducir la falla siempre será bienvenida, no debemos obviar ningún paso que sea relevante para llegar al error en cuestión.

## **¿Cuáles son los problemas más comunes con los informes de defectos?**

- Redactar un defecto de manera excesivamente coloquial y ambigua.
- Dar solo una captura del defecto sin indicar qué se estaba haciendo cuando sucedió.
- No incluir en la descripción del defecto cuál era el resultado esperado para los pasos realizados.
- No determinar un patrón con el cual el defecto ocurre antes de reportar el mismo  
—es importante para ser directos en cuál es el problema—.
- No leer el defecto reportado siguiendo los pasos uno mismo para ver que la descripción es clara.
- No incluir información que dada las características del defecto, la misma es de relevancia.

Cuando se detecta un defecto —como parte de las pruebas estáticas—, o se observa un fallo —como parte de las pruebas dinámicas—, la persona implicada debería recopilar los datos e incluirlos en el informe de defectos. Esta información debería ser suficiente para tres fines:

Gestión del informe durante el ciclo de vida de los defectos.

Evaluación del estado del proyecto, especialmente en términos de calidad del producto y progreso de las pruebas.

Evaluación de la capacidad del proceso.

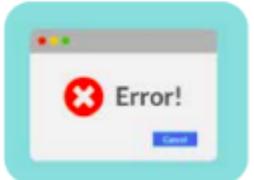
Los datos necesarios para la gestión de los informes de defectos y el estado del proyecto pueden variar en función de cuándo se detecta el defecto en el ciclo de vida, siendo la información requerida menor en etapas anteriores —por ejemplo, revisiones de requisitos y

pruebas unitarias—. No obstante, la información básica recopilada debería ser coherente durante todo el ciclo de vida e, idealmente, en todos los proyectos para permitir una comparación significativa de los datos de defectos del proceso durante el proyecto y en todos los proyectos.

## Partes de un informe de defectos

Atributo	Descripción	Ejemplo
<b>ID</b>	Abreviatura de identificador, un código único e irrepetible que puede ser número o letras.	001 - Test01
<b>Título</b>	El título debe ser corto y específico, que se entienda en este lo que queremos reportar. Cuando el desarrollador o el equipo vean el título pueden interpretar rápidamente qué es, dónde está y cuán importante es ese defecto.	Login - Ingresa con campos en blanco
<b>Descripción</b>	Describir un poco más sobre el error, es decir, desarrollar lo que dejamos afuera en el título lo podríamos explicar acá.	En la pantalla login si dejo vacío los campos nombre y password y apretó ingresar, me lleva a la página principal.
<b>Fecha del informe del defecto</b>	La fecha que detectó el defecto para saber posteriormente el tiempo en que se resolvió.	23/04/21
<b>Autor</b>	El nombre del tester que descubrió el defecto, por si el desarrollador tiene una duda, sabe a quién consultar.	Pepito Román
<b>Identificación del elemento de prueba</b>	Nombre de la aplicación o componente que estamos probando.	Carrito compras
<b>Versión</b>	Es un número que nos indica en qué versión está la aplicación.	1.0.0

Atributo	Descripción	Ejemplo
<b>Entorno</b>	El entorno en el que probamos (desarrollo, QA, producción).	Desarrollo
<b>Pasos a reproducir</b>	Los pasos a seguir para llegar al defecto encontrado.	<ul style="list-style-type: none"> <li>1) Ingresar a la aplicación.</li> <li>2) Dejar en blanco el campo nombre.</li> <li>3) Dejar en blanco el campo password.</li> <li>4) Hacer click en el botón "Ingresar".</li> </ul>
<b>Resultado esperado</b>	Es lo que esperamos que suceda o muestre la aplicación muchas veces según los requerimientos de la misma.	No debe ingresar a la aplicación sin un usuario y una contraseña válidos.
<b>Resultado obtenido o actual</b>	Es lo que sucedió realmente o lo que nos mostró la aplicación. Puede coincidir o no con el resultado esperado, si no coincide, hemos detectado un error o bug.	Ingresar a la aplicación sin usuario y sin contraseña.
<b>Severidad</b>	Cuán grave es el defecto que hemos encontrado, puede ser: bloqueado, crítico, alto, medio, bajo o trivial.	Crítico
<b>Prioridad</b>	Con esto decimos qué tan rápido se debe solucionar el defecto, puede ser: alta, media, baja	Alta
<b>Estado del defecto</b>	Los estados pueden ser: nuevo, diferido, duplicado, rechazado, asignado, en progreso, corregido, en espera de verificación, en verificación, verificado, reabierto y cerrado.	Nuevo

Atributo	Descripción	Ejemplo
<b>Referencias</b>	Link al caso de prueba con el cual encontramos el error.	<a href="https://repositorio.-com.ar/TC-001-User-Login">https://repositorio.-com.ar/TC-001-User-Login</a>
<b>Imagen</b>	Se puede adjuntar una captura de pantalla del error, esto nos permite demostrar que el error sucedió y al desarrollador lo ayuda a ubicar el error.	

### Verificación y validación

Verification is testing that your product meets the specifications / requirements you have written. "Did I build what I said I would?". Validation tests how well you addressed the business needs that caused you to write those requirements. It is also sometimes called acceptance or business testing. (also user needs)

### Casos de prueba

## Caso de Prueba

Un conjunto de pre condiciones, entradas y resultados esperados, desarrollados para impulsar la ejecución de un elemento de prueba para cumplir con los objetivos de la prueba, incluyendo la implementación correcta, la identificación de errores, el chequeo de la calidad y otras informaciones valiosas.

¿Qué es un caso de prueba? Es un documento escrito que proporciona información escrita sobre qué y cómo probar.

## ¿Qué debe contener un caso de prueba?



## Estados de un caso de prueba



\* Los estados dependen de lo definido en el Plan de Pruebas y/o de la herramienta utilizada.

## Características de un buen caso de prueba



### Deben ser simples

Se deben crear casos de prueba que sean lo más simples posibles ya que otra persona que no sea el autor puede ejecutarlos. Utilizar un lenguaje asertivo para facilitar la comprensión y que la ejecución sea más rápida.



### El título debe ser fuerte

Solo leyendo el título, cualquier probador debería comprender el objetivo del caso de prueba.



### Tener en cuenta al usuario final

El objetivo final es crear casos de prueba que cumplan con los requisitos del cliente y que sean fáciles de usar.



### No asumir

No asumir la funcionalidad y las características de la aplicación mientras se prepara el caso de prueba. Se debe ser fiel a los documentos de especificación y ante cualquier duda, hay que consultar.



### Asegurar la mayor cobertura posible

Escribir casos de prueba para todos los requisitos especificados.



### Autonomía

El caso de prueba debe generar los mismos resultados siempre, sin importar quien lo pruebe.



### Evitar la repetición de casos de prueba

Si se necesita un caso de prueba para ejecutar otro, indicar el caso de prueba por su ID.

## Ejemplo de caso de prueba

Este es un caso de prueba positivo para el login de Facebook basado en el siguiente requerimiento y pantalla:

Verificar el login en la página de Facebook para un usuario:

Id		ID-001
Título / Nombre		Login Facebook - Usuario existente
Descripción		Verificar que un usuario existente puede ingresar a Facebook.
Precondición		Contar con los siguientes datos de un usuario existente: - Correo electrónico - Contraseña
Pasos		
#	Acción	Resultado esperado
1	Ingresar a la siguiente dirección: <a href="https://www.facebook.com/">https://www.facebook.com/</a>	Se visualiza la página de login de Facebook.
2	Ingresar el correo electrónico del usuario existente en el campo "Correo electrónico o número de teléfono".	El correo electrónico se visualiza en pantalla.
3	Ingresar la contraseña en el campo "Contraseña".	Cada carácter ingresado se visualiza como un punto.
4	Presionar el botón "Iniciar Sesión".	Se visualiza la página principal de Facebook para el usuario ingresado.
Estado		Revisado
Reportado por		Usuario 1

A continuación, vemos un caso de prueba negativo para el login de Facebook basado en el

Id		ID-002
Título / Nombre		Login Facebook - Usuario no existente
Descripción		Verificar que un usuario que no existe previamente no puede ingresar a Facebook.
Precondición		Los siguientes datos de un usuario no deben existir en Facebook: - Correo electrónico - Contraseña
Pasos		
#	Acción	Resultado esperado
1	Ingresar a la siguiente dirección: <a href="https://www.facebook.com/">https://www.facebook.com/</a>	Se visualiza la página de login de Facebook.
2	Ingresar el correo electrónico de un usuario que no existe en el campo "Correo electrónico o número de teléfono".	El correo electrónico se visualiza en pantalla.
3	Ingresar la contraseña en el campo "Contraseña".	Cada carácter ingresado se visualiza como un punto.
4	Presionar el botón "Iniciar Sesión".	Se muestra el siguiente mensaje: "No encontramos ninguna cuenta que coincida exactamente con los datos que ingresaste".
Estado		Revisado
Reportado por		Usuario 1

## Testing positivo y testing negativo

### Testing positivo (+)

Son aquellos casos de prueba que validan el flujo normal de un sistema bajo prueba. Es decir, flujos que están relacionados a los requisitos funcionales del sistema bajo prueba.

### Testing negativo (-)

Son aquellos casos de prueba que validan flujos no contemplados dentro de los requisitos de un sistema bajo prueba.

## Happy path

### ¿Qué es el happy path testing?

Es el único camino con el que se prueba una aplicación a través de escenarios de prueba cuidadosamente diseñados, que deberían recorrer el mismo flujo que realiza un usuario final cuando usa la aplicación de manera regular. Generalmente es la primera forma de prueba que se realiza en una aplicación y se incluye en la categoría de prueba positiva. Su propósito no es encontrar defectos, sino ver que un producto o procedimiento funcione como ha sido diseñado.



#### Ventajas

Se utiliza para **conocer los estándares básicos** de la aplicación. Es la primera prueba que se realiza.

Se utiliza para determinar la **estabilidad de la aplicación** antes de comenzar con otros niveles de prueba.

Ayuda a identificar cualquier **problema en una etapa temprana** y a ahorrar esfuerzos posteriores.



#### Limitaciones

No garantiza la calidad del producto porque el proceso solo utiliza escenarios de prueba positivos.

Encontrar este camino único requiere un gran conocimiento del uso de la aplicación y necesidades del cliente.

## Casos de uso

Antes de realizar el diseño de los casos de prueba, lo que se debe llevar a cabo es el análisis de los documentos que van a ser la base para la generación de esos casos de prueba. Estos

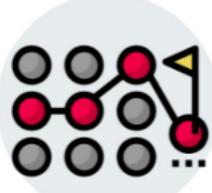
documentos van a asegurar los requisitos del cliente. Generalmente, estos requisitos se encuentran escritos como casos de uso. Un tester debe comprender qué es un caso de uso y cómo diseñar los casos de prueba a partir de estos.

## Caso de uso y caso de prueba



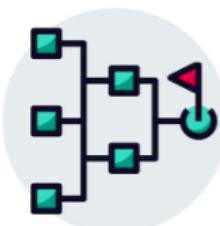
### ¿Qué es un caso de uso?

Un caso de uso cuenta la historia de cómo un usuario interactúa con un sistema de software para lograr o abandonar un objetivo. Cada caso de uso puede contener múltiples rutas que el usuario sigue, estos caminos son denominados escenario de caso de uso.



### ¿Qué es un caso de prueba?

Un caso de prueba cubre el software más en profundidad y con mayor detalle que un caso de uso. Estos incluyen todas las funciones que el programa es capaz de realizar y deben tener en cuenta el uso de todo tipo de datos de entrada/salida, cada comportamiento esperado y todos los elementos de diseño.



### ¿Cómo combinamos los casos de uso con casos de prueba?

Se puede comenzar escribiendo casos de prueba para el "escenario principal" y luego para "escenarios alternativos". Es decir, se escribe uno o más casos de prueba por cada escenario de caso de uso. Luego se puede obtener cada parte de acuerdo a la siguiente tabla:

#### Partes del caso de uso

Nombre del caso de uso

Precondiciones del caso de uso

Secuencia normal y secuencia alternativa

Resultados en la secuencia normal o alternativa. Poscondiciones del caso de uso

#### Partes del caso de prueba

Nombre del caso de prueba

Precondiciones del caso de prueba

Pasos del caso de prueba

Resultado esperado

## ¿Qué son las pruebas de casos de uso?

Es una técnica de **caja negra** donde se verifica si la ruta utilizada por el usuario está funcionando según lo esperado o no. Se pueden crear uno o más casos de prueba para cada comportamiento detallado en los casos de uso –comportamiento básico o normal, excepcionales o alternativos y de tratamiento de errores–.

La cobertura se mide de la siguiente manera:

$$\text{Cobertura} = \frac{\text{Comportamientos o rutas del caso de uso probadas}}{\text{Comportamientos o rutas del caso de uso totales}}$$

Tener en cuenta lo siguiente cuando se utiliza esta técnica de generación de pruebas a partir de casos de uso:

- Solo con las pruebas de casos de uso no se puede decidir la calidad del software.
- Incluso si es un tipo de prueba de extremo a extremo, no garantizará la cobertura completa de la aplicación del usuario.
- Los defectos pueden ser descubiertos posteriormente durante las pruebas de integración.

---

### Niveles y tipos de prueba

En toda actividad es importante conocer el marco global, es decir, tener esa visión general para no perder de vista dónde estamos y cómo seguimos. En este módulo, para lograr esta visión global, comenzaremos por conocer cómo se relacionan en forma lógica y cronológica las actividades que se desarrollan a lo largo del ciclo de vida de las pruebas de software (STLC).

Luego, se agruparán estas actividades con el fin de organizarlas y gestionarlas conjuntamente en los distintos niveles de prueba. Cada nivel de prueba es una instancia del proceso de prueba, desde componentes individuales hasta sistemas completos.

Finalmente, estas actividades de prueba serán agrupadas de acuerdo a características específicas que se necesitan probar en un sistema de software o partes de un sistema. Estos grupos de pruebas con un objetivo específico son llamados tipos de prueba.

### Ciclo de vida de las pruebas de software

No existe un proceso de prueba único y universal, pero existen actividades de prueba comunes que nos ayudan a organizarnos para alcanzar los objetivos establecidos.

### El proceso de prueba en contexto

Algunos factores de contexto que influyen en el proceso de prueba son:

- Modelo de ciclo de vida de desarrollo de software y metodologías de proyecto en uso.
- Niveles y tipos de prueba considerados.
- Riesgos de producto y de proyecto.
- Dominio del negocio.
- Restricciones operativas, incluyendo, pero no limitadas a:
  - Plazos
  - Complejidad

### Tareas principales

El ciclo de vida de las pruebas de software consiste en las siguientes actividades principales –aunque no siempre están agrupadas de esta manera en todos los proyectos de software–:



**Planificación:** En esta actividad se definen los objetivos y el enfoque de la prueba dentro de las restricciones impuestas por el contexto. Algunas **subactividades** realizadas son:

- Determinar el alcance, los objetivos y los riesgos.
- Definir el enfoque y estrategia general.
- Integrar y coordinar las actividades a realizar durante el ciclo de vida del software.
- Definir las especificaciones de técnicas, tareas de prueba adecuadas, las personas y otros recursos necesarios.
- Establecer un calendario de pruebas para cumplir con un plazo límite.

Generar el plan de prueba. **Documentos de salida:**

- Plan de prueba –general y/o por nivel de prueba–.

**Seguimiento y control:** El objetivo de esta actividad es reunir información y proporcionar retroalimentación y visibilidad sobre las actividades de prueba. Como parte del control, se pueden tomar acciones correctivas, como cambiar la prioridad de las pruebas, el calendario y reevaluar los criterios de entrada y salida. Algunas **subactividades** realizadas son:

-

Comprobar los resultados y los registros de la prueba en relación con los criterios de cobertura especificados. • Determinar si se necesitan más pruebas dependiendo del nivel de cobertura que se debe alcanzar. **Documento de salida:** • Informe de avance de la prueba.

**Análisis:** Durante esta actividad se determina “qué probar”. Algunas **subactividades** realizadas son: • Analizar la base de prueba correspondiente al nivel de prueba considerado –información de diseño e implementación, la implementación del componente o sistema en sí, informes de análisis de riesgos, etc.–. • Identificar defectos de distintos tipos en las bases de prueba –ambigüedades, omisiones, inconsistencias, inexactitudes, etc.–. • Identificar los requisitos que se van a probar y definir las condiciones de prueba para cada requisito. • Capturar la trazabilidad entre la base de prueba y las condiciones de prueba.

**Documento de salida:** • Contratos de prueba que contienen las condiciones de prueba.

**Diseño:** Durante esta actividad se determina “cómo probar”. Algunas **subactividades** realizadas son: • Diseñar y priorizar casos de prueba y conjuntos de casos de prueba de alto nivel. • Identificar los datos de prueba necesarios. • Diseñar el entorno de prueba e identificar la infraestructura y las herramientas necesarias. • Capturar la trazabilidad entre la base de prueba, las condiciones de prueba, los casos de prueba y los procedimientos de prueba. **Documento de salida:** • Casos de prueba de alto nivel diseñados y priorizados.

**Implementación:** Se completan los productos de prueba necesarios para la ejecución de la prueba, incluyendo la secuenciación de los casos de prueba en procedimientos de prueba. Algunas **subactividades** realizadas son: • Desarrollar y priorizar procedimientos de prueba. • Crear juegos de prueba (test suite) a partir de los procedimientos de prueba. • Organizar los juegos de prueba dentro de un calendario de ejecución. • Construir el entorno de prueba y verificar que se haya configurado correctamente todo lo necesario. Preparar los datos de prueba y asegurarse de que estén correctamente cargados. • Verificar y actualizar la trazabilidad entre la base de prueba, las condiciones de prueba, los casos de prueba, los procedimientos de prueba y los juegos de prueba. **Documento de salida:** • Procedimientos y datos de prueba. • Calendario de ejecución. • Test suite.

**Ejecución:** Durante esta actividad se realiza la ejecución de los casos de prueba. Algunas **subactividades** realizadas son: • Registrar los identificadores y las versiones de los elementos u objetos de prueba. • Ejecutar y registrar el resultado de las pruebas de forma manual o utilizando herramientas. • Comparar los resultados reales con los resultados

esperados. • Informar sobre los defectos en función de los fallos observados. • Repetir las actividades de prueba, ya sea como resultado de una acción tomada para una anomalía o como parte de la prueba planificada —retest o prueba de confirmación—. • Verificar y actualizar la trazabilidad entre la base de prueba, las condiciones de prueba, los casos de prueba, los procedimientos de prueba y los resultados de la prueba. **Documento de salida:** • Reporte de defectos. • Informe de ejecución de pruebas.

**Conclusión:** Algunas **subactividades** realizadas son: • Comprobar que todos los informes de defecto están cerrados. • Finalizar, archivar y almacenar el entorno de prueba, los datos de prueba, la infraestructura de prueba y otros productos de prueba para su posterior reutilización. • Traspaso de los productos de prueba a otros equipos que podrían beneficiarse con su uso. • Analizar las lecciones aprendidas de las actividades de prueba completadas. • Utilizar la información recopilada para mejorar la madurez del proceso de prueba. **Documento de salida:** • Informe resumen de prueba. • Lecciones aprendidas.



## Conclusión

Se recopilan la información de las actividades completadas y los productos de prueba. Puede ocurrir cuando un sistema de software es liberado, un proyecto de prueba es completado —o cancelado—, finaliza una iteración de un proyecto ágil, se completa un nivel de prueba o se completa la liberación de un mantenimiento.



# Niveles de pruebas

Vamos a conocer los objetivos específicos, las bases de prueba, el objeto de prueba, los defectos y fallos característicos y los enfoques y responsabilidades específicas de cada nivel de prueba.

## Prueba unitaria o de componente

## Prueba de integración

## Prueba de sistema

## Prueba de aceptación

Prueba unitaria o de componente:

### Objetivos específicos

- Reducir el riesgo.
- Verificar que los comportamientos funcionales y no funcionales del componente son los diseñados y especificados.
- Generar confianza en la calidad del componente.
- Encontrar defectos en el componente.
- Prevenir la propagación de defectos a niveles de prueba superiores.

### Bases de prueba

Algunos ejemplos de productos de trabajo que se pueden utilizar como base de prueba incluyen:

- Diseño detallado.
- Código.
- Modelo de datos.
- Especificaciones de los componentes.

### Objeto de prueba

Los objetos de prueba característicos para la prueba de componente incluyen:

- Componentes, unidades o módulos.
- Código y estructuras de datos.
- Clases.
- Módulos de base de datos.

### Defectos y fallos característicos

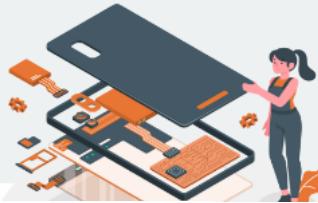
### Enfoques y responsabilidades específicas

Ejemplos de defectos y fallos característicos de la prueba de componente incluyen:

- Funcionamiento incorrecto –por ejemplo, no lo hace de la manera en que se describe en las especificaciones de diseño–.
- Problemas de flujo de datos.
- Código y lógica incorrectos.

En general, el desarrollador que escribió el código realiza la prueba de componente. Los desarrolladores pueden alternar el desarrollo de componentes con la búsqueda y corrección de defectos. A menudo, estos escriben y ejecutan pruebas después de haber escrito el código de un componente. Sin embargo, especialmente en el desarrollo ágil, la redacción de casos de prueba de componente automatizados puede preceder a la redacción del código de la aplicación.

## Prueba de integración:

Objetivos específicos	Bases de prueba	Objeto de prueba
<p>La prueba de integración se centra en las interacciones entre componentes o sistemas.</p> <ul style="list-style-type: none"> <li>• Reducir el riesgo.</li> <li>• Verificar que los comportamientos funcionales y no funcionales de las interfaces sean los diseñados y especificados.</li> <li>• Generar confianza en la calidad de las interfaces.</li> <li>• Encontrar defectos —que pueden estar en las propias interfaces o dentro de los componentes o sistemas—.</li> <li>• Prevenir la propagación de defectos a niveles de prueba superiores.</li> </ul>	<p>Algunos ejemplos de productos de trabajo que pueden utilizarse como base de prueba incluyen:</p> <ul style="list-style-type: none"> <li>• Diseño de software y sistemas.</li> <li>• Diagramas de secuencia.</li> <li>• Especificaciones de interfaz y protocolos de comunicación.</li> <li>• Casos de uso.</li> <li>• Arquitectura a nivel de componente o de sistema.</li> <li>• Flujos de trabajo.</li> <li>• Definiciones de interfaces externas.</li> </ul>	<p>Los objetos de prueba característicos para la prueba de integración incluyen:</p> <ul style="list-style-type: none"> <li>• Subsistemas.</li> <li>• Bases de datos.</li> <li>• Infraestructura.</li> <li>• Interfaces.</li> <li>• Interfaces de programación de aplicaciones –API por sus siglas en inglés–.</li> <li>• Microservicios.</li> </ul> 
Defectos y fallos característicos	Enfoques y responsabilidades específicas	
<ul style="list-style-type: none"> <li>• Datos incorrectos, datos faltantes o codificación incorrecta de datos.</li> <li>• Secuenciación o sincronización incorrecta de las llamadas a la interfaz.</li> <li>• Incompatibilidad de la interfaz.</li> <li>• Fallos en la comunicación entre componentes.</li> <li>• Fallos de comunicación entre componentes no tratados o tratados de forma incorrecta.</li> <li>• Suposiciones incorrectas sobre el significado, las unidades o las fronteras de los datos que se transmiten entre componentes.</li> </ul>	<p>La prueba de integración debe concentrarse en la integración propiamente dicha. Se puede utilizar los tipos de prueba funcional, no funcional y estructural. En general es responsabilidad de los testers.</p> 	

## Prueba de sistema:

Objetivos específicos	Bases de prueba	Objeto de prueba
<ul style="list-style-type: none"> <li>• Reducir el riesgo.</li> <li>• Verificar que los comportamientos funcionales y no funcionales del sistema son los diseñados y especificados.</li> <li>• Validar que el sistema está completo y que funcionará como se espera.</li> <li>• Generar confianza en la calidad del sistema considerado como un todo.</li> <li>• Encontrar defectos.</li> <li>• Prevenir la propagación de defectos a niveles de prueba superiores o a producción.</li> </ul>	<p>Algunos ejemplos de productos de trabajo que se pueden utilizar como base de prueba incluyen:</p> <ul style="list-style-type: none"> <li>• Especificaciones de requisitos del sistema y del software –funcionales y no funcionales–.</li> <li>• Informes de análisis de riesgo.</li> <li>• Casos de uso.</li> <li>• Épicas e historias de usuario.</li> <li>• Modelos de comportamiento del sistema.</li> <li>• Diagramas de estado.</li> <li>• Manuales del sistema y del usuario.</li> </ul>	<ul style="list-style-type: none"> <li>• Aplicaciones.</li> <li>• Sistemas hardware/software.</li> <li>• Sistemas operativos.</li> <li>• Sistema sujeto a prueba (SSP).</li> <li>• Configuración del sistema y datos de configuración.</li> </ul> 

Defectos y fallos característicos	Enfoques y responsabilidades específicas
<ul style="list-style-type: none"> <li>• Cálculos incorrectos.</li> <li>• Comportamiento funcional o no funcional del sistema incorrecto o inesperado.</li> <li>• Control y/o flujos de datos incorrectos dentro del sistema.</li> <li>• Incapacidad para llevar a cabo, de forma adecuada y completa, las tareas funcionales extremo a extremo.</li> <li>• Fallo del sistema para operar correctamente en el/los entorno/s de producción.</li> <li>• Fallo del sistema para funcionar como se describe en los manuales del sistema y de usuario.</li> </ul>	<p>La prueba de sistema debe centrarse en el comportamiento global y extremo a extremo del sistema en su conjunto, tanto funcional como no funcional. Deben utilizar las técnicas más apropiadas para los aspectos del sistema que serán probados. Los probadores independientes, en general, llevan a cabo la prueba de sistema.</p> 

Prueba de aceptación:

Objetivos específicos	Bases de prueba	Objeto de prueba
<p>La prueba de aceptación, al igual que la prueba de sistema, se centra normalmente en el comportamiento y las capacidades de todo un sistema o producto. Los objetivos de la prueba de aceptación incluyen:</p> <ul style="list-style-type: none"> <li>• Establecer confianza en la calidad del sistema en su conjunto.</li> <li>• Validar que el sistema está completo y que funcionará como se espera.</li> <li>• Verificar que los comportamientos funcionales y no funcionales del sistema sean los especificados.</li> </ul>	<p>Entre los ejemplos de productos de trabajo que se pueden utilizar como base de prueba se encuentran:</p> <ul style="list-style-type: none"> <li>• Procesos de negocio.</li> <li>• Requisitos de usuario o de negocio.</li> <li>• Normativas, contratos legales y estándares.</li> <li>• Casos de uso.</li> <li>• Requisitos de sistema.</li> <li>• Documentación del sistema o del usuario.</li> <li>• Procedimientos de instalación.</li> <li>• Informes de análisis de riesgo.</li> </ul>	<ul style="list-style-type: none"> <li>• Sistema sujeto a prueba.</li> <li>• Configuración del sistema y datos de configuración.</li> <li>• Procesos de negocio para un sistema totalmente integrado.</li> <li>• Sistemas de recuperación y sitios críticos –para pruebas de continuidad del negocio y recuperación de desastres–.</li> <li>• Procesos operativos y de mantenimiento.</li> <li>• Formularios.</li> <li>• Informes.</li> <li>• Datos de producción existentes y transformados.</li> </ul>

Defectos y fallos característicos	Enfoques y responsabilidades específicas
<p>Entre los ejemplos de defectos característicos de cualquier forma de prueba de aceptación se encuentran:</p> <ul style="list-style-type: none"> <li>• Los flujos de trabajo del sistema no cumplen con los requisitos de negocio o de usuario.</li> <li>• Las reglas de negocio no se implementan de forma correcta.</li> <li>• El sistema no satisface los requisitos contractuales o reglamentarios.</li> <li>• Fallos no funcionales tales como vulnerabilidades de seguridad, eficiencia de rendimiento inadecuada bajo cargas elevadas o funcionamiento inadecuado en una plataforma soportada.</li> </ul>	<p>A menudo es responsabilidad de los clientes, usuarios de negocio, propietarios de producto u operadores de un sistema, y otros implicados también pueden estar involucrados. La prueba de aceptación se considera, a menudo, como el último nivel de prueba en un ciclo de vida de desarrollo secuencial.</p>

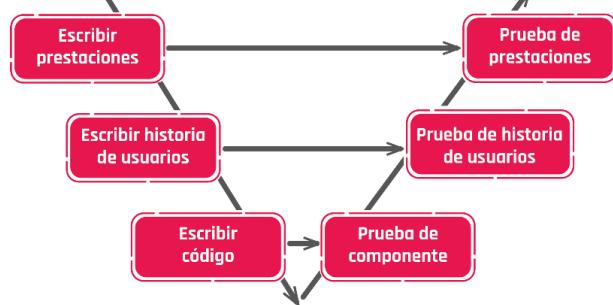


# Desde el análisis a la implementación

## Enfoque tradicional

**VS**

## Enfoque ágil



<b>1</b>	Etapas de procesos de larga duración y secuenciales.	<b>1</b>	Iteraciones cortas definidas como sprints.
<b>2</b>	Pruebas estáticas al principio sobre los documentos que son bases de prueba. Las pruebas dinámicas se realizan como una actividad al final, luego de tener la solución codificada.	<b>2</b>	Pruebas dinámicas y continuas durante la iteración. Los requerimientos son analizados como prestaciones (features) y se crean escenarios de prueba usando un enfoque de desarrollo guiado por comportamiento (BDD).
<b>3</b>	Se realiza una prueba de ambigüedad sobre los requerimientos.	<b>3</b>	Se dividen las prestaciones en historias de usuarios (user stories) y se modelan escenarios de prueba para esas historias.
<b>4</b>	Se dividen los requerimientos en historias pequeñas.	<b>4</b>	Se escriben los unit test y luego se codifica la solución usando un enfoque de desarrollo guiado por pruebas (TDD).
<b>5</b>	Se desarrolla el código en base a la documentación de diseño. Luego de tener el código se desarrollan los unit test.	<b>5</b>	Se enfoca en la ligera documentación priorizando lo que entrega valor al cliente.
<b>6</b>	Existe mayor documentación de cada proceso.	<b>6</b>	Se ejecutan pruebas continuamente y desde etapas tempranas. Toma relevancia la aplicación de integración continua (CI) y distribución o desarrollo continuo (CD).
<b>7</b>	Las pruebas se ejecutan luego de tener el código generado de todo un requerimiento o todo el sistema.	<b>7</b>	Adquieren gran protagonismo las pruebas automatizadas.
<b>8</b>	Las pruebas son generalmente manuales.	<b>8</b>	Además, la integración continua permite tener una retroalimentación activa.

## Modelo en V



## Verificación

- ¿Estamos construyendo el producto correctamente?

## Validación

- ¿Estamos construyendo el producto correcto?

### Tipos de prueba

Un tipo de prueba es un grupo de actividades de pruebas destinadas a probar las características específicas de un sistema de software, o de una parte de un sistema, basados en objetivos de pruebas específicas.

Dichos objetivos pueden incluir:

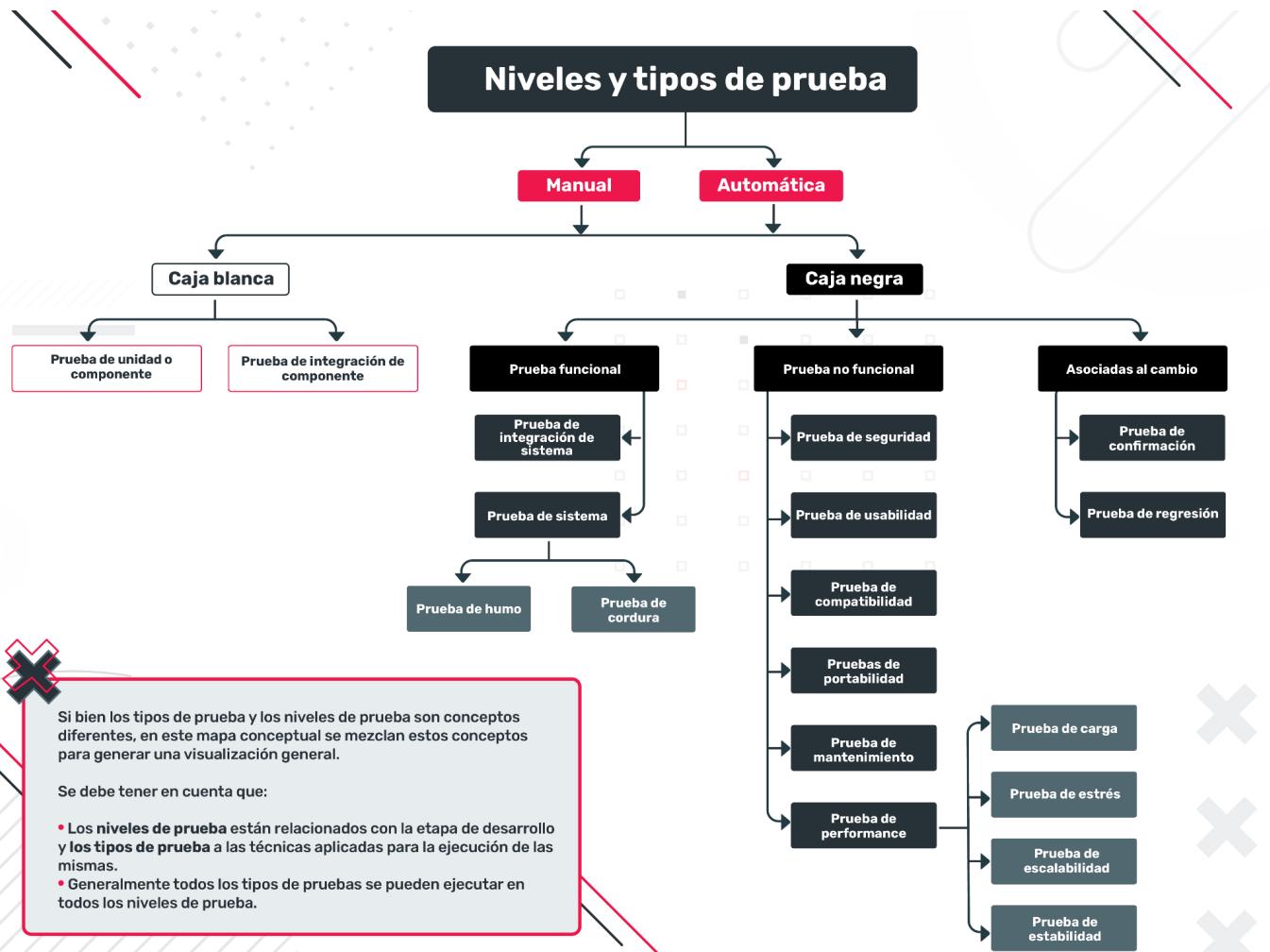
1. Evaluar las características de calidad funcional tales como la completitud, corrección y pertinencia.
2. Evaluar características no funcionales de calidad, tales como la fiabilidad, eficiencia de desempeño, seguridad, confiabilidad y usabilidad.
3. Evaluar si la estructura o arquitectura del componente o sistema es correcta, completa y según lo especificado.

4. Evaluar los efectos de los cambios, tales como confirmar que los defectos han sido corregidos (prueba de confirmación) y buscar cambios no deseados en el comportamiento que resulten de los cambios en el software o en el entorno (prueba de regresión)

	1. Prueba Funcional	2. Prueba No Funcional	3. Prueba Estructurales	4. Prueba Asociada al Cambio
<b>Definición</b>	La prueba funcional de un sistema incluye pruebas que evalúan las funciones que el sistema debe realizar. Las funciones describen <b>qué hace</b> el sistema.	La prueba no funcional prueba " <b>cómo de bien</b> " se comporta el sistema.	Estas pruebas están basadas en la estructura interna del sistema o en su implementación. La estructura interna puede incluir código, arquitectura, flujos de trabajo y/o flujos de datos dentro del sistema	<p>Existen 2 tipos de prueba relacionadas al cambio:</p> <ul style="list-style-type: none"> <li>• <b>Prueba de confirmación:</b> Una vez corregido un defecto, el software se puede probar con todos los casos de prueba que fallaron debido al defecto, que se deben volver a ejecutar en la nueva versión de software. El objetivo de una prueba de confirmación es confirmar que el defecto original se ha solucionado de forma satisfactoria.</li> <li>• <b>Prueba de regresión:</b> Es posible que un cambio hecho en una parte del código, ya sea una corrección u otro tipo de cambio, pueda afectar accidentalmente el comportamiento de otras</li> </ul>
				partes del código, ya sea dentro del mismo componente, en otros componentes del mismo sistema, o incluso en otros sistemas. La prueba de regresión implica la realización de pruebas para detectar estos efectos secundarios no deseados.
<b>Implementación</b>	La prueba funcional observa el comportamiento del software.	El diseño y ejecución de la prueba no funcional puede implicar competencias y conocimientos especiales, como el conocimiento de las debilidades inherentes a un diseño o tecnología -por ejemplo: vulnerabilidades de seguridad asociadas con determinados lenguajes de programación-.	El diseño y la ejecución de este tipo de pruebas pueden implicar competencias o conocimientos especiales, como la forma en que se construye el código, cómo se almacenan los datos, y cómo utilizar las herramientas de cobertura e interpretar correctamente sus resultados.	Especialmente en los ciclos de vida de desarrollo iterativos e incrementales (por ejemplo, Agile), las nuevas características, los cambios en las características existentes y la refactorización del código dan como resultado cambios frecuentes en el código, lo que también requiere pruebas asociadas al cambio.

<b>Niveles de Prueba</b>	Se pueden realizar pruebas funcionales en todos los niveles de prueba.	Se pueden realizar pruebas no funcionales en todos los niveles de prueba	Se puede realizar en el nivel de componente y de integración.	La prueba de confirmación y la prueba de regresión se realizan en todos los niveles de prueba
<b>Alcance</b>	Los requisitos funcionales pueden estar detallados en los siguientes documentos: especificaciones de requisitos del negocio, épicas, historias de usuarios, casos de uso y/o especificaciones funcionales.	La prueba no funcional del sistema evalúa características como la usabilidad, la eficiencia del desempeño o la seguridad.	En el nivel de prueba de integración de componentes, la prueba estructural pueden basarse en la arquitectura del sistema, como las interfaces entre componentes	
<b>Cobertura</b>	La cobertura funcional es la medida en que algún tipo de elemento funcional ha sido practicado por pruebas, y se expresa como un porcentaje del tipo o tipos de elementos cubiertos.	La cobertura no funcional es la medida en que algún tipo de elemento no funcional ha sido practicado por pruebas, y se expresa como un porcentaje del tipo o tipos de elementos cubiertos.	La cobertura estructural es la medida en que algún tipo de elemento estructural ha sido practicado mediante pruebas, y se expresa como un porcentaje del tipo de elemento cubierto.	Los juegos de prueba de regresión se ejecutan muchas veces y generalmente evolucionan lentamente, por lo que la prueba de regresión es un fuerte candidato para la automatización. La cobertura crece a medida que se agregan más funcionales al sistema por lo tanto más pruebas de regresión

## Niveles y tipos de pruebas



## **Ejemplos de tipos de prueba**

Los siguientes ejemplos están basados en una aplicación bancaria.

### **Pruebas funcionales:**

**Prueba de componente:** las pruebas se diseñan con base en la forma en que un componente debe calcular el interés a pagar por un préstamo.

**Prueba de integración de componentes:** las pruebas se diseñan en función de cómo la información de la cuenta capturada en la interfaz de usuario se transfiere a la lógica de negocio.

**Prueba de sistema:** las pruebas se diseñan sobre la base de cómo los titulares de cuentas pueden solicitar una línea de crédito sobre sus cuentas corrientes.

**Prueba de integración de sistemas:** las pruebas se diseñan en función de cómo el sistema utiliza un microservicio externo para comprobar la calificación crediticia del titular de una cuenta.

**Prueba de aceptación:** las pruebas se diseñan con base en la forma en que el empleado del banco tramita la aprobación o rechazo de una solicitud de crédito.

### **Pruebas estructurales:**

**Prueba de componente:** pruebas están diseñadas para lograr una cobertura completa de sentencia y decisión para todos los componentes que realizan cálculos financieros.

**Prueba de integración de componentes:** las pruebas están diseñadas para evaluar cómo cada pantalla de la interfaz del navegador pasa datos a la siguiente pantalla y a la lógica de negocio.

**Prueba de sistema:** las pruebas están diseñadas para cubrir las secuencias de páginas web que pueden ocurrir durante una solicitud de línea de crédito (workflow).

**Prueba de integración de sistemas:** las pruebas están diseñadas para evaluar todos los tipos de consulta posibles que se envían al microservicio de calificación crediticia.

**Prueba de aceptación:** las pruebas están diseñadas para cubrir todas las estructuras de archivos de datos financieros soportados y rangos de valores para transferencias de banco a banco.

### **Pruebas asociadas al cambio:**

**Prueba de componente:** se construyen pruebas de regresión automatizadas para cada componente y se incluyen dentro del marco de integración continua.

**Prueba de integración de componentes:** las pruebas están diseñadas para confirmar la corrección de defectos relacionados con la interfaz a medida que las correcciones se registran en el repositorio de código.

**Prueba de sistema:** las pruebas de un flujo de trabajo dado se ejecutan de nuevo si cambia alguna pantalla de ese flujo de trabajo.

**Prueba de integración de sistemas:** las pruebas de la aplicación que interactúa con el microservicio de calificación de crédito se vuelven a ejecutar diariamente como parte del despliegue continuo de ese microservicio.

**Prueba de aceptación:** todas las pruebas que han fallado previamente se vuelven a ejecutar después de que se haya corregido un defecto encontrado en la prueba de aceptación.

**Es importante ejecutar los tipos de prueba aplicables en cada nivel, especialmente en el nivel más temprano en el que tiene lugar el tipo de prueba.**

---

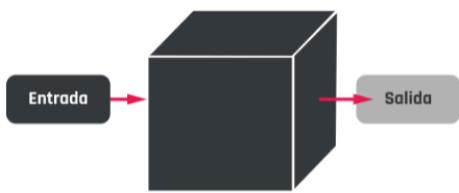
### Técnicas de prueba

Dado que la escritura, diseño e implementación de casos de prueba ocupan un lugar central dentro de un proceso de calidad, estamos descubriendo juntos todo aquello que rodea a estos conceptos.

Comprender y aplicar las técnicas de pruebas más utilizadas en el mercado nos dará una gran ventaja como testers al momento de analizar y dar cobertura a una funcionalidad dentro de un sistema que está bajo prueba.

Conocer estas técnicas de prueba será una gran ventaja en nuestro rol, ya que nos facilitará la tarea de certificar calidad dentro de un proyecto de software. Otra ventaja será el orden que nos dará al momento de escribir casos de prueba empezando desde cero, desde un caso de uso o desde una historia de usuario.

Es por ello que nos centraremos en los fundamentos teóricos escondidos dentro del diseño de casos de pruebas.



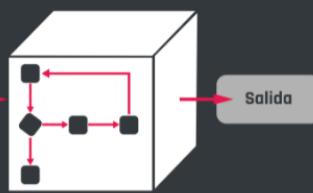
## Caja negra

Las condiciones de prueba, casos de prueba y datos de prueba se deducen de una base de prueba que puede incluir requisitos de software, especificaciones, casos de uso e historias de uso.

Los casos de prueba se pueden utilizar para detectar diferencias entre los requisitos y la implementación de los requisitos, así como desviaciones respecto a los requisitos.

La cobertura se mide en función de los elementos probados en la base de prueba y de la técnica aplicada a la base de prueba.

xx  
xx  
xx



## Caja blanca

Las condiciones de la prueba, los casos de prueba y los datos de prueba se deducen de una base de prueba que puede incluir código, la arquitectura de software, el diseño detallado o cualquier otra fuente de información relacionada a la estructura de software.

La cobertura se mide en base a los elementos probados dentro de la estructura seleccionada (por ejemplo, el código o las interfaces).

Las especificaciones se utilizan a menudo como una fuente adicional de información para detectar el resultado esperado de los casos de prueba.

xx  
xx  
xx

## 1.Categorías de técnicas de prueba

El objetivo de una técnica de prueba es ayudar a identificar las condiciones, los casos y los datos de prueba.

### Elección de una técnica de prueba

La elección de la técnica de prueba a utilizar depende de los siguientes factores:

- Tipo y complejidad del componente o sistema
- Estándares de regulación
- Requisitos del cliente o contractuales
- Clases y niveles de riesgo
- Objetivo de la prueba
- Documentación disponible
- Conocimientos y competencias del probador
- Modelo del ciclo de vida del software
- Tiempo y presupuesto

### Clasificación de las técnicas

**1.Técnicas de caja negra:** se basan en el comportamiento extraído del análisis de los documentos que son base de prueba (documentos de requisitos formales, casos de uso, historias de usuario, etc). Son aplicables tanto para pruebas funcionales como no funcionales. Se concentran en las entradas y salidas sin tener en cuenta la estructura interna.

**2.Técnicas de caja blanca:** se basan en la estructura extraída de los documentos de arquitectura, diseño detallado, estructura interna o código del sistema. Se concentran en el procesamiento dentro del objeto de prueba.

**3.Técnicas basadas en la experiencia:** aprovechan el conocimiento de desarrolladores, probadores y usuarios para diseñar, implementar y ejecutar las pruebas.

## 2.Técnicas de caja NEGRA

### Partición de equivalencia

En esta técnica se dividen los datos en particiones conocidas como clases de equivalencia donde cada miembro de estas clases o particiones es procesado de la misma manera. Las características de esta técnica son:

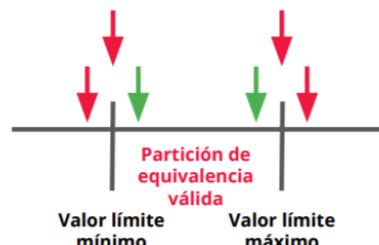
- La “partición de equivalencia válida” contiene valores que son aceptados por el componente o sistema.
- La “partición de equivalencia no válida” contiene valores que son rechazados por el componente o sistema.
- Se pueden dividir las particiones en subparticiones.
- Cada valor pertenece a solo una partición de equivalencia.
- Las particiones de equivalencia no válidas deben probarse en forma individual para evitar el enmascaramiento de fallos.
- La cobertura se mide de la siguiente manera:

$$\text{Cobertura} = \frac{\text{Particiones probadas}}{\text{Particiones identificadas}}$$

Ejemplo: colores RGB en formato String minúscula. El valor de entrada solo puede corresponder a uno de los colores RGB escrito en minúscula, << red >>, << green >>, << blue >>. Se supone que cada una de esas entradas se debería manejar de forma distinta en el programa. Por lo tanto, tendríamos tres entradas de equivalencia válidas, una para cada uno de los valores de entrada: << red >>, << green >> y << blue >>. Una clase de equivalencia inválida incluiría aquellos colores no especificados en la condición.

### Análisis de valores límites

Es una extensión de la técnica de partición de equivalencia que solo se puede usar cuando la **partición está ordenada**, y consiste en **datos numéricos o secuenciales**.



- Se deben identificar los valores límites mínimo y máximo (o valores inicial y final).
- Se pueden utilizar 2 o 3 valores límites.
- Para 2 valores límites se toma el valor que marca el límite (como valor que corresponde a la partición válida), y el valor anterior o posterior que corresponda a la partición de equivalencia inválida.
- Para 3 valores límites se toma el valor que marca el límite, un valor anterior y otro posterior a ese límite.
- La cobertura se mide de la siguiente manera:

$$\text{Cobertura} = \frac{\text{Valores límites probados}}{\text{Valores límites identificados}}$$

Ejemplo: Si una condición para un dato de entrada i especifica un rango de valores definido como  $n \leq i \leq m$ , los casos de prueba resultantes de aplicar el análisis de valores límite corresponden a  $n-1, n, n+1, m-1, m$  y  $m+1$ .

Por ejemplo, si el dato de entrada corresponde a un número que nos indica el día de la semana en el que estamos, deberíamos definir el rango de valores válidos como:  $1 \leq \text{día} \leq 7$ . En ese caso, los casos de prueba resultantes a aplicar serían 0, 1, 2, 6, 7 y 8.

### Tabla de decisión

Esta técnica se utiliza para **pruebas combinatorias**, formadas por reglas de negocio complejas que un sistema debe implementar. Las características de esta técnica son:

- Se deben identificar las condiciones (entradas) y las acciones resultantes (salidas). Estas conforman las filas de la tabla.
- Las columnas de la tabla corresponden a reglas de decisión. Cada columna forma una combinación única de condiciones y la ejecución de acciones asociadas a esa regla.
- Los valores de las condiciones y acciones pueden ser valores booleanos, discretos, numéricos o intervalos de números.
- Ayuda a identificar todas las combinaciones importantes de condiciones y a encontrar cualquier desfase en los requisitos.
- La cobertura se mide de la siguiente manera:

$$\text{Cobertura} = \frac{\text{Número de reglas de decisión probadas}}{\text{Número de reglas de decisión totales}}$$

Condiciones	Reglas							
Condición 1	S	S	S	S	N	N	N	N
Condición 2	S	S	N	N	S	S	N	N
Condición 3	S	N	S	N	S	N	S	N
Acción 1	X	X						
Acción 2				X		X		X
Acción 3			X				X	
Acción 4					X			

Ejemplo:

Cuando un cliente de la empresa paga dentro de los 30 días y la cantidad solicitada no supera el stock, se factura con descuento y se envía la mercadería solicitada. Sin embargo, si el pago se hiciera después de los 30 días, se facturará sin descuento, remitiendo la mercadería. Las mismas acciones se emprenden si se trata de un cliente nuevo. Se debe hacer lo mismo cualquiera sea el plazo de pago.

Si no existe cantidad suficiente de stock y se trata de un cliente de la empresa que paga dentro de los 30 días, facturar con descuento, realizando la entrega de la cantidad de stock y dejar pendiente el resto del pedido. Si el cliente fuera nuevo, no practicar descuento alguno.

En el caso de que el pago se efectuara dentro de los 30 días, cualquiera sea el cliente, se procederá de esta última manera.

Si un cliente que compra por primera vez solicita mayor mercadería que la de stock, cualquiera sea el plazo de pago, no se le practicará descuento alguno, remitiendo la cantidad en stock y dejando pendiente la diferencia.

Separaremos las condiciones y las acciones:

- Condiciones:

- Cliente de la empresa.
- Paga dentro de los 30 días.
- Cantidad solicitada no supera el stock.
- Pago después de 30 días.
- Cliente nuevo.
- Cantidad no supera el stock.

- Cualquiera sea el plazo.
- No hay suficiente cantidad en stock.
- Cliente por primera vez.
- Mayor cantidad de mercadería que la de stock.

- Acciones:

- Facturar con descuento.
- Enviar mercadería solicitada.
- Facturar sin descuento.
- Dejar pendiente lo solicitado menos el stock.
- Enviar stock.

Normalicemos el lenguaje y construyamos la tabla:

Condiciones	1	2	3	4	5	6	7	8
Cliente de la empresa	S	S	N	N	S	N	-	N
Plazo de pago <= 30 días	S	N	N	-	S		N	-
Cantidad solicitada <= cantidad en stock	S	S	S	S	N	N	N	N
Acciones	1	2	3	4	5	6	7	8
Facturar con descuento	X	-	-	-	X	-	-	-
Facturar sin descuento	-	X	X	X	-	X	X	X
Enviar mercadería solicitada	X	X	X	X	-	-	-	-
Enviar cantidad en stock	-	-	-	-	X	X	X	X
Dejar pendiente cantidad solicitada-stock	-	-	-	-	X	X	X	X

Tipos de reglas:

En cada regla se distinguen dos partes. La primera corresponde a las condiciones y la segunda, a las acciones. En la parte de las condiciones se pueden colocar tres tipos de entradas: S (sí), N (no), - (indiferencia). La entrada S (sí) significa que la condición debe satisfacerse o que es cierto que la condición se satisface. La entrada N (no) significa que la condición no debe cumplirse o que es cierto que la condición no se cumple. El - (indiferencia) significa que no importa que la condición se cumpla o no. En la parte de reglas de las acciones hay dos tipos de entradas: X si se debe realizar esa acción y el - si no se debe realizar esa acción. Finalmente, leyendo cada una de las columnas, tendremos los potenciales casos de pruebas para esta funcionalidad.

### Transición de estados

Un diagrama de transición de estado muestra los posibles estados del software, así como la forma en que el software entra, sale y realiza las transiciones entre estados. Las características de esta técnica son:

- Una tabla de transición de estado muestra todas las transiciones válidas y las transiciones potencialmente inválidas entre estados, así como los eventos, las condiciones de guarda y las acciones resultantes para las transiciones válidas.
- Los diagramas de transición de estado, normalmente, sólo muestran las transiciones válidas y excluyen las transiciones no válidas.
- La prueba de transición de estado se utiliza para aplicaciones basadas en menús y es extensamente utilizada en la industria del software embebido. La técnica también es adecuada para modelar un escenario de negocio con estados específicos o para probar la navegación en pantalla.
- La cobertura se mide de la siguiente manera:

$$\text{Cobertura} = \frac{\text{número de estados o transiciones identificados probados}}{\text{número total de estados o transiciones identificados en el objeto de prueba}}$$

Veamos un ejemplo en donde podríamos aplicar la técnica de transición de estados:

Visitás un cajero automático y retirás \$1000. Obtenés tu dinero en efectivo. Ahora te quedás sin saldo y hacés exactamente la misma solicitud de retirar \$1000. Esta vez, el cajero automático se niega a darte el dinero por falta de saldo. Entonces, aquí la transición que causó el **cambio de estado** es la extracción anterior.

### 3. Técnicas basadas en la experiencia

### **Predicción de errores**

Esta técnica se utiliza para anticipar la ocurrencia de equivocaciones, defectos y fallos basados en el conocimiento del probador. Se crea una lista teniendo en cuenta:

- Cómo ha funcionado la aplicación en el pasado.
- Equivocaciones comunes en los desarrolladores.
- Fallos en aplicaciones relacionadas.

En base a esa lista se diseñan pruebas que expongan esos fallos y defectos.

### **Prueba exploratoria**

En esta técnica se diseñan, ejecutan, registran y evalúan de forma dinámica pruebas informales durante la ejecución de la prueba. Los resultados de estas pruebas se utilizan para aprender más sobre el funcionamiento del componente o sistema. Generalmente se utilizan para complementar otras técnicas formales o cuando las especificaciones son escasas, inadecuadas o con restricciones de tiempo.

### **Prueba basada en listas de comprobación**

En esta técnica se diseñan, implementan y ejecutan casos de prueba que cubren las condiciones que se encuentran en una lista de comprobación definida. Se crean basadas en la experiencia y conocimiento de lo que el probador cree que es importante para el usuario y se utilizan debido a la falta de casos de prueba detallados. Durante la ejecución puede haber cierta variabilidad, dependiendo de quién ejecuta la prueba y condiciones del contexto. Esto da lugar a una mayor cobertura. Se utiliza tanto en pruebas funcionales como no funcionales.

---

### **Implementación y ejecución de la prueba**

Al completarse la implementación de los cambios o nuevas funcionalidades solicitados por el cliente, y ya diseñados los casos de pruebas correspondientes, estamos en la etapa de ejecutar las pruebas como parte de la **fase de pruebas** del ciclo de desarrollo de software (SDLC).

El objetivo de la implementación de las pruebas es asegurar que se cumplen los requerimientos del usuario, comprobando que los resultados obtenidos coinciden con los esperados, al mismo tiempo que se identifican y reportan los defectos encontrados. Las tareas dentro de la ejecución se llevan a cabo en forma iterativa hasta conseguir un sistema lo más estable posible.

Durante la ejecución de las pruebas, los conjuntos de pruebas se ejecutan luego del despliegue de cambios en los ambientes de prueba como parte del desarrollo planificado dentro de un sprint. La ejecución de pruebas incluye las siguientes actividades principales:



1. Registrar los identificadores y las versiones de los elementos u objetos de prueba, las herramientas de prueba y los productos de prueba.
2. Ejecutar pruebas de forma manual o utilizando herramientas de ejecución de pruebas.
3. Comparar resultados reales con resultados esperados.
4. Analizar las anomalías para establecer sus causas probables.
5. Informar sobre los defectos en función de los fallos observados.
6. Registrar el resultado de la ejecución de la prueba.
7. Repetir las actividades de prueba, ya sea como resultado de una acción tomada para una anomalía o como parte de la prueba planificada.

#### **Suites de casos de prueba: Pruebas de humo y pruebas de regresión**

Las pruebas de humo y las pruebas de regresión son dos de las pruebas más importantes que se ejecutan a lo largo del desarrollo de un sistema. Ambas son necesarias para el funcionamiento saludable del producto en construcción y relevantes para la calidad final del producto.

Las pruebas de humo se ejecutan para evaluar la estabilidad de las compilaciones de software iniciales o desarrolladas recientemente. Las pruebas de regresión tienen la tarea

de verificar y validar las funcionalidades existentes de la aplicación después de cada modificación o en la adición de nuevas funciones.

Las pruebas de humo son previas a las de regresión. Si se encuentra algún problema durante las de humo, la compilación no se encuentra estable por lo que retorna al equipo de desarrollo hasta que lo sea. Una vez que nos encontramos en una versión estable del sistema, se llevan a cabo las pruebas de regresión sobre las funcionalidades existentes de forma exhaustiva.

**Prueba de humo o smoke test:** cubren las funcionalidades principales de un sistema. Su objetivo es verificar la estabilidad de la aplicación para continuar con pruebas mas exhaustivas. Las pruebas de humo se ejecutan en las etapas iniciales del ciclo de vida de desarrollo de software, cada vez que los desarrolladores entregan una nueva versión al equipo de testing. Controlan de forma sencilla que funcionen los flujos criticos.

**Pruebas de regresión:** son mas detalladas. Nos permiten asegurar que despues de cualquier mejora, actualización o cambio en el código, las interfaces, componentes y sistemas existentes no sufran daños. Son suites mas completas.

Primero se ejecutan las pruebas de humo y luego las de regresión.

## ¿Por qué es importante ejecutar estas pruebas?

- Las pruebas de humo confirman que las funcionalidades básicas no sufrieron fallas.
- Las pruebas de regresión ratifican que todo sigue funcionando de la misma forma.

## Pruebas estáticas y dinámicas

Las pruebas son una combinación de múltiples actividades del ciclo de vida del software relacionadas con la planificación, el diseño y la evaluación del producto de software, con el

objetivo de encontrar los defectos y determinar si el software cumple o no con los requisitos especificados.

Es por ello que en este módulo continuaremos desarrollando otros tipos de pruebas: las pruebas estáticas y las pruebas dinámicas. Estas se complementan entre sí y nos permiten entregar un software con la mejor calidad posible.

Las pruebas estáticas y dinámicas tienen el objetivo de proporcionar una evaluación de calidad de los productos de trabajo e identificar defectos en forma temprana.

### **1. Pruebas estáticas**

Conceptos básicos de la prueba estática

La prueba estática se basa en la evaluación manual de los productos de trabajo (es decir, revisiones) o en la evaluación basada en herramientas del código u otros productos de trabajo (es decir, análisis estático). Este tipo de pruebas **no requieren la ejecución del software que se está probando**.

Se utilizan este tipo de pruebas para examinar cualquier producto de trabajo, como por ejemplo:

- Especificaciones, requisitos de negocio, funcionales y de seguridad.
- Épicas, historias de usuarios y criterios de aceptación.
- Especificaciones de arquitectura y diseño.
- Código.
- Productos de prueba: planes, casos, procedimientos y guiones de prueba.
- Manuales de usuario.
- Contratos, planes de proyecto, calendarios y presupuestos

### **Ventajas de las pruebas estáticas tempranas**

Cuando se aplica al principio del ciclo de vida del desarrollo del software, la prueba estática permite la detección temprana de defectos. Esto genera una reducción de costos y tiempo de desarrollo y prueba. Por el contrario, si el defecto se encuentra luego de las pruebas dinámicas, solucionarlo va a requerir el cambio de código, realizar una prueba de confirmación y luego incluir el mismo en pruebas de regresión, además de los cambios de toda la documentación asociada.

## **Defectos encontrados con pruebas estáticas**

Algunos de los defectos encontrados con pruebas estáticas que son más fáciles y económicos de detectar y corregir son:

- Defectos en los requisitos (inconsistencias, ambigüedades, etc.).
- Defectos de diseño (estructura de base de datos ineficiente, alto acoplamiento, etc.).
- Defectos de codificación (variables con valores no definidos, código inalcanzable o duplicado, etc.).
- Desviaciones con respecto a estándares (falta de uso de estándares de codificación).
- Especificaciones de interfaz incorrectas (unidades de medida diferente, etc.).
- Vulnerabilidades de seguridad (susceptibilidad a desbordamiento de la memoria intermedia).
- Diferencias o inexactitudes en la trazabilidad o cobertura de la base de prueba (falta de pruebas para un criterio de aceptación).
- Defectos de mantenibilidad (mala reutilización de componentes, modularización inadecuada, etc.).

## **2. Pruebas dinámicas**

### **Conceptos básicos de la prueba dinámica**

Las pruebas dinámicas **requieren la ejecución del software**, componente o sistema. Se complementan con las pruebas estáticas debido a que encuentra diferentes tipos de defectos. Para la generación de casos de prueba se utilizan diferentes técnicas de caja negra, caja blanca o basadas en la experiencia de usuario.

**Durante las pruebas dinámicas se ejecuta el software utilizando un conjunto de valores de entrada y su resultado se analiza y compara con el resultado esperado.**

Las fallas más comunes encontradas con este tipo de pruebas son:

- Fallas de funcionalidad.
- Fallas de interacción entre módulos.
- Fallas de rendimiento y seguridad.

# Conclusión

Prueba estática	Prueba dinámica
Se basa en la evaluación manual mediante revisiones y análisis estático	Requieren la ejecución del software, componente o sistema
Detecta los defectos en productos de trabajo.	Detecta los defectos y fallas cuando se ejecuta el software.
Se centra en mejorar la consistencia y la calidad de los productos de trabajo.	Se centra en los comportamientos visibles desde el exterior.
El costo de solucionar un defecto es menor	El costo de solucionar un defecto es mayor

## Proceso de revisión

Dentro de las **pruebas estáticas**, una forma de detectar errores es mediante un proceso de revisión.

Las revisiones consisten en examinar cuidadosamente un producto de trabajo con el principal objetivo de encontrar y remover errores. Pueden ser realizadas por una o más personas.

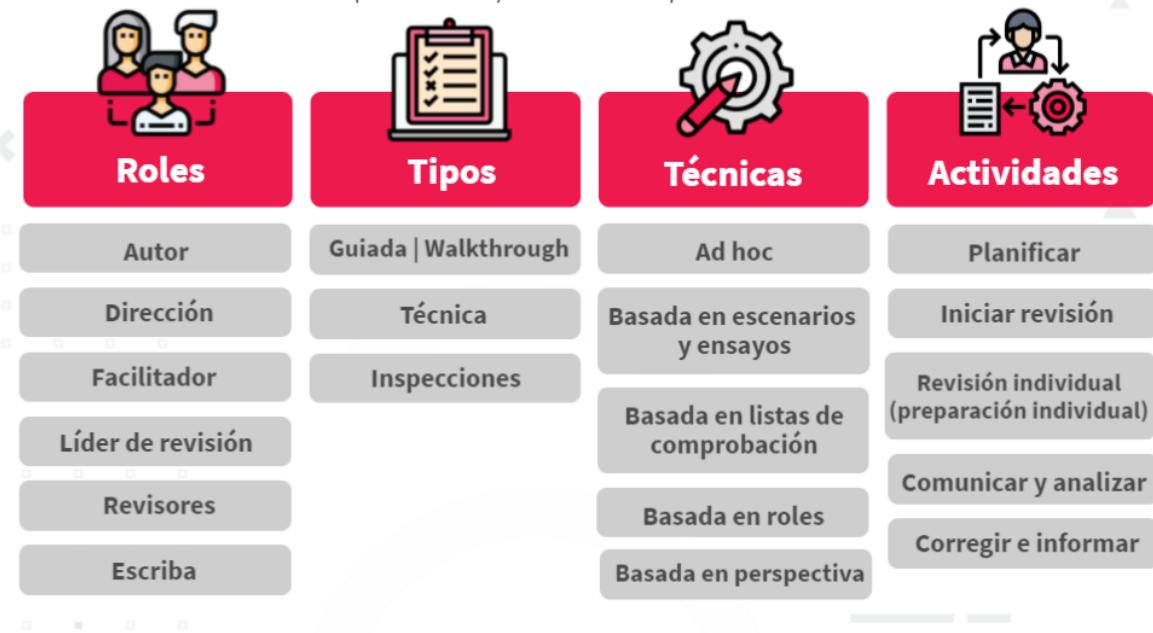
Las revisiones pueden ser:

- Revisiones formales: Tienen roles definidos, siguen un proceso establecido y deben ser documentadas.
- Revisiones Informales: No siguen un proceso definido y no son documentadas formalmente.

El grado de formalidad del proceso de revisión está relacionado con factores, como el modelo del ciclo de vida del desarrollo del software, la madurez del proceso de desarrollo, la complejidad del producto del trabajo que se debe revisar, cualquier requisito legal y/o la necesidad de un rastro de auditoría.

# Proceso de revisiones formales

Como definimos anteriormente, las revisiones pueden ser formales o informales. En el siguiente cuadro conceptual vamos a analizar cada uno de los componentes de las revisiones formales: los roles que intervienen, los tipos de revisiones formales que podemos encontrar, las técnicas de implementación y las actividades que deben incluir.



## Roles

**Autor:** Creador del producto de trabajo bajo revisión y quien corrige los defectos, en caso de ser necesario.

**Director:** Planifica y controla las revisiones.

**Facilitador:** Asegura el funcionamiento efectivo de las reuniones de revisiones y las modera.

**Líder de revisión:** Asume la responsabilidad general de la revisión; decide quiénes estarán involucrados y organiza cuándo y dónde se llevará a cabo.

**Revisores:** Identifican posibles defectos en el producto de trabajo bajo revisión; pueden representar diferentes perspectivas —por ejemplo, probador, programador, usuarios, operador, analista de negocio, experto en usabilidad—.

**Escriba:** Recopila los posibles defectos encontrados, puntos abiertos y decisiones en las revisiones.

### Tipos

**Guiada:** Dirigida por el autor del producto de trabajo, el escriba es obligatorio. El uso de listas de comprobación y la preparación individual previa son opcionales. Puede variar de informal a formal. Puede o no haber documentación de defectos potenciales.

**Técnicas:** Se realiza entre pares técnicos del autor. Si hay una reunión, debe haber una preparación individual previa. Tienen que existir un facilitador y un escriba obligatoriamente, quien idealmente no será el autor. Se elaboran registros de defectos potenciales e informes de revisión.

**Inspecciones:** Los resultados se documentan formalmente. Requieren preparación individual, tienen roles definidos: autor, director, facilitador, escriba, revisores y líder de revisión, pueden incluir también un lector dedicado. El autor no puede actuar como facilitador, líder de revisión, lector o escriba. Se hace uso de listas de comprobación. Se elaboran registros de defectos potenciales e informes de revisión.

### Técnicas

**Ad hoc:** Los revisores leen el producto de trabajo de forma secuencial y a medida que van identificando los defectos, los van documentando, recibiendo poca o ninguna orientación en el proceso.

**Basadas en escenarios y ensayos:** Fundada en el uso esperado del producto de trabajo descrito en un documento, por ejemplo, en un caso de uso.

**basadas en listas de comprobación:** Los revisores detectan defectos a partir de un conjunto de preguntas basadas en defectos potenciales, producto de la experiencia del autor de la lista.

**basadas en roles:** Los revisores evalúan el producto de trabajo desde la perspectiva de usuarios experimentados, inexpertos, adultos, niños o roles específicos en la organización, como un administrador de usuarios, de sistemas o un probador del rendimiento.

**basadas en perspectiva:** Esta técnica es similar a una revisión basada en roles, los revisores adoptan los diferentes puntos de vista del usuario final, del personal de marketing, del diseñador, del probador o del personal de operaciones.

## Actividades

**Planificar:** Definir el alcance, establecer objetivos, roles, tiempo y plazos. Definir y comprobar el cumplimiento de criterios de entrada y salida para revisiones más formales.

**Iniciar revisión:** Distribuir el material e instruir a los participantes.

**Revisión individual:** Revisión del material y tomar notas de los defectos encontrados.

**Comunicar y analizar:** Comunicar defectos a los responsables.

**Corregir e informar:** Comunicar los defectos potenciales encontrados en una reunión de revisión. Elaborar informes de hallazgos, documentar las características de calidad y comprobar criterios de salida.

---

## Requisitos

Una de las revisiones que se realizan en las pruebas estáticas es examinar los requisitos del software. Pero ¿sabemos qué son los requisitos? ¿Qué tipos de requisitos existen?

Un requisito define las funciones, capacidades o atributos intrínsecos de un sistema de software, es decir, describe cómo debe comportarse un sistema. Para decir que un sistema tiene calidad deben cumplirse los requisitos funcionales y no funcionales.

### Requisitos funcionales

Definen lo que un sistema permite hacer desde el punto de vista del usuario. Estos requisitos deben estar especificados de manera explícita. Por ejemplo: "El campo de monto acepta únicamente valores numéricos con dos decimales" (pruebas funcionales y de sistema).

### **Requisitos no funcionales**

Definen condiciones de funcionamiento del sistema en el ambiente operacional. Ejemplos:

- Requisito de usabilidad: la usabilidad se define como el esfuerzo que necesita hacer un usuario para aprender, usar, ingresar datos e interpretar los resultados obtenidos de un software de aplicación (pruebas de usabilidad).
- Requisito de eficiencia: relacionado con el desempeño en cuanto al tiempo de respuesta, número de operaciones por segundo, entre otras mediciones; así como consumo de recursos de memoria, procesador y espacio en disco o red (pruebas de rendimiento, pruebas de carga, estrés y escalabilidad, pruebas de gestión de la memoria, compatibilidad e interoperabilidad).
- Requisito de disponibilidad: disposición del sistema para prestar un servicio correctamente (pruebas de disponibilidad).
- Requisito de confiabilidad: continuidad del servicio prestado por el sistema (pruebas de seguridad).
- Requisito de integridad: ausencia de alteraciones inadecuadas al sistema (pruebas de seguridad, pruebas de integridad).
- Requisito de mantenibilidad: posibilidad de realizar modificaciones o reparaciones a un proceso sin afectar la continuidad del servicio (pruebas de mantenimiento y de regresión)

## **Planificación de la prueba**

## **Organización de la prueba**

Al momento de desarrollar un software nos daremos cuenta de que cuando lo finalicemos, en realidad el trabajo no acaba ahí. Llegará el día en el que el proyecto será publicado y que el cliente lo empiece a usar. Todo debería funcionar bien, pero ¿qué pasa si luego se debe realizar un cambio imprevisto o si se visualiza un error que necesita ser corregido? No podemos hacer los cambios directamente sobre el ambiente en el que el cliente está utilizando el software, porque podríamos romperlo y dejarlo inoperativo. Es por eso que deben existir diferentes ambientes de trabajo, donde se pueda desarrollar y probar los cambios antes de que llegue al ambiente del cliente.

Entendemos como ambiente de trabajo al entorno con todos los recursos necesarios para que se pueda ejecutar un sistema.

### **¿Qué es un ambiente?**

La ejecución de las pruebas se realizan en diferentes espacios de trabajo de acuerdo a la etapa del ciclo de desarrollo en el que se encuentre el sistema en construcción o mantenimiento. Estos entornos son conocidos como ambientes. En otras palabras, hacen referencia a un servidor con ciertos recursos asignados, software y librerías instalados, su propia base de datos y una configuración determinada.

Esto nos permitiría desarrollar aplicaciones de forma segura y con entornos diferenciados para realizar la programación, realizar pruebas, compartir resultados con los clientes y permitirles realizar pruebas y prácticas; y finalmente publicar una aplicación robusta y estable.

Cada uno de los ambientes son utilizados con un fin específico y presentan ciertas ventajas sobre los otros en determinado momento del proceso de trabajo.

### **Niveles de ambientes**

Entonces, podemos decir que es conveniente distinguir los siguientes entornos:

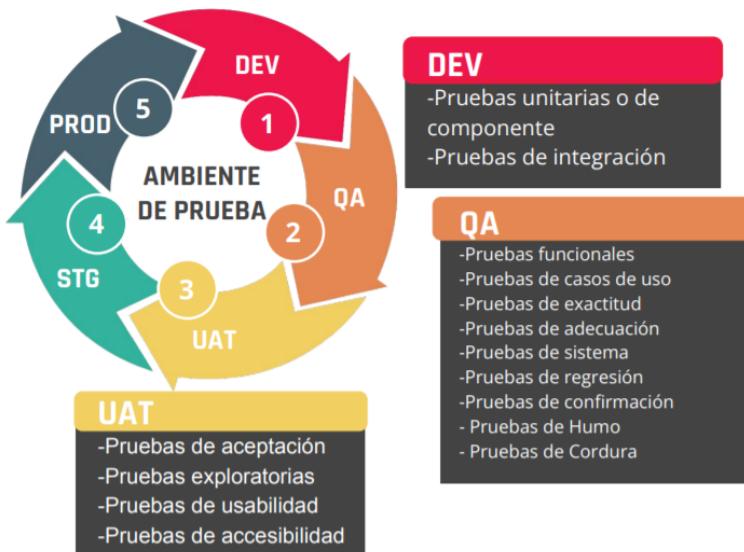
- a) Ambiente de desarrollo o DEV :este entorno es el espacio de trabajo donde el programador desarrolla el código de la aplicación, realiza pruebas iniciales y comprueba si la aplicación se ejecuta correctamente con ese código. Este ambiente puede ser local o en la nube, de acuerdo a la necesidad del proyecto.

- b) Ambiente de pruebas o QA : el entorno de pruebas suele estar ubicado en un servidor en la nube o en una granja de servidores locales (laboratorio). Permite minimizar incidencias en etapas posteriores, ya que el tester ejecutaría las primeras pruebas de funcionalidad en este ambiente.
- c) Ambiente de UAT: el entorno de UAT (o de pruebas de aceptación de usuario) permite a los usuarios del cliente poder verificar que los cambios realizados son los que realmente se solicitaron, evaluando a su vez accesibilidad y usabilidad.
- d) Ambiente de preproducción o STAGE :este entorno debería poseer una configuración técnica idéntica a la que nos encontraremos en el entorno de producción. El propósito principal de este entorno es emular al entorno de producción con el fin de probar las actualizaciones y asegurar que estas no corromperán la aplicación en los servidores en producción cuando sean desplegadas. De esta forma se minimizan las caídas del sistema y corte de los servicios en producción.
- e) Ambiente de producción o PROD :este es el entorno donde finalmente se ejecuta la aplicación, donde acceden los usuarios finales y donde se trabaja con los datos reales de negocio. Es un servidor que posee las mismas características y configuración que tendrá el servidor de preproducción. Aunque, en este caso, puede estar configurado por más de un servidor, para efectos de balanceo de carga en aplicaciones que requieren una infraestructura con capacidad de manejar un tráfico de usuarios pesado y miles de conexiones concurrentes.

**¿Qué tipos de prueba se pueden hacer en cada ambiente?**

## Tipos de prueba según ambiente

STG
-Pruebas de mantenimiento
-Pruebas de seguridad
-Pruebas de rendimiento
-Pruebas de carga, estrés y escalabilidad
-Pruebas de infraestructura
-Pruebas de gestión de la memoria, compatibilidad e interoperabilidad
-Pruebas de migración de datos



En las siguientes pantallas no desarrollaremos el ambiente PROD debido a que:

- En general, los probadores no tienen acceso a este ambiente.
- En el caso de tener acceso y realizar pruebas:
  - ◆ No se deben realizar acciones que generen datos.
  - ◆ Se corre el riesgo de ingresar datos basura.
  - ◆ Se interfiere en los datos de seguimiento.

### 1.DEV

**Pruebas unitarias o de componente:** También se conocen como pruebas de módulo. Se centra en los componentes que se pueden probar por separado. Tiene como objetivo encontrar defectos en el componente y verificar que los comportamientos funcionales y no funcionales del componente son los diseñados y especificados.

**Pruebas de Integración:** Se centra en las interacciones entre componentes o sistemas. Los objetivos de la prueba de integración incluyen encontrar defectos en las propias interfaces o dentro de los componentes o sistemas y verificar que los comportamientos funcionales y no funcionales de las interfaces sean los diseñados y especificados.

## 2.QA

**Pruebas funcionales:** Incluye pruebas que evalúan las funciones que el sistema debe realizar.

Los requisitos funcionales pueden estar descritos en productos de trabajo tales como especificaciones de requisitos de negocio, épicas, historias de usuario, casos de uso y especificaciones funcionales. También pueden estar sin documentar.

**Pruebas de casos de uso:** Proporcionan pruebas transaccionales, basadas en escenarios, que deberían emular el uso del sistema.

**Pruebas de exactitud:** Comprenden el cumplimiento por parte de la aplicación de los requisitos especificados o implícitos y también puede abarcar la exactitud de cálculo.

**Pruebas de adecuación:** Implican evaluar y validar la eficiencia de un conjunto de funciones para la consecución de las tareas especificadas previstas. Estas pruebas pueden basarse en casos de uso.

**Pruebas de sistema:** Se centra en el comportamiento y las capacidades de todo un sistema o producto, a menudo teniendo en cuenta las tareas extremo a extremo que el sistema puede realizar y los comportamientos no funcionales que exhibe mientras realiza esas tareas.

**Pruebas de regresión:** Implican la realización de pruebas para detectar efectos secundarios no deseados, luego de cambios hechos en una parte del código que puedan afectar accidentalmente el comportamiento de otras partes del código.

**Pruebas de confirmación:** Consiste en volver a ejecutar los pasos para reproducir el fallo o los fallos causados por un defecto en la nueva versión de software, una vez corregido el defecto, para así confirmar que el defecto original se ha solucionado satisfactoriamente o detectar efectos secundarios no deseados.

**Pruebas de cordura:** Es una prueba de regresión acotada que se centra en una o unas pocas áreas de funcionalidad. Se utiliza para determinar si una pequeña sección de la aplicación sigue funcionando después de un cambio menor.

**Pruebas de humo:** Se lleva a cabo un smoke test para asegurar si las funciones más importantes de un programa están trabajando correctamente, pero sin molestar con los detalles más finos.

### 3.UAT

**Pruebas de aceptación:** Se centra normalmente en el comportamiento y las capacidades de todo un sistema o producto. Además, pueden producir información para evaluar el grado de preparación del sistema para su despliegue y uso por parte del cliente (usuario final).

**Pruebas exploratorias:** Se diseñan, ejecutan, registran y evalúan de forma dinámica pruebas informales (no predefinidas) durante la ejecución de la prueba. Los resultados de la prueba se utilizan con el objetivo de aprender más sobre el componente o sistema y crear pruebas para las áreas que pueden necesitar ser probadas con mayor intensidad.

**Pruebas de usabilidad:** Evalúan la facilidad con la que los usuarios pueden utilizar o aprender a utilizar el sistema para lograr un objetivo específico en un contexto dado.

**Pruebas de accesibilidad:** Incluyen y evalúan la accesibilidad que presenta un software para aquellos con necesidades particulares o restricciones para su uso. Esto incluye a aquellos usuarios con discapacidades.

### 4.STG

**Pruebas de mantenimiento:** Se centra en probar los cambios en el sistema, así como en probar las piezas no modificadas que podrían haberse visto afectadas por los cambios. El mantenimiento puede incluir lanzamientos planificados y no planificados.

**Pruebas de seguridad:** Las pruebas de seguridad se podrían definir como el conjunto de actividades que se llevan a cabo para encontrar fallas y vulnerabilidades en el sistema, buscando disminuir el impacto de ataques y pérdida de información importante.

**Pruebas de rendimiento:** Se implementan y se ejecutan para evaluar las características relacionadas con el rendimiento del destino de la prueba, como los perfiles de tiempo, el flujo de

ejecución, los tiempos de respuesta y la fiabilidad y los límites operativos. También se pueden realizar en STG.

**Pruebas de carga, estrés y escalabilidad:** Una prueba de carga garantiza que un sistema pueda controlar un volumen de tráfico esperado. Una prueba de estrés es en la que se somete al sistema a condiciones de uso extremas para garantizar su robustez y confiabilidad. Las pruebas de escalabilidad garantizan la escalabilidad de un sistema, es decir, que pueda soportar el incremento de demanda en la operación. También se pueden realizar en QA encontrando el correspondiente escalar con respecto a un ambiente de PROD.

**Pruebas de infraestructura:** Incluyen todos los sistemas informáticos internos, los dispositivos externos asociados, las redes de Internet, la nube y las pruebas de virtualización.

**Pruebas de gestión de la memoria:** Evalúan el estado y la integridad de la memoria del sistema para identificar problemas potenciales.

**Pruebas de compatibilidad:** Incluyen las pruebas para comprobar que el sistema es compatible con todos los navegadores de Internet y todos los sistemas operativos del mercado.

**Pruebas de interoperabilidad:** Se refieren a aquellas donde se realiza la evaluación de la correcta integración entre distintos aplicativos, sistemas, servicios o procesos que conforman una plataforma o solución tecnológica.

**Pruebas de migración de datos:** Incluyen las pruebas realizadas al transferir datos entre tipos de dispositivos de almacenamiento, formatos o sistemas de cómputo.

## Ejemplo

Como parte de mejoras en Netflix, se ha solicitado al equipo de trabajo adicionar una funcionalidad para ver películas/series al mismo tiempo con tus amigos, de forma que se pueda iniciar una sesión compartida de una película o serie, invitar a tus amigos y poder inicializar o pausar el recurso. Siendo parte del equipo de desarrollo (desarrolladores y testers), vinculá los ejemplos con los tipos de prueba que se llevan a cabo a lo largo de los distintos ambientes:

### Ambiente de desarrollo o DEV

<b>Prueba unitaria</b>	Se prueba que se puedan obtener todos los usuarios para utilizarlo luego en la interfaz.
------------------------	--

### Ambiente de pruebas o QA

<b>Prueba funcional</b>	Se prueba que se permita iniciar la sesión compartida de un contenido con éxito.
<b>Prueba de regresión</b>	Se prueba que la reproducción de películas/series por usuario sin sesión compartida, continúe funcionando correctamente.
<b>Prueba de humo</b>	Se prueba que la versión desplegada para el equipo de testing es estable, realizando el login a la aplicación.
<b>Prueba de confirmación</b>	Luego de la resolución de un defecto en la invitación a usuarios se prueba su correspondiente corrección.

### Ambiente de UAT

<b>Prueba de usabilidad</b>	Se prueba que se visualice que el acceso y uso de la funcionalidad sea intuitivo.
-----------------------------	---

### Ambiente de preproducción o STAGE

<b>Prueba de seguridad</b>	Se prueba que no se permite el ingreso de usuarios no invitados a una sesión.
<b>Prueba de rendimiento</b>	Se prueba que la sesión iniciada no se caiga, a medida que se vayan incorporando usuarios.
<b>Prueba de infraestructura</b>	Se prueba que si uno de los usuarios pierde la conexión, los demás usuarios puedan continuar conectados.
<b>Prueba de compatibilidad</b>	Se prueba el inicio de sesión compartida para ver películas en un iPhone y en un celular con Android.