

Calidad.

¿De qué hablamos cuando hablamos de calidad en testing?

Del grado de satisfacción del cliente, esto implica cumplir con sus expectativas y satisfacerlas, sin excedernos en tiempo y presupuesto.

Si cumplimos sus requerimientos por encima de sus expectativas, ¿hablamos de calidad?

Bajo este escenario, no, porque estamos destinando más recursos de los necesarios, disminuyendo la calidad.

Evolución histórica en la calidad del software.

1950 – 1960.

- **Sistemas militares.**
- **Software + hardware.**
- **Dominios estables.**
- **Volumen medio.**

Después de la Segunda Guerra Mundial, los 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 – 1970.

- **Sistemas.**
- **Software independiente.**
- **Propósito general.**
- **Volumen grande.**

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.

1970 – 1980.

- **Aplicaciones masivas.**
- **Procesos IVV.**
- **Métodos iterativos.**
- **Software Factory.**

A partir de los inconvenientes de sobre presupuesto 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.

- **ISO 9126, ISO 12207 e ISO 9000-3.**
- **Orientación a objetos.**
- **UML.**

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.

- **Ciencia, gestión e ingeniería de los Servicios Web 4.0.**
- **Deep learning g.**
- **Machine learning.**

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.

2000 a la Actualidad.

- **Ciencia, gestión e ingeniería de los servicios.**
- **Web 4.0.**
- **Deep learning.**
- **Machine learning.**

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 la seguridad; la importancia de la “ciencia, gestión e ingeniería de los servicios” que requiere un enfoque interdisciplinar —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.

La ingeniería de software es una disciplina joven y que durante su corta vida atravesó diferentes etapas de las cuales aprendió la lección:

La calidad es un valor en sí mismo y no un gasto que las empresas deben realizar para que su negocio prospere.

Esta ingeniería “es como una persona de la cual todos alguna vez se enamoran, pero con la que muy pocos están dispuestos a casarse” porque es muy común encontrarse con proyectos que generan productos de dudosa calidad debido a que los responsables decidieron no realizar tareas de revisión de diseño y código o determinadas pruebas porque dilatan el tiempo de salida al mercado, cuestan dinero y además no habrá diferencia entre uno y otro producto cuando el proyecto termine.

Validación vs. Verificación.

Validación:

Son aquellas cuestiones que un analista de negocio/funcional deberá confirmar con el cliente, garantizando un claro entendimiento de las necesidades antes de pasar a la propia ejecución.

Verificación:

Es el control de calidad sobre un producto, servicio o entregable. En este último caso de una salida que es contrastada con los requerimientos del lado del negocio, de modo de tener una comparación entre lo planteado versus lo realmente logrado.

7 principios del testing.

1. La prueba muestra la presencia de defectos, no su ausencia.

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. La prueba exhaustiva es imposible

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. La prueba temprana ahorra tiempo y dinero.

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. Los defectos se agrupan.

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. Cuidado con la prueba del pesticida.

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. La prueba se realiza de manera diferente según el contexto.

Por ejemplo, el software de control industrial de seguridad crítica se prueba de forma diferente a una aplicación móvil de comercio electrónico.

7. La ausencia de errores es una falacia.

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.

Business analyst / Analista de negocio

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

Software developer / Desarrollador de software.

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.

Grandes empresas - Grandes defectos.

A lo largo de la historia, se registraron grandes errores producidos en grandes empresas.

¿Se acuerdan del famoso error de Samsung en donde los teléfonos Galaxy Note 7 explotaban?

La empresa luego de reconocer el error, publicó un informe con los motivos:

El informe explica que en las primeras baterías se produjo un error en el diseño en la esquina superior derecha, que provocó deformaciones en el electrodo negativo, generando un cortocircuito en algunas de las celdas y un sobrecalentamiento que hacía que el dispositivo se incendiara.

Otra gran empresa que cometió un error que hizo dudar de su fiabilidad a sus clientes —y que produjo una pérdida muy grande de dinero— fue Tesla. Se produjo un error en el sistema de control de crucero de sus vehículos y este se activaba accidentalmente. ¿Qué salió a decir la empresa?

La empresa explica que se produjo un error dentro de su software que producía que se active el control de crucero aleatoriamente en sus vehículos. La solución fue una nueva actualización del software, pero esto generó la desconfianza de sus clientes.

Otro suceso importante le pasó a la famosa plataforma YouTube, el video musical de Gangnam Style rompió la plataforma. ¿Qué pasó aquí?

El contador de YouTube antes usaba un número entero de 32 bits, que es una unidad que se usa para representar datos en la arquitectura de la computadora. Este entero de 32 bits determina que el número máximo de vistas posibles que podía contar era 2,147,483,647. El famoso video excedió este valor máximo y rompió la plataforma.

Hoy en día, YouTube usa un número entero de 64 bits para su contador de videos.

Ciclo de vida de un defecto.

El proceso que gestiona un defecto desde su descubrimiento hasta su solución se denomina ciclo de vida de un defecto.

En cada estado solo existe un responsable del defecto, excepto en estados terminales —cerrado, duplicado—, debido a que no se van a realizar más acciones.

Todos estos sucesos (y muchísimos más registrados a lo largo de la historia) podrían haber sido evitados con un buen plan de testing, de manera de encontrarlos rápidamente y que no lleguen a la sociedad. Es por ello que a lo largo de la materia veremos cómo crear un plan de testing, cómo registrar estos defectos y, por ende, evitar grandes defectos en grandes empresas.

Nuevo/Inicial:

Se recopila 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: “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.

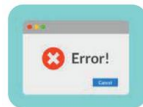
Defectos.

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:

1. Gestión del informe durante el ciclo de vida de los defectos.
2. Evaluación del estado del proyecto, especialmente en términos de calidad del producto y progreso de las pruebas.
3. Evaluación de la capacidad del proceso.

Partes de un informe de defectos.

Atributo	Descripción	Ejemplo
ID	Abreviatura de identificador, un código único e irreplicable que puede ser número o letras.	001 - Test01
Título	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 – Ingresar con campos en blanco
Descripción	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.
Fecha del informe del defecto	La fecha que detectó el defecto para saber posteriormente el tiempo en que se resolvió.	23/04/21
Autor	El nombre del tester que descubrió el defecto, por si el desarrollador tiene una duda, sabe a quién consultar.	Pepito Román
Identificación del elemento de prueba	Identificación del elemento de prueba	Carrito compras
Pasos a reproducir	Los pasos a seguir para llegar al defecto encontrado.	1) Ingresar a la aplicación. 2) Dejar en blanco el campo nombre. 3) Dejar en blanco el campo password. 4) Hacer click en el botón “Ingresar”.
Resultado esperado	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.
Resultado obtenido o actual	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.	Ingresa a la aplicación sin usuario y sin contraseña.
Severidad	Cuán grave es el defecto que hemos encontrado, puede ser: bloqueado, crítico, alto, medio, bajo o trivial.	Crítico
Prioridad	Con esto decimos qué tan rápido se debe solucionar el defecto, puede ser: alta, media, baja	Alta
Estado del defecto	Los estados pueden ser: nuevo, diferido, duplicado, rechazado, asignado, en progreso, corregido, en espera de verificación, en verificación, verificado, reabierto y cerrado.	Nuevo
Referencias	Link al caso de prueba con el cual encontramos el error.	https://repositorio.com.ar/TC-001-User - Login
Imagen	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.	

Caso de prueba.

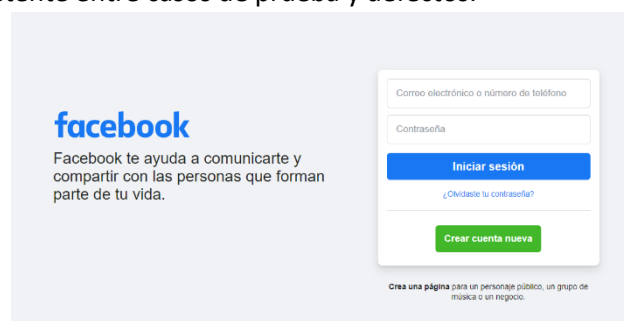
Un caso de prueba es un conjunto de acciones que se ejecutan para verificar una característica o funcionalidad particular de una aplicación de software. Es decir, que todas las características de una aplicación de software van a ser representadas por uno o más casos de prueba. Es por ello que este documento es de vital importancia en el mundo de la calidad.

En esta parte de la materia, aprenderemos cómo crear un caso de prueba de manera correcta mediante el uso de técnicas de prueba. También, nos centraremos en la relación existente entre casos de prueba y defectos.

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: https://www.facebook.com/	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

El siguiente es un caso de prueba negativo para el login de Facebook basado en el mismo requerimiento y pantalla:

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: https://www.facebook.com/	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

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.

Para lograr esta visión global conoceremos 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).

Estas actividades se agruparán 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.

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.

Objetivos específicos.	Bases de prueba.	Objeto de prueba.
<ul style="list-style-type: none">• 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.	<p>Algunos ejemplos de productos de trabajo que se pueden utilizar como base de prueba incluyen:</p> <ul style="list-style-type: none">• Diseño detallado.• Código.• Modelo de datos.• Especificaciones de los componentes.	<p>Los objetos de prueba característicos para la prueba de componente incluyen:</p> <ul style="list-style-type: none">• Componentes, unidades o módulos.• Código y estructuras de datos.• Clases.• Módulos de base de datos.

Defectos y fallos característicos.	Enfoques y responsabilidades específicas.
<p>Ejemplos de defectos y fallos característicos de la prueba de componente incluyen:</p> <ul style="list-style-type: none">• 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.	<p>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.</p>

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">• Reducir el riesgo.• Verificar que los comportamientos funcionales y no funcionales de las interfaces sean los diseñados y especificados.• Generar confianza en la calidad de las interfaces.• Encontrar defectos —que pueden estar en las propias interfaces o dentro de los componentes o sistemas—.• Prevenir la propagación de defectos a niveles de prueba superiores.	<p>Algunos ejemplos de productos de trabajo que pueden utilizarse como base de prueba incluyen:</p> <ul style="list-style-type: none">• Diseño de software y sistemas.• Diagramas de secuencia.• Especificaciones de interfaz y protocolos de comunicación.• Casos de uso.• Arquitectura a nivel de componente o de sistema.• Flujos de trabajo.• Definiciones de interfaces externas.	<p>Los objetos de prueba característicos para la prueba de integración incluyen:</p> <ul style="list-style-type: none">• Subsistemas.• Bases de datos.• Infraestructura.• Interfaces.• Interfaces de programación de aplicaciones —API por sus siglas en inglés—.• Microservicios.

Defectos y fallos característicos	Enfoques y responsabilidades específicas
<ul style="list-style-type: none"> • Datos incorrectos, datos faltantes o codificación incorrecta de datos. • Secuenciación o sincronización incorrecta de las llamadas a la interfaz. • Incompatibilidad de la interfaz. • Fallos en la comunicación entre componentes. • Fallos de comunicación entre componentes no tratados o tratados de forma incorrecta. • Suposiciones incorrectas sobre el significado, las unidades o las fronteras de los datos que se transmiten entre componentes. 	<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"> • Reducir el riesgo. • Verificar que los comportamientos funcionales y no funcionales del sistema son los diseñados y especificados. • Validar que el sistema está completo y que funcionará como se espera. • Generar confianza en la calidad del sistema considerado como un todo. • Encontrar defectos. • Prevenir la propagación de defectos a niveles de prueba superiores o a producción. 	<p>Algunos ejemplos de productos de trabajo que se pueden utilizar como base de prueba incluyen:</p> <ul style="list-style-type: none"> • Especificaciones de requisitos del sistema y del software — funcionales y no funcionales—. • Informes de análisis de riesgo. • Casos de uso. • Épicas e historias de usuario. • Modelos de comportamiento del sistema. • Diagramas de estado. • Manuales del sistema y del usuario. 	<ul style="list-style-type: none"> • Aplicaciones. • Sistemas hardware/software. • Sistemas operativos. • Sistema sujeto a prueba (SSP). • Configuración del sistema y datos de configuración.

Defectos y fallos característicos	Enfoques y responsabilidades específicas
<ul style="list-style-type: none"> • Cálculos incorrectos. • Comportamiento funcional o no funcional del sistema incorrecto o inesperado. • Control y/o flujos de datos incorrectos dentro del sistema. • Incapacidad para llevar a cabo, de forma adecuada y completa, las tareas funcionales extremo a extremo. • Fallo del sistema para operar correctamente en el/los entorno/s de producción. • Fallo del sistema para funcionar como se describe en los manuales del sistema y de usuario. 	<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
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: <ul style="list-style-type: none">• Establecer confianza en la calidad del sistema en su conjunto.• Validar que el sistema está completo y que funcionará como se espera.• Verificar que los comportamientos funcionales y no funcionales del sistema sean los especificados.	Entre los ejemplos de productos de trabajo que se pueden utilizar como base de prueba se encuentran: <ul style="list-style-type: none">• Procesos de negocio.• Requisitos de usuario o de negocio.• Normativas, contratos legales y estándares.• Casos de uso.• Requisitos de sistema.• Documentación del sistema o del usuario.• Procedimientos de instalación.• Informes de análisis de riesgo.	<ul style="list-style-type: none">• Sistema sujeto a prueba.• Configuración del sistema y datos de configuración.• Procesos de negocio para un sistema totalmente integrado.• Sistemas de recuperación y sitios críticos —para pruebas de continuidad del negocio y recuperación de desastres—.• Procesos operativos y de mantenimiento.• Formularios.• Informes.• Datos de producción existentes y transformados.

Defectos y fallos característicos	Enfoques y responsabilidades específicas
Entre los ejemplos de defectos característicos de cualquier forma de prueba de aceptación se encuentran: <ul style="list-style-type: none">• Los flujos de trabajo del sistema no cumplen con los requisitos de negocio o de usuario.• Las reglas de negocio no se implementan de forma correcta.• El sistema no satisface los requisitos contractuales o reglamentarios.• Fallos no funcionales tales como vulnerabilidades de seguridad, eficiencia de rendimiento inadecuada bajo cargas elevadas o funcionamiento inadecuado en una plataforma soportada.	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.

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.

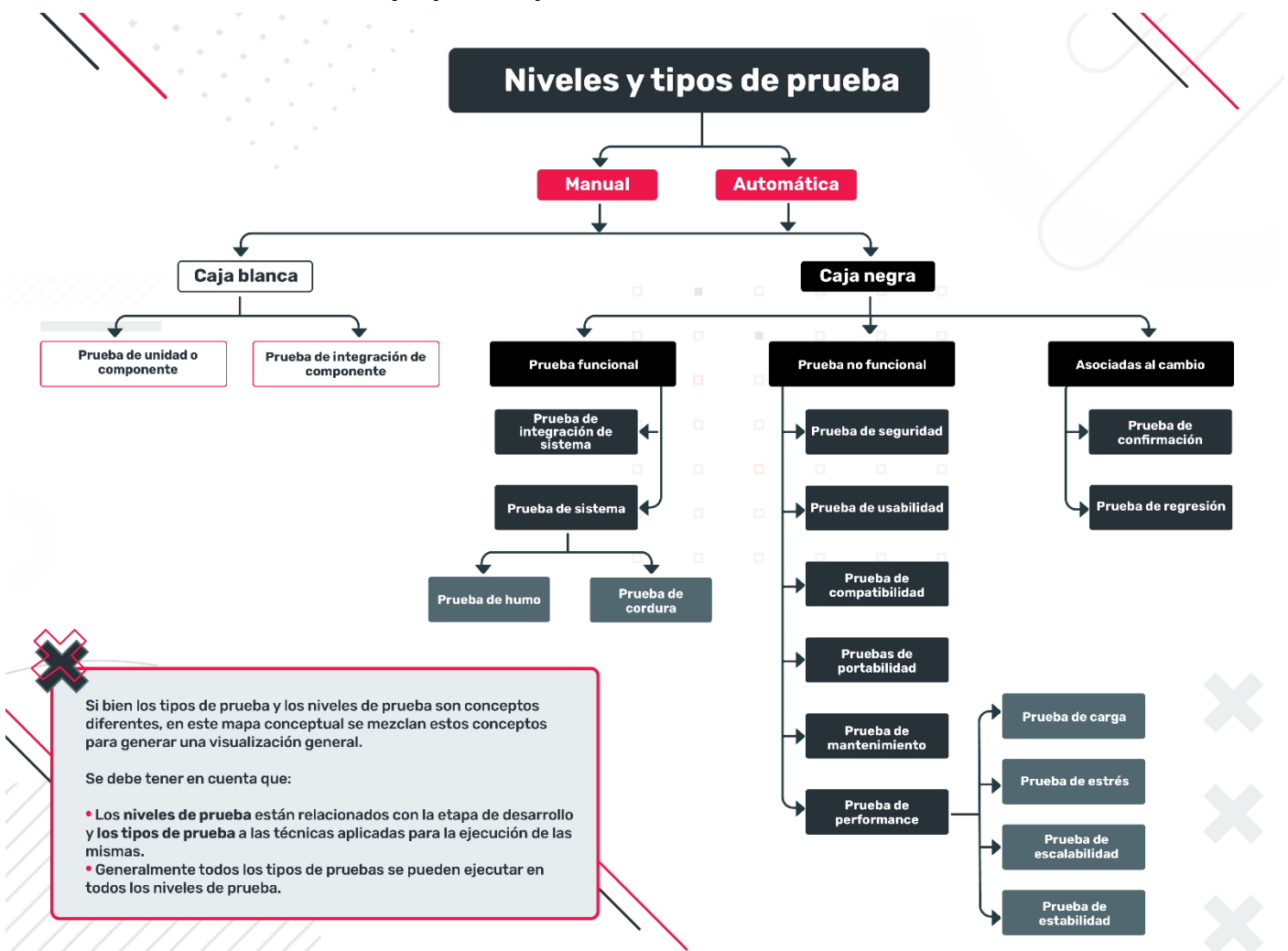
Pruebas no funcionales:

- Prueba de componente: las pruebas de rendimiento están diseñadas para evaluar el número de ciclos de CPU necesarios para realizar un cálculo de intereses totales complejo.
- Prueba de integración de componentes: las pruebas de seguridad están diseñadas para vulnerabilidades de

desbordamiento de memoria intermedia debido a que los datos pasan de la interfaz de usuario a la lógica de negocio.

- Prueba de sistema: las pruebas de portabilidad están diseñadas para comprobar si la capa de presentación funciona en todos los navegadores y dispositivos móviles soportados.

Clasificación de los niveles y tipos de prueba.



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.

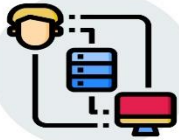


Limitaciones

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

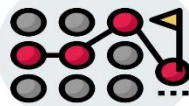
Encontrar este camino único requiere un gran conocimiento del uso de la aplicación y necesidades del cliente.

Caso de uso y caso de prueba



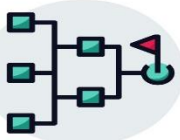
¿Qué es un caso de uso?

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	Partes del caso de prueba
Nombre del caso de uso	Nombre del caso de prueba
Precondiciones del caso de uso	Precondiciones del caso de prueba
Secuencia normal y secuencia alternativa	Pasos del caso de prueba
Resultados en la secuencia normal o alternativa. Poscondiciones del caso de uso	Resultado esperado

La capacidad para crear casos de prueba a partir de los casos de uso y hacer la trazabilidad de unos a otros es una habilidad vital para asegurar un producto de calidad.

¿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.

Clasificación de los tipos de prueba.

Pruebas:

Tipos:

Funcionales: Con este tipo de pruebas testeamos la funcionalidad de nuestro sistema o software. Podemos hacernos preguntas sobre cómo funciona, qué debe estar haciendo y cómo están interactuando los usuarios.

- Pruebas de sanidad.
- Pruebas de aceptación.
- Pruebas de sistema.

No funcionales: Puede que tengamos un sistema funcionando, pero el usuario está experimentando otro tipo de problemas que no son detectados por las pruebas anteriores. Factores como lentitud, problemas con la combinación de colores que producen poca legibilidad, claridad, usabilidad y seguridad, son los que testeamos en este tipo de prueba.

- Pruebas de rendimiento.
- Pruebas de portabilidad.
- Pruebas de usabilidad.
- Pruebas de carga.
- Pruebas de estrés.

Estructurales: Éstas 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.

- Prueba unitaria.
- Prueba de integración.

Asociados al cambio: Este tipo de pruebas están asociadas a los cambios que se producen debido a nuevos requerimientos o cambios en requerimientos existentes. Estos cambios son más frecuentes en ciclos de vida de desarrollo iterativos e incrementales —por ejemplo, Agile—.

- Prueba de confirmación (Re-test).
- Prueba de regresión.

Técnicas:

Caja negra: En este tipo de técnicas el elemento es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno.

- Partición de equivalencia.
- Análisis de valores límites.

Caja blanca: Método por el cual se mira el código y la estructura del producto que se va a probar y se usa ese conocimiento para la realización de las pruebas.

- Pruebas de sentencia.
- Pruebas de decisión.

Formas:

Manual: Las pruebas manuales son ejecutadas directamente por uno o más testers, simulando las acciones del usuario final, apoyándose de las herramientas necesarias.

Automatizadas: Las pruebas automatizadas son ejecutadas por testers con habilidades técnicas, y se apoyan en diversas herramientas para realizar scripts y así, ejecutar las pruebas automáticamente.

