United Certifications
Fundamentos de Pruebas para Ingeniería de Software
(Desarrollo y Administración)

Programa de Estudio - Versión 1.3

Derechos de Autor

Palabras de Agradecimiento

Los autores desean agradecer especialmente a varias personas en la creación de este trabajo. Durante mucho tiempo ha sido un deseo poder compilar o registrar experiencias en esta narrativa. Esto no se habría logrado sin la ayuda de las siguientes personas. Un agradecimiento especial a todos ellos por hacer posible, de una u otra manera, que esta historia se materializara: Kyle Siemens, Rogier Ammerlaan, Daniel van der Zwan, Annelies van Rijn, Valentijn Duijser, John Wittmaekers, Anne Laura Drost-Douma, David de Roo, Sjoerd Walinga, Jose Correia, Khadidja Zegrir, Guino Henostroza, Patricia Osorio Aristizábal. y, por supuesto, a todos los demás colegas de Concept7 y el SSTQB y ITB por apoyar el mérito de este documento.

Versión

Version	Date	Comments
1.0	01-04-2023	First Publication
1.1	30-10-2023	Processed reviewcomments
1.2/1.3	29-08-2024	Processed reviewcomments, included AI

Traducción al español

Versión en español realizada por Claudio Carvajal Z, Master Trainer (Chile)

Índice

Introducción

Propósito de este documento

Este programa de estudios constituye la base para la certificación en *Fundamentos de Pruebas para Ingeniería de Software (Testing Fundamentals for Software Engineers, TFSE – siglas en inglés)*. En este documento se describe lo que necesitas saber para aprobar el examen. Tanto este documento como el programa de formación, incluido el examen, están protegidos por derechos de autor. El examen incluirá únicamente preguntas relacionadas con los conceptos y conocimientos explicados en este documento.

Los diferentes componentes de la formación también están disponibles en el sitio web oficial de *Fundamentos de Pruebas para Ingeniería de Software*. Estos componentes incluyen:

- Una lista completa de proveedores de formación y fechas de cursos disponibles. Aunque se recomienda tomar el curso, no es un requisito obligatorio para presentar el examen.
- El syllabus (programa de estudios, este documento) disponible para descargar.
- Un examen de práctica completo con 40 preguntas y un documento con las respuestas, para usar como preparación para el examen real.
- Nuestro objetivo es ofrecer estos documentos en varios idiomas. Mantente atento al sitio web para futuras actualizaciones.

Propósito de este Programa de Estudios

El testing no es una ciencia exacta. Existen innumerables formas de lograr el objetivo: en general, una aplicación libre de errores graves y potencialmente costosos. Este curso no describe todas las facetas del testing ni enseña todos los aspectos que existen sobre esta disciplina.

Existen numerosos cursos que abordan aspectos específicos del testing en detalle. Puede que se requieran cursos más especializados si deseas trabajar en la ingeniería de pruebas o profundizar en el testing de manera más exhaustiva.

El curso *Certificación Fundamentos de Pruebas para Ingeniería de Software* proporciona lo esencial. Presenta un enfoque práctico y pragmático, ofreciendo métodos que se pueden aplicar directamente en la mayoría de los proyectos cotidianos.

El objetivo de este curso es enseñar habilidades específicas de testing a desarrolladores y administradores. Es adecuado para personas con distintos niveles de experiencia, independientemente de su grado de seniority.

Resultados esperados después de asistir a esta formación (Objetivos de Negocio)

BO 1	Aprender aspectos prácticos sobre pruebas y calidad como desarrolladores o administradores.
BO 2	Comprender los fundamentos de las pruebas y la calidad como desarrolladores o administradores.
BO 3	Entender los requisitos previos y principios para realizar pruebas de calidad.
BO	Aprender una variedad de herramientas para mejorar la calidad como desarrolladores o

4	administradores.
BO 5	Aprender diversas formas de crear un script de prueba como desarrolladores o administradores.

Objetivos de Aprendizaje

Los objetivos de aprendizaje son descripciones breves de lo que se necesita recordar después de leer el texto correspondiente. Existen 3 niveles **[I]**:

- K1: Recordar
- K2: Comprender
- K3: Aplicar

A continuación, se resumen todos los objetivos de aprendizaje (Learning Objective, LO) de este curso:

LO1	Recordar qué significa testing. (K1)
LO2	Recordar qué significa calidad. (K1)
LO3	Recordar que la calidad puede evaluarse de diferentes formas usando los atributos de calidad. (K1)
LO4	Comprender la necesidad de realizar pruebas. (K2)
LO5	Comprender la diferencia entre el enfoque de los desarrolladores y el de los testers. (K2)
LO6	Aplicar prácticas clave de testing. (K3)
LO7	Memorizar algunos términos específicos de pruebas. (K1)
LO8	Comprender la práctica de no hacer suposiciones. (K2)
LO9	Comprender la práctica de que el testing es pensamiento lógico. (K2)
LO10	Comprender la importancia de las pruebas de regresión. (K2)
LO11	Recordar los pros y contras de la automatización de pruebas. (K1)
LO11.1	Comprender cómo la IA puede beneficiar al testing y cuáles son sus limitaciones. (K1)
LO12	Comprender que probar todo es imposible. (K2)
LO13	Comprender la importancia de tener una estrategia de pruebas. (K2)
LO14	Aplicar análisis de riesgos en las pruebas. (K3)
LO15	Comprender en qué consiste el análisis de Pareto. (K2)
LO16	Comprender cómo capturar la causa de los incidentes puede mejorar tu SDLC. (K2)
LO17	Aplicar la práctica de ser específico. (K3)
LO18	Comprender la práctica de realizar pruebas lo antes posible. (K2)
LO19	Conocer los tipos de entornos, incluyendo pruebas. (K2)
LO20	Comprender la importancia de un buen entorno de pruebas. (K2)
LO21	Comprender la importancia de los datos de prueba. (K2)
LO22	Comprender la práctica de comenzar con pruebas pequeñas y expandir gradualmente su alcance. (K2)
LO23	Recordar los niveles y tipos de pruebas. (K1)
LO24	Comprender cómo los niveles y tipos de pruebas se aplican a una estrategia de pruebas. (K2)
LO25	Recordar un proceso básico de pruebas. (K1)
LO26	Comprender el valor de documentar las pruebas. (K2)
LO27	Aprender a crear un script de prueba mediante la representación del proceso/programa. (K3)
LO28	Aprender a crear un script de prueba usando pruebas semánticas. (K3)
LO29	Aprender a probar funcionalidades utilizando tablas de decisión. (K3)

LO30	Aprender a profundizar en las pruebas mediante el análisis de valores límite. (K3)
LO31	Aprender a profundizar en las pruebas mediante clases de equivalencia. (K3)
LO32	Aprender a probar usando una checklist. (K3)
LO33	Aprender la técnica de pruebas combinatorias (pairwise testing). (K3)
LO33.1	Aprender a usar IA al diseñar pruebas. (K3)
LO34	Comprender la importancia de una buena comunicación en todas las actividades como tester. (K3)
LO35	Aprender los fundamentos de las pruebas de seguridad. (K2)
LO36	Aprender los fundamentos de las pruebas de usabilidad. (K2)
LO37	Aprender los fundamentos de las pruebas de rendimiento. (K1)

Requisitos previos

No se requiere conocimiento previo específico; sin embargo, resulta útil tener experiencia en una o más de las siguientes áreas: desarrollo de software, gestión de proyectos, gestión de software, aceptación de software y/o testing de software.

Notas Generales

Para mantener la fluidez en el texto de este documento y del material del curso, los autores pueden referirse a:

- "**Software**" en algunos casos donde se pretende indicar "Producto, Servicio y/o Sistema".
- "**Ingeniero/as de software**" en algunos casos donde se pretende indicar desarrolladores o administradores.
- "**SDLC**" como la abreviatura en inglés del Ciclo de Vida del Desarrollo de Software.
- **"Administrador/a"** con este término nos referimos a un administrador de TI, administrador de sistemas, administrador de redes, empleado de operaciones, gestor de aplicaciones, gestor funcional o especialista en soporte de TI. Alguien que trabaja en TI y es responsable de gestionar, mantener y dar soporte a sistemas informáticos, redes y aplicaciones de software dentro de una organización. Este rol incluye tanto tareas técnicas (como configuración de sistemas, seguridad y resolución de problemas) como tareas funcionales (como garantizar que el software cumpla con las necesidades del usuario, proporcionar soporte a los usuarios y coordinar entre los departamentos de TI y del negocio).

Nota del traductor

Se utiliza la forma masculina de los términos repetidos en este documento para simplificar su lectura. Por ej. Se usa "los desarrolladores" para referirse a todas las personas quienes trabajan en el desarrollo de software.

Estándares para técnicas de pruebas

En este programa de estudio se explican varias técnicas de pruebas. Se ha tomado una decisión consciente de mantener esto limitado y sencillo. Se enseñan los fundamentos de algunas técnicas porque este programa de estudio solo describe los aspectos básicos del testing. Para técnicas de prueba adicionales y un uso más avanzado, se hace referencia al estándar ISO **[XX]** u otras fuentes.

Aplicabilidad en Agile y DevOps

Este programa de estudio hace menos referencia a diversas metodologías de desarrollo de sistemas (Cascada, Agile, DevOps). Esta es una elección deliberada. Queremos establecer aquí la esencia de las pruebas y, en nuestra opinión, esta esencia no varía según la metodología de desarrollo utilizada. Habrá diferencias en cómo se ejecutan las actividades descritas en el programa de estudios dependiendo del proyecto, del momento o del entorno donde se realicen. También notarás que en otros tipos de organizaciones o con diferentes tipos de software, el énfasis puede variar, a veces porque las organizaciones ya han alcanzado un nivel de madurez más avanzado. Sin embargo, los fundamentos del testing y la importancia de estas habilidades básicas no cambiarán según la metodología de desarrollo de sistemas elegida. Dado que este programa de estudios está dirigido a desarrolladores, administradores y otras personas que inician en el testing, se ha optado por representar la esencia y no profundizar demasiado en temas que se alejan de los fundamentos.

Nota del autor principal Mattijs Kemmink:

Más de 20 años antes de iniciar la creación de este programa de estudios, comencé mi carrera como tester de software. En realidad, quería ser desarrollador de software, pero mi empleador seguía una política según la cual los desarrolladores de software debían comenzar como testers. En ese momento, tuve que preguntar qué era el testing porque no había oído hablar del concepto como función. Sin embargo, me alegré de conseguir un trabajo permanente con una gran empresa. Y así comenzaron los primeros pasos en mi camino en el mundo de las pruebas de software. En los años siguientes, tuve oportunidades de cambiar de tester a desarrollador varias veces, pero seguí volviendo a mis raíces en el mundo de las pruebas.

A lo largo de este programa, espero llevarte de viaje a través de algunas de mis experiencias en el mundo de las pruebas, relacionándolas con la teoría de metodologías de prueba conocidas. Presentaré los conceptos de la forma más sencilla posible, intentando hacerlo tal como los experimenté a lo largo de mi carrera sin un equipaje innecesario. Sin embargo, ten en cuenta que las pruebas nunca son sencillas, ya que siempre se personalizan y dependen de la situación y el entorno relevantes. Ten en cuenta que este curso no reemplaza los cursos teóricos estándar (por ejemplo, el ISTQB), ya que el campo de control de calidad es simplemente demasiado amplio para eso. Mi objetivo en este curso es proporcionar a los participantes una experiencia relevante para comprender mejor el papel del tester y la importancia del campo. Mi objetivo es ayudar a las personas a mejorar la calidad general de los productos que colaboran para producir y brindarles el conocimiento que desearía haber tenido al comienzo de mi carrera. Espero enseñarte aspectos prácticos que sean inmediatamente aplicables en tus prácticas diarias. A lo largo de mi carrera, me he topado con muchos desarrolladores y administradores que se preocupaban mucho por la calidad y querían aprender más sobre cómo involucrarse, pero no sabían por dónde empezar. Estas son las personas a las que espero ayudar con este curso de certificación.

Capítulo 1: Familiarizándose con las pruebas (el Testing)

LO1	Recordar qué se entiende por testing. (K1)
LO2	Recordar qué se entiende por calidad. (K1)
LO3	Recordar que se puede evaluar la calidad de diferentes maneras utilizando los atributos de calidad. (K1)
LO4	Entender la necesidad de realizar pruebas. (K2)

¿Qué es exactamente el testing?

El "testing" tiene muchas definiciones. En este curso, se define de la siguiente manera: "Un proceso que proporciona información y asesoramiento sobre la calidad y los riesgos asociados" **[II]**

Esta definición comporta dos componentes valiosos: **"calidad"** y **"riesgo"**. La calidad se define y detalla en la sección que sigue, pero primero es importante señalar que se podría asumir que el testing es la única forma de obtener información sobre la calidad. Sin embargo, existen otras maneras de alcanzar la calidad. El testing, en sí mismo, es una medida reactiva porque se realiza cuando el producto ya ha sido creado. No obstante, algo que puede ser más útil y eficiente que el testing, en muchos casos, es la **revisión**. Esto se debe a que a menudo puedes realizar una revisión incluso antes de que el producto sea desarrollado. Al llevar a cabo una revisión, puedes aumentar la calidad desde el principio, mejorando los requisitos ya creados, generando aquellos que faltan o eliminando requisitos redundantes.

La segunda parte de la definición de **"testing"** que merece ser destacada es **"riesgos"**. Los riesgos ya existen y el testing proporciona información sobre ellos. Si se ha realizado un análisis de riesgos para un proyecto específico, el testing puede verificar en qué medida esos riesgos siguen siendo relevantes. El testing también puede revelar nuevos riesgos. A través del testing, las organizaciones buscan identificar, eliminar o mitigar riesgos relacionados con sus operaciones. Además, el testing puede llevarse a cabo para cumplir con requisitos regulatorios o para garantizar que los procesos empresariales, productos y soluciones de TI ofrezcan calidad.

¿Qué es la Calidad?

En este curso, la **"calidad"** se define de la siguiente manera: *"La calidad es el conjunto de atributos y características de un producto o servicio que son importantes para satisfacer necesidades establecidas u obvias"* **[III]**. En términos más simples, podríamos decir que la calidad se compone de atributos y características, y que la calidad corresponde a expectativas que pueden estar establecidas o no.

Esta definición de **"calidad"** hace referencia a uno de los aspectos más complicados del testing: las **"necesidades no establecidas"**. Al trabajar con una lista de requisitos, pueden existir necesidades no establecidas que sean obvias para algunas partes interesadas y no tanto para otros. Por ejemplo, cerrar una aplicación usando el botón **X** (salir) en un entorno Windows frente a un entorno Apple: la misma acción se encuentra en diferentes partes de la pantalla.

Atributos de Calidad

Si consideramos que la **"Calidad"** se relaciona con **"atributos y características"** (tomado de la cita anterior **[III]**), debemos entender que los **"Atributos de Calidad"** son diferentes formas de analizar la calidad. La norma ISO 25010 **[IV]** describe varios atributos de calidad que pueden ser útiles en el testing de software. Algunos ejemplos de estos atributos son: funcionalidad, seguridad, rendimiento, usabilidad, mantenibilidad y portabilidad. La clasificación y subdivisión de la calidad en atributos de la calidad según la ISO 25010 es muy útil en el testing de software. Analizaremos algunos de estos atributos de calidad con mayor detalle en un capítulo posterior.

Dependencia del Software

El software está en todas partes hoy en día; es casi imposible vivir sin él. Solo piensa en pasar un día sin tu smartphone: para muchas personas, esto sería un gran desafío. Probablemente podrías sobrevivir sin tu teléfono, pero la vida con un smartphone es mucho más rápida y sencilla.

Así como tu smartphone hace tu vida diaria más eficiente al depender del software, si observamos el panorama general, nuestra sociedad también tiene procesos altamente dependientes y mejorados gracias al uso del software. Piensa en todos los procesos automatizados del gobierno, como el caso de la recaudación de impuestos. Si volviéramos a realizar declaraciones de impuestos sin computadoras, el proceso sería mucho más largo, costoso y demandaría una gran cantidad de recursos humanos. Otro ejemplo son los sistemas de trenes. El software controla los horarios, los accesos y los boletos de temporada; sería un gran retroceso manejar estos procesos sin software. Dado que queremos que la sociedad siga funcionando de manera eficiente, debemos garantizar que el software sea confiable y siga operando correctamente.

Además, y precisamente porque dependemos de él, queremos que el software funcione según lo previsto. Por ejemplo, considera el software que activa los airbags de un automóvil en caso de accidente. Queremos que esto suceda exactamente como se espera. Si los airbags se despliegan demasiado tarde, habrá daños; si se despliegan demasiado pronto o de manera inesperada, también habrá daños. En este ejemplo, el daño más crítico que los airbags buscan prevenir es el daño físico; sin embargo, también puede haber daños materiales al vehículo, daños inmateriales como consecuencias psicológicas, o daños financieros significativos. Este ejemplo ilustra cómo el testing de software es esencial para prevenir daños, demostrar calidad, identificar y mitigar riesgos, y garantizar la continuidad.

Definiciones

Testing	Un proceso que proporciona información y asesoramiento sobre la calidad y los riesgos relacionados.
Calidad	El conjunto de atributos y características de un producto o servicio que son importantes para satisfacer necesidades establecidas u obvias.
Atributo de Calidad	Una característica de un sistema de información.

Riesgo	Un factor compuesto por probabilidad e impacto que puede tener consecuencias.
Funcionalidad	La utilidad de algo, o qué tan bien realiza el trabajo para el que está destinado.
Mantenibilidad	El grado en que un producto o sistema puede ser modificado de manera efectiva y eficiente por administradores designados.
Portabilidad	El grado en que un sistema, producto o componente puede ser transferido de manera efectiva y eficiente de un entorno operativo o de uso de hardware, software u otro, a otro.
Revisión	La actividad de evaluar un producto o proceso con el objetivo de encontrar errores o realizar mejoras.

Capítulo 2: Tener el Enfoque Correcto para las Pruebas

LO5	Comprender la diferencia entre el enfoque de los desarrolladores y el enfoque de los testers. (K2)

Para un desarrollador, puede ser difícil realizar pruebas de manera efectiva. El principal objetivo de un desarrollador suele ser crear un producto funcional basado en las especificaciones dadas. El enfoque del desarrollador está en entregar un producto definido por el cliente y adecuado a sus necesidades. Dado que los desarrolladores se concentran principalmente en los resultados finales o las soluciones, es menos probable que detecten problemas periféricos.

Por otro lado, un tester tiene un objetivo completamente diferente: examinar la aplicación de todas las maneras posibles y verificar si funciona correctamente. El enfoque de un tester podría considerarse opuesto al de un desarrollador, lo que explica por qué existe una diferenciación natural entre ambas funciones. Por esta razón, combinar estas dos perspectivas tan diferentes al mismo tiempo puede resultar complicado.

Sin embargo, si es tu responsabilidad realizar ambas funciones, hay algunos consejos que puedes seguir para facilitarlo. Por ejemplo, podrías acordar con un colega que prueben mutuamente su trabajo para obtener una perspectiva más independiente y separarte del enfoque de desarrollo. También podrías designar un día específico para realizar pruebas o reservar un momento en el día en el que te enfoques conscientemente en las pruebas en lugar del desarrollo. Tener el enfoque correcto es crucial; en el próximo capítulo se discutirán algunos consejos concretos para lograrlo.

Como administrador, también puedes aprovechar esta conciencia sobre los desafíos para comprender mejor por qué las pruebas son a veces excelentes y otras veces insuficientes. Una de las cosas que puedes hacer es preguntar a los desarrolladores sobre su proceso de pruebas, por ejemplo, si emplean testers, si los desarrolladores también realizan pruebas, etc. Si todavía estás en la fase de negociación del contrato, incluso puedes proponer requisitos explícitos sobre esto. Por supuesto, si eres un administrador que realiza desarrollo o configuración de software, puedes aplicar los consejos anteriores para el desarrollo en tu propio trabajo.

Capítulo 3: Las Ocho Prácticas de Pruebas

LO6	Aplicar prácticas clave de pruebas. (K3)
LO7	Memorizar algunos términos específicos de pruebas. (K1)

Para hacer que el tema de las pruebas sea lo más accesible posible, hemos formulado una serie de prácticas basadas en experiencias del mundo de las pruebas. Estas prácticas están vinculadas a una serie de actividades concretas y aplicables que buscan proporcionar una visión que te ayude a maximizar la calidad del software.

La adhesión a estas prácticas puede reducir significativamente la probabilidad de errores. Incluso si no eres un tester, podrás mejorar la calidad del software, los procesos y los requisitos. Los principios que queremos explicar son:

1. No Hagas Suposiciones.
2. El testing es Pensamiento Lógico.
3. Probar Todo es Imposible.
4. Sé Específico.
5. Realiza Pruebas lo Antes Posible.
6. Empieza con Algo Pequeño y Expande Gradualmente el Alcance de tus Pruebas.
7. Documenta tus Pruebas.
8. Entiende la Importancia de una Buena Comunicación.

Cada una de estas prácticas será explicada en las secciones siguientes. Además de estas prácticas, existe otro principio universal importante que aplica: las pruebas siempre dependen del contexto. En este caso, "contexto" se refiere a los factores ambientales y las condiciones del proyecto. Los factores ambientales incluyen el sector o industria específica donde se realizan las pruebas; la naturaleza del software que estás creando, la organización o proyecto en cuestión, los requisitos legales, normativas, guías y estándares de la industria aplicables. Las condiciones del proyecto abarcan: tiempo, presupuesto y calidad esperada.

Además, ten en cuenta que puedes aplicar estas prácticas personalmente o asegurarte de que quienes desarrollan o prueban software para tu organización las utilicen. También puedes exigir su aplicación a un proveedor, ya sea solicitándolas directamente o incluyéndolas como una condición contractual.

Aplicación en Pruebas Ágiles y DevOps

En las organizaciones de TI modernas y con mayor madurez, a veces se observa que las pruebas y la calidad han evolucionado, y parece que algunos de los aspectos de este temario no se aplican o se aplican en menor medida. En ciertos casos, esto puede ser cierto. Es un hecho que existen otras herramientas y recursos que se utilizan en estos entornos. Sin embargo, el objetivo de este temario es claramente establecer los fundamentos, la esencia de las pruebas y la calidad del software. También buscamos hacer que las pruebas sean accesibles para un grupo con poca experiencia en el campo. Incluso en organizaciones con un mayor nivel de madurez o que operan con un proceso **Ágil** o **DevOps**, se puede aplicar

esta base a través de las prácticas presentadas. La diferencia radica en que será necesario revisar la estrategia de pruebas por sprint o por historia de usuario y posiblemente ajustarla específicamente para ese sprint o historia de usuario. Por ejemplo, si estás trabajando en historias con un alto riesgo dentro de tu sprint, podrías optar por una técnica de prueba más sólida que logre una cobertura más amplia.

En el caso de **DevOps**, podrías realizar los diferentes niveles de prueba, tal vez incluso en el mismo entorno. Pero, al menos, debes asegurarte de llevarlas a cabo para evitar que omitirlas afecte la calidad. Para el uso de otras herramientas y recursos en un entorno Ágil o DevOps, recomendamos encarecidamente formaciones más completas adaptadas a estas metodologías. Por ejemplo: **TMap, ISTQB, Agile United o DevOps United**, que pueden complementar este temario.

Práctica 1: No Hagas Suposiciones

LO8	Comprender la práctica de prueba de no hacer suposiciones. (K2)

La regla básica para todos los testers es "no hagas suposiciones": esta es la respuesta que obtendrás de cualquier persona en el ámbito de pruebas. Como tester, necesitas tener una actitud extremadamente crítica hacia lo que estás probando, además de poseer una dosis saludable de desconfianza. Como tester, deberías verificar básicamente todo, es decir, comprobar si el sistema funciona tal como se describe. También debes verificar si lo que se describió es lo que el cliente realmente necesita. En el mundo de las pruebas, lo primero se llama "verificación": el proceso de comprobar que el software hace lo que se especificó. Lo segundo se llama "validación": comprobar que la especificación satisface lo que el cliente necesita y/o espera. **[V]**

Como tester, a veces tendrás que asumir que lo que está descrito es lo que el cliente desea. En otras ocasiones, deberás asumir que las especificaciones del proveedor son correctas. En cuanto a las mejores prácticas:

- Asume que todo lo que se describe o dice sobre el software también debe verificarse;
- También debe validarse que lo que el cliente quiere haya sido construido.

Un ejemplo típico de una suposición es pensar que, si algo funciona en un entorno, también funcionará en otro. En realidad, esto a menudo no es cierto. Por ejemplo, en términos de gestión de configuraciones, existen diferentes versiones de componentes web para distintos entornos, y algunos componentes pueden no estar disponibles en ciertos entornos. Por estas razones, es posible que algunos componentes ni siquiera se detecten en determinados entornos.

Tomemos como ejemplo un sistema relativamente simple (ver Diagrama: Resumen de Entornos) en el que tenemos un sistema de información que consta de una base de datos, código, funciones y un componente de software externo, todos publicados en una página web. En la figura, las marcas verdes representan lo que está funcionando correctamente y lo que ha sido probado. Los círculos rojos indican defectos.

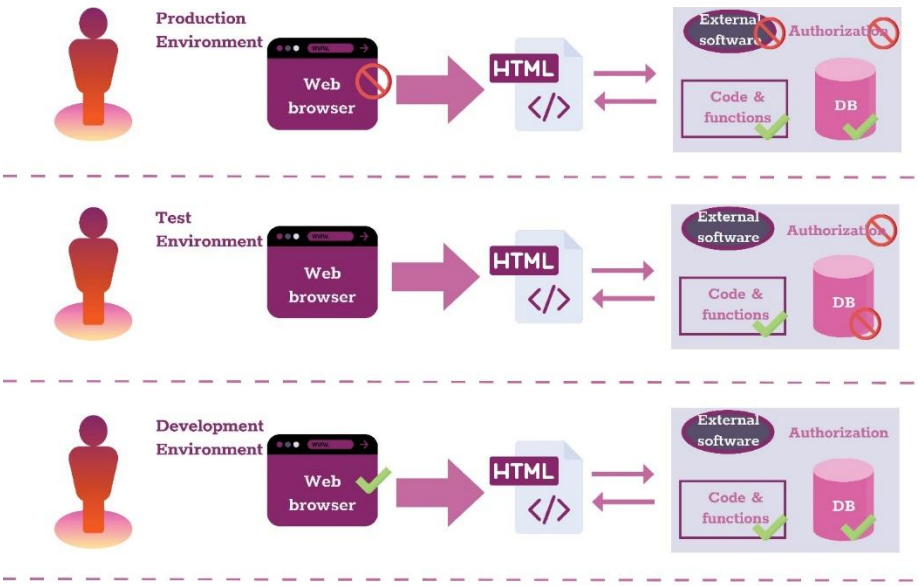


Diagrama: Resumen de Entornos

Si estás construyendo este sistema, puedes probarlo exhaustivamente en tu entorno de desarrollo, como debería ser.

Sin embargo, después de realizar las pruebas en el entorno de desarrollo, necesitas implementarlo en el entorno de pruebas. Como ya has comprobado que la funcionalidad funciona, es probable que confíes en que también funcionará en el entorno de pruebas. Pero surgen varias preguntas: ¿Qué pasa si olvidas entregar un componente? ¿Qué pasa si necesitas usar una dirección diferente para el software de terceros (piensa en ciertos plugins)? ¿Qué pasa si necesitas un certificado diferente en un entorno distinto? También será necesario reasignar autorizaciones. ¿Están correctas desde el primer momento? Si es así, ¿cómo sabes que los diferentes roles que definiste también funcionan? A veces, las pruebas realizadas con el rol de administrador funcionan bien, pero luego fallan para un usuario normal. Además de todo esto, el cliente podría tener un navegador web gravemente desactualizado o sin probar que podría generar errores que no encontraste durante las pruebas en el entorno de desarrollo.

Esto no significa que tengas que repetir toda la prueba en el entorno de pruebas y luego nuevamente en producción: ya has demostrado que la funcionalidad funciona. Sin embargo, aunque hayas probado todo extensamente en el entorno de desarrollo, ahora podrías probar adicionalmente algunos casos específicos para demostrar que la entrega está completa y que todos los componentes están presentes. Lo importante aquí es que, si verificas dinámicamente que funciona, tienes la mayor probabilidad de hacerlo bien. Si la verificación dinámica no puede realizarse, también puedes verificar o validar de forma estática.

Además, es importante comunicarte. Si tienes dudas sobre si algo está intencionado, contacta con el tester o el autor del requisito. La verificación/prueba estática y dinámica se explicará con más detalle en la práctica 5 de este programa de estudios.

Más adelante en este programa también discutiremos la comunicación en mayor detalle. Por ahora, es importante recordar no hacer suposiciones. Además, si eres un desarrollador o administrador, debes dejar de lado suposiciones como: "las pruebas son difíciles", "las pruebas son para testers", "no hay tiempo para probar", "ya hice todas las pruebas unitarias", "hay buen registro (logging)", "los testers también deberían tener algo que hacer", "probar es aburrido" y "funciona en mi entorno". Sin formación, es posible alegar desconocimiento, pero después de asistir a esta formación, sabrás mejor.

En resumen: si es posible, intenta verificar todo. Si es posible, hazlo dinámicamente llevándolo a cabo. Ten cuidado de no duplicar pruebas. Por ejemplo, ya no necesitarás probar el código de tu componente muy extensamente en los entornos de prueba o producción. Si la verificación dinámica no es posible, intenta validar. En cualquier caso, lo más importante es que comuniques lo que decides hacer.

Definiciones

Suposición	Una suposición, premisa o hipótesis que no ha sido probada.
Gestión de configuración	La gestión de configuración se centra en la versionado de elementos a lo largo del ciclo de vida del desarrollo de software (SDLC). Permite rastrear qué instancia específica de un diseño pertenece a qué instancia específica de código.
Pruebas dinámicas	Pruebas realizadas ejecutando código (de un componente o sistema) **[V]**.
Pruebas estáticas	Pruebas de una parte (de un producto o elemento de trabajo) del SDLC sin ejecutar código. **[V]** Por ejemplo, revisión de código, requisitos o análisis estático.
Ambiente de Prueba	Un entorno que contiene hardware, instrumentación, simuladores, programas de software y otros elementos de soporte necesarios para realizar una prueba.
Validación	El proceso de comprobar que la especificación satisface lo que el cliente necesita o espera.
Verificación	El proceso de comprobar que el software hace lo que está especificado.

Práctica 2: Las pruebas son Pensamiento Lógico

LO9	Comprender la práctica de que las pruebas son pensamiento lógico. (K2)
LO10	Comprender la importancia de las pruebas de regresión. (K2)
LO11	Recordar los pros y contras de la automatización de pruebas. (K1)
LO11.1	Comprender cómo la IA puede beneficiar las pruebas de software y cuáles son sus limitaciones.

Las pruebas de software no son muy complicadas y, a menudo, se reducen a una lógica básica. Considera la siguiente afirmación:

'SI llueve, ENTONCES te mojas.'

Si quisieras desarrollar este requisito, podrías convertirlo en código rápidamente. Luego, probablemente ejecutarías una prueba unitaria para verificarlo, como sueles hacer al desarrollar software. Esto último es una suposición y, por supuesto, nunca está de más preguntar si se probó y cómo se hizo. Sin embargo, ten en cuenta que, en el desarrollo, el enfoque está en realizar cosas y convertir los requisitos en código. El objetivo principal es encontrar la solución.

Si deseas ponerte en el lugar de un tester, deberás abordar el requisito de manera muy diferente. Por ejemplo: ¿Qué pasa si no llueve? ¿Te mojas? ¿Y si llueve pero no te mojas? ¿Qué debería ocurrir en esa situación?

Como tester, querrás verificar las siguientes cosas:

Escenario	Llueve	Te mojas
1.	Verdadero	Verdadero
2.	Verdadero	Falso
3.	Falso	Verdadero
4.	Falso	Falso

Tabla: Posibles Resultados de una Afirmación

Necesitarías validar que el segundo escenario no pueda ocurrir y que los demás escenarios puedan ocurrir para confirmar que la afirmación es verdadera.

El ejemplo anterior no trata sobre TI. Ahora tomemos, como otro ejemplo, el siguiente requisito:

*'SI soy **administrador** de este sistema, ENTONCES quiero poder modificar **usuarios**.'*

Escenario	Soy administrador	Puedo modificar usuarios
1	Verdadero	Verdadero
2	Verdadero	Falso
3	Falso	Verdadero
4	Falso	Falso

Tabla: Posibles Resultados de una Afirmación

Con este ejemplo, entendemos de inmediato que poder añadir, modificar o eliminar usuarios siendo un no-administrador no es deseable.
Esto definitivamente necesitaría ser probado. Además de enfocarte en cumplir con el requisito, intenta también centrarte en los escenarios "¿Qué pasa si...?" o en las situaciones "Falsas".

Si tomamos el ejemplo de subir un archivo .pdf a un sitio web, la importancia de las pruebas lógicas se vuelve aún más clara. Consideremos el siguiente requisito:

*' SI soy **administrador**, ENTONCES quiero poder añadir un archivo .PDF a un sitio web.'*

Desde la perspectiva de un desarrollador, las pruebas necesarias pueden parecer limitadas. Sin embargo, como tester, este caso debería involucrar mucho más. Subir archivos con una extensión diferente puede tener consecuencias no deseadas. También querrás verificar que los usuarios sin acceso de administrador no puedan subir archivos.

La siguiente tabla ilustra lo que esto implica:

	1	***2***	***3***	***4***
Soy administrador	S	S	N	N
El archivo es un .pdf	S	N	S	N
Puede subirse el archivo	X			
Sin acción	X	X	X	

Tabla: tabla de decisión para subir un archivo

Ahora queda claro que al menos 4 casos de prueba deberían ser ejecutados. Pero una revisión rápida muestra que hay varios roles además de "administrador" y "usuario," por lo que el camino "**3**" debería ser probado con todos esos roles. Además, hay otras extensiones de archivos que querrás excluir por razones legales o de seguridad, como .doc, .docx, .ppt, .exe, .zip, .rar, etc. Esto genera más situaciones en el camino "**2**" de lo que se pensaría a simple vista.

Además, simplemente verificar una extensión puede ser insuficiente. Un archivo .pdf también puede contener contenido malicioso. Por lo tanto, deberías sugerir que el requisito

se amplíe para incluir un requisito adicional relacionado con la integridad del archivo. Por simplicidad, dejemos esto como una nota al margen por ahora.

Alguien podría argumentar que no hay necesidad de realizar una revisión extensa en este caso, ya que se trata de un administrador. Después de todo, los permisos de un administrador están reservados para pocas personas que, en general, saben lo que hacen. Sin embargo, la funcionalidad aquí podría ser reutilizada en otro lugar o la contraseña de un administrador podría ser descubierta, por lo que una revisión exhaustiva podría ser beneficiosa.

Regresión

Ya podemos observar que incluso para funcionalidades relativamente simples, pueden surgir muchas razones para probar algo. También queda claro que el proceso de pruebas puede volverse bastante extenso. Ahora, imagina que el requisito en este ejemplo fue modificado después de que el proceso de pruebas para el requisito original se completó. Supongamos que ahora se permite otro tipo de archivo. La nueva funcionalidad sería la siguiente:

	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***
Soy administrador	S	S	S	S	N	N	N	N
El archivo es un .pdf	S	S	N	N	S	S	N	N
El archivo es un .doc	S	N	S	N	S	N	S	N
Puede subirse el archivo	X	X	X					
Sin acción	X	X	X	X	X			

Tabla: Tabla de decisión para subir un archivo con un nuevo requisito

Pensaste que ya tenías muchos casos de prueba en la situación original, pero ahora ves que el número está aumentando. Una razón importante a tener en cuenta aquí es que no puedes asumir que los cambios en el código no tuvieron efectos secundarios no deseados en la funcionalidad existente; por lo tanto, los casos de prueba originales también deben ejecutarse.

Recuerda considerar la funcionalidad que deseas probar en busca de regresiones con cada nuevo cambio. Las pruebas para verificar que la funcionalidad existente no se altere se denominan pruebas de regresión. Las pruebas de regresión generalmente son adecuadas para la automatización de pruebas debido a su naturaleza repetitiva.

Automatización de Pruebas

Antes de que las pruebas puedan ser automatizadas, siempre se requiere realizar pruebas manuales para comprender el sistema y saber qué se necesita para crear scripts de prueba automatizados. La automatización de pruebas generalmente se refiere a la ejecución automatizada de pruebas, una definición relativamente limitada. Sin embargo, esta definición podría ampliarse para incluir "la automatización de partes del proceso de pruebas," ya que a menudo es deseable automatizar tareas que consumen mucho tiempo en el proceso de

pruebas de una manera sencilla y rentable. Por ejemplo, la implementación de software de un entorno a otro, a menudo integrada en pipelines de Integración Continua/Despliegue Continuo (CI/CD), garantiza que las pruebas automatizadas se ejecuten de manera consistente y eficiente con cada actualización del software.

Aunque existen muchos beneficios en la automatización de tareas de prueba, también hay argumentos en contra que deben considerarse. Por ejemplo, para automatizar algo en el SDLC, se necesita un proceso maduro, así como una inversión de tiempo, dinero y recursos. Esto puede significar que podría pasar algún tiempo antes de obtener un retorno de la inversión inicial.

La automatización de pruebas puede ser valiosa, por ejemplo, cuando se realizan grandes cantidades de pruebas (de regresión) en entornos de software complejos y maduros durante largos períodos de tiempo. Otros ejemplos de situaciones que se prestan a la automatización de pruebas incluyen proyectos grandes en los que las mismas áreas deben ser probadas repetidamente (proyectos con muchas iteraciones). La automatización de pruebas también puede ser útil en proyectos que ya han pasado por un proceso inicial de pruebas manuales. Implementar la automatización de pruebas en un producto de software maduro puede mejorar la eficiencia en costos de las pruebas y aumentar la cobertura general de las mismas.

En general, la automatización de pruebas es adecuada si los procesos básicos de prueba están maduros, o dicho de otra manera, si existe una estrategia de pruebas bien establecida que incluya diferentes tipos de pruebas para cubrir diversos objetivos.

Algunas trampas típicas de la automatización de pruebas que deben evitarse:

- La automatización es un medio; sin embargo, a veces se percibe como un fin.
- Implica construir otra aplicación que necesita ser diseñada, aprendida, gestionada y mantenida.
- Con la automatización de pruebas, a menudo se crea una aplicación que puede distraer del objetivo principal: probar la aplicación deseada.
- Los scripts de prueba automatizados requieren mantenimiento, y descuidar este aspecto puede volverlos inmanejables.
- Ten en cuenta que, antes de poder automatizar un caso de prueba, primero debe diseñarse manualmente.
- Pensar y desarrollar una estrategia de pruebas te da una mejor comprensión de qué partes del sistema son adecuadas para la automatización. Además de las pruebas de regresión, esto incluye pruebas donde la aplicación necesita ser evaluada en diferentes sistemas operativos, navegadores web y dispositivos.

Existen muchos beneficios en la automatización de ciertas pruebas; sin embargo, la clave está en encontrar el equilibrio adecuado sobre qué es más apropiado para automatizar. En ocasiones, la **Automatización en las Pruebas** es más lógica e incluso más adecuada que la automatización de pruebas en sí misma. La automatización de pruebas es un tema amplio y,

si estás interesado, existen cursos completos sobre este tema que incluyen herramientas como Selenium.

Herramientas de Prueba

Existen muchas herramientas disponibles que pueden ser de gran apoyo en diferentes partes del proceso de pruebas o en varias etapas del SDLC. Para capturar casos de prueba y defectos, así como para la gestión de configuraciones, las herramientas adecuadas pueden ser muy beneficiosas. En el caso de las pruebas de rendimiento, incluso se podría argumentar que el uso de herramientas es una necesidad. Sin embargo, ten en cuenta que, aunque las herramientas ofrecen soluciones increíbles, suelen abarcar un amplio espectro de tareas dentro del SDLC, por lo que es posible que no ofrezcan exactamente la funcionalidad que necesitas.

Si planeas usar herramientas, realiza un piloto y un buen análisis de riesgos sobre la herramienta, considerando cuidadosamente los costos (incluyendo licencias) y los costos de mantenimiento. Las herramientas no siempre son lo que parecen, y los costos no siempre están claramente especificados. También existen riesgos asociados con herramientas de código abierto, ya que podrían no contar con un equipo de soporte dedicado. Trata de evitar crear proyectos adicionales y extensos solo para satisfacer la implementación de herramientas.

Volviendo a la idea de que "las pruebas son pensamiento lógico", esta filosofía también aplica a la automatización de pruebas y a las herramientas de prueba. Es fundamental tomar las acciones apropiadas antes de la implementación para garantizar el éxito. Como el objetivo principal de cada proyecto es obtener software de calidad, es importante mantener el enfoque en este objetivo y evitar desviarse hacia la implementación de herramientas y automatizaciones adicionales que podrían ser innecesarias.

Inteligencia Artificial en las Pruebas de Software

La inteligencia artificial (IA) se refiere a la simulación de la inteligencia humana, o quizás de manera más precisa, "la simulación de un cerebro biológico" en computadoras y máquinas. La IA permite a estos sistemas realizar tareas que usualmente requieren funciones cognitivas, como comprender el lenguaje, reconocer patrones, aprender de la experiencia, tomar decisiones y resolver problemas. La IA funciona utilizando algoritmos y datos para imitar funciones cognitivas, lo que permite a las máquinas adaptarse a nueva información y mejorar con el tiempo.

La inteligencia artificial se está integrando cada vez más en las prácticas de pruebas de software para mejorar la eficiencia, precisión y escalabilidad. Las herramientas de pruebas impulsadas por IA (modelos conversacionales específicos) pueden automatizar tareas repetitivas y que consumen mucho tiempo, como la creación de artefactos de prueba (por ejemplo, planes de prueba, casos de prueba, datos de prueba, informes de prueba), la ejecución de pruebas y la detección de defectos. La IA también puede ser muy útil para depurar scripts de pruebas automatizadas gracias a su capacidad para escribir código de programación.

La inteligencia artificial también puede redactar requisitos de manera eficiente, por ejemplo: requisitos para entornos de prueba, herramientas de prueba o incluso los requisitos del sistema. Sin embargo, estos requisitos deben ser evaluados y enriquecidos por un humano para mejorarlos y adaptarlos específicamente a la situación.

Lo mismo puede aplicarse al análisis de riesgos: al proporcionar una descripción genérica del sistema, la IA puede generar eficazmente una lista de riesgos genéricos que luego se pueden evaluar y utilizar como base para el análisis de riesgos.

Al analizar datos históricos de pruebas y patrones de uso de aplicaciones, los algoritmos de IA pueden predecir posibles áreas de fallo, optimizar la cobertura de pruebas y priorizar los esfuerzos de prueba. Esta capacidad predictiva ayuda a identificar rutas críticas y áreas que requieren más atención, reduciendo el tiempo total del ciclo de pruebas.

Ejemplo:
Imagina una aplicación de banca móvil que ha estado en uso durante varios años. Al analizar datos históricos de pruebas de versiones anteriores y patrones de uso recopilados de usuarios reales, los algoritmos de inteligencia artificial pueden identificar qué características y funciones son más propensas a tener problemas. Por ejemplo, la inteligencia artificial podría detectar que las áreas donde más errores se han reportado son la función de inicio de sesión y el procesamiento de transacciones. Con esta información, el software de IA puede predecir que estas áreas son puntos potenciales de fallo en futuras actualizaciones.

Como resultado, la solución de inteligencia artificial puede generar automáticamente más casos de prueba para estas áreas críticas, asegurando que se prueben exhaustivamente antes del lanzamiento. Además, la IA puede priorizar los esfuerzos de prueba en estas áreas de alto riesgo, garantizando que las funciones más críticas se prueben primero. Este enfoque específico ayuda a optimizar la cobertura de las pruebas y reduce el tiempo necesario para completar el proceso general de pruebas, lo que lleva a lanzamientos de software más rápidos y confiables.

Además, las herramientas impulsadas por inteligencia artificial (IA) pueden generar automáticamente scripts de prueba significativos, realizar pruebas visuales mediante el reconocimiento de cambios en la interfaz de usuario (UI), asistir en el análisis de las causas raíz y simular el comportamiento del usuario para validar el rendimiento de la aplicación en diferentes escenarios.

El uso de la IA en las pruebas también se extiende a las técnicas de Procesamiento del Lenguaje Natural (NLP, por sus siglas en inglés), lo que permite la automatización de la generación de informes de defectos y la documentación de pruebas basándose en entradas proporcionadas por los usuarios y registros de conversaciones.

Las capacidades de aprendizaje continuo de la IA le permiten adaptarse a entornos de software en constante evolución, asegurando que las prácticas de prueba sigan siendo sólidas y relevantes.

El Buen Diseño de Prompts Influye en los Resultados
El diseño de prompts se refiere a la práctica de crear entradas específicas y efectivas para guiar a los sistemas de inteligencia artificial (IA) en la generación de resultados precisos y relevantes. En el contexto de las pruebas de software, el diseño de prompts puede utilizarse para instruir de manera más detallada a los modelos de inteligencia artificial sobre cómo generar artefactos de prueba específicos, según los deseos del usuario, crear datos de prueba o incluso escribir scripts para pruebas automatizadas.

Al proporcionar prompts bien estructurados y claros, los testers pueden aprovechar todo el potencial de la IA para realizar tareas precisas e influir en la calidad de los resultados generados por la inteligencia artificial. Un diseño de prompts adecuado garantiza que las salidas de la IA estén alineadas con las necesidades específicas del proceso de pruebas, mejorando tanto la relevancia como la calidad de los artefactos generados por la IA.

Por ejemplo, se podría indicar que la herramienta debe utilizar la terminología de ISTQB. Esto resultará en mejores artefactos que cumplan con los estándares establecidos. Sin embargo, sigue siendo fundamental que los resultados producidos a través del diseño de prompts sean revisados y validados por testers humanos para asegurarse de que cumplan con los estándares deseados y reflejen con precisión los requisitos de prueba.

Ejemplo de Prompt de IA para Pruebas de Software:
"Genera casos de prueba para una aplicación bancaria móvil, enfocándote en la funcionalidad de inicio de sesión y el procesamiento de transacciones. Utiliza la terminología de ISTQB e incluye tanto casos de prueba funcionales como de valores límite. Asegúrate de cubrir casos extremos, como intentos incorrectos de inicio de sesión y saldo insuficiente en la cuenta durante las transacciones."

El uso de la inteligencia artificial en las pruebas de software ofrece claros beneficios, como una mayor eficiencia y precisión. Sin embargo, es crucial verificar los resultados generados por la IA, ya que pueden requerir validación humana.

Además, algunas organizaciones pueden tener restricciones para utilizar herramientas de IA de acceso público, especialmente cuando se manejan datos específicos de la empresa. Es importante ser cautelosos, ya que no siempre se sabe cómo se procesan los datos o quién tiene acceso a ellos.

Asegúrate siempre de que los datos se gestionen de forma segura y verifica cómo se utilizan dentro del sistema para evitar accesos no autorizados o posibles filtraciones de datos. Ten en cuenta las regulaciones de privacidad y las políticas de la empresa para proteger la información sensible.

Definiciones

Acción	El evento que está ocurriendo.
Inteligencia Artificial	Máquinas que imitan comportamientos de la inteligencia humana.

Automatización en pruebas	Automatización de pasos del proceso de prueba o del ciclo de vida del desarrollo de software (SDLC).
Condición	Un estado que puede o no cumplirse.
Decisión	El resultado de una condición.
DevOps	Una práctica que fusiona desarrollo y operaciones para mejorar la colaboración, automatizar procesos y acelerar la entrega de software.
Integración Continua	Una práctica de desarrollo de software donde los cambios en el código se prueban y se integran automáticamente en la rama principal regularmente, asegurando que el nuevo código se integre sin problemas y se valide desde el principio.
Despliegue Continuo	Una extensión de la integración continua (CI) que despliega automáticamente cada cambio validado en producción, asegurando que el software esté siempre en un estado desplegable y que las actualizaciones se entreguen rápidamente a los usuarios.
Aprendizaje Automático	Computadoras que aprenden a partir de patrones de datos.
Procesamiento del Lenguaje Natural (NLP)	El procesamiento del lenguaje natural combina técnicas estadísticas con métodos de aprendizaje automático. Con NLP, podemos extraer información de manera inteligente y automática de grandes cantidades de texto no estructurado. Por ejemplo, podemos extraer fácilmente palabras clave de un texto.
Diseño de prompts	El proceso de diseñar y crear entradas específicas, claras y efectivas para guiar a los modelos de IA en la generación de resultados precisos y relevantes. Implica crear instrucciones bien estructuradas para optimizar la respuesta de la IA en tareas como la generación de texto, casos de prueba o datos.
Automatización de pruebas	El uso de software para ejecutar o apoyar actividades de prueba.
Herramientas de prueba	Software o hardware que respalda una o más actividades de prueba.
Pruebas de regresión	Repruebas de funcionalidad preexistente para determinar si funciona como lo hacía antes de un cambio dado.
Requisito	Descripción de lo que es necesario.

Práctica 3: Probar Todo es Imposible

LO12	Entender la práctica de que probar todo es imposible. (K2)
LO13	Comprender la importancia de tener una estrategia de pruebas. (K2)
LO14	Aplicar el análisis de riesgos en tus pruebas. (K3)
LO15	Entender qué implica el análisis de Pareto. (K2)
LO16	Comprender cómo capturar la causa de los incidentes puede ayudarte a mejorar tu SDLC. (K2)

En el software y las aplicaciones que se desarrollan, existen innumerables opciones y posibilidades. Además, las aplicaciones pueden visualizarse en diversos dispositivos. Por ejemplo, incluso una **interfaz de programación de aplicaciones (API)** relativamente simple puede tener rápidamente varias docenas de opciones para ingresar datos. Es imposible ejecutar todos los casos de prueba potenciales, por lo que tendrás que tomar decisiones y diferenciarlas. Estas decisiones pueden parecer difíciles al principio, pero algunos trucos pueden ayudarte.

Consideremos que estamos cambiando la funcionalidad de un campo de número de teléfono. En un sitio web dado, la recopilación de un número de teléfono a través de un campo específico es una parte integral del recorrido del cliente, utilizado para verificar la identidad del cliente al realizar un pedido en la tienda en línea. Se deben considerar las pautas para la recopilación correcta de números de teléfono en un campo, ya que son esenciales para que la funcionalidad de validación de identidad del cliente sea posible. Esto no siempre se realiza correctamente, ni por los usuarios que ingresan el número de teléfono manualmente, ni por los desarrolladores de aplicaciones que programan los campos. A continuación, se presentan algunos ejemplos de formatos de números de teléfono:

País	**Ejemplo de Notación**	
Países Bajos	+31 059 660 0233	00(31)059 660 0233
	+(31) 059 660 0233	00(31)59 660 0233
	+31 59 660 0233	00 31 059 660 0233
	+31 (0)59 660 0233	00 31 59 660 0233
Estados Unidos	+(1)(425) 555-0100	+14255550100
Reino Unido	+(44)(20) 1234 5678	+442912345678
Prefijo Móvil Reino Unido	+(44) 07412 123 456	
China	+(86)(10) 1234 5678	+861012345678
Singapur	+(65) 1234 5678	+6512345678

Tabla: Formatos de números de teléfono

En este ejemplo, una API podría gestionarse de varias maneras, permitiendo muchos mensajes diferentes de SOAP (Protocolo Simple de Acceso a Objetos) y muchos mensajes diferentes de REST (Transferencia de Estado Representacional) para distinguir entre números de teléfono fijo y números de teléfono móvil.

Dependiendo del mensaje SOAP o REST (que tienen funcionalidades muy diferentes), anteriormente había dos o cuatro campos de número de teléfono en este tipo de mensajes. El campo destinado a números de teléfono móvil siempre tenía que comenzar con un prefijo

móvil. Además, había un requisito estricto para el campo de número de teléfono móvil: un máximo de 10 posiciones, comenzando siempre con un signo "+" seguido de un código de país de dos dígitos, que en este caso también debía tener el valor fijo del país.

Todos esos requisitos ahora han sido eliminados, y el campo ahora permite un máximo de 20 posiciones. También se eliminaron los requisitos del código de país y del prefijo móvil, lo que significa que ahora se puede ingresar un número de teléfono fijo en el campo que antes era exclusivo para números móviles y viceversa. Si analizas las distintas formas en las que podrías escribir un número de teléfono, llegas a una cantidad infinita de situaciones.

Cabe destacar que lo que se muestra en la tabla anterior (ejemplos de casos de prueba de números de teléfono) es solo una parte del conjunto de notaciones posibles para números de teléfono fijo, y los números móviles están en gran medida excluidos de la ecuación. Un desarrollador puede resolver esto de manera relativamente sencilla al construir una verificación en el campo que excluya todos los caracteres especiales y agregue automáticamente un signo "+". Sin embargo, como tester, tendrás que dedicar bastante esfuerzo a verificar todas las entradas válidas.

También tendrías que probar diferentes entradas en todos los campos, ya que no puedes asumir que todos los campos funcionan con el mismo mecanismo. Como desarrollador, puedes validar esto en el código, pero desde la perspectiva de las pruebas, el código es una caja negra. Al final, esto puede resultar en decenas de miles de situaciones de prueba.

Es imposible probar todas esas situaciones funcionales, ya que esto generalmente tomaría demasiado tiempo o simplemente costaría demasiado. Tendrás que tomar decisiones sobre qué es necesario probar. Debe quedar claro de qué consta técnicamente el objeto de prueba (el campo de número de teléfono), ya que esto ayudará a obtener una imagen clara de lo que necesita ser probado, lo cual debe documentarse en una **estrategia de pruebas**.

¿Qué es una Estrategia de Pruebas?

Una estrategia de pruebas puede considerarse como las directrices tácticas generales que deberían explicar todos los detalles y aspectos relacionados con lo que necesita ser probado a lo largo del ciclo de vida del desarrollo de software (SDLC).

Un aspecto importante a recordar al crear una estrategia de pruebas es asegurarse de que proporcione información útil sobre el **objeto de prueba**.

Creación de una Estrategia de Pruebas: Recopilación de Información

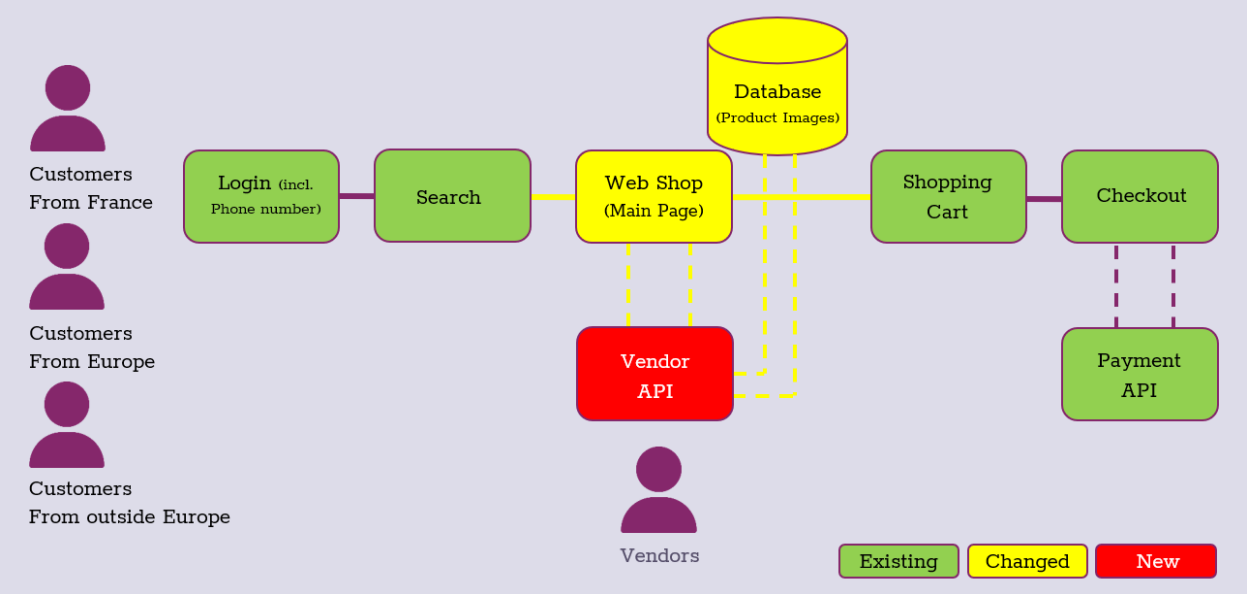
Información útil al crear una estrategia de pruebas (y al decidir los objetivos de prueba):

- Requisitos: Sirven como base para nuestras pruebas, por ejemplo, diseños de alto nivel, épicas, historias de usuario, casos de uso, documentación técnica, manuales de usuario, etc.
- Riesgos: Ayudan a tener una comprensión más sólida de dónde podrían existir posibles debilidades del sistema. Estos pueden obtenerse de evaluaciones de riesgos existentes.
- Atributos de calidad: Permiten identificar mejor las áreas que pueden requerir mayor atención. Por ejemplo: si se procesan información financiera y personal de clientes (Seguridad), o si muchos usuarios usan el sistema al mismo tiempo para comprar boletos (Rendimiento), etc.
- Casos de prueba de alto nivel existentes: Sirven como ejemplos y pueden tomarse de un conjunto de regresión existente.
- Informes de incidentes (tickets de servicio): Representan problemas comunes en el proceso o sistema que deben ser considerados.

Creación de una Estrategia de Pruebas: Esquematizar el Sistema

Puede ser útil incluir (o dibujar) un esquema que describa cómo funciona el objeto de prueba (incluyendo las partes que existen y su entorno). Incluir un esquema del sistema puede ayudar a aclarar qué partes del sistema son nuevas y cuáles ya existen.

También puede ser beneficioso usar un código de colores en el esquema, como se muestra en el siguiente ejemplo:



Este esquema puede utilizarse para asignar niveles y tipos de prueba, lo cual discutiremos en la "Práctica 6: Comenzar pequeño y expandir gradualmente el alcance de las pruebas".

Una vez que tengamos un esquema que explique cómo funciona el sistema y hayamos obtenido una mayor comprensión de la propia aplicación, podría ser beneficioso profundizar nuestro análisis de los riesgos para ajustar el enfoque de nuestra estrategia de pruebas en consecuencia.

Desarrollo de una Estrategia de Pruebas Básica mediante Análisis de Riesgos

Al decidir qué probar, debemos desarrollar una estrategia básica de pruebas basada en riesgos. Existen varias formas de crear este tipo de estrategia, que pueden variar en longitud y nivel de detalle. A continuación, profundizaremos en algunos enfoques básicos que te ayudarán a construir tu estrategia de pruebas basada en riesgos.

Para poder crear una estrategia de pruebas basada en riesgos, es importante primero definir lo que consideramos un riesgo. Según TMap **[XII]**, un riesgo se compone de dos elementos: "probabilidad de fallo" y "daño". La probabilidad de fallo incluye la frecuencia con la que se utiliza el proceso y la complejidad del propio proceso. Considera la siguiente fórmula:

Riesgo = Probabilidad de fallo (Frecuencia + Complejidad) * Daño

Analicemos cómo podemos identificar mejor los riesgos a través de la evaluación de riesgos.

Evaluación de Riesgos para Construir una Estrategia de Pruebas

La evaluación de riesgos **[XIX]** es un proceso básico que consta de tres pasos esenciales para construir una estrategia de pruebas basada en riesgos: identificación, análisis y mitigación.

Una buena fuente para **identificar riesgos** es el cliente (el propietario de la aplicación). Como principal parte interesada, el cliente generalmente comprende mejor el negocio para el cual estamos construyendo (o cambiando) un sistema. Por lo tanto, el cliente debería ser consciente de los riesgos y particularidades del negocio en cuestión y puede ayudar a decidir cuáles son los riesgos más importantes.

Durante el proceso de evaluación de riesgos, también es importante incluir a otros interesados del lado del cliente, como administradores de sistemas, desarrolladores y usuarios clave. Estas otras perspectivas pueden mejorar drásticamente la validez del **análisis de riesgos** en su conjunto. Tienden a tener conocimientos relevantes sobre el mantenimiento, uso y desarrollo de la funcionalidad, lo que les permite elaborar sobre la frecuencia de uso, la complejidad o los problemas comunes con la funcionalidad.

Una vez que los riesgos son analizados, podemos comenzar a enfocarnos en la **mitigación de riesgos**. Las pruebas en sí mismas son una medida de mitigación de riesgos. Existen otras medidas para mitigar los riesgos, por ejemplo, la implementación de monitoreo después del despliegue en el entorno de producción.

Usando el ejemplo anterior de cambiar la funcionalidad de un campo de número de teléfono, nuestro cliente podría confirmar que el 90% de sus pedidos entrantes provienen de Europa.

Con esta valiosa información, sabemos que una funcionalidad bien operativa es muy importante para los números de teléfono europeos. Por lo tanto, podemos usar esta información para dividir nuestra funcionalidad y, por ende, nuestros riesgos.

Consideremos los riesgos que hemos identificado:

Nr.	Riesgo
1	La funcionalidad de números de teléfono dentro de Europa no funciona.
2	La funcionalidad de números de teléfono fuera de Europa no funciona.

Probar todos los formatos de números de teléfono de cada país en Europa ya cubre mucho terreno, por lo que sería beneficioso limitar aún más el alcance. Haciendo preguntas más específicas al cliente sobre los mercados más importantes dentro de Europa, podemos descubrir qué formatos de números de teléfono requieren más pruebas.

Por ejemplo, si nuestro cliente dice que el 70% de los ingresos provienen de Francia, podemos crear subdivisiones adicionales del riesgo:

Nr.	Riesgo
1	La funcionalidad de números de teléfono de Francia no funciona.
2	La funcionalidad de números de teléfono de otros países en Europa no funciona.
3	La funcionalidad de números de teléfono fuera de Europa no funciona.

Para mantener nuestro ejemplo simple, nos detendremos en estos riesgos identificados (en la realidad podrían ser muchos más).

Ahora debemos tomar estos riesgos identificados y analizarlos adecuadamente. El análisis de riesgos implica aprender sobre la probabilidad de fallo y el daño.

La probabilidad de fallo se compone de los factores de frecuencia y complejidad. Con frecuencia, nos referimos al intervalo en el que se utiliza el programa (o proceso). Al determinar la frecuencia de uso, una buena regla general es obtener las cifras de uso de la funcionalidad específica desde la aplicación en el entorno de producción. Si estos datos están disponibles, proporcionarán una visión factual sobre el uso de las diferentes funcionalidades. A menudo, esas cifras se pueden obtener revisando registros o informes de inteligencia empresarial para la funcionalidad mencionada en producción. Al agregar estos datos, obtendrás una comprensión más factual de la frecuencia con la que se utiliza una funcionalidad en particular.

Con complejidad, nos referimos a la dificultad de utilizar un programa (o proceso) determinado, o a la complejidad del propio programa. Si combinamos los resultados de frecuencia y complejidad, llegaremos a una puntuación final para la probabilidad de fallo.

Es importante recordar que el análisis de riesgos es siempre una forma subjetiva de estimar algo, razón por la cual deberíamos involucrar al cliente tanto como sea posible. Esta subjetividad es necesaria porque estamos tratando de asignar un valor a la prioridad que debe darse a los riesgos identificados.

En el análisis de riesgos, es una práctica común usar una escala de tres puntos con los valores “alto” (H), “medio” (M) y “bajo” (L) para todos los componentes. Es importante tener en cuenta que no todo puede ser listado como “alto”, ya que no todo es igualmente importante. Considera la siguiente **matriz de riesgos** para nuestro ejemplo de funcionalidad de campo de número de teléfono:

Risk table			**Function**	Login 2FA email	Login 2FA sms	View insurance details	Apply for insurance	Change insurance
			Frequency	L	H	H	L	M
			Complexity	H	H	L	M	H
			Total	**M**	**H**	**M**	**L**	**H**
Process	**System**	**Damage**						
Car insurance	Webportal, backend	**M**		B	A	B	C	A
Healthcare insurance	Webportal, backend	**H**		A	A	A	B	A
Boat insurance	Webportal, backend	**L**		C	B	C	C	B

Matriz de Riesgo: Funcionalidad del campo telefónico

En esta matriz de riesgos, la parte inferior, “Proceso”, “Sistema” y “Daño”, se refiere a nuestros riesgos subdivididos. “Francia” está listada bajo “Daño” como “H” (alto), porque el 70% de los pedidos provienen de este país. A través de una deducción matemática, si el 10% de los pedidos provienen de “Fuera de Europa”, podemos estimar que el 20% de los pedidos provienen del resto de Europa (“Resto de Europa”). Por lo tanto, “Resto de Europa” está listado bajo “Daño” como “M” (medio) con un 20%, y “Fuera de Europa” está listado bajo “Daño” como “L” (bajo) con un 10%.

Mientras discutíamos el proceso con las partes interesadas, entendimos que la “Frecuencia” es “H” (alta) porque el campo se utiliza en cada pedido para validar la identidad del cliente. Como el cambio de un campo de número de teléfono es un procedimiento relativamente común y directo, uno podría (recordando que es subjetivo) considerar que la “Complejidad” es “M” (media).

La estimación “Total” para la probabilidad de fallo de este riesgo es “M” (media). Al observar la esquina inferior derecha de la matriz de riesgos, las letras “A”, “B” y “C” se refieren al nivel de importancia de la funcionalidad al mitigar el riesgo.

- “A” se refiere a los riesgos que deberían recibir la mayor cobertura para mitigar el daño que tendría el mayor impacto en la empresa.
- “C” se refiere a los riesgos que requieren la menor cobertura (ya que el riesgo general para la empresa es menor).
- “B” se utiliza para los riesgos que se encuentran en un punto intermedio.

Objetivos de Prueba

Los objetivos de prueba, a menudo llamados "metas de prueba", pueden ser una funcionalidad dada, parte de un sistema de software, o incluso un atributo de calidad que debería recibir especial atención durante las pruebas; por lo tanto, no existe una ciencia exacta o una regla fija para deducirlos.

Como hemos visto en nuestro ejemplo de funcionalidad del campo de número de teléfono, durante el análisis de riesgos podemos observar que la división del riesgo en diferentes partes nos proporcionó algunos puntos de vista valiosos sobre la funcionalidad. Estos son: "Pedidos desde Francia", "Pedidos desde otras partes de Europa" y "Pedidos desde fuera de Europa". Aquí hemos listado solo estos, pero considerando una tienda en línea donde se pueden pedir artículos, es evidente que podríamos nombrar otras partes de la tienda en línea como posibles (e incluso valiosos) objetivos de prueba también.

Algunos ejemplos podrían ser: "La facilidad de uso de la interfaz", "El carrito de compras", "El proceso de pago", "Las opciones de pago", "El motor de búsqueda", "El rendimiento", "La API para proveedores externos", "La documentación de la API". Todos estos son ejemplos que podrían volverse importantes si se profundizara más en este ejemplo.

Estrategia de Pruebas Basada en Riesgos

Una vez que sabemos dónde se requieren los diferentes niveles de cobertura, podemos dividir el sistema en diferentes áreas de enfoque (objetivos de prueba) y utilizarlas para crear una estrategia de pruebas. Crear una estrategia de pruebas implica diseñar un enfoque estructurado para las pruebas mediante el uso de diferentes técnicas de prueba que puedan aplicar diversos niveles de cobertura a los distintos objetivos de prueba.

En nuestro ejemplo de funcionalidad del campo de número de teléfono, hemos limitado nuestros objetivos de prueba para mantener el ejemplo lo más simple posible, pero si se estuviera creando una estrategia de prueba real para este caso, tendríamos que considerar que podrían haber muchos más factores a tener en cuenta.

Mientras creas tu estrategia de pruebas y, por ejemplo, analizas informes de incidentes, si se hace evidente que la lista de objetivos de prueba es muy extensa o resulta muy desafiante o aparentemente imposible incluirlos todos en la estrategia de pruebas, el "análisis de Pareto" es un ejemplo de un método que podría ser útil.

Análisis de Pareto (Ejemplo de Método de Evaluación de Riesgos)

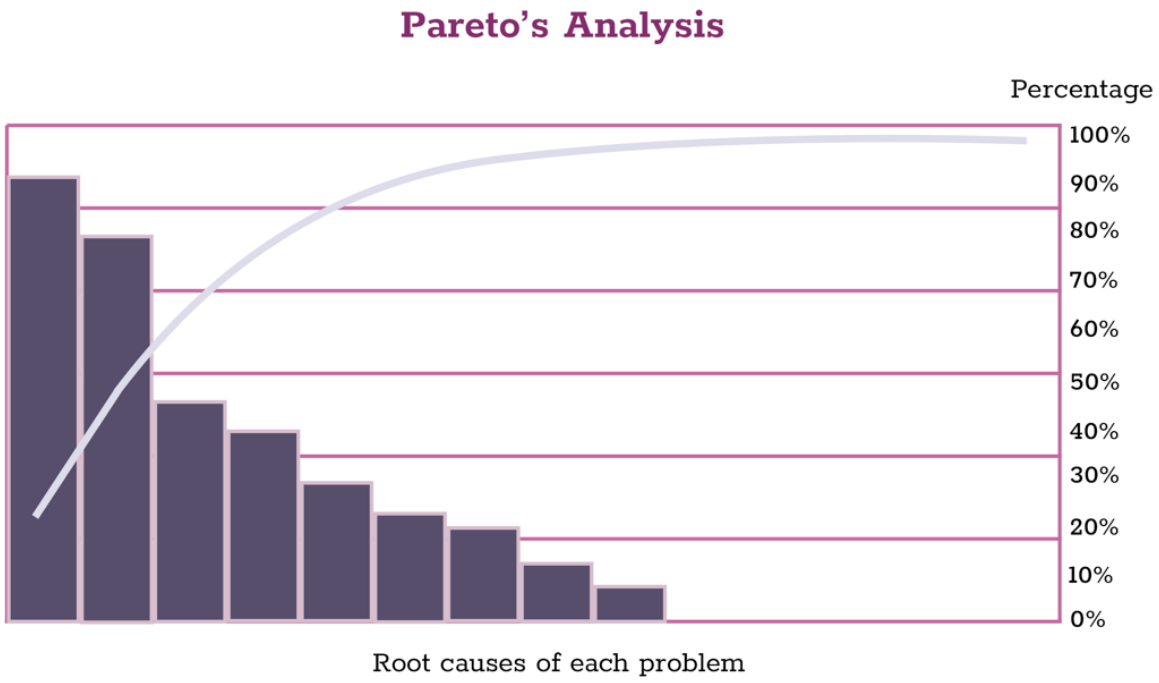
El análisis de Pareto puede hacer que la evaluación de riesgos sea más eficiente **[VI]**. Este análisis se basa en la idea de que el 80% de los beneficios de un proyecto se pueden lograr realizando el 20% del trabajo. Inversamente, el 80% de los defectos pueden estar relacionados con el 20% de las causas.

Pasos para implementar el análisis de Pareto:

1. Identificar y describir los problemas que han ocurrido.
2. Identificar la causa raíz de cada problema.

3. Asignar una puntuación relativa a cada problema.
4. Agrupar causas similares y sumar la puntuación de esos grupos.
5. Una vez identificados los tipos más comunes de problemas, tomar medidas específicas sobre ellos.

Un análisis de Pareto dará lugar a este tipo de diagrama lineal:



Al observar el ejemplo anterior del "Análisis de Pareto", vemos que, con este método, la consideración de que probar todo es imposible se vuelve visualmente clara. Es importante considerar "cuándo el software es adecuado para su propósito".

Análisis de Causa Raíz

El análisis de causa raíz es la identificación de la causa inicial de los defectos, incidentes o problemas. Al registrar las causas raíz y revisarlas periódicamente, puedes observar los tipos de errores que ocurren con mayor frecuencia y cómo podrías evitarlos en el futuro, lo que constituye una herramienta poderosa que puede mejorar significativamente tu proceso de desarrollo de software.

Aplicar el análisis de Pareto a las causas raíz agrupadas puede ser beneficioso al realizar un análisis de causa raíz, ya que ayuda a resolver el 80% de los problemas comunes con, generalmente, el 20% del esfuerzo.

A continuación, se presenta una descripción general de las causas raíz comunes que podrían considerarse al aplicar un análisis de causa raíz:

Category cause	Main cause	Description
Code	Error in code	The functionality is not working because of an error in the code.
	Built differently than designed	The built functionality differs from the description in the design.
	Built more than designed	The built functionality was not described in the design.
	Text & Layout	Errors in spelling, punctuation, design, alignment and usability.
	Forgot to execute	Some of the functionality was forgotten to be developed.
	Unintended modification	Modified code unintentionally changed functionality.
	Merge/integrate error	Due to an integration of different systems, an error occurred.
Design	Design not updated	The draft was not updated in a timely manner.
	Interpretation error	The description of functionality in the design was misinterpreted by the developer.
	Design gaps	The design did not describe a necessary component of functionality.
	Unclear specification	The description of functionality is not clear enough in the design.
	Incorrect specification	There is an error in the specification.
Environment	Hardware	The finding was caused by hardware problems.
	Software	The finding was caused by software problems.
	Configuration	The finding was caused by misconfiguration of (parts of) the test environment.
Test	Incorrect test data	The test data used to run the test case was incorrect.
	Incorrect test case	The test case performed is incorrect.
Unknown		Cause not yet clear.

Independientemente de cómo tú (o tu equipo) decidan desarrollar su análisis de riesgos y estrategia de pruebas, es fundamental recordar que probar todo es imposible. Por esta razón, necesitamos asegurarnos de que estamos enfocando la mayor parte de nuestros esfuerzos de prueba en las partes más importantes y valiosas de nuestro software.

Definiciones

API	Interfaz de Programación de Aplicaciones (API) es un conjunto de reglas y herramientas que permite a diferentes programas de software comunicarse y colaborar entre sí.
Complejidad	El grado en que un componente o sistema tiene un diseño y/o estructura interna que es difícil de entender, mantener y verificar [V].
Daño	La consecuencia adversa de un evento.
Probabilidad de Fallo	La probabilidad estadística de que algo salga mal.
Frecuencia	La frecuencia con la que algo sucede o ocurre dentro de un período de tiempo determinado.
REST	Transferencia de Estado Representacional (REST) es una forma de API donde las computadoras pueden comunicarse de manera simple y estandarizada.
Riesgo	Un factor que podría resultar en consecuencias negativas en el futuro.
Evaluación de Riesgos	El proceso de identificación, análisis y mitigación de riesgos.
Matriz de Riesgos	Una representación gráfica para evaluar los riesgos potenciales que se utiliza para decidir el nivel de cobertura requerido.
Análisis de Causa Raíz	Una técnica de análisis destinada a identificar la causa inicial de los errores (defectos).

SOAP	Protocolo Simple de Acceso a Objetos (SOAP) es un método estandarizado a través del cual las computadoras pueden comunicarse entre sí.
Base de Pruebas	La base sobre la cual se fundamentan las pruebas, incluyendo documentación, un sistema existente, requisitos o el conocimiento de una persona que sabe cómo funciona.
Objeto de Prueba	El producto que se debe probar.
Objetivo(s) de Prueba	Las partes del producto que se deben probar.
Estrategia de Pruebas	Las directrices tácticas generales que explican todos los detalles de lo que necesita ser probado a lo largo del ciclo de vida del desarrollo de software (SDLC).

Práctica 4: Sé Específico

LO17	Aplica la práctica de ser específico. (K3)

Cuando diseñes casos de prueba, requisitos y diseños funcionales, es muy importante ser preciso. Mientras que la programación podría considerarse una ciencia exacta (descomponiendo todo en 1s y 0s), este no es siempre el caso para las pruebas, como veremos en los capítulos siguientes. Es importante invertir este tiempo para tener una base sólida para el desarrollo y las pruebas en el futuro.

Ser específico asegura que tú (como tester) tomes decisiones bien fundamentadas. Si se realiza correctamente, la práctica de "ser específico" también garantiza que estés describiendo las cosas correctas con precisión y con el nivel de detalle adecuado. Si los requisitos que recibiste no son específicos, puedes hacerlos específicos traduciéndolos en casos de prueba. Si surgen preguntas o ambigüedades mientras analizas los requisitos y los traduces en casos de prueba, deben aclararse con el autor de dichos requisitos. Este tipo de comunicación puede conducir a una mejor comprensión y, por ende, a una implementación más precisa de los requisitos.

Un truco que puedes usar al ser específico es SMART, que significa "específico", "medible", "aceptable", "realista" y "limitado en el tiempo". La tabla en la sección de definiciones de este capítulo explica los componentes de SMART **[VII]**.

En algunos casos, podría ser difícil o casi imposible aclarar más los requisitos "vagos" o los casos de prueba. Si este es el caso, una cosa que puedes hacer es describir el riesgo asociado al requisito que no puedes probar adecuadamente o al caso de prueba que no puedes describir correctamente. En otras palabras, debes especificar exactamente lo que no puedes probar o medir porque tu incapacidad para hacerlo aumenta el riesgo.

Por ejemplo, si se te requiere determinar que el fondo de una pantalla permanece "morado" cuando ocurre una acción dada y no te han proporcionado los parámetros específicos de color RGB, ¿cómo puedes determinar que el fondo de la pantalla tiene el tono correcto de morado? Si aceptaras todos los tonos de morado (porque no solicitaste los parámetros exactos de color al cliente), podrías cumplir con el requisito inicial, pero habrías perdido el propósito del mismo, ya que aceptarías todo el espectro comprendido de colores morados. Sin embargo, el tono particular probablemente sea importante para la identidad corporativa del cliente.

Ejemplo de Requisito Empresarial:

Requisito:
"Se crearán flujos de trabajo para las diversas cartas y correos electrónicos. Estos flujos de trabajo se podrán ejecutar en función de las selecciones realizadas utilizando la búsqueda avanzada."

Hay mucha información poco clara en el requisito anterior. La información "vaga" podría aclararse con las siguientes preguntas (entre otras posibles):

- *¿Cuántos flujos de trabajo se crearán?*
- *¿Cuántas cartas hay?*
- *¿Cuántos correos electrónicos hay?*
- *¿Cuántas selecciones hay?*
- *¿Qué es un flujo de trabajo?*
- *¿Qué flujos de trabajo conducen a qué cartas?*
- *¿Qué flujos de trabajo conducen a qué correos electrónicos?*
- *¿Todos los flujos de trabajo conducen solo a cartas?*
- *¿Todos los flujos de trabajo conducen solo a correos electrónicos?*
- *¿Dónde puedo iniciar los flujos de trabajo?*
- *¿Cómo funciona el inicio de los flujos de trabajo?*
- *¿Cuál es la relación entre las selecciones y la búsqueda avanzada?*

Haciendo estas preguntas, obtendrás una comprensión mucho mejor de lo que está pidiendo el requisito, lo que te permitirá entender mejor qué debes probar. Esto ayudará a que el requisito cumpla con los estándares SMART mencionados anteriormente.

El siguiente texto es una mejora de este requisito en particular:

- *Se pueden crear varios flujos de trabajo en la aplicación. Un flujo de trabajo es parte del paquete estándar y puede configurarse para un uso específico. Hay tres flujos de trabajo: flujo de trabajo A, flujo de trabajo B y flujo de trabajo C. El flujo de trabajo A es para la carta de aceptación o el correo electrónico (si el correo electrónico es conocido), el flujo de trabajo B es para la carta de rechazo, y el flujo de trabajo C es para el envío de la factura.*
- *Estos flujos de trabajo pueden ser ejecutados por el rol de "Encargado de Contacto con el Cliente" desde la pantalla "CER012". Esta pantalla muestra todos los flujos de trabajo. Al presionar el botón "Iniciar flujo de trabajo", el flujo de trabajo se activa. Habrá tres botones, uno para cada flujo de trabajo. Los flujos de trabajo pueden estar activos durante un máximo de un minuto.*
- *Al presionar el botón "Iniciar flujos de trabajo", se activa una selección predefinida en la búsqueda avanzada. Estas selecciones son las siguientes: para el flujo de trabajo A: DATOS DE NOMBRE o CORREO ELECTRÓNICO y NOMBRE, FECHA DE CARTA18, FIRMA4; para el flujo de trabajo B: DATOS DE NOMBRE, FECHA DE CARTA14, FIRMA2; para el flujo de trabajo C: DATOS DE NOMBRE, FECHA DE CARTA11, DATOS FINANCIEROS, FIRMA1.*
- *Si un flujo de trabajo no se puede ejecutar o si un flujo de trabajo está activo durante más de un minuto, se debe mostrar un mensaje de error con el siguiente texto: "el flujo de trabajo X no pudo ejecutarse" al usuario que inició el flujo de trabajo. En el lugar de la X, se debe mostrar el nombre del flujo de trabajo en cuestión.*

El texto anterior puede llevar más tiempo generarlo, pero proporciona información específica que es mucho más fácil de comprender con precisión y desarrollar. Esto también facilitará saber qué se debe probar. Es muy probable que ahorre tiempo general de desarrollo del

proyecto, ya que el producto será más probable que cumpla con las demandas precisas del cliente.

Ser Específico: Al Informar un Defecto

Al redactar un informe de defecto, es importante ser lo más específico posible para que los defectos puedan ser reproducidos o sometidos a nuevas pruebas por otras personas.

Un informe de defecto bien especificado debe contener:

- Un título claro
- Una descripción SMART de los pasos para reproducir el defecto
- El resultado esperado
- El resultado real generado
- Prueba del defecto: registros (logs), capturas de pantalla, etc.
- Información relevante: un número único que identifique el(los) caso(s) de prueba asociado(s), el nombre de la pantalla, trazabilidad (o referencia) al requisito dado, el entorno donde ocurre el defecto, el número de versión del software utilizado.
- Gravedad del defecto: “Bloqueante”, “Grave”, “No grave”, “Cosmético”.
- Prioridad del defecto: “Baja”, “Media”, “Alta”.

Dado que generalmente múltiples partes están involucradas en la creación y prueba de nuevo software a lo largo del ciclo de vida del desarrollo de software (SDLC), es importante mantener estándares sólidos de comunicación, como ser específico. Esto ayuda a los miembros del equipo a ahorrar tiempo valioso (y recursos) y a realizar las tareas más rápidamente, manteniendo al mismo tiempo la motivación a lo largo del proyecto.

Definiciones

Específico	Sé lo más detallado posible y evita generalidades. Evita palabras de referencia como “esto”, “estos”, “eso” y “aquellos”, porque si un texto se copia o modifica, el significado puede no ser claro. Siempre formula de manera positiva y singular. Considera formular rutas de error.
Medible	El requisito o caso de prueba debe ser medible. Evita cláusulas vagas como “rápido”, “instantáneo”, “muchos”, “fácil” o “similar”. Vincula una unidad medible a cada caso para determinar si es correcto. Ejemplos: “los datos en la pantalla CER012 deben cargarse completamente en 1 segundo tras abrir la pantalla o solicitar un registro”, o “el proceso debe computar 100,000 transacciones diarias entre las 23:00 y 5:00”.
Aceptable	Un requisito o caso de prueba siempre debe ser aceptado por el cliente. Además, los requisitos deben ser ética y moralmente aceptables y cumplir con los requisitos legales y regulaciones.
Realista	Un requisito o caso de prueba siempre debe ser realista. Se deben formular objetivos alcanzables dentro del rango de tiempo, finanzas y recursos disponibles.
Limitado en el	Muchos requisitos carecen de una indicación de tiempo. Al hacer los

tiempo	parámetros de tiempo concretos, puedes medir mejor si un requisito es satisfactorio. Incluye unidades de tiempo claramente definidas, como días, horas, minutos, segundos, milisegundos, etc.

Práctica 5: Probar lo Antes Posible

LO18	Entender la práctica de probar lo antes posible. (K2)
LO19	Tipos de entornos, incluidos los de pruebas. (K2)
LO20	Comprender la importancia de un buen entorno de pruebas. (K2)
LO21	Comprender la importancia de los datos de prueba. (K2)

La práctica de probar lo antes posible permite beneficios significativos en términos de tiempo y costo. Esta práctica tiene sus raíces en la investigación realizada por Barry Boehm en 1977 **[IX]**. El estudio de Boehm demuestra que el costo de un defecto aumenta exponencialmente cuanto más tarde se detecte el defecto. Ergo, mientras antes se encuentren los defectos, más barato será corregirlos. Considera el siguiente gráfico que muestra los costos de corregir defectos en las diferentes fases del ciclo de vida del desarrollo de software (SDLC):

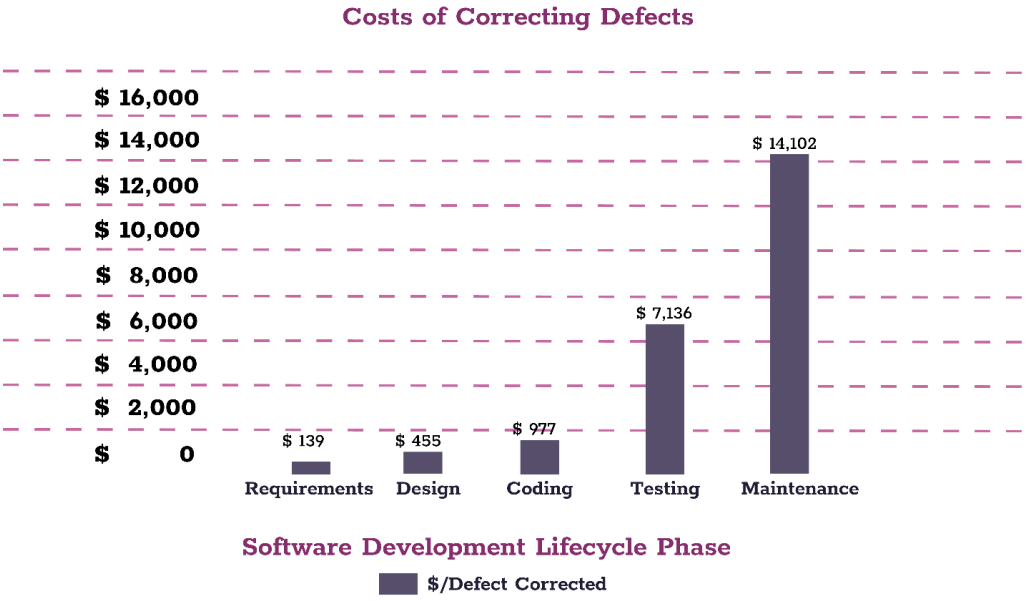


Figura: Costo de corrección de defectos

Como podemos ver arriba, los costos de corrección crecen exponencialmente a lo largo del SDLC. Por lo tanto, si un defecto se encuentra y corrige en las primeras fases del SDLC, uno puede imaginar las consecuencias positivas que esto puede tener en el tiempo y presupuesto general del proyecto.

Sabiendo esto, los esfuerzos para prevenir defectos deberían comenzar lo antes posible. A continuación, se presentan algunos ejemplos de cómo implementar esta práctica de probar lo antes posible:

- Si tú (o tu equipo) no pueden probar dinámicamente, entonces pueden intentarlo estáticamente mediante revisiones (por ti mismo o con varias personas en una

reunión de revisión), en un walkthrough o con una demostración temprana de tu producto.
- Una prueba del sistema en el entorno de desarrollo siempre es más barata y rápida que probar algo en una etapa posterior, por ejemplo, en un entorno de aceptación. Es de tu interés probar dinámicamente tan pronto como puedas, por ejemplo, para evitar que los defectos sean desplegados al entorno de aceptación en primer lugar.
- No pospongas ninguna actividad de prueba con la intención de "volveremos a probar esto más tarde". En su lugar, al probar temprano, incluso si significa que puedes tener que repetir algunas pruebas más adelante, puedes evitar construir sobre código defectuoso que podría permanecer sin ser detectado durante un período de tiempo más largo.

Puede ser tentador e incluso a veces "más divertido" comenzar a desarrollar temprano; sin embargo, recordar revisar los requisitos puede ser increíblemente rentable.

Investigaciones más recientes que las de Boehm también confirman esto. Por lo tanto, como desarrollador o administrador, tenlo en cuenta. Otras medidas que podrías aplicar incluyen hacer que los usuarios revisen los requisitos, realizar revisiones automáticas de código estático y ejecutar pruebas de integración de unidades con otros desarrolladores. Por ejemplo, considera la integración entre el front-end y el back-end de una aplicación web, que a menudo puede realizarse en el entorno de desarrollo pero que en la práctica puede llevarse a cabo más tarde.

Otro punto importante a considerar, que a menudo se subestima a lo largo del SDLC, es asegurarse de que el entorno de pruebas funcione correctamente y esté bien gestionado.

Entornos de Prueba

Por lo general, hay cuatro tipos principales de entornos de prueba (o cuatro tipos principales de entornos en los que se llevan a cabo las pruebas). El entorno que siempre se encuentra primero es el entorno de desarrollo.

El **entorno de desarrollo** es el primer nivel y es utilizado por los desarrolladores para desarrollar el software. En el entorno de desarrollo, típicamente se realizan pruebas técnicas como las pruebas unitarias. Este entorno generalmente está disponible solo para los desarrolladores y, a menudo, tiene poca o ninguna integración con componentes o partes externas de software.

El segundo nivel suele ser el **entorno de prueba**. Este entorno está destinado a ser utilizado para pruebas técnicas, por ejemplo, pruebas de sistema e integración. Por lo tanto, este entorno necesita más componentes externos para que puedan ejecutarse pruebas de integración. Principalmente es utilizado por testers dedicados, aunque los desarrolladores y encargados de mantenimiento también podrían tener acceso.

El tercer nivel suele ser el **entorno de aceptación**. Este entorno es generalmente más extenso que el de desarrollo y/o prueba, ya que incluye casi todos los componentes relevantes del paisaje de software. El objetivo principal del entorno de aceptación es lograr

que el software recién creado o modificado sea aceptado, lo que significa que las pruebas suelen ser realizadas por los usuarios finales. Dado que a menudo es el entorno más completo antes de la etapa de producción, algunos administradores tienden a usarlo para pruebas de instalación y otras pruebas relacionadas con el mantenimiento.

El último nivel es el **entorno de producción**. Generalmente, no se recomienda realizar pruebas en el entorno de producción, ya que esto podría impactar a los usuarios reales en tiempo real. Sin embargo, podría haber una o dos excepciones a esta regla. Por ejemplo, la construcción de un sistema completamente nuevo podría requerir pruebas de rendimiento. La dificultad con las pruebas de rendimiento radica en que se necesita una similitud técnica con el entorno de producción. La similitud técnica es necesaria porque el cálculo de la carga tiende a estar correlacionado con la cantidad o cantidades de componentes técnicos utilizados en el entorno (hardware, software, switches, CPU, memoria). El entorno que incluye todos estos componentes en las proporciones correctas es el entorno de producción. La similitud técnica, siendo clave para el rendimiento, es lo que hace difícil lograr esto en otros entornos. Cuando se trata de pruebas en el entorno de producción, siempre es importante ser extremadamente cauteloso con los riesgos involucrados. Siempre que sea posible, se recomienda realizar cualquier tipo de prueba en los entornos previos a producción.

Estos cuatro entornos (Desarrollo, Prueba, Aceptación y Producción) a menudo se denominan el **principio DTAP**. El principio DTAP implica la creación del software y luego garantizar que se implemente en cada entorno en orden (Desarrollo → Prueba → Aceptación → Producción), ingresando a producción solo después de haber pasado por todos los entornos previos. Al seguir este principio, podemos asegurarnos de que todas las versiones de software en los entornos dados sean las mismas (excepto por los cambios que están en proceso de desarrollo o prueba).

DTAP en Agile y DevOps

El párrafo anterior describe una situación hacia la cual muchas organizaciones con un nivel inicial de madurez en TI y/o pruebas podrían desarrollarse. Este modelo amplía las posibilidades de realizar pruebas extensivas y exhaustivas desde las primeras etapas del proceso de desarrollo de software. Sin embargo, esto no significa que el modelo DTAP sea obligatorio. Si existen suficientes oportunidades para ejecutar los niveles de prueba definidos en tu estrategia de pruebas en menos entornos, como ocurre en equipos Agile, esto también puede ser una buena solución.

También hay organizaciones que siguen un enfoque DevOps. DevOps es una metodología donde el desarrollo y las operaciones se manejan dentro de un mismo equipo, y los usuarios también están representados en estos equipos. Esto a veces conduce a situaciones en las que el modelo DTAP ya no se utiliza completamente. Por ejemplo, considera el desarrollo de aplicaciones donde se implementan nuevas funcionalidades en producción para un número limitado de usuarios, y estas funcionalidades pueden revertirse fácilmente en caso de problemas. En tales casos, no siempre es necesario tener una canalización DTAP completa.

La elección de una canalización DTAP siempre será un equilibrio adecuado entre costos, beneficios y riesgos.

Comprender la Importancia de un Buen Entorno de Pruebas

Uno o más entornos de prueba bien proporcionados y bien gestionados pueden aumentar significativamente la calidad de tu software al permitirte realizar pruebas de manera rápida, precisa y frecuente. En buenos entornos de prueba, puedes eliminar muchas suposiciones que de otro modo podrían no detectarse hasta la etapa de producción.

Un entorno de prueba es importante porque reúne varias prácticas. En el entorno de prueba se combinan requisitos, código desplegado, casos de prueba, datos de prueba y otros software necesarios. Si el entorno de prueba no existe, o si faltan partes, el software no se puede probar y, por lo tanto, se tendrán que hacer suposiciones (por ejemplo, que algunos componentes funcionan), lo cual es contrario a la *Práctica 1: No Hagas Suposiciones*.

Para probar lo antes posible, es necesario asegurarse de que el entorno de prueba esté lo más completo posible, evitando así pruebas tardías en un entorno de nivel superior. Los entornos de nivel superior son más extensos en el SDLC y, por lo tanto, tienden a tener más dependencias, lo que puede limitar severamente su disponibilidad (en términos de tiempo) para pruebas en una etapa posterior. La disponibilidad limitada de entornos de nivel superior puede deberse a muchas razones, como dependencias internas del proyecto o el uso por otros proyectos. Incluso el hecho de que el cronograma se vuelva más ajustado a medida que se acerca la fecha de lanzamiento podría ser un factor importante.

A veces hay razones financieras comprensibles para no crear un entorno de prueba separado. Sin embargo, es importante tener en cuenta que todos los defectos que no se detectan o se encuentran más tarde también cuestan dinero. Algunos de estos costos podrían no ser evidentes de inmediato (relacionados con correcciones adicionales y despliegues), pero pueden surgir mucho más tarde cuando el software ya está en producción, afectando la reputación, las ventas y la calidad general.

Cuando se explican claramente los riesgos de no contar con un entorno de prueba, muchos clientes están dispuestos a invertir más, incluso sin un caso de negocio completamente sólido. Se recomienda encarecidamente hacer que los riesgos sean lo más concretos posible y describir esta urgencia de manera adecuada.

Uso de Stubs y Drivers

Para resolver el problema de no tener todos los componentes disponibles en entornos de nivel inferior, podemos utilizar stubs y drivers para simular esos componentes. La simulación de las partes faltantes del sistema te permite realizar pruebas más extensivas en una etapa temprana y, por lo tanto, encontrar defectos antes en el SDLC. Los stubs y drivers, dependiendo de la situación y tecnología, son relativamente simples de crear y, por ende, pueden ser viables desde el punto de vista del caso de negocio al evaluar los ahorros en los costos de corrección de defectos en una etapa posterior.

Consideraciones clave sobre los entornos de prueba:

- La omisión de un entorno de prueba siempre generará un riesgo. Los riesgos también pueden surgir incluso si solo se omite una parte del entorno de prueba.
- Los riesgos deben hacerse claros a la persona que es, en última instancia, responsable (el cliente).
- Los entornos de prueba deben parecerse lo más posible al entorno de producción.
- Recuerda utilizar stubs y drivers en lugar de los componentes que faltan.
- Respeta el principio DTAP y hazlo cumplir (técnicamente) siempre que sea posible.
- Establece una gestión de configuración y mantén los entornos de manera adecuada. También es importante mantener la gestión de configuración de los componentes relevantes.
- Siempre que sea posible, automatiza la transferencia de software de un entorno a otro. Esto puede prevenir acciones manuales innecesarias que podrían dañar la calidad y previsibilidad de tus despliegues.
- Es beneficioso automatizar la creación de entornos. Al hacerlo, solo necesitas gestionar los componentes de infraestructura una vez, lo que permite la creación sencilla de nuevos entornos específicos para cambios o versiones que pueden descartarse fácilmente. Esto se recomienda especialmente al trabajar con entornos en la nube.

Cuando se trata de entornos de prueba, hay innumerables problemas y situaciones que se pueden encontrar. Podemos imaginar un entorno de prueba como una configuración de laboratorio: una sustancia (software) se está creando mediante un proceso muy preciso (SDLC). Necesitas tener la cantidad adecuada de sustancias originales que deben usarse correctamente: deben calentarse a la temperatura exacta, vaporizarse, etc. En el proceso, no es posible retirar parte de la sustancia original, ya que afectaría el producto final.

Datos de Prueba

La calidad de los datos de prueba disponibles determina en gran medida la calidad de tus pruebas, por lo que contar con datos de prueba de buena calidad es esencial. Siempre que sea posible, es beneficioso utilizar datos de producción. Sin embargo, al trabajar con datos de producción, recuerda considerar la legislación local, como el cumplimiento del GDPR en Europa.

Desde una perspectiva de pruebas, probar con datos de producción es beneficioso porque proporciona muchas variaciones de casos de prueba funcionales. Utilizar una variedad más amplia de datos puede ser una forma completa y eficiente de probar el software en cuestión.

Ya sea que los datos de producción sean una opción o no, es importante asegurarse de que todas las variantes funcionales estén presentes en los datos de prueba. Por ejemplo, si estás probando el software de una póliza de seguro de automóviles, es esencial contar con datos que incluyan todas las variaciones en dirección, tipo de automóvil, años sin reclamaciones, etc.

Para ser efectivo y eficiente, es importante probar lo antes posible, de modo que los defectos se detecten temprano y se evite que persistan a lo largo del SDLC, donde podrían convertirse en defectos más costosos de corregir más adelante.

Definiciones

Principio DTAP	DTAP significa Desarrollo, Prueba, Aceptación y Producción, un principio en el cual el software pasa por todos estos entornos antes de ser puesto en uso.
Revisión	Una forma de prueba estática en la que un producto (intermedio) del SDLC es evaluado por una o más personas.
Stubs y Drivers	Un componente simulado temporalmente que se utiliza para probar otro componente de software. Un driver se coloca antes de un componente y un stub viene después.
Datos de Prueba	Los datos utilizados para la ejecución de pruebas.
Entorno de Prueba	Un entorno validado, usable y estable, que se asemeja lo más posible al entorno de producción final, utilizado para ejecutar casos de prueba o reproducir errores.

Práctica 6: Comienza Pequeño y Expande Gradualmente el Alcance de tus Pruebas

LO22	Entender la práctica de comenzar pequeño y expandir gradualmente el alcance de tus pruebas. (K2)
LO23	Recordar los niveles de prueba y los tipos de prueba. (K1)
LO24	Comprender cómo los niveles de prueba y los tipos de prueba se aplican a una estrategia de pruebas. (K2)

El concepto de "comenzar pequeño y expandir gradualmente" también se considera una práctica recomendada en la gestión de proyectos. Esto también es cierto cuando consideramos la manera más ventajosa de probar software. La práctica asume que has revisado completamente una pequeña parte (unidad) del software para asegurarte de que funciona correctamente.

Las unidades confirmadas como funcionales pueden integrarse con otra unidad del software, siempre que la otra unidad ya haya sido revisada de manera independiente, lo cual aumenta tu alcance. Esto tiene como ventaja que los defectos que surjan de la integración de las dos unidades deben provenir de la integración en sí. Esto simplifica la investigación al buscar la causa de los defectos encontrados.

Esta práctica refuerza la Práctica 5: Probar lo antes posible, ya que puedes probar más temprano y de manera más completa si divides el sistema en unidades más pequeñas para probar.

En general, hay cuatro niveles de prueba:

- Nivel de Prueba 1 - Pruebas Unitarias
- Nivel de Prueba 2 - Pruebas de Integración
- Nivel de Prueba 3 - Pruebas de Sistema
- Nivel de Prueba 4 - Pruebas de Aceptación

Nivel de Prueba 1 - Pruebas Unitarias

La práctica de probar unidad por unidad se llama **pruebas unitarias** y, según los estándares internacionales de pruebas (como el ISTQB), es el primer **nivel de prueba**. Es una práctica común que las pruebas unitarias sean realizadas por desarrolladores en el entorno de desarrollo. Las pruebas unitarias se ejecutan dinámicamente y se llevan a cabo hacia la finalización del desarrollo de la unidad, mientras el software aún está en desarrollo. El alcance y la cobertura de las pruebas unitarias están siempre limitados a la propia unidad.

Nivel de Prueba 2 - Pruebas de Integración

El siguiente nivel de prueba implica la prueba de dos unidades juntas, lo que se denomina **pruebas de integración**. Estas pruebas solo pueden generar defectos que estén relacionados con la integración de las dos unidades, ya que los defectos en ambas unidades deberían

haberse detectado por separado en las pruebas unitarias previas. Seguir este proceso permite que la detección de errores sea considerablemente más sencilla.

Nota: donde usamos el término "unidad", también podríamos usar "componente". Como las pruebas de integración se centran principalmente en la interacción entre dos componentes o sistemas, el alcance debe cubrir todas las posibles interacciones (y situaciones derivadas de ellas) entre las dos unidades.

Nivel de Prueba 3 - Pruebas de Sistema

Las **pruebas de sistema** son el nivel de prueba que generalmente se recomienda después de las pruebas de integración. Estas pruebas suelen realizarse en el entorno de prueba e implican la prueba de diferentes integraciones juntas. Las pruebas de sistema a veces pueden involucrar una cadena de procesos, lo que se conoce como **pruebas de cadena**. El alcance de las pruebas de sistema está orientado hacia el flujo y/o la consistencia de las diferentes integraciones y unidades del sistema.

Nivel de Prueba 4 - Pruebas de Aceptación

El siguiente nivel que generalmente se recomienda es el de **pruebas de aceptación**. Estas pruebas suelen ser realizadas por usuarios dentro de la organización para la cual se desarrolla el sistema. El alcance de las pruebas de aceptación es confirmar si el sistema apoya a los usuarios y a los procesos organizacionales según lo previsto.

Las pruebas de **Extremo a Extremo** (E2E) son una forma de prueba que evalúa los principales escenarios del flujo de un sistema de principio a fin. Las pruebas E2E generalmente se llevan a cabo en el nivel de aceptación del SDLC, donde existen muchas dependencias.

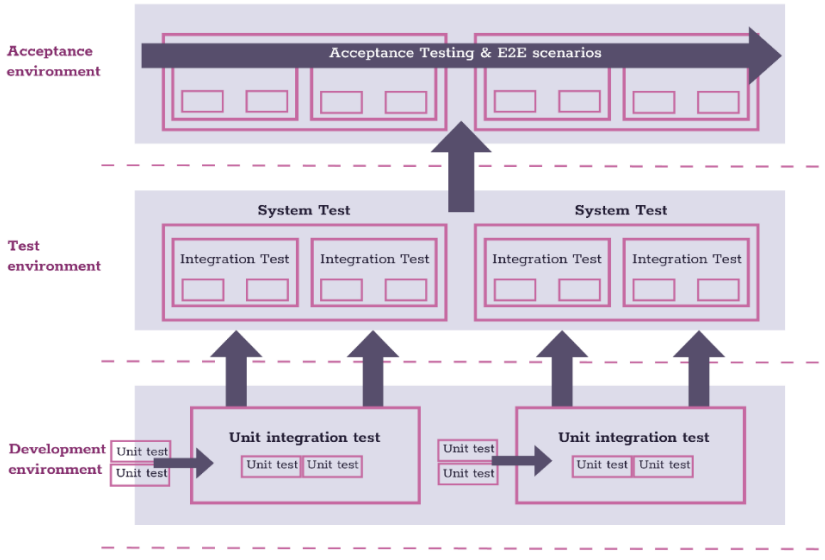


Figura: Niveles de Prueba vs. Entorno de Prueba

En la figura: Niveles de Prueba vs. Entorno de Prueba, cada cuadro (unidad) representa una prueba. Cuando se ha completado la prueba en un cuadro, este se combina con otro cuadro (ya probado). Este proceso continúa siguiendo los niveles de prueba (1-4) explicados previamente, que se construyen uno sobre otro.

Pruebas de Integración del Sistema

Un nivel de prueba que se utiliza frecuentemente, ubicado entre las pruebas de sistema y las pruebas de integración, se llama **Pruebas de Integración del Sistema** (PIS). El software que requiere integración con el software de otros equipos u organizaciones demanda especial atención. Cuando los componentes del software se desarrollan por separado, aumenta la probabilidad de que ocurran malentendidos o fallos de comunicación. Estos errores de comunicación pueden llevar a que partes del software se interpreten de manera diferente, perjudicando la calidad general del software.

Pruebas de Extremo a Extremo (E2E)

Ya hemos visto cómo comenzar pequeño y expandir gradualmente el alcance de las pruebas facilita la detección de errores, pero el principio de probar temprano también se aplica aquí. Una prueba E2E es una forma de prueba tardía y, por lo tanto, relativamente costosa, por lo que debe ejecutarse de la manera más fluida y eficiente posible. Las pruebas E2E solo deben encontrar errores que provengan de la integración de todos los componentes del proceso. Por lo tanto, es de tu interés haber ejecutado completamente los tipos de prueba anteriores antes de comenzar a realizar pruebas E2E.

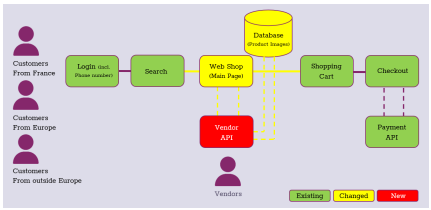
Tipos de Pruebas

Los **tipos de pruebas** se centran en objetivos específicos de prueba, así como en características específicas de un componente o sistema **[V]**. Por lo general, representan ciertas características de calidad (funcionalidad, seguridad, rendimiento, usabilidad, etc.), pero también pueden centrarse en conceptos como regresión o cambios. Algunos de estos tipos de pruebas se desarrollarán más en el siguiente capítulo, que se enfoca en las características de calidad.

Los diversos tipos de pruebas pueden ejecutarse en cualquiera de los cuatro niveles de prueba, pero es útil considerar que ciertos tipos de pruebas son más beneficiosos en ciertos entornos. Los tipos de pruebas suelen decidirse después del análisis de riesgos o el análisis de la base de pruebas, cuando queda claro que ciertos aspectos del sistema requieren atención especial.

Implementación de Niveles y Tipos de Pruebas en la Estrategia de Pruebas

Cuando continuamos nuestro ejemplo de la práctica 3 y construimos nuestra estrategia de pruebas, debemos analizar las diferentes partes del sistema (objetivos de prueba) para decidir qué niveles de prueba aplicar a cada una. Considerando nuestro esquema de ejemplo anterior, nuestros objetivos de prueba son los diversos componentes: “Clientes”, “Inicio de sesión”, “Búsqueda”, “Tienda en línea (Webshop)”, etc.



Recordatorio - Bosquejo de la práctica 3: Probar todo es imposible

Para aplicar los diferentes niveles y tipos de pruebas a nuestros objetivos de prueba, es beneficioso crear una tabla de estrategia de pruebas que utilice la información obtenida en nuestro análisis de riesgos para determinar la cobertura de cada objetivo de prueba en cada nivel de prueba. Es importante considerar qué tipos de pruebas son relevantes al probar tu software en diversos niveles.

Para nuestro ejemplo dado, nuestra estrategia de pruebas podría verse así:

Tabla de Estrategia de Pruebas		**Niveles de Prueba**			
Objetivo de prueba/tipo de prueba	**Prueba Estática**	**Prueba Unitaria**	**Integración**	**Sistema**	**Aceptación**
Pedidos desde Francia	Revisión de Código	Cobertura Alta	Cobertura Alta	Cobertura Media	Incluir en todos los escenarios E2E
Pedidos desde otras partes de Europa		Cobertura Media	Cobertura Media	Cobertura Baja	Probar Escenarios Principales
Pedidos desde otras partes del Mundo		Cobertura Baja	Cobertura Baja	Cobertura Baja	Probar Algunos Escenarios
Inicio de Sesión (Login)	..	..	..	..	..
Búsqueda	..	..	..	..	..
Usabilidad	Prototipado	Revisión Guiada		Seguimiento Ocular	Probar por Grupos de Usuarios
Documentación de la API	Revisión				

Cobertura

En la tabla de estrategia de pruebas, hemos determinado la cobertura para diferentes procesos. Esta cobertura se divide en 'Alta,' 'Media' y 'Baja.' Utilizando varias técnicas de prueba, podemos lograr esta cobertura deseada. En la 'Práctica 7: Documenta tus Pruebas,' prestaremos atención a este aspecto.

Las estrategias de pruebas son un tema muy amplio, y tu conocimiento crecerá a lo largo de tu carrera en pruebas. Si mantienes el enfoque y "comienzas pequeño y gradualmente expandes tu alcance de pruebas," ya estarás en el camino correcto para crear estrategias de prueba bien enfocadas y estructuradas en tu trabajo diario.

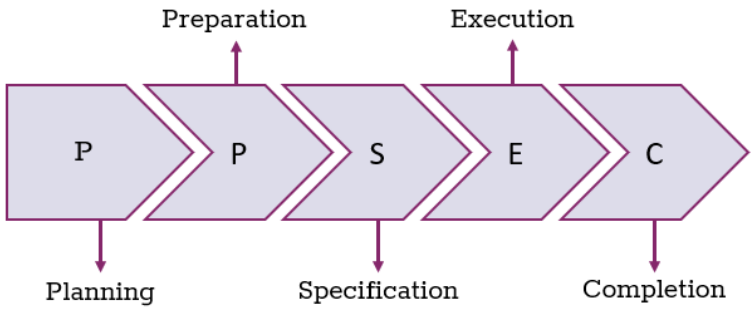
Definiciones

Prueba de Aceptación	Un nivel de prueba destinado a determinar si el sistema es aceptado. **[V]**
Prueba de Cadena	Un tipo de prueba que incluye múltiples sistemas.
Prueba de Extremo a Extremo (E2E Test)	Un tipo de prueba en la que los procesos empresariales se prueban de principio a fin en condiciones similares a producción. **[XXI]**
Prueba de Integración	Un nivel de prueba que se centra en las interacciones entre componentes o sistemas. **[V]**
Prueba de Sistema	Un nivel de prueba diseñado para verificar que un sistema en su conjunto cumple con los requisitos especificados. **[V]**
Nivel de Prueba	Una representación específica de un proceso de prueba. [V]
Tipo de Prueba	Un conjunto de actividades de prueba basado en objetivos específicos de prueba y centrado en características específicas de un componente o sistema, por ejemplo, una prueba de rendimiento o seguridad, o una prueba de aceptación del usuario. **[V]**
Prueba de Unidad/Componente	Un nivel de prueba que se centra en componentes individuales de hardware o software. **[V]**

Práctica 7: Documentar tus Pruebas

LO25	Recordar un proceso básico de pruebas (K1)
LO26	Comprender el valor de documentar tus pruebas (K2)

Las metodologías de prueba reconocidas existentes distinguen varias fases diferentes en el proceso de pruebas, que generalmente incluyen: planificación, preparación, especificación, ejecución y finalización.



Testing Process

Generalmente, entre el 60% y el 70% del tiempo dedicado a las pruebas se destina a los tres primeros procesos (planificación, preparación y especificación) **[XII]**. Las fases de ejecución y finalización de las pruebas en sí mismas suelen constituir solo entre el 20% y el 30% del tiempo. Estos porcentajes indican que una cantidad considerable de tiempo se invierte en la planificación, preparación y especificación.

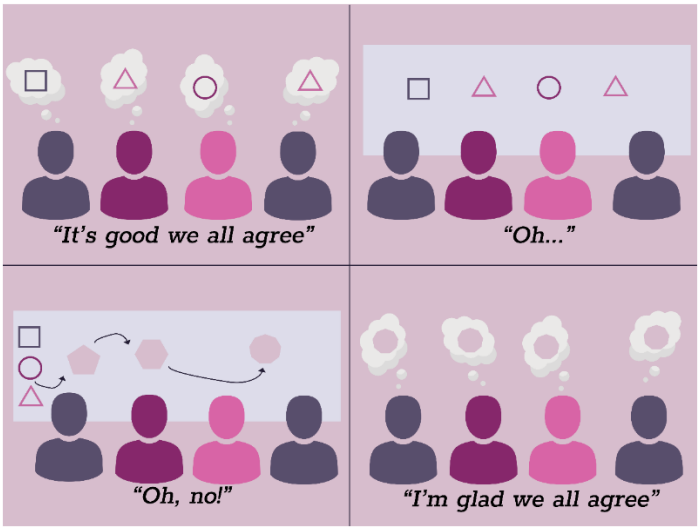
La fase de preparación es crucial, ya que consiste en seleccionar y proponer pruebas adecuadas. Es aquí donde se deciden aspectos importantes de todo el proceso de pruebas, como probar las cosas correctas y en el momento adecuado.

A veces, es necesario elaborar un plan integral con una estrategia de pruebas detallada, en el que se describan en detalle los acuerdos y decisiones relacionados con todo el proyecto de pruebas. Esto puede convertirse en una actividad bastante extensa, con un plan general de pruebas para todos los tipos de pruebas, planes de prueba detallados para cada tipo de prueba, así como acuerdos sobre tiempo, recursos, cronogramas, presupuesto, etc.

Los planes extensos pueden ser valiosos en ciertas circunstancias, especialmente en entornos como grandes corporaciones o instituciones gubernamentales. Una planificación completa también puede ser útil si hay muchas partes diferentes involucradas en las pruebas y se necesita acordar qué parte prueba qué elementos para evitar superposición o duplicación de pruebas, o para prevenir que ciertas partes prueben un área específica.

Sin embargo, ten en cuenta que dichos planes extensos no deben hacerse por costumbre o simplemente porque sea la forma de trabajar en una organización en particular: este tipo de planificación solo debe hacerse cuando aporta valor. Nunca debería ser una prioridad crear un plan general de pruebas excesivamente extenso simplemente por tenerlo, ya que es muy probable que no sea leído.

En otras circunstancias, una estrategia de prueba menos extensa de solo unas pocas páginas será suficiente. Estos esquemas más breves deben documentar tus pensamientos y discusiones con compañeros de trabajo, y cumplen su propósito porque facilitan la comunicación o la referencia a ellos. Este tipo de documentación también ayuda a prevenir malentendidos en los acuerdos.



De ahí la práctica: "Documentar tus pruebas." El significado de esta práctica tiene dos aspectos: primero, documentar tu estrategia de pruebas. Segundo, documentar los casos de prueba que vas a probar o que ya has probado. Para documentar las pruebas, es útil contar con requisitos documentados. Esto puede parecer obvio, pero no siempre es una práctica común. Documentar los requisitos es una excelente medida de calidad. Al documentar los requisitos, haces que la información sea transferible entre personas, evitas malentendidos y puedes crear una comprensión común de los requisitos. Estos requisitos pueden ser refinados más fácilmente y utilizados como base para las pruebas. Por lo tanto, siempre es una buena práctica solicitar que los clientes documenten los requisitos y ofrecer tu ayuda para hacerlo. Esta acción, por sí sola, puede añadir una calidad considerable.

Una buena manera de formular requisitos es convertirlos en historias de usuario en el formato: Como <*rol*> Quiero <*funcionalidad*> Para <*añadir valor*>. Por ejemplo:

*'**COMO** administrador **QUIERO** poder añadir un archivo .PDF a un sitio web **PARA** mostrar este archivo .PDF a los visitantes del sitio web.'* **[X]**

Al formular los requisitos de esta manera, la práctica de “documentar tus pruebas”, junto con la práctica de “ser específico”, nos permitirá crear requisitos comprensibles. Estos requisitos proporcionan posteriormente las pautas para desarrollar, probar y mantener el software.

La cantidad de detalle con la que se documenten los requisitos variará según las organizaciones y proyectos. El nivel de detalle de los requisitos determinará cuán rigurosas y detalladas podrían ser nuestras pruebas, algo a considerar al construir una estrategia de pruebas y seleccionar técnicas de prueba para evaluar ciertas partes del sistema.

Documentar las pruebas derivadas de la base de pruebas ofrece un enfoque estructurado, lo que asegura que no olvidemos casos de prueba relevantes. Documentar exactamente lo que se ha probado hace que las pruebas sean reproducibles y repetibles. La replicabilidad es una parte muy importante de las pruebas. A menudo, otra persona (por ejemplo, el desarrollador o el coordinador de pruebas) tiene que reproducir tu prueba para presenciar o experimentar un defecto específico. Al documentar exactamente lo que hiciste, facilitas y aceleras este análisis del sistema, lo que puede ahorrar tiempo y dinero.

Además, documentar lo que has probado hace que sea más fácil repetir esas pruebas tú mismo. Recordar lo que hiciste a veces es prácticamente imposible debido a la complejidad de la situación o al número de casos y rondas de prueba que realizaste. Realizar grabaciones de pantalla mientras ejecutas tus pruebas es también una buena forma de capturar la ejecución y los resultados de las pruebas, siempre que la grabación sea clara, relevante y que puedas buscar y recuperar fácilmente los diferentes casos de prueba realizados.

Otra razón importante para documentar tus pruebas. A menudo, deseas crear un guion claro en preparación para las pruebas que será utilizado durante la ejecución de las mismas. Por ejemplo, la ejecución de las pruebas solo puede comenzar cuando el software se entrega al entorno de pruebas. Esto podría ocurrir días o semanas después de la creación de los casos de prueba. Por lo tanto, al crear el guion de prueba antes de la ejecución, las pruebas pueden ser realizadas de forma independiente por otros miembros del equipo (o cualquier otra persona) diferente al autor.

Razón final para documentar: Tener una predicción precisa y exacta del resultado. Debemos escribir claramente nuestra predicción de los resultados antes de realizar las pruebas. De esta manera, no estará sesgada por el resultado real de la prueba, y será más probable notar desviaciones respecto a la predicción inicial. Al documentar, es menos probable que incluyas el comportamiento del producto observado durante las pruebas en el guion. Es importante recordar que el comportamiento observado del producto durante las pruebas no siempre es el comportamiento correcto. Si el comportamiento esperado se establece de antemano, las posibilidades de incluir un comportamiento no deseado en el guion de prueba son menores, y es más probable detectar defectos.

Por ejemplo, piensa en construir un artículo de IKEA. Si sigues cuidadosamente las instrucciones, obtendrás el resultado correcto (la construcción adecuada del artículo). Si simplemente comienzas a juntar piezas sin seguir las instrucciones, hay una alta probabilidad de que termines con algunas piezas al revés o mal colocadas.

Uso de Técnicas de Prueba

Existen varias formas de crear diseños y guiones de prueba, conocidas como “técnicas de prueba”. Diferentes técnicas de prueba se aplican a distintas situaciones, pruebas y componentes de software, algunas de las cuales se discuten a continuación. A continuación, se enumeran varios aspectos importantes a considerar al aplicar técnicas de prueba. Estos se aplican al usar una técnica de prueba y al crear un guion de prueba:

- Aplica las prácticas de prueba.
- Diseña tus casos de prueba antes de ejecutarlos, en lugar de hacerlo al mismo tiempo.
- Usa una técnica para crear un diseño de prueba de alto nivel.
- A partir del diseño de prueba de alto nivel, añade más detalle al crear los casos de prueba.
- Lingüísticamente, los casos de prueba deben formularse de manera singular.
- Separa un caso de prueba lógico de un caso de prueba físico.
- Considera las condiciones iniciales (precondiciones).
- Documenta una predicción del resultado.
- Documenta el resultado.

Los elementos enumerados anteriormente se explicarán con más detalle en las secciones siguientes, que se centran en varias técnicas de prueba.

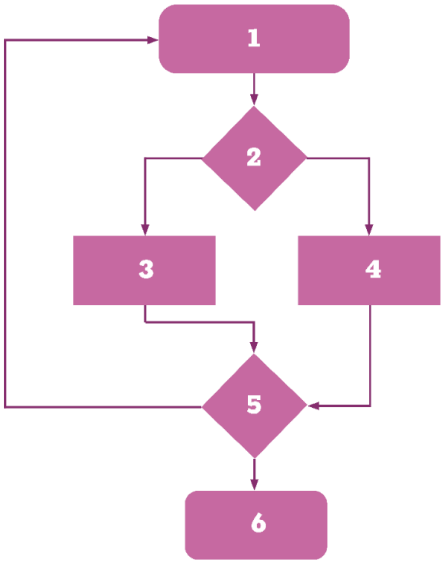
Prueba del Flujo de Proceso

LO27	Aprenda a crear un script de prueba mediante el diseño del proceso/programa. (K3)

La primera técnica de prueba que vamos a discutir se llama la técnica de "flujo de proceso" **[XI]**. Esta técnica es útil para probar flujos en código, aplicaciones y procesos o cadenas de procesos.

Las pruebas de flujo de proceso te permiten delinear la estructura de un programa o proceso, incluyendo momentos de decisión, así como caminos alternativos y de error. La técnica proporciona una visión clara de los caminos dentro de un proceso o programa. Esto te permite verificar si el flujo principal de las decisiones más importantes funciona correctamente, así como los caminos de error más relevantes. Vale la pena mencionar (aunque pueda parecer innecesario) que dicho diagrama de proceso también puede dibujarse basándose en el código del programa.

Ejemplo de Flujo de Proceso:



Descripción:

1. Abrir la aplicación móvil y mostrar la pantalla de inicio.
2. ¿Es el cliente un cliente existente?
3. Sí, mostrar la pantalla de inicio de sesión.
4. No, mostrar la pantalla de creación de cuenta.
5. ¿Inicio de sesión (Crear cuenta e iniciar sesión) exitoso? Sí -> 6 No -> volver al paso 1.
6. Mostrar pantalla de continuación.

Pasos para Construir un Diagrama de Flujo de Proceso

Comienza analizando la información, el proceso o el código del programa. Al hacerlo, puedes dibujar un diagrama similar al ejemplo de flujo de proceso. Durante el proceso de diagramación, podrías encontrar dificultades para conectar las líneas, lo que podría significar que has encontrado un defecto. Mientras construyes el diagrama, intenta ser lo más exhaustivo posible. En las secciones donde falte información o no puedas conectar las líneas, recopila información preguntando a los miembros del equipo y/o al cliente sobre el comportamiento esperado de la parte o camino faltante.

En el diagrama de flujo de proceso, un rectángulo representa una acción y un rombo representa una decisión. Una línea con una flecha se dibuja desde la acción o decisión hacia la acción o decisión que le sigue. Dibujar el diagrama de esta manera facilita determinar qué caminos existen en el proceso o código y si es posible probarlos.

Por ejemplo, imagina una aplicación móvil en la que deseas probar el proceso de inicio de sesión. Este proceso está representado esquemáticamente en el diagrama de flujo de proceso mencionado anteriormente.

Dibujar un diagrama de esta forma te permite aclarar cuáles flujos existen y cuáles puedes probar. El siguiente paso, etiquetado como "Descripción" en este ejemplo, incluye una versión escrita en un lenguaje comprensible. Esto hace que el proceso sea aún más claro y debería permitir que cualquier miembro del equipo pueda ejecutar la prueba también.

Creación del Guion de Prueba

Al crear el guion de prueba, el primer paso es escribir todos los pasos de alto nivel de los diferentes flujos deducidos del diagrama.

Caso de Prueba	**Esquema**	**Descripción**
1	1-2-3-5-6	Intento exitoso de inicio de sesión; cliente existente.
2	1-2-3-5-1	Intento fallido de inicio de sesión; cliente existente.
3	1-2-4-5-6	Intento exitoso de inicio de sesión; cliente nuevo.
4	1-2-4-5-1	Intento fallido de inicio de sesión; cliente nuevo.

Una vez que estos pasos han sido delineados, habrás creado casos de prueba de alto nivel. Estos casos de prueba se llaman "casos de prueba lógicos", ya que describen únicamente la lógica utilizada para construirlos. Después de completar esta etapa, puedes proceder a crear los pasos detallados y escribir los casos de prueba detallados.

Caso de Prueba 1: Intento exitoso de inicio de sesión; cliente existente	
Paso 1: Abrir la pantalla con un cliente existente. (pre-condición)	
Paso 2: Mostrar la pantalla de inicio de sesión. (pre-condición)	
Paso 3: Iniciar sesión con credenciales válidas. (pre-condición)	
Prueba: Verificar que se muestre la pantalla de continuación (predicción del resultado).	
Resultado (Esperado)	Ok/No Ok

Con este nivel de detalle, es fácil identificar las diferentes partes del caso de prueba. Estas diferentes partes son las condiciones iniciales o "pre-condiciones". En este caso de prueba, las pre-condiciones son los pasos 1-3: describen las condiciones que deben cumplirse para permitir la ejecución del caso de prueba. La siguiente parte es la prueba en sí, junto con la predicción del resultado. La parte final del caso de prueba es el (resultado) esperado, en el cual la conclusión de la predicción del resultado sería verdadera (Ok) o falsa (No Ok). Ahora hemos explicado un caso de prueba detallado. Este caso de prueba, al igual que su versión de nivel superior, todavía se denomina caso de prueba lógico. Para ser exhaustivos, los otros casos de prueba detallados son los siguientes:

Caso de Prueba 2: Intento fallido de inicio de sesión; cliente existente	
Paso 1: Abrir la pantalla con un cliente existente.	
Paso 2: Mostrar la pantalla de inicio de sesión.	
Paso 3: Iniciar sesión con credenciales inválidas.	
Prueba: Verificar que se regrese a la pantalla de inicio (predicción del resultado).	
Resultado (Esperado)	Ok/No Ok

Caso de Prueba 3: Intento fallido de inicio de sesión; cliente nuevo	
Paso 1: Abrir la pantalla de cliente nuevo.	
Paso 2: Mostrar la pantalla de inicio de sesión.	
Paso 3: Iniciar sesión con credenciales inválidas.	
Prueba: Verificar que efectivamente se regrese a la pantalla de inicio (predicción del resultado).	
Resultado (Esperado)	Ok/No Ok

Caso de Prueba 4: Intento exitoso de inicio de sesión; cliente nuevo	
Paso 1: Abrir la pantalla de cliente nuevo.	
Paso 2: Mostrar la pantalla de inicio de sesión.	
Paso 3: Iniciar sesión con credenciales válidas.	
Prueba: Verificar que se muestre la pantalla de continuación (predicción del resultado).	
Resultado (Esperado)	Ok/No Ok

Una vez que los casos de prueba han sido descritos, pueden ejecutarse, pero para ello es necesario buscar en la base de datos los datos correctos para implementarlos. En el caso de ejemplo, estos son los datos del cliente. Esto puede hacerse con datos existentes o datos manipulados para cumplir con las precondiciones de los casos de prueba. Los casos de prueba en los que se han implementado los datos correctos se denominan "casos de prueba físicos".

Técnica de Prueba Semántica

LO28	Aprender a crear un guion de prueba utilizando la técnica de prueba semántica. (K3)

Otra técnica de prueba que puedes utilizar para crear casos de prueba es la técnica de prueba semántica [XII]. Esta técnica utiliza las palabras SI (IF), ENTONCES (THEN) y SINO (ELSE) para describir casos de prueba. Las pruebas semánticas son útiles para probar funcionalidades simples y de complejidad media. Las pruebas semánticas también se adaptan bien para probar requisitos individuales, así como requisitos que son co-dependientes de otros requisitos, por ejemplo, diferentes requisitos de control en pantalla como: "la edad del cliente debe ser < 16" o "la verificación de crédito del cliente debe dar un resultado positivo".

Para dar un ejemplo de uso, utilizaremos la misma descripción de funcionalidad utilizada para la técnica de flujo de proceso:

Descripción:

1. Abrir la aplicación móvil y mostrar la pantalla de inicio.
2. ¿Es el cliente un cliente existente?
3. Sí -> Mostrar la pantalla de inicio de sesión.
4. No -> Mostrar la pantalla de creación de cuenta.
5. ¿Inicio de sesión exitoso? Sí -> 6; No -> Volver al paso 1.
6. Mostrar la pantalla de continuación.

Usando este ejemplo, llegarías al siguiente guion de prueba semántico:

1.	SI se abre la aplicación ENTONCES mostrar la pantalla de inicio SINO no realizar ninguna acción
2.	SI el cliente es un cliente existente ENTONCES mostrar la pantalla de inicio de sesión SINO mostrar la pantalla de creación de cuenta.
3.	SI el cliente es un cliente existente con un inicio de sesión exitoso ENTONCES mostrar la pantalla de continuación SINO mostrar la pantalla de inicio.

Con co-dependencia, sería así:

SI se abre la aplicación

```
ENTONCES mostrar la pantalla de inicio
    SI el cliente es un cliente existente
    ENTONCES mostrar la pantalla de inicio de sesión
        SI hay un intento exitoso de inicio de sesión
        ENTONCES mostrar la pantalla de continuación
        SINO mostrar la pantalla de inicio
    SINO mostrar la pantalla de creación de cuenta
SINO no realizar ninguna acción
```

Observa la alineación aquí: SI, ENTONCES y SINO están relacionados entre sí. Esto se denomina una declaración anidada y también se utiliza en el código de programación. Además, ten en cuenta que un SI siempre tiene un ENTONCES y un SINO. De esta manera, es posible describir todas las posibilidades de la funcionalidad.

El SI describe una pre-condición: una condición que un caso de prueba debe cumplir para poder ejecutarse. El ENTONCES describe la acción que debería ocurrir cuando se cumple la condición. Finalmente, el SINO describe lo que debería suceder si la condición no se cumple, y hace referencia a un camino de error o una funcionalidad subsecuente.

Ahora que hemos descrito los casos de prueba como casos de prueba de alto nivel, podemos continuar, como hicimos con la técnica de flujo de proceso, detallándolos aún más:

Nr.	**Caso de prueba de alto nivel**	**Nr.**	**Caso de prueba detallado**
1.	SI se abre la aplicación ENTONCES mostrar la pantalla de inicio SINO no realizar ninguna acción.	1.	Verificar que se muestre la pantalla de inicio.
		2.	Comprobar que no se muestre ninguna pantalla cuando no se realiza ninguna acción.
2.	SI el cliente es un cliente existente ENTONCES mostrar la pantalla de inicio de sesión SINO mostrar la pantalla de creación de cuenta.	3.	Verificar que la pantalla de inicio de sesión se muestre para un cliente existente.
		4.	Asegurarse de que la pantalla de creación de cuenta se muestre para un cliente nuevo.
3.	SI el cliente es un cliente existente con un inicio de sesión exitoso ENTONCES mostrar la pantalla de continuación. SINO mostrar la pantalla de inicio.	5.	Verificar que, tras un intento exitoso de inicio de sesión, se muestre la pantalla de continuación.
		6.	Comprobar que, tras un intento fallido de inicio de sesión, se muestre la pantalla de inicio.

A continuación, podemos crear los pasos más detallados del caso de prueba:

```
Caso de Prueba Detallado 1: Verificar que se muestre la pantalla de inicio.
```

Paso 1: Abrir la aplicación.	
Paso 2: Mostrar la pantalla de inicio.	
Verificación: Que la pantalla de inicio se muestre (predicción del resultado).	
Resultado	Ok/No Ok

Caso de Prueba Detallado 2: Comprobar que no se muestre ninguna pantalla cuando no se realiza ninguna acción.	
Paso 1: Instalar la aplicación, pero no abrirla.	
Verificación: Que no se abra nada (predicción del resultado).	
Resultado	Ok/No Ok

El segundo caso de prueba puede parecer redundante; esto puede ser cierto en este ejemplo, pero a menudo tiene sentido verificar explícitamente que no se realice ninguna acción.

Tablas de Decisión

LO29	Aprender a probar funcionalidades utilizando una tabla de decisión (K3)

Las tablas de decisión [VIII] pueden utilizarse para probar la estructura y lógica de un programa o proceso. En las tablas de decisión, se diferencian las condiciones que se cumplen o no, y se describen las acciones resultantes de esas decisiones. De esta manera, es posible verificar adecuadamente la funcionalidad completa. Las tablas de decisión son una técnica compleja que permite realizar pruebas exhaustivas; a menudo se utilizan para probar procesos complicados o en entornos de alto riesgo, donde es importante distinguir todos los posibles resultados.

Las tablas de decisión se construyen escribiendo todas las condiciones. Al formular las condiciones en una tabla de decisión, es importante tener en cuenta algunos aspectos. Las condiciones deben formularse en un lenguaje simple y en términos singulares. Además, deben redactarse de manera que el resultado de la pregunta sea fácilmente comprensible si se responde con “Sí” o “No”. De esta forma, la tabla será fácil de entender y replicar.

Ejemplo de entrada a una atracción de parque de diversiones:

Tienes el siguiente requisito: “Si la edad > 15 o la altura > 135, entonces tienes acceso a la atracción.”

Si tienes un boleto VIP, tienes acceso directo a la atracción. Esto conduce a la siguiente tabla de decisión:

Condiciones								
Edad> 15 años								
Altura > 135 cm								
Boleto VIP								
Acciones								

...								

El siguiente paso es completar todas las acciones que resultan de las condiciones. Una vez que las acciones están listadas, puedes llenar las decisiones con 'S' (Sí) y 'N' (No). Excel es una buena herramienta para ayudarte con esto. En una tabla de decisión, las condiciones son booleanas (S/N). El número de "diferentes decisiones posibles" es, por lo tanto, 2 elevado al número de condiciones. Por ejemplo, con 3 condiciones en nuestra tabla, será 2 elevado a la 3, es decir, $2^3 = 2 \times 2 \times 2 = 8$. Con 4 condiciones, $2^4 = 2 \times 2 \times 2 \times 2 = 16$.

Una vez que las columnas han sido determinadas, puedes llenar las columnas con 'S' y 'N'. Una forma sencilla de hacerlo es llenar la primera mitad de la primera condición con 'S' y la segunda mitad con 'N'. En nuestro ejemplo, tenemos 3 condiciones, lo que resulta en 8 columnas (2 x 2 x 2), donde las columnas 1-4 serían 'S' y las columnas 5-8 serían 'N'.

Condiciones	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Edad > 15 años	S	S	S	S	N	N	N	N
Altura > 135 cm								
Boleto VIP								
Acciones								
Acceso Directo								
Conceder Acceso								
Sin Acceso								

Una vez que hayas llenado la primera condición (fila), puedes pasar a la siguiente condición. Una buena regla a seguir es copiar las decisiones de la condición anterior en la fila siguiente y cambiar la segunda mitad de las porciones consecutivas de 'S' a 'N', y la primera mitad de las porciones consecutivas de 'N' a 'S', como se muestra aquí:

Condiciones	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Edad > 15 años	S	S	S	S	N	N	N	N
Altura > 135 cm	S	S	N	N	S	S	N	N
Boleto VIP								
Acciones								
Acceso Directo								
Conceder Acceso								
Sin Acceso								

Puedes continuar con las siguientes condiciones hasta llegar a la última condición, como se muestra aquí:

Condiciones	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Edad > 15 años	S	S	S	S	N	N	N	N
Altura > 135 cm	S	S	N	N	S	S	N	N
Boleto VIP	S	N	S	N	S	N	S	N
Acciones								

Acceso Directo								
Conceder Acceso								
Sin Acceso								

Luego, puedes emparejar el resultado de las diferentes condiciones con una acción. En nuestro ejemplo, una persona debe tener 15 años o más, o medir más de 135 cm para obtener acceso a la atracción. Además, es necesario tener un boleto VIP para acceder directamente a la atracción. La columna 1 describe a una persona que tiene 15 años o más, mide más de 135 cm y también tiene un boleto VIP. Por lo tanto, se debe conceder acceso directo al individuo descrito en la columna 1. Escribe una 'X' en el campo que indique la acción correcta.

A continuación, puedes completar la segunda columna. Dado que el requisito de "Boleto VIP" es un 'N', no se concederá acceso directo a un individuo que cumpla con estos criterios. Este individuo debe esperar en la fila para el acceso habitual. Todas las demás acciones en la tabla deben completarse ahora hasta que haya una acción para cada columna. Ahora tienes los 8 casos de prueba para probar la funcionalidad exhaustivamente, como se muestra aquí:

Condiciones	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Edad > 15 años	S	S	S	S	N	N	N	N
Altura > 135 cm	S	S	N	N	S	S	N	N
Boleto VIP	S	N	S	N	S	N	S	N
Acciones								
Acceso Directo	X		X		X			
Conceder Acceso		X		X		X		
Sin Acceso							X	X

Análisis de Valores Límite

LO30	Aprender a profundizar tus pruebas utilizando el análisis de valores límite. (K3)

El análisis de valores límite **[XIV & XV]** es un método para cubrir los límites de valores en campos de entrada, condiciones, APIs, procesos por lotes y complementos en tus pruebas. En este método, se busca específicamente el límite para determinar distinciones en las diferencias de funcionalidad utilizada en el software. Realizar un análisis de valores límite te permite verificar de manera adecuada la completitud y el correcto funcionamiento de la funcionalidad.

Para ilustrar el análisis de valores límite, imaginemos el uso de un servicio a demanda con distinción por edad para contenido violento. Una de las reglas es que una persona debe tener 18 años o más para ver cierto contenido. Esto podría verificarse probando con una edad seleccionada al azar por encima de 18, como 23, pero quizás una mejor alternativa sería verificar con una edad de 17 (sin acceso), 18 (acceso) y 19 (acceso). Usarías estos valores específicamente porque muchos errores surgen de valores incorrectos en los límites: considera el uso de los símbolos '<', '>', '=', '<=', y '>=' en el código de programación. Una buena forma de aplicar el análisis de valores límite es crear 3 casos de prueba por límite: un caso de prueba en el límite, un caso de prueba justo a la izquierda (probablemente menor que) del límite y un caso de prueba justo a la derecha (probablemente mayor que) del límite.

Veamos otro ejemplo de ciudadanos mayores que reciben un descuento en el transporte público para ilustrar cómo se puede aplicar el análisis de valores límite. Tomemos el siguiente requisito: *'Los ciudadanos de 65 años o más reciben un 20% de descuento en la tarifa de su billete de transporte público.'*

Cuando aplicamos el análisis de valores límite a esto, dará lugar a 3 casos de prueba por límite, generando los siguientes casos de prueba:

- 64, 65, 66 (invalido, valido, valido)

Particionamiento en Clases de Equivalencia

LO31	Aprender a profundizar tus pruebas utilizando el particionamiento en clases de equivalencia. (K3)

El particionamiento en clases de equivalencia, o clases de equivalencia **[XIII & XIV]**, es una técnica de prueba que distingue diferentes clases de valores de entrada u otros requisitos para una aplicación. Los valores dentro de las mismas clase(s) a menudo conducen al mismo tipo de procesamiento. Se distingue entre clases de equivalencia "válidas" y clases de equivalencia "inválidas". Los valores de entrada de las clases de equivalencia válidas se procesan correctamente, mientras que los valores de entrada de las clases de equivalencia inválidas podrían generar un mensaje de error. Al aplicar este principio, al menos un caso de prueba debe basarse en cada clase de equivalencia separada.

Tomemos el ejemplo de niños que acceden a una estructura de juego especial. Un control de entrada sobre la edad tiene el siguiente requisito: 6 < edad < 12 (la edad debe ser mayor que 6 pero menor que 12).

En este ejemplo, hay tres clases de equivalencia:

- Edad > 6 años
- Edad en el rango de 7 a 11 años
- Edad < 12 años

Al aplicar las clases de equivalencia genera los siguientes casos de prueba:

- Edad = 4 años (inválido)
- Edad = 8 años (válido)
- Edad = 15 años (inválido)

Es común en el particionamiento en clases de equivalencia que haya menos casos de prueba en comparación con el análisis de valores límite. En otras palabras, la cobertura de los casos de prueba aplicados en el análisis de valores límite es mucho mayor. En el ejemplo del acceso a la estructura de juego, tendríamos 5, 6, 7, 11, 12 y 13 como límites en el análisis de valores límite, mientras que solo tenemos 3 en el particionamiento en clases de equivalencia. No obstante, el particionamiento en clases de equivalencia es útil si deseas realizar pruebas con una cobertura ligeramente menor o si necesitas probar valores de entrada alfanuméricos, por ejemplo, al considerar diversos documentos utilizados para identificación: pasaporte, licencia de conducir, tarjeta de identificación, etc.

Técnica de Prueba Basada en Listas de Verificación

LO32	Aprender a realizar pruebas utilizando una lista de verificación. (K3)

La técnica de prueba basada en listas de verificación **[XVI]** implica la creación de una lista de verificación que sirva como base para las pruebas. Ejemplos prácticos de la aplicación de esta técnica se pueden observar en actividades repetitivas como la entrega de software a un entorno de pruebas, aceptación o producción. Ciertos aspectos relacionados con la seguridad también son problemas que pueden abordarse fácilmente con una lista de verificación. En la lista de verificación, anotas los controles que realizas en una columna, junto a columnas con las opciones 'Ok' y 'No Ok' para ser marcadas según corresponda. Las listas de verificación pueden crearse basándose en la experiencia y ampliarse agregando las causas de incidentes previos. Usar listas de verificación en los procesos proporciona predictibilidad y consistencia; además, son una excelente forma de capturar conocimientos de empleados con mucha experiencia y transferir ese conocimiento a empleados junior.

Ejemplo de la técnica de prueba basada en listas de verificación:

Nr,	Description of test case	Result (Ok/Not ok)	Note
1.	Verify that the login screen is displayed.	Ok	
2.	Make sure that the "login" field is displayed.	Ok	
3.	Make sure that the "password" field is displayed.	Ok	
4.	Verify that the entries in the "login" field with 35 positions are accepted.	Ok	
5.	Check that the entries in the "login" field with 36 positions are not accepted.	Not ok	The login field allows more than 36 positions, see defect 123.
6.	Verify that the entries in the "password" field with 35 positions are accepted.	Ok	
7.	Check that the entries in the "password" field with 36 positions are not accepted.	Not ok	The password field allows more than 36 positions, see defect 138.
8.	Check that if a user enters a password of fewer than 16 characters, the message "your password must be at least 16 characters" appears.	Ok	
9.	Check that if a user enters a password with no special characters, the message "your password must contain at least 1 of the following special characters -_()*&^%$#@!?/><" appears.	Ok	
10.	Check that if a user enters a password without a capital letter, the message "your password must contain at least 1 capital letter" appears.	Ok	
11.	Verify that a user can log in with the role "administrator"		
12.	...		

Con las listas de verificación, los empleados con menos experiencia pueden revisar las mismas cosas que sus colegas más experimentados. Las pruebas basadas en listas de verificación pueden combinarse bastante bien con otras técnicas de prueba como la matriz CRUD, el análisis de valores límite, el particionamiento en clases de equivalencia o la técnica de prueba semántica. Sin embargo, una vez realizadas, las listas de verificación deben mantenerse regularmente, ya que algunas de las acciones repetitivas podrían cambiar con el tiempo.

Usos prácticos de la técnica de prueba basada en listas de verificación:

- Diseño de Interfaz.
- Valores de entrada (pantallas y archivos).
- Valores de salida.
- Validación de archivos y campos.
- Mensajes de error correctos.
- Campos faltantes.
- Longitud de campo incorrecta.
- Tipo de campo incorrecto (tipo de dato).
- Posición incorrecta.

Otra aplicación práctica de esta técnica sería verificar la completitud de la entrega a otro entorno. Cuando se construye una aplicación, a menudo consta de diferentes componentes y archivos. Después de su creación, estos componentes deben recopilarse para su implementación en el siguiente entorno. Aunque existen herramientas que pueden ser de ayuda, todavía podrían realizarse acciones manuales, y cada una de estas acciones representa un riesgo para la calidad del software, así como para las pruebas en el siguiente entorno.

Por lo tanto, en estas situaciones, una lista de verificación en la que se describan meticulosamente todas las acciones necesarias puede ser útil. Además de crear una lista de

verificación, también puedes utilizar listas de verificación existentes para probar aplicaciones. Por ejemplo, hay excelentes listas de verificación en línea disponibles para seguridad y usabilidad que puedes incorporar fácilmente en tu proyecto.

Técnica de Prueba por Pares (Pairwise Testing)

LO33	Aprender la técnica de prueba por pares. (K3)

La prueba por pares es un método para encontrar defectos combinando dos valores de una variable. Esta técnica de prueba asume que la mayoría de los defectos son causados por un solo factor o por la interacción de dos factores diferentes. Cuando probar todas las posibles combinaciones de valores de entrada sería imposible o muy tedioso, la prueba por pares puede ser una forma efectiva de probar cada combinación de dos factores aleatorios y fomentar una mayor variedad de datos de prueba.

Tomemos un ejemplo para ilustrar cómo funciona la prueba por pares. Imaginemos un sistema con tres variables de entrada definidas: “formato de archivo”, “nivel de acceso” y “rol”. Estas variables definidas tienen varios valores listados en la siguiente tabla:

Formato de archivo	**Nivel de acceso**	**Rol**
.pdf	Verde	Empleado
.gif	Naranja	Administrador
.docx	Púrpura	
.jpeg		

En total, tenemos 4 x 3 x 2 valores, lo que da un total de 24 combinaciones únicas para las variables de entrada.

TC	**Formato de archivo**	**Nivel de acceso**	**Rol**
1	.pdf	Verde	Empleado
2	.pdf	Naranja	Administrador
3	.pdf	Púrpura	Empleado
4	.pdf	Verde	Administrador
5	.pdf	Naranja	Empleado
6	.pdf	Púrpura	Administrador
7	.gif	Verde	Empleado
8	.gif	Naranja	Administrador
9	.gif	Púrpura	Empleado
10	.gif	Verde	Administrador
11	.gif	Naranja	Empleado
12	.gif	Púrpura	Administrador
13	.docx	Verde	Empleado
14	.docx	Naranja	Administrador

15	.docx	Púrpura	Empleado
16	.docx	Verde	Administrador
17	.docx	Naranja	Empleado
18	.docx	Púrpura	Administrador
19	.jpeg	Verde	Empleado
20	.jpeg	Naranja	Administrador
21	.jpeg	Púrpura	Empleado
22	.jpeg	Verde	Administrador
23	.jpeg	Naranja	Empleado
24	.jpeg	Púrpura	Administrador

Como la aplicación de la técnica de prueba por pares implica centrarse en solo dos valores de entrada a la vez, en el primer paso, considerando solo las dos primeras columnas, se obtendrá la siguiente tabla de combinaciones:

Formato de archivo	**Nivel de acceso**
.pdf	Verde
.pdf	Naranja
.pdf	Púrpura
.gif	Verde
.gif	Naranja
.gif	Púrpura
.docx	Verde
.docx	Naranja
.docx	Púrpura
.jpeg	Verde
.jpeg	Naranja
.jpeg	Púrpura

Aquí tenemos cada valor de entrada de la columna “Formato de archivo” combinado con cada valor de entrada diferente en la columna “Nivel de acceso”, lo que proporciona doce posibilidades. Como hay cuatro valores de entrada diferentes para “Formato de archivo” y tres valores de entrada diferentes para “Nivel de acceso”, entonces 4 valores de entrada × 3 valores de entrada = 12 casos de prueba diferentes hasta ahora.

Una vez que todas las combinaciones posibles para los valores de entrada de las dos primeras columnas estén claras, necesitamos aplicar la misma técnica a las columnas segunda y tercera para determinar cuántas combinaciones son posibles. Como hay tres valores de entrada diferentes para “Nivel de acceso” y dos valores de entrada diferentes para “Rol”, llegamos a 3 valores de entrada × 2 valores de entrada = 6 casos de prueba, como se muestra aquí:

Nivel de acceso	**Rol**
Verde	Empleado

Verde	Administrador
Naranja	Empleado
Naranja	Administrador
Púrpura	Empleado
Púrpura	Administrador

Podríamos suponer en este punto que tienes 18 casos de prueba; sin embargo, este no es el caso en la prueba por pares. El valor de esta técnica radica en obtener la cobertura más relevante posible con una menor cantidad de casos de prueba.

Si consideramos las 24 combinaciones posibles y ahora implementamos la prueba por pares, simplemente necesitamos asegurarnos de que todas las combinaciones listadas anteriormente en las dos tablas anteriores que comparan dos valores de entrada a la vez estén representadas dentro de los casos de prueba, por lo que todos los demás casos de prueba pueden eliminarse.

Por ejemplo, si destacamos en verde los casos de prueba posibles que satisfacen la primera tabla al comparar los valores de entrada de "Formato de archivo" y "Nivel de acceso", destacaríamos lo siguiente:

TC	Formato de archivo	Nivel de acceso	Rol
1.	.pdf	Verde	Empleado
2.	.pdf	Verde	Administrador
3.	.pdf	Púrpura	Empleado
4.	.pdf	Púrpura	Administrador
5.	.pdf	Naranja	Empleado
6.	.pdf	Naranja	Administrador
7.	.gif	Verde	Empleado
8.	.gif	Verde	Administrador
9.	.gif	Púrpura	Empleado
10.	.gif	Púrpura	Administrador
11.	.gif	Naranja	Empleado
12.	.gif	Naranja	Administrador
13.	.docx	Verde	Empleado
14.	.docx	Verde	Administrador
15.	.docx	Púrpura	Empleado
16.	.docx	Púrpura	Administrador
17.	.docx	Naranja	Empleado
18.	.docx	Naranja	Administrador
19.	.jpeg	Verde	Empleado
20.	.jpeg	Verde	Administrador
21.	.jpeg	Púrpura	Empleado
22.	.jpeg	Púrpura	Administrador
23.	.jpeg	Naranja	Empleado
24.	.jpeg	Naranja	Administrador

Ahora tomaríamos la siguiente tabla que compara los siguientes dos valores de entrada, "Nivel de acceso" y "Rol", y nos daríamos cuenta de que ya hemos resaltado tres de las seis combinaciones. Por lo tanto, en la prueba por pares, simplemente necesitamos asegurarnos de que las otras tres combinaciones también estén representadas. Por lo tanto, podemos resaltar en amarillo cualquiera de las filas que cumplan con esto, intentando cubrir la mayor cantidad de combinaciones posible. Las filas restantes pueden eliminarse, como se muestra aquí:

TC	Formato de archivo	Nivel de acceso	Rol
1.	.pdf	Verde	Empleado
~~2.~~	~~.pdf~~	~~Verde~~	~~Empleado~~
3.	.pdf	Púrpura	Administrador
~~4.~~	~~.pdf~~	~~Púrpura~~	~~Administrador~~
5.	.pdf	Naranja	Empleado
~~6.~~	~~.pdf~~	~~Naranja~~	~~Administrador~~
~~7.~~	~~.gif~~	~~Verde~~	~~Empleado~~
8.	.gif	Verde	Administrador
~~9.~~	~~.gif~~	~~Púrpura~~	~~Empleado~~
10.	.gif	Púrpura	Administrador
~~11.~~	~~.gif~~	~~Naranja~~	~~Empleado~~
12.	.gif	Naranja	Administrador

Aunque ya hemos aplicado completamente el principio de la prueba por pares, podemos hacer lo mismo con los casos de prueba 13 al 24. Aquí también combinamos las combinaciones únicas de las columnas 'Formato de archivo' y 'Nivel de acceso' con las combinaciones únicas de las columnas 'Nivel de acceso' y 'Autorización'. Esto conduce a la siguiente tabla en la que se pueden eliminar cuatro casos de prueba ('14', '16', '18', '19', '21', '23'):

13.	.docx	Verde	Empleado
~~14.~~	~~.docx~~	~~Verde~~	~~Administrador~~
15.	.docx	Púrpura	Empleado
~~16.~~	~~.docx~~	~~Púrpura~~	~~Administrador~~
17.	.docx	Naranja	Empleado
~~18.~~	~~.docx~~	~~Naranja~~	~~Administrador~~
~~19.~~	~~.jpeg~~	~~Verde~~	~~Empleado~~
20.	.jpeg	Verde	Administrador
~~21.~~	~~.jpeg~~	~~Púrpura~~	~~Empleado~~
22.	.jpeg	Púrpura	Administrador
~~23.~~	~~.jpeg~~	~~Naranja~~	~~Empleado~~
24.	.jpeg	Naranja	Administrador

Esto deja los siguientes doce casos de prueba:

TC	Formato de archivo	Nivel de acceso	Rol
1.	.pdf	Verde	Empleado

3.	.pdf	Púrpura	Empleado
5.	.pdf	Naranja	Empleado
8.	.gif	Verde	Administrador
10.	.gif	Púrpura	Administrador
12.	.gif	Naranja	Administrador
13.	.docx	Verde	Empleado
15.	.docx	Púrpura	Empleado
17.	.docx	Naranja	Empleado
20.	.jpeg	Verde	Administrador
22.	.jpeg	Púrpura	Administrador
24.	.jpeg	Naranja	Administrador

En este ejemplo, queda claro que tenemos casos de prueba que cubren una amplia variedad de datos de prueba, pero no hemos probado, por ejemplo, la combinación de .pdf, Naranja y Administrador (caso de prueba "6").

Una ventaja de la prueba por pares es que la cobertura de tus casos de prueba puede ser bastante alta; sin embargo, no se prueban todas las dependencias entre las diferentes variables. Además, se puede imaginar que al aplicar esta técnica también se pueden generar combinaciones que no sean realistas para la situación en cuestión **[XVII]**.

Uso de la IA para Aplicar Técnicas de Prueba

LO33.1	Aprender a usar IA al diseñar pruebas. (K3)

La Inteligencia Artificial (IA) se utiliza cada vez más para automatizar y optimizar técnicas de prueba tradicionales, como tablas de decisión, pruebas por pares, análisis de valores límite, entre otras. Las herramientas impulsadas por IA pueden analizar los requisitos y generar automáticamente casos de prueba que cubran diversas combinaciones de entradas y salidas. Al aprovechar la IA, los testers pueden lograr una mayor eficiencia, precisión y cobertura en sus procesos de prueba.

Beneficios potenciales:

1. **Automatización del diseño de pruebas**: La IA puede generar automáticamente tablas de decisión, combinaciones por pares y casos de prueba de valores límite, reduciendo el esfuerzo manual y el tiempo dedicado al diseño de pruebas. Sin embargo, estos siempre deben ser verificados por un humano, ya que no podemos confiar completamente en la IA, y se sabe que las aplicaciones de IA pueden cometer errores en la información.
2. **Mejora de la cobertura de pruebas**: Los algoritmos de IA pueden analizar rápidamente espacios de entrada complejos y garantizar que todas las combinaciones relevantes y casos límite estén cubiertos, mejorando la exhaustividad de las pruebas.
3. **Detección de errores**: La IA puede identificar inconsistencias, casos faltantes y errores lógicos en tablas de decisión y otros artefactos de prueba, mejorando la calidad y la fiabilidad de los casos de prueba.

4. **Escalabilidad**: La IA puede manejar grandes conjuntos de datos y escenarios complejos de manera eficiente, lo que la hace adecuada para requisitos extensos de datos de prueba y aplicaciones con numerosas variables de entrada.
5. **Consistencia**: La IA mantiene la consistencia en la generación de casos de prueba, minimizando el riesgo de errores humanos y podría usarse como un medio de estandarización en diferentes proyectos.

Desventajas potenciales:

1. **Dependencia excesiva de la IA**: Confiar únicamente en la IA para la generación de casos de prueba puede llevar a pasar por alto conocimientos y perspectivas específicos del dominio que solo los testers humanos pueden proporcionar. La IA podría no entender siempre las sutilezas de ciertas reglas o requisitos comerciales. Además, la IA puede cometer errores al diseñar pruebas, por lo que siempre se debe verificar su trabajo.
2. **Complejidad en la interpretación**: Los casos de prueba y resultados generados por la IA pueden ser complejos y difíciles de interpretar, especialmente si los testers no están familiarizados con los enfoques de pruebas basados en IA.
3. **Restricciones de integridad y seguridad de los datos**: Las grandes organizaciones con políticas estrictas de seguridad de datos pueden enfrentar desafíos al adoptar herramientas de IA, especialmente soluciones de IA de código abierto, debido a preocupaciones sobre la integridad de los datos y el cumplimiento normativo. Las restricciones de seguridad pueden limitar el uso de herramientas de IA basadas en la nube, requiriendo soluciones locales que cumplan con los estándares de seguridad internos. Esta limitación puede restringir los tipos de herramientas de IA disponibles, reduciendo potencialmente la flexibilidad y efectividad de las pruebas impulsadas por IA en estos entornos.
4. **Configuración inicial, capacitación y costos**: Implementar herramientas de prueba basadas en IA requiere una inversión inicial en configuración y capacitación, lo cual puede ser costoso. Las organizaciones deben presupuestar la integración de herramientas de IA en sus flujos de trabajo existentes, lo que incluye la compra o licencia de software de IA, la configuración de la infraestructura para soportar operaciones de IA y la capacitación de los testers para usar eficazmente los modelos de IA. Estos costos iniciales pueden ser una barrera significativa, especialmente para empresas pequeñas o proyectos con presupuestos limitados.

La integración de la IA en técnicas de prueba tradicionales ofrece beneficios significativos en términos de eficiencia, cobertura, consistencia y detección de errores. Sin embargo, es importante equilibrar el uso de la IA con la experiencia humana para garantizar pruebas completas y efectivas. Al comprender las fortalezas y limitaciones de la IA en las pruebas, las organizaciones pueden aprovechar mejor sus capacidades para mejorar sus procesos de prueba.

Relación entre la cobertura en la estrategia de pruebas y las técnicas de prueba

En la Práctica 6, discutimos el desarrollo de una estrategia de pruebas. Existe una conexión entre la estrategia de pruebas y la aplicación de técnicas de prueba. Mientras que la estrategia de pruebas identifica diferentes clases de riesgo, ahora podemos usar técnicas de prueba para cubrir estas clases de riesgo.

Algunas técnicas de prueba ofrecen una cobertura extensa y pueden utilizarse bien para cubrir objetivos de prueba con alto riesgo. Otras proporcionan una cobertura media o baja y pueden aplicarse a riesgos medios o bajos. También es posible variar dentro de una técnica de prueba para lograr una cobertura mayor, media o menor.

La tabla a continuación describe las características de diferentes técnicas de prueba y dónde pueden utilizarse. Es importante tener en cuenta que el uso de una técnica de prueba depende de varios factores. Por ejemplo, necesitas tener una base de prueba adecuada para aplicar la técnica. Nota que la técnica de prueba CRUD será discutida en el Capítulo 4.

Técnicas de prueba	**Adecuación para qué procesos y cobertura**
Prueba de Flujo de Proceso	Adecuada para procesos donde es crucial probar el flujo completo de eventos y acciones, como aplicaciones de flujo de trabajo y procesos de negocio, o código complejo que se desee probar dentro del flujo. Puede aplicarse bien en pruebas de extremo a extremo, cadenas o aceptación.
Prueba Semántica	Adecuada para procesos donde el significado e interpretación de datos y mensajes son cruciales, como protocolos de comunicación y sistemas de intercambio de datos. A menudo útil en pruebas de sistema e integración.
Tabla de Decisión	Adecuada para procesos con lógica de negocio compleja y reglas de decisión, como sistemas financieros o cálculos de seguros. Puede aplicarse en múltiples niveles de prueba para probar software complejo. Combina bien con el Análisis de Valores Límite o la Partición de Equivalencias para mayor cobertura.
Análisis de Valores Límite	Adecuada para procesos con valores límite críticos, como transacciones financieras, aplicaciones de seguridad, etc. La aplicación de esta técnica generalmente proporciona una cobertura de media a alta.
Partición de Equivalencias	Adecuada para procesos estructurados y bien definidos donde las variables de entrada pueden dividirse en clases. Proporciona menor cobertura que el Análisis de Valores Límite.
Pruebas Basadas en Checklists	Adecuada para procesos donde se pueden generar y aplicar listas estructuradas de elementos de prueba, como en pruebas de aceptación de usuario y auditorías de seguridad. La cobertura depende de cuán específica sea la checklist.
Prueba por Pares	Adecuada para procesos con muchas combinaciones de variables de entrada, como pruebas de configuración y compatibilidad en aplicaciones de software. También puede aplicarse para generar datos de prueba. Proporciona una cobertura media, que suele ser representativa y eficiente.
Matriz CRUD	Adecuada para procesos donde las interacciones con bases de datos y las operaciones en los datos son cruciales, como en aplicaciones de gestión de datos. Es bien aplicable a modelos

	de autorización y proporciona una alta cobertura si se realiza completamente.

En la siguiente tabla, se muestran algunos ejemplos de cómo vincular estas diferentes técnicas de prueba con nuestra tabla de estrategia de pruebas previamente establecida. Esto nos permite implementarlas para lograr coberturas 'Alta', 'Media' y 'Baja'. Es importante tener en cuenta que la implementación y efectividad de una técnica dependen en gran medida del contexto específico del proyecto de pruebas, la base de pruebas y el software que se va a probar. Estos factores deben ser considerados al tomar una decisión.

Tabla de estrategia de pruebas		**Niveles de la Prueba**			
Objetivo de prueba/tipo de prueba	**Prueba Estática**	**Prueba Unitaria**	**Prueba de Integración**	**Prueba de Sistema**	**Prueba de Aceptación**
Pedidos desde Francia	Revisión de código	Prueba semántica + Análisis de Valores Límite	Se deben probar todas las posibilidades y excepciones	Tabla de Decisión + Análisis de Valores Límite	Prueba de flujo de proceso de todos los escenarios E2E
Pedidos desde distintas partes de Europa		Pruebas por pares	Pruebas por pares	Prueba semántica	Prueba de flujo de proceso incluyendo los escenarios más importantes
Pedidos desde otras partes del mundo		Checklist con baja cobertura	Checklist con baja cobertura	Prueba de flujo con baja cobertura	Prueba de flujo de proceso con algunos escenarios
API del proveedor	Revisión de código	CRUD		Tablas de Decisión + Análisis de Valores Límite	
Inicio de Sesión	..	..	..	..	..
Búsqueda	..	..	..	..	..
Usabilidad	Prototipado	Revisión Guiada		Seguimiento ocular	Grupos de Usuarios
API Documentation	Revisión				

En esta práctica, hemos discutido varias técnicas de prueba y explicado cómo aplicarlas. También hemos descrito la relación entre la estrategia de prueba y el despliegue de las técnicas de prueba. La práctica "Documenta tus pruebas" te enseña cómo aplicar esto en tu trabajo diario.

Definiciones

Booleano	Un resultado que solo puede tener uno de dos valores posibles: verdadero o falso, sí o no.
Cobertura	El alcance en el que tus casos de prueba cubren la funcionalidad a probar (funcionalidad probada vs. funcionalidad existente máxima).
Caso de Prueba Detallado	Un caso de prueba en un nivel de abstracción muy detallado. Los casos de prueba detallados casi siempre consisten en precondiciones, una predicción de resultados y un resultado.
Caso de Prueba de Alto Nivel	Un caso de prueba en un nivel alto de abstracción que generalmente se utiliza para derivar casos de prueba detallados (o para proporcionar una visión general de alto nivel) de la aplicación. Existe una relación entre los niveles de prueba y el nivel de abstracción de los casos de prueba.
Caso de Prueba Lógico	Un caso de prueba derivado de una base de prueba que consiste en precondiciones (las entradas) y postcondiciones (las acciones o resultados). Los casos de prueba lógicos se basan en la lógica, no en datos concretos.
Predicción de Resultados	El resultado esperado de un caso de prueba.
Caso de Prueba Físico	Un caso de prueba lógico con datos de prueba añadidos.
Precondición	Una condición que debe cumplirse para ejecutar un caso de prueba.
Resultado (Real)	El resultado real de un caso de prueba: 'OK' o 'No OK'.
Proceso de Prueba	La colección de actividades interrelacionadas que consisten en planificación de pruebas, monitoreo y control de pruebas, análisis de pruebas, diseño de pruebas, implementación de pruebas, ejecución de pruebas y finalización de pruebas.
Caso de Prueba	Un conjunto de precondiciones, valores de entrada, acciones (si las hay), resultados esperados y postcondiciones, desarrollados a partir de condiciones de prueba.
Técnica de Prueba	Una forma estructurada de derivar casos de prueba a partir de una base de prueba.
Guion de Prueba	Un conjunto de instrucciones para llevar a cabo una prueba.
Diseño de Pruebas	La actividad que deriva y especifica casos de prueba a partir de condiciones de prueba. [V]
Variable	Un elemento, característica o factor que es susceptible de variar o cambiar.

Práctica 8: Comprender la Importancia de una Buena Comunicación

LO34	Comprender la importancia de una buena comunicación en todas tus actividades como tester. (K3)

Ahora hemos aprendido una gran cantidad de habilidades para probar software de manera efectiva y estructurada. Una habilidad sirve para reforzar las prácticas anteriores: una buena comunicación.

Las habilidades básicas de comunicación incluyen hablar, escribir, escuchar y leer. Cuando los mensajes se comunican a través de canales verbales o escritos, es importante verificar que el mensaje se haya entregado correctamente y que haya sido comprendido.

Los mensajes siempre tienen un emisor y un receptor. Una buena comunicación implica verificar, como emisor del mensaje, que el mensaje se haya entregado correctamente y que haya sido comprendido. Como receptor, verificar que tu comprensión del mensaje comunicado es la misma que la intención del emisor también es una buena práctica de comunicación. Además de la comunicación verbal y escrita, también existe la comunicación no verbal, que también se puede utilizar para verificar que los mensajes han sido recibidos y comprendidos.

Validar la comunicación es necesario porque todos tienen filtros personales que pueden hacer que los mensajes se procesen de maneras inesperadas. Los filtros de cada persona se basan en su experiencia personal, antecedentes, diferencias culturales, educación y creencias. Como resultado, un mensaje que se intente enviar de una manera puede ser interpretado de diferentes formas por diferentes receptores. Al realizar pruebas, es importante verificar que los mensajes que envíes hayan sido comprendidos tal como se pretendía y que la comprensión de los mensajes que recibas sea la misma que la intención del emisor.

La comunicación a veces puede ser difícil por varias razones: verificar la comprensión puede hacer que las personas sientan que el emisor las está controlando, que es demasiado meticuloso o que está interfiriendo demasiado en su trabajo. Además, las personas pueden parecer muy seguras de sí mismas, lo que puede hacer que te sientas reacio a hacer más preguntas. Por lo tanto, una buena comunicación también requiere reflexión, tacto y habilidades sociales. Esta práctica es especialmente importante porque el desarrollo de software es un conjunto complejo de actividades que a menudo no son tangibles.

Veamos algunos ejemplos de cómo podrías aplicar esta práctica en conjunto con las prácticas de prueba. La práctica de "no hacer suposiciones" también asume un concepto similar porque, con esta práctica, verificas y/o validas todo y no asumes nada.

La práctica de "probar lo antes posible" resalta la importancia de la revisión. Revisar o hacer que los productos sean revisados es una excelente manera de aumentar la calidad de estos productos en una etapa temprana, y la revisión se trata de verificar que el producto sea preciso, de buena calidad y completo. "Productos" podría significar cualquier cosa, desde diseño funcional, requisitos, guiones de prueba, el resultado del análisis de riesgos, etc. Al enviar productos para su revisión, la buena comunicación es importante porque confirmar si el receptor entiende lo que el producto implica, así como lo que se espera de ellos, puede mejorar enormemente los resultados obtenidos al final. Al hacerlo, toda la revisión será más valiosa.

La práctica de "documentar tus pruebas" también es relevante para la práctica de una buena comunicación. Al documentar tu trabajo y describir los requisitos o casos de prueba de manera clara, no solo se puede hacer referencia a la documentación en un momento posterior, sino que también hay una mayor probabilidad de que tú y los demás entiendan lo que se quiere decir.

Otra parte de la buena comunicación es la gestión de expectativas. Es importante en nuestro trabajo como testers que se establezcan expectativas razonables y realistas y se comuniquen claramente y de manera comprensible a las diversas partes interesadas. Gestionar las expectativas es importante porque minimiza sorpresas y evita que las expectativas y los resultados finales estén demasiado alejados. Por lo tanto, es importante comunicar en el momento adecuado si, por ejemplo, prevés que la creación de un guion de prueba o la ejecución de una prueba tomará mucho más tiempo de lo esperado inicialmente.

Usar una comunicación efectiva en un proceso de pruebas es crucial.

A lo largo de un proceso de pruebas, hay muchas instancias donde la comunicación juega un papel vital. Hemos aprendido en prácticas anteriores que aplicar la comunicación es importante, incluso en la práctica de 'Ser específico' y 'Documentar tus pruebas.' En la práctica de 'Probar todo es imposible,' aprendimos cómo establecer una Estrategia de Pruebas Basada en Riesgos. La comunicación está implícita en todas estas prácticas. Por

supuesto, después de ejecutar las pruebas, necesitamos comunicarnos con el cliente sobre los resultados obtenidos en las pruebas. Sería peculiar entregar el software al final del proceso sin proporcionar a las partes interesadas una visión de los resultados obtenidos durante las pruebas del software. Incluso durante un largo proceso de desarrollo y pruebas, mantener informadas a las partes interesadas es esencial.

Esto se puede lograr a través de informes simples. En el informe, esencialmente necesitamos abordar tres temas: el estado de los casos de prueba, defectos y riesgos. Para los casos de prueba, puedes incluir los estados descritos en la tabla 'Ejemplo de estados de casos de prueba' a continuación en el informe.

Casos de prueba	
Estado	**Descripción**
No iniciado	Casos de prueba que aún no han sido iniciados.
En progreso	Casos de prueba que han sido iniciados, pero aún están en progreso.
No aplicable	Casos de prueba que han sido diseñados pero ya no son aplicables.
Fallido	Casos de prueba que no fueron exitosos y resultaron en un defecto.
Aprobado	Casos de prueba que se han completado con éxito.
Total	El conteo total de casos de prueba.

'Ejemplo de estado de caso de prueba'

En el caso de los defectos, puedes considerar los siguientes estados en tu informe. Ten en cuenta que, a veces, pueden existir más estados que los descritos en la tabla, pero para un proceso de prueba simple, puedes utilizar los que se indican a continuación.

Defectos	
Estado	**Descripción**
Nuevo	Defectos que acaban de ser descubiertos.
Abierto	Defectos que aún están abiertos.
Asignado	Defectos que han sido asignados a alguien para su investigación y determinación del impacto.
En progreso	Defectos que están siendo reparados actualmente.
Listo para retestear	Defectos que están listos para ser retesteados.
Retesteando	Defectos que están siendo retesteados actualmente.
Retesteo OK	Defectos que han sido resueltos después del retesteo.
Retesteo NO OK	Defectos que no han sido resueltos después del retesteo.
Estacionado	Defectos para los cuales se ha tomado la decisión de no abordar inmediatamente. Por ejemplo, debido a un alto impacto en el sprint actual o la necesidad de más análisis para una solución adecuada.

'Ejemplo de estado de defecto'

En el caso de los defectos, es recomendable incluir la gravedad en el informe. A continuación, encontrarás un ejemplo de los distintos niveles de gravedad que podrías utilizar.

Severidad de los Defectos	
Severidad	**Descripción**
Bloqueante	El defecto bloquea otra funcionalidad, impidiendo que se pruebe, creando incertidumbre sobre el funcionamiento de esos casos de prueba.
Grave	El defecto es grave y necesita ser solucionado, pero no bloquea ninguna otra funcionalidad.
No grave	El defecto es válido, pero no es lo suficientemente grave. Hay una solución alternativa aceptable disponible.
Cosmético	El defecto es cosmético o un error tipográfico.

'Ejemplo de gravedad de los defectos'

Los informes sobre los riesgos se pueden realizar demostrando cuánto esfuerzo se ha invertido en cubrir adecuadamente los objetivos de prueba identificados en el análisis de riesgos. Para ello, tomamos la tabla creada en la Práctica 3 y realizamos algunos ajustes. Básicamente, se informa sobre la estrategia de prueba establecida y se proporciona a las partes interesadas retroalimentación sobre cómo y en qué medida se han mitigado los riesgos identificados durante el análisis de riesgos.

Reporte de Riesgos				
Riesgo (Objetivo de Prueba)	**Consecuencia**	**Severidad**	**Medidas**	**Comentarios**
Pedidos desde Francia no procesados	Pérdida significativa de ingresos debido a la imposibilidad de vender estos productos. Daño sustancial a la reputación de la empresa entre los clientes.	Alta	Pruebas exhaustivas	Se han implementado todas las medidas descritas en el proceso de pruebas. Como resultado, este riesgo ha sido completamente mitigado.
Pedidos desde otras partes de Europa	Pérdida moderada de ingresos, daño a la imagen.	Media	Este riesgo se ha reducido significativamente mediante las medidas de calidad implementadas.	
Pedidos desde otras partes del mundo	Pérdida menor de ingresos, daño leve a la reputación.	Baja	Este riesgo ha sido casi eliminado debido a las medidas implementadas.	
API de proveedores	Los vendedores pueden no ser capaces de presentar correctamente sus productos, lo que podría llevar a una pérdida de	Alta	Trayectoria de prueba con los vendedores, monitoreo de disponibilidad después de la liberación a producción	A pesar de un proceso de pruebas exhaustivo, sigue existiendo un riesgo residual, principalmente relacionado con el uso correcto de la API. Para facilitar esto de manera efectiva para los vendedores, queremos establecer

	ingresos o errores en los productos ofrecidos.			monitoreo y soporte activo. Esto incluye configurar nuestro servicio de soporte técnico de API para hacer un seguimiento proactivo y asistir si detectamos un número significativo de errores durante el monitoreo.
...	..			

'Ejemplo de informe de riesgos'

Ten en cuenta que tus informes también pueden abordarse por nivel de prueba o tipo de prueba. En nuestro ejemplo, utilizamos todo el proceso de prueba. Sin embargo, también podrías informar solo sobre la revisión de código, las pruebas unitarias o las pruebas de aceptación. A continuación, se ofrecen algunos ejemplos:

Revisión de Código		
Líneas de código revisadas	Defectos encontrados	Defectos aún abiertos
16.268	68	5

'Ejemplo de informe de revisión de código'

Casos de Prueba – Prueba Unitaria					
No Iniciados	**En Proceso**	**No Aplicable**	**Fallidos**	**Aprobados**	**Total**
15	26	8	5	145	199

'Example Unit test report'

Reporte de Defectos									
	Nuevo	**Abierto**	**Asignado**	**Resolviendo**	**Listo para re-prueba**	**Re-prueba**	**Re-prueba OK**	**Re-prueba no OK**	**Aparcado**
Bloqueante	0	1	1	0	2	1	5	0	0
Serio	0	1	2	0	0	1	5	1	1
No Serio	0	0	1	2	0	0	4	1	3
Cosmético	1	0	0	1			2	0	1
Total	0	2	4	3	2	2	16	2	5

'Ejemplo de Informe de Defectos'

Estos informes son ejemplos. Puedes usarlos o crear tu propio informe combinando elementos de estos ejemplos. Por supuesto, puedes visualizar este informe utilizando gráficos o reportando el progreso del proceso de pruebas y los cambios de estado por día o semana. Al informar en los intervalos adecuados (diarios, semanales, mensuales) sobre el progreso de tu proceso de pruebas y cómo mitiga los riesgos, puedes adherirte

efectivamente a la práctica de 'La importancia de una buena comunicación' y proporcionar a los interesados una visión clara de tu proceso de pruebas.

Definiciones

Gestión de Expectativas	El proceso de clarificar o ajustar ciertos resultados esperados.
Comunicación no verbal	El lenguaje corporal que usamos al expresarnos
Receptor	El destinatario de un mensaje o registro, ya sea escrito o hablado.
Emisor	El remitente de un mensaje o registro, ya sea escrito o hablado.
Comunicación verbal	Las palabras que elegimos para expresarnos.

Capítulo 4: Atributos de Calidad, enfocados en Seguridad, Usabilidad y Rendimiento

En este capítulo, discutimos más a fondo algunos ejemplos importantes de atributos de calidad. Para comprender mejor qué son los atributos de calidad, nos centraremos en tres que son relevantes para casi todos los proyectos: seguridad, usabilidad y rendimiento.

Estos atributos de calidad son importantes de incluir al crear una estrategia de pruebas. Son aspectos de la calidad que se encuentran en casi todos los proyectos de software y deben ser considerados al formular una estrategia de pruebas. En nuestro ejemplo elegido de un sitio web de comercio electrónico que vende productos, la seguridad, la usabilidad y el rendimiento también jugarán un papel importante. Por ejemplo, el riesgo asociado con medidas de seguridad inadecuadas podría llevar a vendedores no autorizados a usar incorrectamente la API. En cuanto a la usabilidad y el rendimiento de la tienda en línea, si no están en orden, podríamos recibir menos pedidos de usuarios porque podrían abandonar la plataforma debido a una interfaz deficiente o una velocidad de procesamiento lenta. Existen más atributos de calidad además de los mencionados aquí. Sin embargo, elegimos excluirlos de este plan de estudios, excepto por la funcionalidad y los atributos de calidad mencionados en el Capítulo 1, para mantener el contenido manejable y centrarnos en lo básico.

LO35	Aprender los conceptos básicos de las pruebas de seguridad. (K2)
LO36	Aprender los conceptos básicos de las pruebas de usabilidad. (K1)
LO37	Aprender los conceptos básicos de las pruebas de rendimiento. (K1)

Atributo de Calidad: Seguridad

Una herramienta muy útil al probar la seguridad en aplicaciones es el Top Ten de OWASP **[XVIII]**. OWASP (Open Web Application Security Project) es una fundación sin fines de lucro enfocada en mejorar la seguridad del software. Cada pocos años, OWASP publica una lista de los diez riesgos de seguridad más críticos, llamada "Top Ten de OWASP". El Top Ten de OWASP es ampliamente reconocido como el primer paso para cualquier empresa interesada en mejorar el nivel de seguridad en el software que desarrollan, así como los métodos por los cuales aquellos que crean el software realizan su trabajo. El enfoque no está

solo en la seguridad del software que se está creando, sino también en los procesos empresariales y las mejores prácticas en el lugar de trabajo. Por ejemplo, no sería una buena práctica tener contraseñas anotadas cerca del espacio de trabajo. El Top Ten de OWASP incluye muchas sugerencias valiosas sobre cómo incluir diversas mejores prácticas en los procesos de prueba (o desarrollo). Información más específica, incluidos ejemplos e información detallada sobre cómo probar fallas de seguridad, se puede encontrar en https://www.owasp.org.

Conceder permisos es fácil y permite a los usuarios hacer muchas cosas en una trayectoria de desarrollo y pruebas. A menudo, esto se hace desde el punto de vista de "el usuario es el rey". Para evitar el controles de acceso rotos, a los usuarios se les deben otorgar permisos para hacer solo lo necesario para cumplir con sus tareas.

Atributo de Calidad Seguridad: Ejemplo de OWASP Top Ten: Control de Acceso Roto

El control de acceso roto generalmente significa vulnerabilidades en roles y permisos. El control de acceso bien diseñado debe mantenerse y crearse para que los usuarios no puedan actuar fuera de su autorización prevista. El fracaso al diseñar el control de acceso adecuadamente podría llevar a la divulgación no autorizada, actualización o eliminación de datos, o el uso de funciones que están fuera de las autorizaciones previstas para los usuarios. Por ejemplo, tomemos una pequeña empresa en la que a los usuarios del departamento de administración financiera se les otorgan todos los permisos para la administración financiera. Estos permisos podrían llevar a que puedan aumentar sus salarios. Por eso es importante reiterar que el sistema de roles y permisos debe estar correctamente diseñado, alineado y probado. Una buena regla general aquí es otorgar solo los permisos que se necesitan—no más

El rol de Administrador debe usarse solo cuando sea necesario y no debe ser compartido con un número infinito de personas. Se debe crear una cuenta específica para acomodar otras autorizaciones (posiblemente de administrador) y también establecer un procedimiento que asegure que los permisos puedan ser revocados si es necesario, por ejemplo, cuando un usuario deja la empresa.

Atributo de Calidad Seguridad: Matriz CRUD

Una buena manera de probar (y también diseñar y alinear) roles y permisos es configurar una simple matriz de autorización. Esta matriz de autorización también se llama matriz CRUD (Crear, Leer, Actualizar, Eliminar). Una matriz de autorización se crea colocando todos los roles y permisos, vinculados a la funcionalidad que deben ejercer, dentro de una tabla.

	Datos de Cliente	**Datos delPedido**	**Datos de Producto**
Administrador	CRUD	CRUD	CRUD
Empleado	CRU	CRU	RU
Cliente	RU	CR	

Ejemplo de matriz CRUD simple

Al utilizar una matriz CRUD, puedes encontrar pasos faltantes o verificar si los datos son (permitidos ser) accesibles. Al trabajar a través de la matriz CRUD y realizar pruebas con todos los diferentes roles (uno por uno), podemos verificar todos los permisos otorgados, así como las acciones que no están permitidas, y luego confirmar nuevamente que lo que no está permitido realmente no es posible. Si tomamos nuestra matriz CRUD de ejemplo y seguimos la fila de "Empleado", para las entidades "Datos de Cliente" y "Datos de Pedido", vemos que necesitamos probar la creación, lectura y actualización de datos, pero no debemos olvidar confirmar si el Empleado no puede eliminar datos en estas entidades.

Atributo de calidad de seguridad: ejemplo de OWASP Top Ten: Fallos criptográficos

Los fallos criptográficos se centran en errores que son el resultado de la encriptación. Esto puede estar relacionado con el proceso de encriptación en sí o con la falta de encriptación necesaria. Estos fallos pueden llevar a la exposición de datos sensibles en tránsito o en reposo, por ejemplo, una base de datos de contraseñas que utiliza una encriptación simple, rota o unidireccional para almacenar contraseñas.

Un buen punto de partida para resolver estos problemas es una clasificación adecuada de los datos: esto determina el grado de confidencialidad de los datos. Por esta razón, las contraseñas, los registros médicos, los números de tarjetas de crédito, la información personal y la información confidencial de negocios requieren protección adicional, especialmente si esos datos están cubiertos por leyes de privacidad como el Reglamento General de Protección de Datos (GDPR) de la UE u otros estándares.

Después de la clasificación, el siguiente paso es asegurar adecuadamente los datos con métodos de encriptación modernos e inquebrantables. Estas acciones se aplican tanto a enlaces internos como externos de la aplicación. Consulta el sitio web de OWASP para más sugerencias de prevención y consejos para realizar pruebas.

"Control de acceso roto" y "Fallos criptográficos" son solo dos de los elementos del OWASP Top Ten para hacer que tu software y entorno de trabajo sean seguros. Es casi imposible excluir la posibilidad de ser hackeado, pero el objetivo debe ser evitarlo tanto como sea posible. El desafío es detectar rápidamente los hackeos y asegurarse de mantener el acceso (y el control) sobre tus datos. El monitoreo adecuado, los procedimientos de respaldo, los procedimientos de contingencia y los procedimientos de restauración son clave para mantener los datos seguros. Si estos procedimientos están implementados, practicados y probados regularmente, ya estás en un buen camino hacia mejores prácticas de seguridad.

El sitio web de OWASP menciona varios consejos y trucos más para el desarrollo seguro y las pruebas de seguridad: https://www.owasp.org.

Atributo de calidad: Usabilidad

La **usabilidad** se refiere al grado en que un producto o sistema puede ser utilizado por usuarios especificados para lograr de manera efectiva, eficiente y exitosa los objetivos especificados en un contexto de uso determinado.

Si consideramos que estamos revisando un sitio web, una aplicación o un sistema en las instalaciones, la pantalla principal suele ser la página más importante en cuanto a usabilidad. No solo es la parte más visitada del software, sino que también determina si el usuario puede (o incluso querrá) seguir utilizando el software para acceder a todas las demás partes. Aquí es donde los usuarios formarán su primera opinión sobre el software, por lo que el reconocimiento de la idoneidad es extremadamente importante. Una vez que se satisfacen las necesidades de los usuarios, intentarán lograr su(s) objetivo(s) utilizando el sistema. Para maximizar la satisfacción del usuario, se deben proporcionar entradas claras y mostrar las tareas más importantes. La pantalla principal también es un lugar ideal para diferenciarte de la competencia.

Para obtener los mejores resultados, una navegación clara y una arquitectura de la información son esenciales. La clave es centrarse en la tarea en sí. Puedes orientarte hacia la tarea si conoces la necesidad de un usuario específico. Por ejemplo, si se te concede acceso a algo, la navegación en ese espacio debe ser clara. Los usuarios quieren saber dónde están, qué pueden hacer y adónde pueden ir. También quieres que esta información se entregue de la manera más reconocible. Los elementos para un sitio web podrían incluir el enlace “inicio”, un menú hamburguesa, una categorización lógica (miguitas de pan), un mapa del sitio, etc.

Es importante hacer coincidir el modelo mental del diseñador con el modelo mental del usuario. Para tener esto en cuenta, considera cuál es el principal enfoque del usuario: este debe ser el CTA (Call To Action – Llamado a la Acción) más importante. Un CTA es el botón más importante para lograr un objetivo. Este objetivo suele ser lo que el visitante busca, y este mismo objetivo generalmente beneficia al negocio porque contribuye a lograr los objetivos comerciales. Por lo tanto, asegúrate de establecer claramente cuáles son las tareas más importantes.

En un formulario, debe ser claro dónde rellenar algo y qué debe completarse. Se debe proporcionar retroalimentación inmediata tanto para las entradas correctas como incorrectas. Considera cómo mostrarías si algo es obligatorio o en qué formato debe ingresarse un número de teléfono o un código postal. Si algo sale mal, se debe dar retroalimentación directamente al usuario para mejorar.
Al considerar el atributo de calidad de la usabilidad, el enfoque principal debe estar siempre en los objetivos del usuario y los objetivos generales del cliente. En el mejor de los casos, estos objetivos se superpondrían y el software debería funcionar de la manera más fluida posible para que se cumplan esos objetivos.

Atributo de calidad: Rendimiento

Las **pruebas de rendimiento** se pueden dividir en dos temas principales: pruebas de carga y pruebas de estrés. Las pruebas de carga consisten en simular una carga determinada de usuarios y observar los tiempos de respuesta bajo esta carga. Las pruebas de estrés implican encontrar el límite de lo que el sistema puede manejar en términos de carga máxima.

Una buena prueba de rendimiento requiere conocimientos específicos y herramientas: conocimiento detallado de la infraestructura y cómo se relaciona con la infraestructura del

entorno de producción. Para una buena prueba de rendimiento, es crucial tener un entorno que sea similar o casi idéntico al entorno de producción. Los componentes de infraestructura, como interruptores, hardware y cortafuegos, y su dimensionamiento deben tenerse en cuenta para simular con precisión la carga del sistema. Para lograr esta carga, a menudo se requieren herramientas específicas para simular una gran cantidad de usuarios simultáneos.

Otra forma de obtener una buena indicación del rendimiento es creando un perfil de carga de usuarios del entorno futuro. Esto puede proporcionar una mejor comprensión de qué carga requiere el sistema y en qué momentos del día. Además, establecer una monitorización adecuada, incluidos los sellos de tiempo, es una medida mitigante en las pruebas de rendimiento.

Definiciones

Llamada a la Acción	La acción más importante que deseas que realice el visitante.
Prueba de carga	Un tipo de prueba de rendimiento realizada para evaluar el comportamiento de un componente o sistema bajo diferentes cargas, generalmente entre condiciones esperadas de uso bajo, normal y pico.
Pantalla principal	La pantalla más importante de una aplicación desde la que se pueden realizar la mayoría de las acciones, generalmente la primera pantalla después de iniciar sesión en una aplicación.
Interacción de usuario perfecta	El grado en que una interfaz de usuario permite que el usuario tenga una interacción agradable y satisfactoria.
Rendimiento	La velocidad con la que el sistema de información maneja las transacciones.
Reconocimiento de idoneidad	El grado en que los usuarios pueden reconocer si un producto o sistema es adecuado para sus necesidades.
Seguridad	Garantía de que la consulta o modificación de los datos solo sea posible por aquellos autorizados para hacerlo.
Prueba de estrés	Una forma de prueba de rendimiento que tiene como objetivo evaluar un componente o sistema en o más allá de los límites de la carga de trabajo esperada o especificada para él, o con recursos limitados como memoria o capacidad del servidor.
Usabilidad	El grado en que un producto o sistema puede ser utilizado de manera efectiva, eficiente y satisfactoria por los usuarios.

Referencias

Referencia	Fuente
[I]	**Libro:** L. Anderson, P. W. Airasian, and D. R. Krathwohl, A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, 2001, Allyn & Bacon, ISBN 03-21084-05-5
[II]	**Sitio Web:** ISO 9000:2015 Standard information: https://www.iso.org/standard/45481.html
[III]	**Sitio Web:** American Society for Quality glossary: https://asq.org/quality-resources/quality-glossary/
[IV]	**Sitio Web:** ISO 25010 Standard information: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[V]	**Sitio Web:** ISTQB® glossary application: https://glossary.istqb.org/
[VI]	**Sitio Web:** Wikipedia page for Vilfredo Pareto: https://en.wikipedia.org/wiki/Vilfredo_Pareto
[VII]	**Artículo:** Doran, G. T. (1981). "There's a S.M.A.R.T. Way to Write Management's Goals and Objectives", Management Review, Vol. 70, Issue 11, pp. 35-36.
[VIII]	**Artículo:** Jan Vanthienen, "The History of Modeling Decisions using Tables (Part 1)" Business Rules Journal, Vol. 13, No. 2, (Feb. 2012): https://www.brcommunity.com/articles.php?id=b637
[IX]	**Libro:** Boehm, B. (1981) Software Engineering Economics, Prentice-Hall Inc., ISBN 01-38221-22-7
[X]	**Sitio Web:** Agile Alliance glossary: https://www.agilealliance.org/glossary/user-story-template/
[XI]	**Sitio Web:** Wikipedia page for Flowchart: https://en.wikipedia.org/wiki/Flowchart
[XII]	**Libro:** Pol. M., Teunissen. R., Veenendaal van. E., Testen volgens TMap 2e druk, p326, ISBN 90-72194-58-6
[XIII]	**Libro:** Beizer, B. (1990) Software Testing Techniques. 2nd Edition, Van Nostrand Reinhold, New York. ISBN 18-50328-80-3
[XIV]	**Libro:** Burnstein, Ilene (2003), Practical Software Testing, Springer-Verlag, ISBN 0-387-95131-8
[XV]	**Libro:** G. Myers, The Art of Software Testing; John Wiley, New York, 1979. ISBN 0-471-04328-1
[XVI]	**Sitio Web:** ISTQB Foundation syllabus, 2019 version, p61: https://www.istqb.org/certifications/certified-tester-foundation-level
[XVII]	**Sitio Web:** Software Testing Help regarding for Pairwise Testing: https://www.softwaretestinghelp.com/what-is-pairwise-testing/
[XVIII]	**Sitio Web:** OWASP Top10: https://owasp.org/Top10/
[XIX]	**Artículo:** The New Religion of Risk Management, Bernstein, 1996: https://hbr.org/1996/03/the-new-religion-of-risk-management
[XX]	**Sitio Web:** ISO29119-4:https://www.iso.org/obp/ui/en/#iso:std:iso-iec-ieee:29119:-4:ed-2:v1:en

[XXI] **Sitio Web**: TMap Glossary Online: https://www.tmap.net/page/tmap-glossary-online