

MATERIAL DE LECTURA

QA Trainees Program

★ MÓDULO 1: Fundamentos de Testing	4
Unidad 1	4
Introducción al mundo del testing	4
7 principios de las pruebas de software:	5
1- No es posible realizar pruebas exhaustivas	5
2- Agrupación de defectos	5
3- Paradoja de los pesticidas	6
4- Las pruebas muestran la presencia de defectos	6
5- Ausencia de Error – falacia	6
6- Pruebas tempranas	6
7- Las pruebas dependen del contexto	7
¿Es lo mismo calidad y testing?	7
Proceso de calidad de software	8
¿Qué es un error?	9
Unidad 2	11
Categorías de las pruebas: Funcionales - No Funcionales - Mantenimiento	11
Pruebas Funcionales:	11
Pruebas No Funcionales:	12
Mantenimiento:	12
Pruebas Manuales	13
Tipos de pruebas	13
Integración	14
Verificación	15
Sistema	15
Aceptación	16
Descripción de pruebas	16
Pruebas de Funciones de negocio	16
Performance	17
Seguridad	17
Interfaz de usuario	18
Stress	18
Volumen	19
Usabilidad	19
Regresión	20
Smoke / Pruebas de Humo	20
Unidad 3	20
Proceso de Pruebas	20
Análisis de requisitos:	21
Planificación de pruebas	21

Desarrollo de casos de prueba	22
Configuración del entorno de prueba	22
Ejecución de pruebas	22
Cierre del ciclo de prueba	23
Identificar Funcionalidades	23
Técnicas para identificar funcionalidades:	23
Escenario de prueba	25
¿Por qué crear escenarios de prueba?	26
Cómo escribir escenarios de prueba	27
Ejemplo: Escenario de prueba para la aplicación de comercio electrónico. Link: https://www.saucedemo.com/	27
Escenario de prueba 1: comprobar la funcionalidad de inicio de sesión.	27
Escenario de prueba 2: verificar la descripción del producto	28
Escenario de prueba 3: Comprobar el historial de pedidos	29
Casos de Prueba (diseño, ejecución, resultado)	29
Matriz de trazabilidad	31
Generación de Datos	32
Métricas	32
★ MÓDULO 2: Fundamentos de Testing	33
Unidad 1	33
Testing continuo en el ciclo DEVOPS	33
Los 10 principios del Agile Testing	36
Testing Manifiesto	37
Planificación de pruebas en contextos ágiles:	39
Unidad 2	40
Guía de desarrollo con ejemplos:	40
BDD	40
ATDD	41
SBE	42
Habilitando la Colaboración	42
Dinámica de los 3 amigos	42
Dinámica del Impact Mapping	44
Dinámica del Example Mapping	46
Pruebas Exploratorias	47
Atributos de la Calidad del Testing	48
Desarrollo	48
Operacionales	49

Material de Lectura:

DESARROLLO POR MÓDULOS (SPRINT'S)

★ MÓDULO 1: Fundamentos de Testing

Objetivo: Que los participantes adquieran los conocimientos y conceptos básicos de pruebas de software.

Que mejoren sus habilidades y conocimientos en diseñar un plan y casos de prueba manuales lo que le permitirá desarrollar una carrera exitosa en el campo de la tecnología.

Unidad 1

Introducción al mundo del testing

El testing es una parte crucial en el desarrollo de software que implica la evaluación de un sistema o una aplicación para asegurar que cumpla con los requisitos y estándares de calidad. El proceso de testing puede variar desde pruebas manuales simples hasta pruebas automatizadas complejas, y se lleva a cabo en diferentes etapas del ciclo de vida del software, incluyendo la etapa de desarrollo, pruebas de aceptación del usuario y mantenimiento.

El objetivo principal del testing es garantizar que el software funcione como se espera y que no tenga errores que puedan afectar su rendimiento y/o seguridad. Esto incluye probar tanto la funcionalidad como el comportamiento del software en diferentes entornos y situaciones. El testing también ayuda a identificar y corregir errores, mejorando así la calidad del software.

El testing puede ser manual o automatizado, y se puede utilizar una combinación de ambos métodos para obtener los mejores resultados.

En resumen, el testing es una parte crucial en el desarrollo de software que ayuda a garantizar la calidad y la funcionalidad de una aplicación o sistema.

Pero, ¿cómo determina que está siguiendo la estrategia correcta para las pruebas? Para eso, debe apegarse a algunos principios básicos de prueba.

7 principios de las pruebas de software:

1- No es posible realizar pruebas exhaustivas

Necesitamos la cantidad óptima de pruebas en función de la evaluación de riesgos de la aplicación.

Y ¿cómo se determina este riesgo?

Considere un escenario en el que está moviendo un archivo de la carpeta A a la carpeta B. Piense en todas las formas posibles en que puede probar esto. Además de los escenarios habituales, también puede probar las siguientes condiciones:

- Intentando mover el archivo cuando está abierto
- No tiene los derechos de seguridad para pegar el archivo en la Carpeta B
- La carpeta B está en una unidad compartida y la capacidad de almacenamiento está llena.
- La carpeta B ya tiene un archivo con el mismo nombre, de hecho, la lista es interminable
- O suponga que tiene 15 campos de entrada para probar, cada uno con 5 valores posibles, el número de combinaciones a probar sería 5^{15} .

Si tuviera que probar todas las combinaciones posibles, el TIEMPO DE EJECUCIÓN Y LOS COSTOS del proyecto aumentan exponencialmente. Necesitamos ciertos principios y estrategias para optimizar el esfuerzo de prueba.

2- Agrupación de defectos

Agrupación de defectos que establece que una pequeña cantidad de módulos contiene la mayoría de los defectos detectados. Esta es la aplicación del Principio de Pareto a las pruebas de software: aproximadamente el 80% de los problemas se encuentran en el 20% de los módulos.

Por experiencia, puede identificar tales módulos riesgosos. Pero este enfoque tiene sus propios problemas.

Si las mismas pruebas se repiten una y otra vez, eventualmente los mismos casos de prueba ya no encontrarán nuevos errores.

3- Paradoja de los pesticidas

El uso repetitivo de la misma mezcla de pesticidas para erradicar insectos durante la agricultura conducirá con el tiempo a que los insectos desarrollen resistencia al pesticida. Lo mismo se aplica a las pruebas de software. Si se realiza el mismo conjunto de pruebas repetitivas, el método será inútil para descubrir nuevos defectos.

Para superar esto, los casos de prueba deben revisarse regularmente, agregando casos de prueba nuevos y diferentes para ayudar a encontrar más defectos.

Debe buscar continuamente mejorar los métodos existentes para que las pruebas sean más efectivas. Pero incluso después de todo este sudor y trabajo duro en las pruebas, nunca podrá afirmar que su producto está libre de errores.

Para ejemplificar este punto, sugiero visualizar este ejemplo:

<https://www.youtube.com/watch?v=yeUyxjLhAxU>

Hasta una empresa como MICROSOFT ha probado su sistema operativo a fondo pero falló durante su lanzamiento público.

4- Las pruebas muestran la presencia de defectos

Las pruebas de software reducen la probabilidad de que queden defectos no descubiertos en el software, pero incluso si no se encuentran defectos, no es una prueba de ausencia de los mismos.

5- Ausencia de Error – falacia

Las pruebas de software no consisten simplemente en encontrar defectos, sino también en comprobar que el software satisface las necesidades del negocio. Encontrar y corregir defectos no ayuda si la construcción del sistema no se puede usar y no cumple con las necesidades y requisitos del usuario.

6- Pruebas tempranas

Pruebas tempranas: las pruebas deben comenzar lo antes posible en el ciclo de vida del desarrollo de software. Para que cualquier defecto en la fase de requisitos o diseño se capture en etapas tempranas. Es mucho más económico corregir un defecto en las primeras etapas de la prueba, en el momento en que se definen los requisitos.

7- Las pruebas dependen del contexto

Todos los software desarrollados no son idénticos. Puede utilizar un enfoque, metodologías, técnicas y tipos de pruebas diferentes según el tipo de aplicación. Por ejemplo, probar un sistema para una pequeña empresa, no será lo mismo que probar el sistema de un cajero automático, o para una empresa que solicita el uso de la aplicación para millones de usuarios.

Es falso pensar que los principios de prueba no son necesarios en la práctica. Estos principios son fundamentales para crear una estrategia de prueba efectiva y para desarrollar casos de prueba que detecten errores. Aprender estos principios es como aprender a conducir: al principio prestamos atención a todos los detalles, pero con la experiencia se vuelven naturales. Los probadores experimentados aplican estos principios sin siquiera pensar en ellos, lo que demuestra que son esenciales en la práctica. Por lo tanto, no hay verdad en el mito de que los principios de prueba no se utilizan en la práctica.

¿Es lo mismo calidad y testing?

La calidad y el testing son dos conceptos relacionados pero distintos. La calidad se refiere a la capacidad de un producto o servicio de cumplir con las expectativas del cliente y de cumplir con los requisitos establecidos. Por otro lado, el testing se refiere a la evaluación y verificación del software para determinar si cumple con los requisitos y especificaciones.

El testing es una actividad dentro del proceso de aseguramiento de la calidad del software. A través del testing, se busca descubrir y corregir los errores y defectos del software antes de que este sea entregado a los usuarios finales. El objetivo final es garantizar que el software cumpla con los requisitos establecidos y funcione de manera correcta y eficiente.

La calidad, por su parte, es un concepto más amplio que abarca muchos aspectos diferentes del software, incluyendo la usabilidad, la confiabilidad, la escalabilidad, la seguridad, la eficiencia, entre otros. El testing es solo una parte del proceso de aseguramiento de la calidad, aunque es una parte importante y necesaria.

Es importante destacar que aunque el testing no garantiza la calidad del software, es una herramienta fundamental para lograrla.

La calidad es un concepto subjetivo que se define en base a un conjunto de propiedades, que pueden ser ponderadas de distinta forma según cada persona y situación. Jerry Weinberg define la calidad como valor para una persona, y Cem Kaner agrega que es valor para una persona a la que le interesa. Es importante tener en cuenta la falacia del Nirvana y no comparar el software con una solución ideal inexistente, sino con una solución que dé un valor real a las personas que lo utilizarán. La frase de Voltaire "lo mejor es enemigo de lo bueno" nos recuerda que buscar la perfección puede impedirnos lograr algo de calidad. En conclusión, el objetivo del testing es aportar calidad al producto que se está verificando, teniendo en cuenta que la calidad es un concepto subjetivo y complejo.

Muchas personas confunden el Plan de Calidad con el Test Plan, pero el Plan de Calidad va más allá de solo pruebas y se enfoca en cómo se cumplirán los estándares de calidad del proyecto y producto. El Control de Calidad se enfoca en medir el rendimiento del producto contra los estándares establecidos, mientras que el Aseguramiento de la Calidad se enfoca en controlar los procesos utilizados para generar los entregables.

Proceso de calidad de software

El proceso de calidad de software se refiere al conjunto de actividades planificadas y sistemáticas que se llevan a cabo durante todo el ciclo de vida del software para garantizar que se cumplan los requisitos de calidad establecidos para el producto final.

El proceso de calidad de software implica la definición de estándares y prácticas para el desarrollo de software, la revisión de los artefactos producidos durante el desarrollo del software, la prueba de software y la validación del software producido.

Es esencial para garantizar la entrega de un producto de calidad que cumpla con las expectativas del cliente y los requisitos del proyecto. Se debe realizar un estudio profundo de las características y envergadura del proyecto, de manera que pueda determinar cuáles son los tipos y niveles de pruebas que son necesarios para llevar a cabo una evaluación efectiva.

Enfocándonos en las tareas que efectúa un tester, en la planificación de las pruebas, es importante tener en cuenta el momento en que se encuentre el proyecto, y que tipo de prueba se debe aplicar en una etapa específica. Por ejemplo, las pruebas de aceptación probablemente se realizarán en una etapa más avanzada del proyecto, cuando las

funcionalidades del software estén más definidas y se pueda evaluar el comportamiento del sistema en su totalidad.

Asimismo, la realización de pruebas automatizadas puede resultar conveniente en algunos casos, pero en otros, puede no ser la mejor opción debido a la complejidad del proyecto, los costos involucrados y la necesidad de tener un equipo de especialistas en automatización.

En cuanto a las pruebas no funcionales, es fundamental tener en cuenta la relevancia de cada una de ellas y aplicarlas en el momento adecuado. Por ejemplo, las pruebas de compatibilidad se realizan para asegurarse de que el software funcione correctamente en diferentes dispositivos y sistemas operativos, pero pueden no ser necesarias en la etapa inicial del proyecto.

En resumen, el proceso de calidad de software implica un estudio detallado de las características del proyecto para determinar los tipos y niveles de pruebas necesarios, y aplicarlos en el momento adecuado. De esta manera, se garantiza la entrega de un producto de calidad que cumpla con las expectativas del cliente y los requisitos del proyecto.

¿Qué es un error?

Para dar una definición precisa de que es un error, debemos diferenciar el concepto con otras palabras que parecen similares pero tienen otro significado: fallo y defecto.

Un error es una equivocación cometida por una persona, que introduce un defecto o problema en el software. Este defecto puede causar un fallo en el sistema cuando se ejecuta en ciertas circunstancias.

Por si estos conceptos no quedan claros, vamos a explicarlos mejor.

Un error es una equivocación que comete la persona, ya sea en el código o en alguna otra parte del proceso.

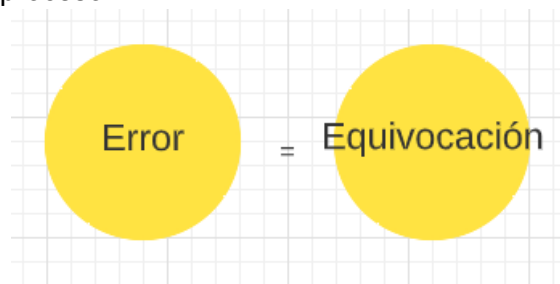


Figura 1: Error es sinónimo de equivocación. Fuente:

<https://www.diariodeqa.com/post/quiero-ser-qa-que-debo-aprender-errores-defectos-y-fallos>

Cuando se produce el error, se introduce un defecto, que es también llamado BUG. Es un problema, un inconveniente, una imperfección.



Figura 2: Sinónimo de defecto. Fuente:

<https://www.diariodeqa.com/post/quiero-ser-qa-que-debo-aprender-errores-defectos-y-fallos>

Y si ese fragmento de código que contiene un BUG se ejecuta, produce una Falla

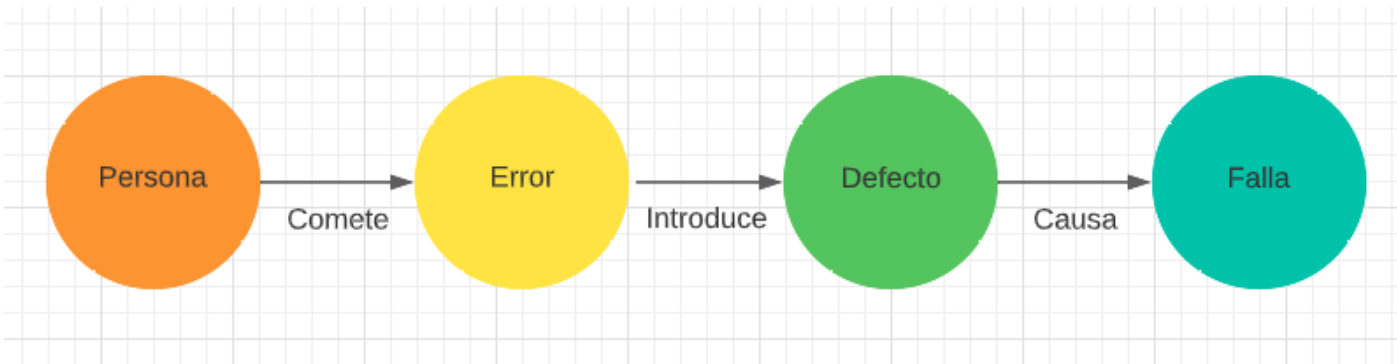


Figura 3: Diferencia de error, defecto y falla. Fuente:

<https://www.diariodeqa.com/post/quiero-ser-qa-que-debo-aprender-errores-defectos-y-fallos>

Ejemplo práctico:

Se le pide a un desarrollador que construya un sitio donde puedan acceder personas de 18 años o mayores a esa edad y él escribe lo siguiente:

```
if(edad>18){  
  console.log("Bienvenido al sitio web")  
}else{  
  console.log("No puede acceder a este sitio")  
}
```

Figura 4: Ejemplo de error . Fuente: Fragmento de código realizado por autor en Visual Studio Code

¿Puedes ver que está mal en el código?

En la consigna que se le da al desarrollador, se incluye a las personas de 18 años, y el algoritmo, no los incluye. Por ese motivo, el dev cometió un error que introdujo un defecto:

AÑOS	RESULTADO ESPERADO	RESULTADO ACTUAL
17	No puede acceder al sitio	No puede acceder al sitio
18	Bienvenido al sitio web	No puede acceder al sitio
19	Bienvenido al sitio web	Bienvenido al sitio web

Unidad 2

Categorías de las pruebas: Funcionales - No Funcionales - Mantenimiento

El propósito de las pruebas de software es identificar errores, brechas o requisitos faltantes en contraste con los requisitos reales. El producto de software debidamente probado garantiza la confiabilidad, la seguridad y el alto rendimiento, lo que además se traduce en ahorro de tiempo, rentabilidad y satisfacción del cliente.

Las pruebas de software tienen varios beneficios importantes, entre ellos:

Rentabilidad: Las pruebas de software pueden ayudar a ahorrar dinero a largo plazo al detectar errores y problemas temprano en el proceso de desarrollo, lo que significa que cuesta menos corregirlos.

Seguridad: Las pruebas de software pueden mejorar la seguridad del producto final al identificar y eliminar riesgos y problemas antes de que el producto se lance al mercado.

Calidad del producto: Las pruebas de software pueden mejorar la calidad del producto final al detectar y corregir errores y problemas antes de que lleguen a los clientes.

Satisfacción del cliente: Las pruebas de UI/UX pueden mejorar la experiencia del usuario y, por lo tanto, aumentar la satisfacción del cliente con el producto final.

En el contexto del testing de software, por lo general, las pruebas se clasifican en tres categorías.

Pruebas Funcionales:

Las pruebas funcionales están diseñadas para verificar si el software se comporta de acuerdo a lo esperado y cumple con los requisitos de funcionalidad. Estas pruebas se realizan para comprobar que el software realiza las funciones para las que ha sido diseñado de manera correcta y sin errores. Algunas de las técnicas utilizadas para realizar pruebas funcionales incluyen:

Pruebas de unidad: se prueban los componentes individuales del software para asegurarse de que funcionan correctamente.

Pruebas de integración: se prueban los diferentes módulos del software en conjunto para asegurarse de que funcionan de manera coordinada y sin errores.

Pruebas de aceptación: se realizan pruebas para verificar que el software cumple con los requisitos de funcionalidad y se comporta de acuerdo a lo esperado por el cliente.

Estos conceptos, están explicados de manera más detallada en los párrafos siguientes.

Pruebas No Funcionales:

Las pruebas no funcionales están diseñadas para evaluar aspectos del software que no están relacionados con su funcionalidad. Estas pruebas se realizan para evaluar la calidad del software en términos de aspectos como el rendimiento, la escalabilidad, la seguridad y la usabilidad. Algunas de las técnicas utilizadas para realizar pruebas no funcionales incluyen:

Pruebas de rendimiento: se realizan para evaluar el rendimiento del software en términos de velocidad, capacidad de procesamiento y uso de recursos del sistema.

Pruebas de seguridad: se realizan para evaluar la seguridad del software en términos de vulnerabilidades, ataques y medidas de protección implementadas.

Pruebas de usabilidad: se realizan para evaluar la facilidad de uso del software y la satisfacción del usuario.

Pruebas de compatibilidad: se realizan para evaluar la compatibilidad del software con diferentes sistemas operativos, navegadores y dispositivos.

Mantenimiento:

Las pruebas de mantenimiento en el testing de software son esenciales para garantizar que el software funcione correctamente después de las actualizaciones, correcciones y mejoras en el código fuente. Se realizan después de la fase de desarrollo y pruebas iniciales y antes de la entrega del software.

Estas pruebas tienen como objetivo asegurarse de que las nuevas actualizaciones o cambios no introduzcan errores o problemas adicionales en el software existente.

También incluyen la actualización de los casos de prueba existentes para asegurarse de que sigan siendo efectivos después de los cambios realizados.

Son importantes porque el software es un producto en constante evolución, y los errores pueden surgir en cualquier momento. Ayudan a garantizar que el software se mantenga en buen estado y siga cumpliendo con los requisitos y expectativas de los usuarios.

Además, también pueden ayudar a identificar oportunidades de mejora en el software y a garantizar que el software sea compatible con las últimas tecnologías y sistemas operativos.

Entre ellas podemos mencionar, las pruebas de regresión, que como acabamos de describir, son una técnica que se centra en asegurar que las funcionalidades existentes no se hayan visto afectadas después de cambios en el software.

Pruebas Manuales

Las pruebas manuales se refieren a la realización de pruebas de software sin la ayuda de herramientas automatizadas. Es decir, se llevan a cabo mediante la interacción humana con el software, simulando el comportamiento de un usuario real en diferentes escenarios y comprobando si el software funciona según lo esperado.

En las pruebas manuales, el tester o probador se encarga de diseñar casos de prueba, ejecutarlos y documentar los resultados obtenidos.

Las pruebas manuales son necesarias en situaciones donde la automatización no es posible o no es rentable. Además, las pruebas manuales son más flexibles y adaptables a cambios de requisitos en comparación con las pruebas automatizadas. Sin embargo, pueden ser más propensas a errores humanos y requieren más tiempo y recursos para su realización.

Tipos de pruebas

También llamadas pruebas de componentes, o pruebas de módulos, se centran en probar componentes individuales que se pueden probar por separado. Los objetivos de las pruebas de componentes incluyen:

- Encontrar defectos en el componente.

- Garantizar que el código y las estructuras de datos funcionen según lo previsto y especificado

- Identificación de defectos y fallas típicos

Verificar que se cumplan las especificaciones del componente

Reduciendo el riesgo

Generando confianza en la calidad del componente

Estas pruebas se centran en probar la funcionalidad de una unidad de código aislada, como una clase o un método. Las pruebas unitarias suelen ser escritas por el propio desarrollador y se ejecutan a menudo durante el proceso de desarrollo para detectar y corregir errores temprano.

Considere el desarrollo basado en pruebas (TDD). Es altamente iterativo y se basa en ciclos de desarrollo de casos de prueba automatizados, luego compila e integra pequeñas piezas de código, luego ejecuta las pruebas de componentes, corrige cualquier problema y refactoriza el código. Este proceso continúa hasta que el componente se haya construido por completo y todas las pruebas del componente hayan pasado.

En estas pruebas se pueden distinguir dos tipos principales: positivas y negativas.

En las pruebas unitarias positivas, se prueban casos en los que se espera que la unidad de código funcione correctamente. Por ejemplo, para un caso de prueba positivo, se podría probar la función sumando dos números enteros pequeños y comprobando que el resultado es la suma correcta. Luego, se podrían probar combinaciones de números más grandes, números negativos y cero, asegurándose de que la función produzca el resultado esperado para cada caso. Además, se deben incluir pruebas para casos límite como el valor mínimo y máximo que puede tomar un entero.

Por otro lado, en las pruebas unitarias negativas, se prueban casos en los que se espera que la unidad de código falle. Esto se hace para asegurar que la unidad de código maneje adecuadamente situaciones inesperadas o errores. Por ejemplo, se podrían realizar pruebas con contraseñas que no cumplen con los requisitos mínimos, como una contraseña que sea demasiado corta o no contenga caracteres especiales. También se pueden realizar pruebas con contraseñas que ya hayan sido utilizadas anteriormente, ya que esto debería ser invalidado por el sistema.

Además, se pueden realizar pruebas de límites, como intentar ingresar contraseñas extremadamente largas o con caracteres especiales que no estén permitidos por el sistema.

Al realizar pruebas unitarias negativas, se busca identificar posibles vulnerabilidades o debilidades en la función, permitiendo que se realicen correcciones antes de que el software sea lanzado al mercado.

Integración

Las pruebas de integración son un tipo de prueba de software que se centra en las interacciones e interfaces entre componentes o sistemas. El objetivo de estas pruebas es encontrar defectos en la comunicación y la interoperabilidad entre los componentes y garantizar que el comportamiento funcional y no funcional de las interfaces sea el esperado.

Hay dos niveles diferentes de pruebas de integración: las pruebas de integración de componentes y las pruebas de integración del sistema. Las pruebas de integración de componentes se centran en las interacciones e interfaces entre los componentes integrados, mientras que las pruebas de integración del sistema cubren las interacciones y las interfaces proporcionadas por organizaciones externas, como servicios web.

Las pruebas de integración son importantes porque ayudan a reducir el riesgo en el sistema y a generar confianza en la calidad de las interfaces. Además, evitan que los defectos se escapen a niveles de prueba más altos y ayudan a encontrar defectos en las propias interfaces o dentro de los componentes o sistemas.

Las pruebas de integración suelen ser realizadas por el equipo de QA y pueden ser automatizadas o manuales.

Verificación

También conocidas como pruebas estáticas, no implican la ejecución del software, sino que se centran en revisar el código y la documentación para identificar posibles errores o problemas. Algunos ejemplos de pruebas de verificación incluyen:

Revisiones de código: los desarrolladores y otros miembros del equipo revisan el código para detectar errores, problemas de rendimiento y otros problemas.

Análisis de estática de código: se utilizan herramientas automatizadas para revisar el código fuente en busca de posibles problemas, como variables no utilizadas, código muerto o problemas de seguridad.

Inspecciones de documentos: se realizan revisiones de documentos, manuales de usuario, manuales técnicos y otros documentos para garantizar que sean precisos y claros.

Es importante tener en cuenta que las pruebas de verificación no reemplazan completamente las pruebas de validación, ya que no pueden garantizar que el software funcione correctamente en todas las situaciones posibles. Sin embargo, son una parte importante del proceso de pruebas y pueden ayudar a identificar problemas antes de que el software se pruebe en un entorno de prueba completo.

Sistema

Estas pruebas se centran en probar el comportamiento y capacidades del sistema completo y se realizan después de las pruebas de integración. Las pruebas de sistema suelen probar cómo el sistema se comporta en situaciones reales y pueden incluir pruebas de rendimiento, pruebas de seguridad, pruebas de usabilidad, considerando tareas de extremo a extremo y comportamientos no funcionales que pueda exhibir.

Suelen ser realizadas por el equipo de QA y pueden ser automatizadas o manuales.

Los objetivos de las pruebas del sistema son validar que el sistema está completo y funciona como se espera, generar confianza en la calidad del sistema como un todo, reducir el riesgo y evitar que los defectos escapen a niveles de prueba o producción más altos.

Aceptación

Estas pruebas se centran en verificar que el sistema cumple con los requisitos del cliente y es aceptable para su uso, es decir, para evaluar si un sistema o producto cumple con el uso previsto y las especificaciones.

Suelen ser realizadas por el cliente o el equipo de QA en nombre del cliente.

Los objetivos de las pruebas de aceptación incluyen establecer la confianza en la calidad y la integridad del sistema, validar que el sistema funcione y se comporte como se especifica, y cumplir con los requisitos legales o reglamentarios. Los tipos comunes de pruebas de aceptación incluyen pruebas de aceptación del usuario (UAT), pruebas de aceptación operativa (OAT), pruebas alfa y beta, y pruebas de aceptación regulatorias y contractuales. Los probadores pueden usar una variedad de técnicas y herramientas para realizar pruebas de aceptación, incluidas pruebas de carga, pruebas de rendimiento, pruebas de copia de seguridad y restauración, y pruebas de vulnerabilidad.

Hay que enfatizar la importancia de involucrar a los evaluadores desde el principio en el refinamiento de las historias de los usuarios o en las actividades de pruebas estáticas, como las revisiones, para reducir la incidencia de falsos positivos y negativos en las pruebas de aceptación.

Si bien encontrar defectos durante las pruebas de aceptación no siempre es el objetivo principal, se pueden descubrir defectos durante las pruebas que pueden mejorar la calidad y la confiabilidad del sistema.

Descripción de pruebas

Pruebas de Funciones de negocio

Este tipo de prueba se enfoca en verificar que el software cumpla con los requisitos funcionales especificados, que cumpla con los requisitos de negocio establecidos. Es decir, se asegura de que las funciones del software se comporten como se espera que lo hagan.

Las pruebas de Funciones de negocio se realizan para probar la funcionalidad del software en diferentes escenarios, lo que permite evaluar el comportamiento del software en situaciones reales. Estas pruebas se basan en casos de prueba diseñados específicamente para probar cada uno de los requisitos de negocio del software.

El proceso de prueba de Funciones de negocio incluye la identificación de requisitos funcionales, el diseño de casos de prueba, la ejecución de pruebas y la presentación de informes de resultados.

Performance

O también llamadas pruebas de rendimiento se centran en evaluar la capacidad del software para cumplir con los requisitos de rendimiento especificados. Es decir, miden la capacidad del software para responder a una carga de trabajo específica. Estas pruebas se llevan a cabo para medir la velocidad, escalabilidad, estabilidad y capacidad del software bajo diferentes cargas de trabajo.

Durante estas pruebas, se realizan diferentes tipos de pruebas, como la prueba de carga, la prueba de estrés y la prueba de volumen, para evaluar el rendimiento del software en diferentes escenarios. La prueba de carga se enfoca en medir el rendimiento del software bajo una carga de trabajo determinada, mientras que la prueba de estrés se enfoca en evaluar el rendimiento del software bajo condiciones extremas de carga. La prueba de volumen se enfoca en evaluar la capacidad del software para manejar grandes cantidades de datos y transacciones.

Estas pruebas son útiles para identificar cuellos de botella y áreas de mejora en el rendimiento del software. También son útiles para predecir cómo se comportará el software en situaciones de uso en tiempo real.

Seguridad

Las pruebas de seguridad son un tipo de prueba de software que tiene como objetivo evaluar la capacidad del software para resistir diferentes tipos de ataques, incluyendo ataques de hackers, virus y malware. Estas pruebas se realizan para garantizar que el software sea seguro y no pueda ser explotado por atacantes malintencionados.

Las pruebas de seguridad pueden incluir diferentes técnicas, como pruebas de penetración, pruebas de vulnerabilidad, pruebas de autenticación y pruebas de autorización. Las pruebas de penetración implican intentar explotar las vulnerabilidades del software utilizando técnicas de hacking. Las pruebas de vulnerabilidad se enfocan en identificar posibles debilidades en el software. Las pruebas de autenticación se enfocan en evaluar la fortaleza de los sistemas de autenticación del software, mientras que las pruebas de autorización se enfocan en evaluar la capacidad del software para restringir el acceso no autorizado a los datos.

Interfaz de usuario

Estas pruebas evalúan la usabilidad del software. Se enfocan en la experiencia del usuario al utilizar el software, la facilidad de uso, la navegación y la apariencia visual. Se realizan para garantizar que el software tenga una interfaz de usuario intuitiva, fácil de usar y estéticamente atractiva.

Stress

Las pruebas de estrés son una técnica utilizada en la ingeniería de software para evaluar el comportamiento del software bajo condiciones extremas de carga. Estas pruebas buscan detectar los límites del software en cuanto a su capacidad de respuesta y su capacidad para mantener su funcionamiento estable.

En general, las pruebas de estrés se realizan para identificar cualquier debilidad en la infraestructura del software que pueda causar un fallo del sistema en condiciones extremas. Estas pruebas pueden involucrar la simulación de una gran cantidad de usuarios accediendo simultáneamente al sistema, una gran cantidad de solicitudes o transacciones procesadas por segundo, o cualquier otro factor que pueda tener un impacto significativo en el rendimiento del software.

Existen diferentes tipos de pruebas de estrés, como las pruebas de carga, las pruebas de rendimiento y las pruebas de capacidad. En general, estas pruebas se realizan en diferentes etapas del ciclo de vida del software, desde las pruebas unitarias hasta las

pruebas de aceptación. Las pruebas de carga simulan la actividad normal del sistema bajo una carga de trabajo específica y mide cómo el sistema se comporta y responde a esta carga. Se realizan para determinar la capacidad de un sistema para manejar la cantidad de usuarios o transacciones que se esperan en condiciones normales. También se pueden utilizar para identificar cuellos de botella y mejorar el rendimiento del sistema. Las Pruebas de rendimiento miden la velocidad, capacidad de respuesta y escalabilidad de un sistema bajo diferentes cargas de trabajo. Se utilizan para evaluar el rendimiento del sistema en términos de tiempo de respuesta, latencia, velocidad de transferencia de datos y capacidad de procesamiento. También pueden ayudar a identificar problemas de red, servidores y hardware, y optimizar la configuración del sistema para lograr un mejor rendimiento. Las pruebas de capacidad evalúan la capacidad máxima de un sistema para manejar una carga de trabajo específica. Estas pruebas se utilizan para identificar los límites del sistema y para determinar cuántos usuarios o transacciones el sistema puede manejar antes de que se produzcan fallas o se degrade el rendimiento. También pueden ayudar a identificar problemas de infraestructura, como la necesidad de más recursos de hardware o software, para asegurar que el sistema pueda manejar la carga de trabajo esperada.

Las pruebas de estrés son importantes para garantizar la calidad del software y su capacidad para responder a las demandas de los usuarios en situaciones extremas. Sin embargo, es importante tener en cuenta que estas pruebas también pueden tener un impacto en la infraestructura del software, por lo que deben ser planificadas cuidadosamente y realizadas por profesionales con experiencia en este tipo de pruebas.

Volumen

Las pruebas de volumen son una técnica de pruebas de carga que se enfoca en evaluar la capacidad del software para manejar grandes cantidades de datos y transacciones. Estas pruebas se realizan para simular situaciones de alta demanda, donde el sistema debe procesar grandes cantidades de datos y transacciones simultáneamente.

El objetivo principal de las pruebas de volumen es identificar los límites de capacidad del software, es decir, cuántos datos o transacciones el sistema puede manejar sin afectar el rendimiento y la funcionalidad del software. También se evalúa la capacidad del sistema para escalar, es decir, para manejar mayores volúmenes de datos y transacciones en el futuro.

Además de evaluar la capacidad del software, las pruebas de volumen también pueden ayudar a identificar problemas de infraestructura, como la necesidad de más recursos

de hardware o software, para asegurar que el sistema pueda manejar grandes cantidades de datos y transacciones.

Usabilidad

Las pruebas de usabilidad son una técnica de pruebas de software que se enfoca en evaluar la facilidad de uso del software y la satisfacción del usuario al utilizarlo. Estas pruebas se realizan para asegurar que el software sea fácil de usar y que cumpla con las expectativas de los usuarios.

Durante las pruebas de usabilidad, se evalúan diferentes aspectos del software, como la facilidad de aprendizaje, la eficiencia de uso, la facilidad de recordar y la satisfacción general del usuario.

El objetivo principal de las pruebas de usabilidad es identificar problemas en el diseño y la usabilidad del software, para poder corregirlos y mejorar la experiencia del usuario.

Regresion

Las pruebas de regresión son una técnica de pruebas de software que se enfoca en asegurarse de que las funcionalidades existentes no se hayan visto afectadas después de realizar cambios en el software. Estas pruebas son importantes para garantizar que el software siga funcionando correctamente y que los cambios realizados no hayan causado problemas en otras partes del sistema.

Durante las pruebas de regresión, se ejecutan una serie de pruebas diseñadas para evaluar las funcionalidades críticas del software que no deberían haberse visto afectadas por los cambios recientes.

Smoke / Pruebas de Humo

Las pruebas de humo (smoke tests) son una técnica de pruebas de software que consiste en ejecutar un conjunto básico de pruebas para verificar que el software funciona correctamente después de una actualización o corrección menor. Estas pruebas son muy superficiales y rápidas, y están diseñadas para detectar errores críticos que puedan impedir que se realicen pruebas más exhaustivas.

Las pruebas de humo se ejecutan antes de que se realicen pruebas más detalladas para asegurarse de que el software esté estable y funcional. Generalmente, las pruebas de humo se enfocan en verificar la funcionalidad básica del software y se realizan sobre

una versión inicial del software. Estas pruebas no cubren todas las funcionalidades del software, sino sólo las más importantes y críticas.

El objetivo de las pruebas de humo es detectar problemas críticos de forma temprana, para evitar que se propaguen a otras áreas del software y se conviertan en problemas mayores.

Unidad 3

Proceso de Pruebas

El proceso de pruebas en testing se refiere a las diferentes etapas que se siguen para asegurar que un software cumpla con los requisitos y estándares de calidad esperados. El proceso de pruebas implica la planificación, diseño, ejecución, monitoreo y reporte de las pruebas realizadas sobre el software.

El proceso de pruebas suele seguir un enfoque sistemático y se enfoca en identificar los posibles problemas y errores del software, como fallos de funcionalidad, de rendimiento, de seguridad o de usabilidad, entre otros. Una vez que se detectan estos errores, se deben documentar, reportar y priorizar para su posterior corrección por parte del equipo de desarrollo.

Es importante tener en cuenta que el proceso de pruebas puede variar dependiendo del tipo de software y de los requisitos específicos de cada proyecto.

Un tipo de proceso de pruebas es Fases STLC (Ciclo de vida de prueba de software). Donde cada etapa tiene un criterio definido de Entrada y Salida, Actividades y Entregables asociados con ella.

Análisis de requisitos:

Durante esta fase, el equipo de control de calidad interactúa con diferentes partes interesadas para comprender los requisitos en detalle y determinar qué requisitos son comprobables. Los requisitos pueden ser funcionales o no funcionales.

Las actividades clave en la fase de requisitos de prueba incluyen la identificación de los tipos de pruebas que se realizarán, la recolección de detalles sobre las prioridades y el enfoque de las pruebas, la preparación de la Matriz de Trazabilidad de Requerimientos (RTM) para garantizar que cada requisito esté cubierto por al menos una prueba, la identificación de los detalles del entorno de prueba y el análisis de la factibilidad de automatización si se requiere.

Los entregables de la fase de requisitos de prueba incluyen la RTM que rastrea la cobertura de las pruebas para cada requisito y un informe de viabilidad de automatización si se determina que la automatización es factible y necesaria.

Planificación de pruebas

Implica la determinación de la estrategia del plan de pruebas para el proyecto, la estimación de los esfuerzos y costos del proyecto, la identificación de los recursos necesarios y la selección de las herramientas de prueba adecuadas. También se determinan el entorno de prueba, las limitaciones y el programa de pruebas durante esta fase.

Las actividades clave en la planificación de pruebas incluyen la preparación del plan de prueba o documento de estrategia para varios tipos de pruebas, la selección de las herramientas de prueba adecuadas, la estimación del esfuerzo de prueba necesario, la planificación de los recursos y la determinación de los roles y responsabilidades de los miembros del equipo de pruebas, así como la identificación de los requisitos de formación.

Los entregables de la planificación de pruebas incluyen el plan de prueba o documento de estrategia que establece la estrategia de pruebas para el proyecto y el documento de estimación de esfuerzo que describe los esfuerzos y costos previstos para la realización de las pruebas.

Desarrollo de casos de prueba

Implica la creación, verificación y reelaboración de casos de prueba y scripts de prueba una vez que el plan de prueba ha sido establecido. En esta fase, los datos de prueba son identificados, creados, revisados y re-trabajados en base a las condiciones previas. Luego, el equipo de control de calidad inicia el proceso de desarrollo de casos de prueba para unidades individuales.

Las actividades clave en la fase de desarrollo de casos de prueba incluyen la creación de casos de prueba y scripts de automatización (si es necesario), la revisión y línea base de los casos de prueba y scripts, y la creación de datos de prueba si el entorno de prueba está disponible.

Los entregables de la fase de desarrollo de casos de prueba incluyen los casos de prueba o guiones desarrollados y los datos de prueba creados para ser utilizados en la ejecución de las pruebas.

Configuración del entorno de prueba

Es la actividad en la que se establecen las condiciones de software y hardware necesarias para llevar a cabo las pruebas de un producto de trabajo. Es esencial para el proceso de prueba y se puede llevar a cabo en paralelo con la fase de desarrollo de casos de prueba. El equipo de prueba debe verificar la preparación del entorno (mediante una prueba de humo) antes de comenzar a realizar las pruebas. Las actividades en esta fase incluye comprender la arquitectura requerida, la configuración del entorno, preparar una lista de requisitos de hardware y software, configurar el entorno de prueba y los datos de prueba, y realizar una prueba de humo. Los entregables de esta fase incluyen el entorno de prueba configurado y los datos de prueba, así como los resultados de la prueba de humo.

Ejecución de pruebas

Se realiza la prueba de la compilación del software en función de los planes de prueba y los casos de prueba preparados. El proceso consiste en la ejecución del script de prueba, el mantenimiento del script de prueba y la notificación de errores. Si se informan errores, se devuelve al equipo de desarrollo para su corrección y se realizan nuevas pruebas.

Los entregables clave de la fase de ejecución de pruebas son el RTM (Matriz de requerimientos) actualizado con el estado de ejecución, casos de prueba actualizados con resultados e informes de defectos.

Cierre del ciclo de prueba

Verificación de que los errores hayan sido corregidos y que el software cumpla con los requisitos de calidad establecidos.

Los resultados de la evaluación del ciclo de prueba, incluyendo las métricas de prueba, se documentan en el informe de cierre de prueba. Además, se documentan las lecciones aprendidas durante el proyecto y se proporciona un análisis de los resultados de las pruebas, que incluyen la distribución de los defectos por tipo y gravedad. También se proporciona un informe cualitativo y cuantitativo de la calidad del producto del trabajo al cliente.

Identificar Funcionalidades

La identificación de funcionalidades es un proceso crucial que implica comprender las diversas funciones y características que se espera que un software o sistema proporcione.

Comienza con la comprensión de los requisitos del usuario y la documentación, y se extiende hasta la creación de casos de prueba para validar la funcionalidad. El objetivo

principal es asegurarse de que el software cumpla con las necesidades y expectativas del usuario final.

Para identificar las funcionalidades, se pueden utilizar diversas técnicas, como entrevistas con el cliente, revisión de documentos, análisis de requisitos, casos de uso, diagramas de flujo y mapas mentales. Es importante tener una comprensión completa de las funcionalidades y requisitos para poder desarrollar casos de prueba sólidos que cubran todas las posibles situaciones y escenarios.

Una vez que se han identificado las funcionalidades, es importante priorizarlas y enfocarse en las más críticas y de alta prioridad. Esto ayuda a asegurar que el software se entregue en el plazo requerido y con las funcionalidades necesarias.

Técnicas para identificar funcionalidades:

- Entrevistas con el cliente: Esta técnica implica realizar entrevistas con el cliente para entender sus necesidades y expectativas en cuanto a las funcionalidades del producto. Durante la entrevista, se puede hacer preguntas abiertas y cerradas para obtener información detallada.
- Encuestas: Consiste en hacer preguntas a un grupo de personas, generalmente usuarios potenciales del sistema o personas que están familiarizadas con el dominio del sistema. Las preguntas pueden ser abiertas o cerradas y deben estar diseñadas para obtener información sobre las necesidades y expectativas de los usuarios con respecto al sistema.
Para diseñar una encuesta efectiva para identificar funcionalidades, es importante tener en cuenta lo siguiente:
 - Identificar el grupo objetivo: Es importante saber quiénes serán los destinatarios de la encuesta, para poder elaborar preguntas específicas y relevantes para su perfil.
 - Diseñar preguntas claras y precisas: Las preguntas deben ser claras y precisas para que los encuestados puedan entenderlas fácilmente y proporcionar respuestas útiles.
 - Incluir preguntas abiertas y cerradas: Es importante incluir una mezcla de preguntas abiertas y cerradas para obtener tanto información cuantitativa como cualitativa.
 - Limitar la cantidad de preguntas: Es importante no hacer demasiadas preguntas para no abrumar a los encuestados.
- Revisión de documentos: La revisión de documentos implica analizar cualquier documento relacionado con el proyecto, como especificaciones de requisitos, casos de uso anteriores, plan de proyecto, etc. Esto puede ayudar a identificar las funcionalidades que se han incluido anteriormente y a evitar duplicaciones innecesarias.

- **Análisis de requisitos:** Esta técnica implica identificar, analizar y documentar los requisitos del sistema. Los requisitos se clasifican en funcionales y no funcionales y se documentan en una especificación de requisitos.
- **Casos de uso:** Los casos de uso se utilizan para describir cómo un usuario interactúa con el sistema para realizar una tarea específica.
- **Diagramas de flujo:** Los diagramas de flujo son una representación gráfica de un proceso o sistema. Los diagramas de flujo se pueden utilizar para representar el flujo de datos y las interacciones entre los componentes del sistema.
- **Mapas mentales:** Los mapas mentales son una técnica visual para organizar ideas y conceptos. Los mapas mentales se pueden utilizar para identificar cómo se relacionan entre sí las distintas funcionalidades. Esta técnica también puede ayudar a identificar las áreas que necesitan más investigación y análisis.
- **Prototipo:** es una versión preliminar del sistema que permite a los usuarios probar y proporcionar comentarios sobre las funcionalidades que les gustaría ver en el sistema final. Los comentarios y las solicitudes de funcionalidades se pueden utilizar para ajustar el diseño y mejorar la experiencia del usuario.

Cada técnica puede ser útil por sí misma o en combinación con otras para identificar las funcionalidades necesarias para un sistema. Depende del equipo de proyecto elegir las técnicas adecuadas según las necesidades del proyecto y el tiempo y los recursos disponibles.

Entonces, ¿proceso de calidad de software y procesos de pruebas, son lo mismo?
Te sugiero que tomes unos minutos, y de acuerdo a lo leído hasta este momento, respondas esa pregunta...

...El proceso de calidad de software y el proceso de pruebas son dos procesos distintos que forman parte del ciclo de vida del software.

El proceso de calidad de software incluye todas las actividades que se llevan a cabo para asegurar que el software cumpla con los estándares y requisitos de calidad establecidos. Esto incluye la planificación de calidad, la gestión de calidad, la garantía de calidad y el control de calidad. En este proceso se definen las políticas y procedimientos necesarios para garantizar la calidad del software y se realizan revisiones y auditorías para asegurarse de que se están cumpliendo los estándares de calidad.

Por otro lado, el proceso de pruebas es una parte importante del proceso de calidad de software y se enfoca en detectar defectos, errores y sugerir mejoras en el software antes de su lanzamiento al mercado. El objetivo de las pruebas es identificar y corregir problemas para garantizar que el software funcione correctamente y cumpla con los requisitos de calidad.

En resumen, el proceso de calidad de software es un proceso más amplio que incluye el proceso de pruebas como una de sus partes esenciales.

Escenario de prueba

Un caso de prueba y un escenario de prueba son términos utilizados en la prueba de software para describir dos conceptos diferentes, aunque relacionados.

Un caso de prueba es una descripción de los pasos que se deben seguir para probar una funcionalidad específica y los resultados esperados.

Un caso de prueba debe incluir los siguientes elementos:

- Nombre del caso de prueba: El nombre debe ser lo suficientemente descriptivo para identificar la funcionalidad o característica que se está probando.
- Objetivo del caso de prueba: El objetivo debe describir lo que se espera lograr al realizar el caso de prueba.
- Descripción del caso de prueba: La descripción debe incluir los pasos detallados que se deben seguir para completar el caso de prueba.
- Entradas: Las entradas son los datos que se necesitan para completar el caso de prueba.
- Resultados esperados: Es la salida o respuesta que se anticipa obtener después de realizar una acción o evento específico en un sistema o proceso determinado. Es la expectativa de un resultado en base a las condiciones y entradas de un escenario o prueba, y se utiliza como una medida para determinar si el sistema o proceso está funcionando correctamente o no. Es la respuesta o comportamiento previsto de un sistema o proceso en una situación determinada.
- Criterios de aceptación: Los criterios de aceptación establecen las condiciones que deben cumplirse para considerar el caso de prueba como aprobado.

Los casos de prueba son útiles porque ayudan a garantizar que cada funcionalidad o característica del sistema se pruebe adecuadamente.

Por otro lado, un escenario de prueba se define como cualquier funcionalidad que se puede probar. También se denomina condición de prueba o posibilidad de prueba. Debe ponerse en el lugar del usuario final y descubrir los escenarios del mundo real y los casos de uso de la aplicación bajo prueba.

Es un método en el que se utilizan escenarios reales para probar la aplicación de software en lugar de casos de prueba. El propósito de las pruebas de escenarios es probar escenarios de extremo a extremo para un problema complejo específico del

software. Los escenarios ayudan de una manera más fácil a probar y evaluar problemas complicados de extremo a extremo.

¿Por qué crear escenarios de prueba?

Los escenarios de prueba se crean por las siguientes razones,

- La creación de escenarios de prueba garantiza una cobertura de prueba completa
- Los escenarios de prueba pueden ser aprobados por varias partes interesadas, como Business Analyst, Developers, Customers para garantizar que la aplicación bajo prueba se pruebe a fondo. Garantiza que el software funcione para los casos de uso más comunes.
- Sirven como una herramienta rápida para determinar el esfuerzo de trabajo de prueba y, en consecuencia, crear una propuesta para el cliente u organizar la fuerza de trabajo.
- Ayudan a determinar las transacciones de extremo a extremo más importantes o el uso real de las aplicaciones de software.
- Para estudiar el funcionamiento integral del programa, el escenario de prueba es fundamental.

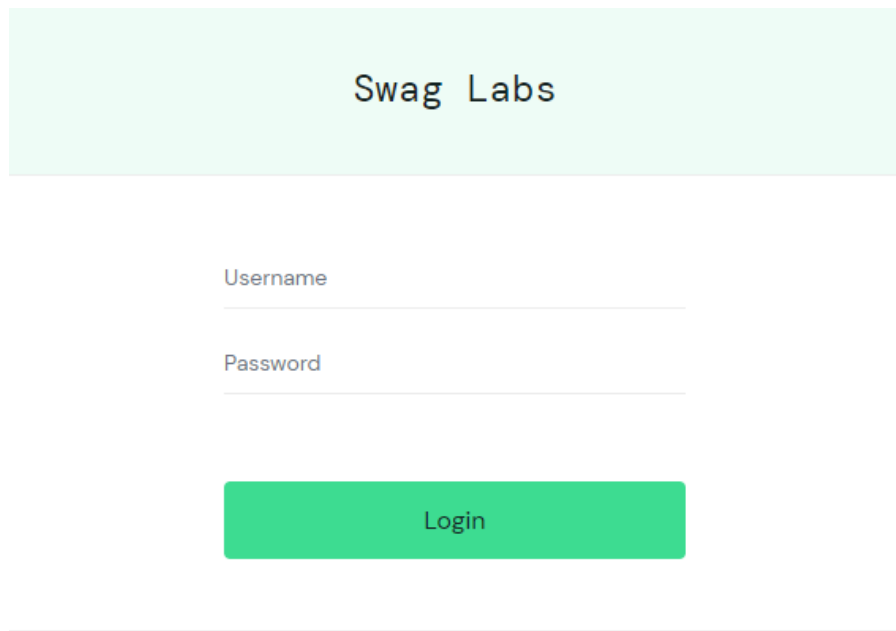
Cómo escribir escenarios de prueba

1. Lea los Documentos de requisitos del Sistema bajo prueba. También se pueden referir casos de uso, libros, manuales, etc. de la aplicación a probar.
2. Para cada requisito, descubra las posibles acciones y objetivos de los usuarios. Determinar los aspectos técnicos del requerimiento. Determinar posibles escenarios de abuso del sistema y evaluar a los usuarios con mentalidad de hacker.
3. Después de leer el documento de requisitos y realizar el debido análisis, enumera diferentes escenarios de prueba que verifiquen cada característica del software.
4. Una vez que haya enumerado todos los escenarios de prueba posibles, se crea una matriz de trazabilidad para verificar que todos y cada uno de los requisitos tengan un escenario de prueba correspondiente.

Ahora vamos a los ejemplos para que quede más claro lo explicado hasta aquí:

Ejemplo: Escenario de prueba para la aplicación de comercio electrónico. Link: <https://www.saucedemo.com/>

Escenario de prueba 1: comprobar la funcionalidad de inicio de sesión.



Swag Labs

Username

Password

Login

Figura 5: Login de Página de prueba. Fuente: <https://www.saucedemo.com/>

Para ayudarlo a comprender la diferencia entre el escenario de prueba y los casos de prueba, los casos de prueba específicos para este escenario de prueba serían:

- Verifique el comportamiento del sistema cuando se ingresa una identificación de correo electrónico y una contraseña válidas.
- Verifique el comportamiento del sistema cuando se ingresa una identificación de correo electrónico no válida y una contraseña válida .
- Verifique el comportamiento del sistema cuando se ingresa una identificación de correo electrónico válida y una contraseña no válida .
- Verifique el comportamiento del sistema cuando se ingresa una identificación de correo electrónico y una contraseña no válidas .
- Verifique el comportamiento del sistema cuando la identificación de correo electrónico y la contraseña se dejan en blanco y se ingresa Iniciar sesión.
- Comprobar Olvidó su contraseña está funcionando como se esperaba

Como es evidente, los casos de prueba son más específicos.

Además de este escenario, otros posibles podrían ser:

Escenario de prueba 2: verificar la descripción del producto

Y en el, anotaremos los posibles casos de prueba. ¿Se te ocurren algunos casos de prueba?

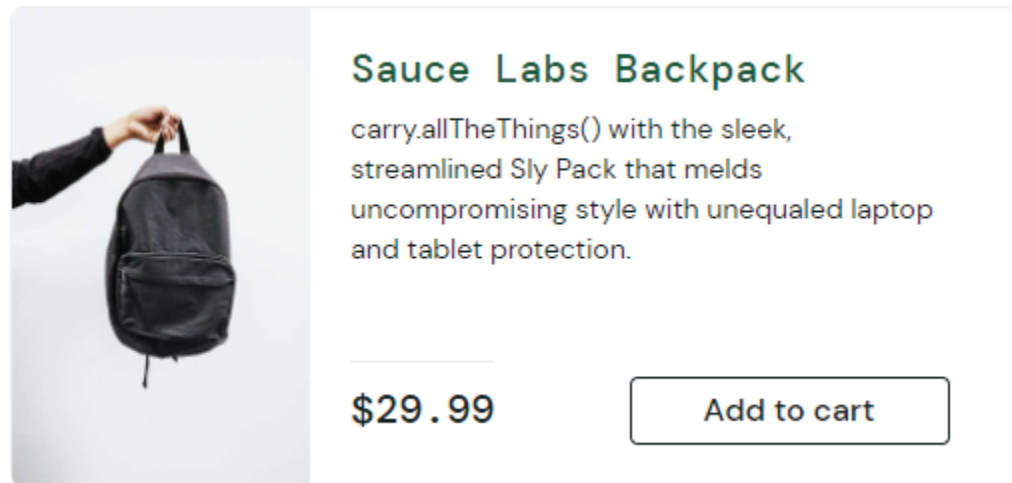


Figura 6: Detalle de producto de la página de prueba Suacedemo. Fuente: <https://www.saucedemo.com/inventory.html>

Algunos ejemplos de casos de pruebas serían:

- Verificar que la información del producto se muestra correctamente en la página, incluyendo el nombre del producto, el valor del precio, el formato de cada una de estas partes.
- Verificar que la imagen se visualiza correctamente, que el tamaño es el correcto.
- Verificar la funcionalidad del Botón "Add to cart"

Escenario de prueba 3: Comprobar el historial de pedidos

Algunos ejemplos de casos de pruebas serían:

- Verificar que el boton de carrito funciona correctamente
- Verificar que en el listado se visualice correctamente todos los productos agregados, entre otros casos.

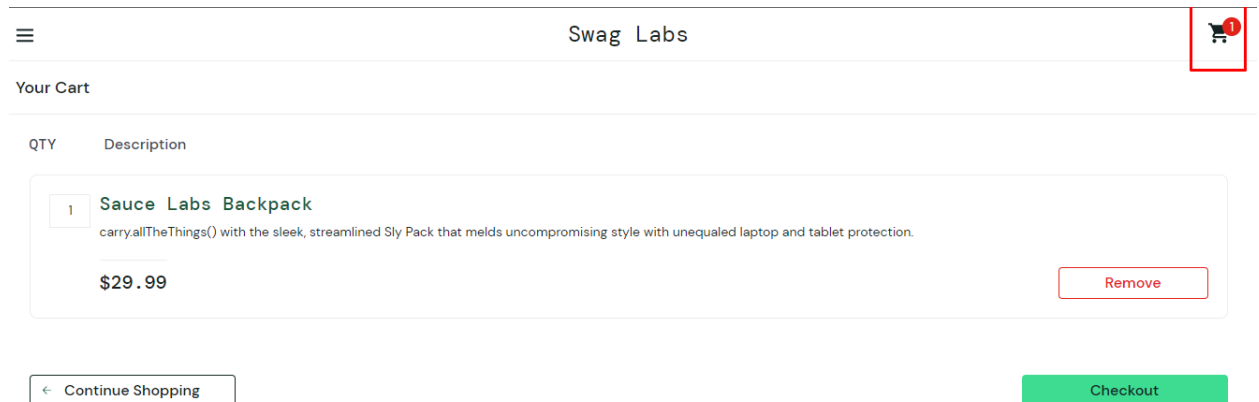


Figura 7: Listado de Carrito de página Saucedemo. Fuente: <https://www.saucedemo.com/cart.html>

Casos de Prueba (diseño, ejecución, resultado)

En este apartado aprenderemos a realizar un caso de prueba, ejecutarlo y ver los resultados.

Los escenarios de prueba son bastante vagos y cubren una amplia gama de posibilidades. La prueba trata de ser muy específica.

Donde se realizan los casos de prueba? Depende de cada caso, tamaño y complejidad del proyecto y equipo.

Pero las herramientas que usamos pueden ser entre otras: un documento word o excel, aplicaciones como “TestCollab” (que tienen una versión gratuita y otra paga), Jira o, ya un poco más complejo, realizar los casos de prueba de manera automatizada (Selenium, Cypress, etc).

Para explicar la manera de diseñar y ejecutar un caso de prueba, lo haremos de manera práctica con un ejemplo.

A continuación se muestra un paso a paso un manera de cómo diseñar un caso de prueba:

- Identificar el objetivo: El primer paso para diseñar un caso de prueba es identificar el objetivo del mismo. ¿Qué funcionalidad o requisito se va a probar? ¿Cuál es el resultado esperado?
- Identificar las entradas: Identifique todas las entradas necesarias para ejecutar el caso de prueba. Esto incluye cualquier información o datos necesarios para la ejecución del caso de prueba.

En este caso se deben tener en cuenta todas las características de cada entrada: Cantidad mínima y máxima de caracteres, tipo de caracteres que admite el campo (numéricos, alfabéticos, especiales), si admite campo vacío o

no, validaciones necesarias (Ejemplo: campo edad, si es para mayores de 18 años, que incluya mensaje si el usuario ingresa 17).

- Identificar las salidas: Identifique las salidas que se esperan del caso de prueba. ¿Qué resultados se esperan de la ejecución del caso de prueba?
- Diseñar los pasos del caso de prueba: Una vez que se han identificado las entradas y las salidas, es hora de diseñar los pasos del caso de prueba. Esto incluye las acciones que se deben realizar para ejecutar el caso de prueba. Es decir, un paso a paso que debe ser claro, detallado y específico.
- Identificar los criterios de éxito: Identifique los criterios de éxito que se deben cumplir para que el caso de prueba se considere exitoso. Esto puede incluir la verificación de resultados, el tiempo de respuesta o la usabilidad.
- Establecer las condiciones previas: Identifique cualquier condición previa que se deba cumplir antes de la ejecución del caso de prueba. Esto puede incluir la configuración del sistema o la creación de datos de prueba.
- Establecer las condiciones posteriores: Identifique cualquier condición posterior que se deba cumplir después de la ejecución del caso de prueba. Esto puede incluir la restauración del sistema a su estado anterior o la eliminación de datos de prueba.
- Revisión y validación: Una vez que se ha diseñado el caso de prueba, es importante revisarlo y validarlo con otros miembros del equipo de prueba. Esto ayuda a identificar cualquier error o problema antes de la ejecución del caso de prueba.

Ahora vamos con un ejemplo de cómo diseñar un caso de prueba para una funcionalidad de **inicio de sesión en un sitio web**:

- Identificar el objetivo: El objetivo de este caso de prueba es verificar si el usuario puede iniciar sesión correctamente en el sitio web utilizando un nombre de usuario y una contraseña válidos, y si no puede iniciar sesión con un nombre de usuario o contraseña inválidos.
- Identificar las entradas: Las entradas para este caso de prueba son el nombre de usuario y la contraseña del usuario.
- Identificar las salidas: La salida esperada es que el usuario sea redirigido a la página de inicio después de iniciar sesión correctamente, y que no pueda iniciar sesión si el nombre de usuario o la contraseña son inválidos.
- Diseñar los pasos del caso de prueba:
 - A. Navegar al sitio web de inicio de sesión.
 - B. Introducir el nombre de usuario válido en el campo correspondiente.
 - C. Introducir la contraseña válida en el campo correspondiente.
 - D. Hacer clic en el botón "Iniciar sesión".
 - E. Verificar que el usuario ha sido redirigido a la página de inicio.
 - F. Cerrar sesión.

- G. Repetir los pasos a, b y c pero introduciendo un nombre de usuario o contraseña inválidos.
- H. Verificar que el usuario no pueda iniciar sesión y reciba un mensaje de error.
- Identificar los criterios de éxito: El criterio de éxito para este caso de prueba es que el usuario pueda iniciar sesión correctamente con un nombre de usuario y una contraseña válidos, y que no pueda iniciar sesión con un nombre de usuario o contraseña inválidos.
- Establecer las condiciones previas: La condición previa para este caso de prueba es que el sitio web esté disponible y que el usuario tenga un nombre de usuario y una contraseña válidos.
- Establecer las condiciones posteriores: La condición posterior para este caso de prueba es que el usuario pueda acceder al home de la página cuando ingresa usuario y contraseña válidos y que pueda leer el mensaje de error, informando que el usuario y contraseña son incorrectos, dependiendo en cada caso, según corresponda.
- Revisión y validación: El caso de prueba debe ser revisado y validado por otros miembros del equipo de prueba para asegurarse de que es completo y preciso.

Con este caso de prueba, se puede verificar que la funcionalidad de inicio de sesión en el sitio web funciona correctamente tanto con entradas válidas como inválidas. Al incluir una prueba unitaria negativa, el equipo de prueba también puede detectar posibles errores en la validación de entradas en el sitio web.

Matriz de trazabilidad

Una matriz de trazabilidad es una herramienta de gestión de proyectos que ayuda a mantener el control de los requisitos del proyecto y a garantizar que se cumplan. Es un documento que relaciona los requisitos del proyecto con los casos de prueba, la documentación del usuario final y otros documentos clave para el proyecto. Esto ayuda a garantizar que todas las partes interesadas del proyecto estén en la misma página y que se logren los objetivos del proyecto de manera efectiva. La matriz de trazabilidad es útil para verificar la integridad de las relaciones de muchos a muchos entre los documentos relacionados y para asegurarse de que no se deje ningún requisito sin probar.

La matriz de trazabilidad de requisitos (RTM) es un documento que ayuda a garantizar que todos los requisitos del usuario se hayan probado adecuadamente a través de casos de prueba y que se hayan verificado en el software desarrollado. En otras palabras, la RTM es una herramienta que asegura que todos los requisitos del usuario se hayan cubierto y se hayan cumplido antes de la entrega del software final. Además, la RTM también puede ayudar a identificar y rastrear cualquier cambio en los requisitos durante el ciclo de vida del desarrollo del software.

Generación de Datos

Los datos de prueba son la información proporcionada a un programa de software durante su prueba para verificar su funcionamiento. Estos datos pueden ser afectados o ser influenciados por la ejecución del software durante la prueba. Los datos de prueba se utilizan tanto para comprobar el correcto funcionamiento del software con entradas conocidas, como para evaluar su capacidad de manejar entradas inusuales o excepcionales.

Es importante tener en cuenta que si los datos de prueba no están bien diseñados, puede ser que no se prueben todos los escenarios posibles y esto afectaría la calidad del software. Por lo tanto, es esencial contar con una selección adecuada de datos de prueba para lograr una prueba exhaustiva y efectiva del software.

Dependiendo de su entorno de prueba, es posible que necesite CREAR datos de prueba (la mayoría de las veces) o al menos identificar datos de prueba adecuados para sus casos de prueba (si los datos de prueba ya están creados).

Por lo general, los datos de prueba se crean en sincronización con el caso de prueba para el que se pretende utilizar.

Métricas

Las métricas en el testing se utilizan para medir la calidad y eficacia de los procesos de pruebas de software. Algunas de las métricas comunes en el testing incluyen:

- Cobertura de prueba: mide el porcentaje de código o funcionalidad que se ha probado. Puede ser medido en términos de líneas de código, funciones, clases, requisitos, etc.
- Tiempo de respuesta: mide el tiempo que tarda una aplicación en responder a una solicitud del usuario. Esta métrica es importante para evaluar la calidad del rendimiento de la aplicación.
- Defectos encontrados: mide el número de defectos encontrados durante las pruebas. Esta métrica ayuda a determinar la calidad del software y el éxito de las pruebas.
- Tiempo de resolución de defectos: mide el tiempo que tarda el equipo de desarrollo en resolver los defectos encontrados durante las pruebas.
- Estabilidad del sistema: mide la capacidad del sistema para funcionar de manera confiable durante un período de tiempo determinado.
- Calidad de los datos: mide la precisión y la integridad de los datos que se utilizan en las pruebas.
- Satisfacción del usuario: mide la satisfacción del usuario con la funcionalidad, la usabilidad y el rendimiento de la aplicación.

Es importante seleccionar las métricas adecuadas en función de los objetivos específicos de prueba y de los requisitos del proyecto.

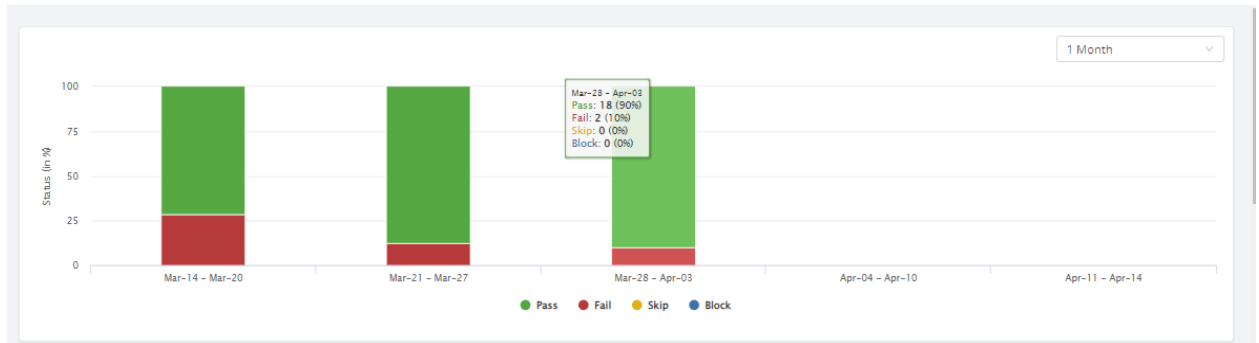


Figura 8: Gráfico donde se visualiza el porcentaje de casos de pruebas ejecutados, discriminados por aprobados y fallidos. Fuente: TestCollab.

★ MÓDULO 2: Fundamentos de Testing

Objetivo: Que los participantes adquieran los conocimientos en prácticas de testing colaborativas, para la detección temprana de errores y problemas, en el ciclo de vida del desarrollo, lo que permite una corrección más rápida y efectiva.

Unidad 1

Testing continuo en el ciclo DEVOPS

Primero deberíamos definir qué es el Agile Testing. Se refiere a la aplicación de prácticas de testing que se realizan de manera colaborativa y constante, desde el inicio hasta la entrega del producto y más allá. El objetivo principal es garantizar que se entregue valor de manera frecuente a nuestros clientes, asegurando la calidad del producto. Las actividades de testing se enfocan en crear calidad desde el inicio del proceso, y utilizan ciclos rápidos de retroalimentación para validar nuestro entendimiento y corregir cualquier problema lo antes posible.

En el enfoque de Agile Testing, se fortalece la idea de que todo el equipo es responsable de la calidad del producto, no sólo el equipo de testing. Por lo tanto, se fomenta la colaboración y la comunicación constante entre los miembros del equipo, lo que permite una mayor eficiencia en la identificación y resolución de problemas. En resumen, Agile Testing es un enfoque de testing colaborativo y continuo que busca garantizar la calidad del producto y la satisfacción del cliente.

Segundo deberíamos entender que es Devops y que es Agilidad.

DevOps es una metodología o conjunto de prácticas que busca integrar y coordinar los equipos de desarrollo de software y operaciones de tecnología para mejorar la eficiencia y velocidad de entrega de software, así como también la calidad y la estabilidad de los sistemas. El término DevOps es una combinación de "development" (desarrollo) y "operations" (operaciones), y se enfoca en la colaboración, la comunicación, la automatización y la medición de resultados para lograr una entrega de software más rápida, más segura y más confiable. En resumen, DevOps busca unir las áreas de desarrollo y operaciones de TI, y aplicar principios ágiles para mejorar la entrega de software y los resultados empresariales.

La agilidad en el testing es una práctica dentro del proceso de desarrollo ágil que se enfoca en la iteración continua de pruebas y validación de software. Esta práctica se integra en el proceso SDLC para asegurar la calidad del software a medida que se desarrolla. La metodología ágil se enfoca en desarrollar el software de manera iterativa, incremental y evolutiva, dividiendo el producto en piezas más pequeñas y luego integrándose para su prueba final. Esto permite que los equipos de desarrollo de software puedan recibir comentarios y realizar ajustes a medida que avanzan en el proceso de desarrollo. Las metodologías ágiles más populares para el testing son Scrum, Kanban y XP.

Ágil vs. DevOps

Partes interesadas y cadena de comunicación en un proceso típico de TI.



Figura 9: Comunicación en un proceso TI. Fuente: <https://www.guru99.com/agile-vs-devops.html>

Agile aborda las brechas en las comunicaciones con el cliente y el desarrollador



DevOps aborda las brechas en las comunicaciones de operaciones de TI y desarrolladores



Figura 10 : Brechas de comunicación Ágil y DevOp. Fuente: <https://www.guru99.com/agile-vs-devops.html>

Al incorporar el testing en todas las etapas del ciclo de vida de desarrollo, podemos aprender cómo los usuarios interactúan con nuestro software y mejorar continuamente su calidad. En resumen, el testing es una parte integral del enfoque DevOps, en el que trabajamos en colaboración para crear, entregar y mantener software de alta calidad de manera rápida y eficiente.

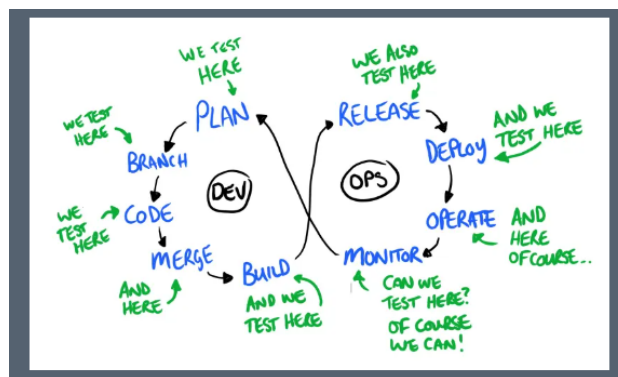


Figura 11: Ciclo Devops e interacción con Testing Ágil. Fuente: <https://danashby.co.uk/2016/10/19/continuous-testing-in-devops/>

¿Qué está diciendo esta imagen? En el mundo de DevOps, es importante probar cada etapa del proceso de desarrollo de software. Podemos probar el plan o diseño del

software, lo que implica una exploración exhaustiva de todas las posibles interacciones, variables y riesgos que pueden surgir. También podemos probar ramas de desarrollo individuales y estrategias de ramificación, y revisar el código para investigar errores y verificar si cumple con las expectativas. Las fusiones también deben probarse y verificarse cuidadosamente para evitar conflictos de combinación y riesgos ocultos de diferentes integraciones.

Además, podemos probar el proceso de construcción y el producto final en diferentes entornos. Las pruebas exploratorias son particularmente efectivas para evaluar la calidad percibida del producto. Es importante probar el proceso de lanzamiento y la implementación en un entorno de prueba y en producción para asegurarse de que no haya problemas de usuario. Y una vez que el software está en vivo, debemos estar atentos al monitoreo y la información que nos proporciona. Podemos explorar diferentes ideas y estar atentos a cualquier información que nos brinde para asegurarnos de que estamos buscando las cosas correctas. En resumen, en el mundo de DevOps, cada etapa del proceso de desarrollo de software debe ser probada cuidadosamente para garantizar la calidad y la satisfacción del cliente.

Los 10 principios del Agile Testing

El Agile Testing es un enfoque de pruebas de software dentro del marco ágil, que se enfoca en la colaboración, la retroalimentación constante y la entrega continua de valor al cliente.

Los 10 principios para los Agile Testers son una serie de directrices que ayudan a los miembros del equipo, no solo a los testers, a trabajar de manera más efectiva en un entorno ágil. Estos principios son los siguientes:

1. Proporciona un feedback continuo: los testers deben proporcionar retroalimentación frecuente a los miembros del equipo sobre el progreso del proyecto, los riesgos y cualquier problema que puedan surgir.
2. Entrega valor al cliente: el equipo debe centrarse en ofrecer valor al cliente en todo momento y asegurarse de que las pruebas se realicen con ese objetivo en mente.
3. Permite la comunicación cara a cara: la comunicación directa y efectiva es esencial para el éxito en un equipo ágil, y los testers deben estar dispuestos a comunicarse con los miembros del equipo.
4. Ten coraje: los testers deben estar dispuestos a tomar decisiones difíciles y defender lo que creen que es correcto, incluso si eso significa plantear diferentes puntos de vista a otros miembros del equipo.
5. Sim: mantener la simplicidad en todo momento y evitar la complejidad innecesaria.

6. Práctica la mejora continua: siempre hay margen de mejora, y los testers deben estar dispuestos a aprender de sus errores y trabajar en su crecimiento profesional.
7. Responde al cambio: en un entorno ágil, los cambios son inevitables, y los testers deben estar preparados para adaptarse rápidamente a los cambios en los requisitos y objetivos del proyecto.
8. Auto-organízate: los testers deben ser capaces de gestionar su propio trabajo de manera efectiva, sin necesidad de supervisión constante. También está relacionado con la proactividad que requiere la metodología Ágil.
9. Céntrate en la gente: el éxito en un equipo ágil depende en gran medida de las relaciones interpersonales y los testers deben enfocarse en construir relaciones positivas con los demás miembros del equipo.
10. ¡Disfruta! Por último, los testers deben recordar que trabajar en un entorno ágil puede ser divertido y gratificante.

Testing Manifiesto

Creado por Karen Greaves y Samantha Laing, el Testing Manifiesto refleja el cambio de mentalidad necesario para un enfoque de Agile Testing exitoso.

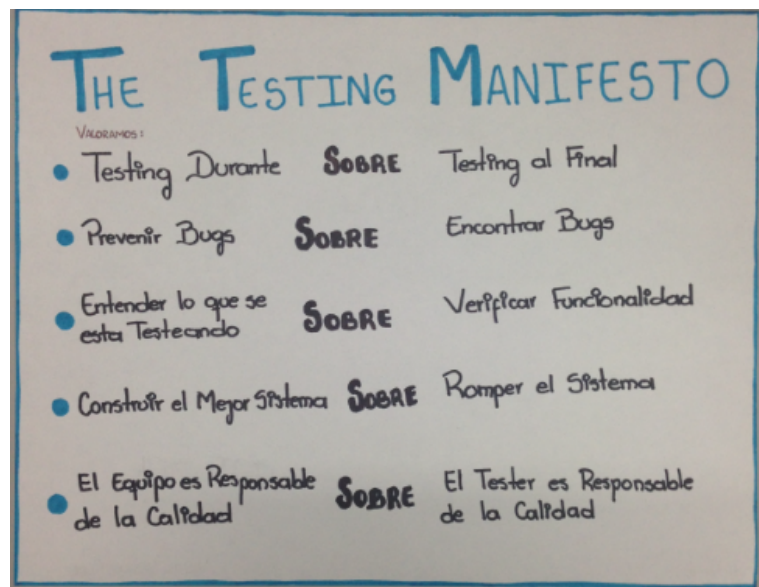


Figura 12: Qué características prevalecen en el Testing Manifiesto.

Fuente: <https://valexamp.wordpress.com/2016/10/31/agile-testing-manifesto/>

Los valores que se pregonan son:

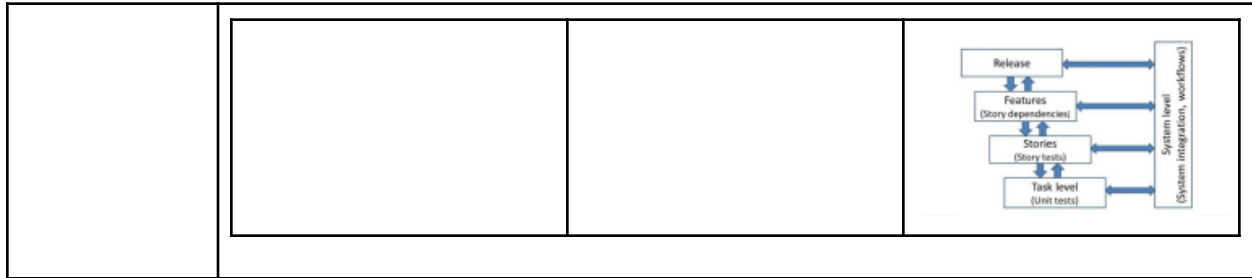
- **Testing durante SOBRE testing al final:** Priorizar la realización de tareas de testing desde el inicio hasta que se da por hecha por sobre realizar testing como una etapa al finalizar el sprint.
Tradicionalmente el testing ha sido una fase que se ha producido al final del desarrollo. Sin embargo, en la agilidad el testing es una actividad que necesita darse junto con la codificación, la documentación y el resto de operaciones.
- **Prevenir bugs SOBRE encontrar bugs:** Incentivar la discusión y revisión grupal de las tareas, tener en claro que hay que hacer, como y porque, evitar todo tipo de suposiciones y trabajar realmente en equipo con el desarrollador para que los bugs se puedan evitar de antemano por sobre esperar a que el desarrollo esté completo para revisar y estudiar el mismo, así como evitar discutir con el desarrollador sobre que se va a testear.
- **Entender lo que se está testeando SOBRE verificar funcionalidad:** La necesidad de entender qué es lo que quiere el usuario, para que lo quiere y como él lo va a usar, para realizar pruebas que den valor agregado por sobre solo ver que las especificaciones se cumplen (ver que $1+1$ sea igual a 2) en una tarea más de checklist que de testing.
Los testers deben ser los representantes del cliente en cada decisión del diseño, asegurándose que la feature satisfaga las verdaderas necesidades de estos y no solo las especificaciones
- **Construir el mejor sistema SOBRE romper el sistema:** Poner el trabajo de equipo en generar el ambiente necesario para poder mejorar lo que se está haciendo por sobre solo intentar romper el sistema.
Tradicionalmente los Testers se han centrado en buscar la forma de romper el sistema, enfocándose en el Testing Negativo. Sin embargo, un tester se debe enfocar en el objetivo principal: crear un producto que aporte valor y cumpla su función como se espera.
- **El equipo es responsable de la calidad SOBRE el tester es responsable de la calidad:** En agile la clave de entregar un producto con calidad es que el equipo se comprometa con todas las actividades y trabajen conjuntamente para generar valor, por eso es importante que la responsabilidad sobre la calidad sea del equipo por sobre que la calidad recaiga sobre un solo rol como es el del tester.
Los testers por sí solos no pueden mejorar la calidad. Su función es la de determinar el nivel de calidad e informar de ella a los interesados. Sólo puedes mejorar la calidad uniendo tus esfuerzos a los del equipo.

Es importante aclarar, que al igual que el manifiesto agile la idea es ponderar una postura sobre la otra, en lugar de eliminar una postura para ser reemplazada por otra.

Planificación de pruebas en contextos ágiles:

Para planificar las pruebas de forma efectiva, un equipo necesita considerar su contexto. Para entender tu contexto piensa en tres aspectos

EQUIPO	<p>No todos los equipos que se crean son iguales. Se debe tener en cuenta las habilidades y capacidades del equipo de pruebas. Es importante tener en cuenta la experiencia en pruebas de los miembros del equipo, su conocimiento técnico y su capacidad para colaborar y trabajar en equipo. Con base en esto, se pueden definir roles y responsabilidades claras para los miembros del equipo de pruebas. Además, se pueden dar 2 casos:</p>					
	<table><tr><td><p>1. Equipo pequeños.</p><p>a. Ubicados en la misma sala/edificio</p><p>b. Buenas oportunidad para aprender los valores del resto y compartirlos</p><p>c. La planificación es mucho más fácil</p></td><td colspan="2"><p>2. Equipos grandes y distribuidos.</p><p>a. Trabajan en múltiples proyectos y con múltiples equipos</p><p>b. Surge la necesidad de tener especialistas en el equipo</p><p>c. Se incrementan las dependencias</p><p>d. La planificación se vuelve más compleja</p></td></tr></table>			<p>1. Equipo pequeños.</p> <p>a. Ubicados en la misma sala/edificio</p> <p>b. Buenas oportunidad para aprender los valores del resto y compartirlos</p> <p>c. La planificación es mucho más fácil</p>	<p>2. Equipos grandes y distribuidos.</p> <p>a. Trabajan en múltiples proyectos y con múltiples equipos</p> <p>b. Surge la necesidad de tener especialistas en el equipo</p> <p>c. Se incrementan las dependencias</p> <p>d. La planificación se vuelve más compleja</p>	
<p>1. Equipo pequeños.</p> <p>a. Ubicados en la misma sala/edificio</p> <p>b. Buenas oportunidad para aprender los valores del resto y compartirlos</p> <p>c. La planificación es mucho más fácil</p>	<p>2. Equipos grandes y distribuidos.</p> <p>a. Trabajan en múltiples proyectos y con múltiples equipos</p> <p>b. Surge la necesidad de tener especialistas en el equipo</p> <p>c. Se incrementan las dependencias</p> <p>d. La planificación se vuelve más compleja</p>					
PRODUCTO	<p>Se refiere al producto o sistema que se está desarrollando y a las necesidades y expectativas del cliente. Es importante entender el propósito y los objetivos del producto, así como sus funcionalidades clave y las expectativas del cliente en cuanto a calidad y rendimiento. Con base en esto, se pueden definir los casos de prueba necesarios para garantizar la calidad del producto y su capacidad para satisfacer las necesidades del cliente. Consideraciones del producto:</p>					
	<table><tr><td><p>Tamaño:</p><p>¿Cuánta gente lo va a usar?,</p><p>¿Está integrado con Apps Externas?</p></td><td colspan="2"><p>Riesgos: Para reducir riesgos los equipos necesitan conocer el dominio del negocio.</p><ul style="list-style-type: none">• Los expertos de dominio deben ayudar en la planificación de las pruebas.• ¿Tiene idea el equipo de como se usa el producto?• ¿Tienen todos los miembros del equipo conocimiento del dominio?</td></tr></table>			<p>Tamaño:</p> <p>¿Cuánta gente lo va a usar?,</p> <p>¿Está integrado con Apps Externas?</p>	<p>Riesgos: Para reducir riesgos los equipos necesitan conocer el dominio del negocio.</p> <ul style="list-style-type: none">• Los expertos de dominio deben ayudar en la planificación de las pruebas.• ¿Tiene idea el equipo de como se usa el producto?• ¿Tienen todos los miembros del equipo conocimiento del dominio?	
<p>Tamaño:</p> <p>¿Cuánta gente lo va a usar?,</p> <p>¿Está integrado con Apps Externas?</p>	<p>Riesgos: Para reducir riesgos los equipos necesitan conocer el dominio del negocio.</p> <ul style="list-style-type: none">• Los expertos de dominio deben ayudar en la planificación de las pruebas.• ¿Tiene idea el equipo de como se usa el producto?• ¿Tienen todos los miembros del equipo conocimiento del dominio?					
NIVEL DE DETALLE	<p>Los niveles identificados son:</p>					
	<p>Nivel de Release/Feature: las actividades deben ser coordinadas entre equipos</p>	<p>Nivel de Historias: emplear la Specification By Example para incrementar el entendimiento de la historia y convertirla en pruebas</p>	<p>Nivel de Tareas: Usar TDD para escribir pruebas unitarias de bajo nivel. También se puede emplear DDD como alternativa.</p>			



Unidad 2

Guia de desarrollo con ejemplos:

BDD

BDD (Behavior Driven Development o Desarrollo Guiado por el Comportamiento) es una metodología de desarrollo de software que se enfoca en la colaboración entre los diferentes roles involucrados en el proceso de desarrollo y en la creación de una especificación ejecutable a partir del comportamiento esperado de la aplicación.

Se utiliza un lenguaje de dominio específico (DSL, por sus siglas en inglés) para describir el comportamiento esperado de la aplicación en términos de escenarios y ejemplos concretos.

Su sintaxis es:

GIVEN: Precondiciones -> "Dado que"

WHEN: Disparadores -> "Cuando"

THEN: Post Condiciones -> "Entonces"

y se organizan en historias de usuario.

Por ejemplo, supongamos que estamos desarrollando una aplicación de compras en línea. Una historia de usuario podría ser:

Como cliente, quiero poder agregar productos a mi carrito de compras para poder comprarlos más tarde.

Un escenario concreto de esta historia de usuario podría ser:

Dado que soy un cliente en la página de un producto,
 Cuando hago clic en el botón "Agregar al carrito",
 Entonces debería ver que el producto se agrega a mi carrito.

Este escenario se puede escribir en un formato entendible tanto para desarrolladores como para usuarios y puede ser utilizado para guiar la implementación y las pruebas de la funcionalidad descrita.

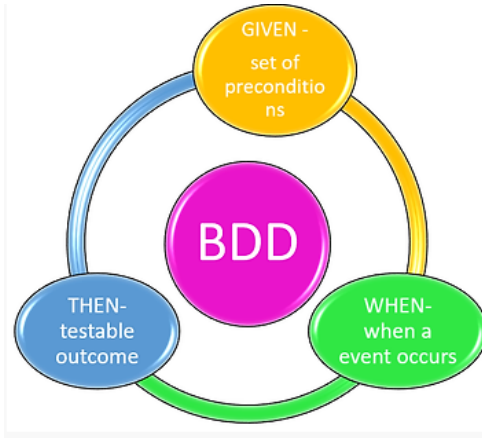


Figura 13: Imagen BDD. Fuente: <https://www.numpyninja.com/post/insight-on-bdd>

ATDD

ATDD (Acceptance Test-Driven Development) es una forma más genérica de guiar el desarrollo con ejemplos sin un lenguaje o reglas estrictas.

Es una práctica de desarrollo ágil que se enfoca en definir las pruebas de aceptación antes de implementar la funcionalidad. La idea es trabajar en colaboración con el equipo de negocio y el equipo técnico para definir las expectativas de la funcionalidad y asegurarse de que la solución cumple con los requerimientos y expectativas del negocio.

Se utilizan pruebas de aceptación para definir las funcionalidades que se van a implementar y para validar que el software cumple con los requerimientos del negocio. Estas pruebas se realizan utilizando un lenguaje de negocio entendible por todos los involucrados en el proyecto, incluyendo a los desarrolladores, analistas y expertos del negocio.

Se lleva a cabo:

ANTES de Implementar una necesidad todos los miembros del equipo contribuyen para generar escenarios de cómo se comportará dicha necesidad.

DESPUÉS el equipo convierte esos escenarios en pruebas de aceptación.

Un ejemplo de este caso, sería:

El equipo de negocio puede definir una prueba de aceptación como: "Dado que soy un cliente de la tienda en línea, cuando busco un producto, quiero que la búsqueda me muestre todos los productos disponibles que coincidan con mi búsqueda".

El equipo técnico utiliza esta prueba para definir los casos de prueba y validar que la búsqueda funciona como se espera. En este ejemplo, los casos de prueba podrían

incluir la verificación de que la búsqueda devuelve los resultados correctos y en el orden correcto, que los resultados son relevantes para la búsqueda realizada y que la búsqueda funciona correctamente en diferentes navegadores y dispositivos.

SBE

SBE se refiere a "Specification By Example" en inglés, lo que significa "Especificación por Ejemplo" en español. Es una técnica utilizada en la ingeniería de software para definir y comunicar los requisitos de un software a través de ejemplos concretos de cómo debe funcionar el software en diferentes situaciones.

En lugar de simplemente escribir los requisitos en un documento, la técnica de SBE se centra en crear ejemplos concretos de cómo se espera que el software se comporte en diferentes situaciones y cómo los usuarios interactúan con él. Esto ayuda a asegurar que los requisitos sean claros y comprensibles para todos los miembros del equipo de desarrollo de software, incluidos los desarrolladores, los testers y los responsables de la calidad del software.

A medida que avanza el desarrollo, los ejemplos se perfeccionan y se convierten en especificaciones ejecutables que permiten validar el producto con frecuencia.

Estos ejemplos ejecutables junto con BDD y ATDD, se convierten en documentación viva de la aplicación.

Un ejemplo de SBE podría ser el de un equipo de desarrollo de software que está trabajando en una aplicación móvil de compras en línea. En lugar de simplemente escribir los requisitos en un documento, el equipo utiliza SBE para definir cómo debe funcionar la aplicación en diferentes situaciones. Por ejemplo, podrían crear ejemplos de cómo se espera que el proceso de pago funcione, incluyendo cómo se deben manejar diferentes tipos de errores y excepciones. Esto ayuda a asegurar que todos los miembros del equipo tengan una comprensión clara de cómo debe funcionar la aplicación en diferentes situaciones, lo que a su vez ayuda a garantizar que el software entregado cumpla con los requisitos del cliente.

Habilitando la Colaboración

La colaboración dentro del equipo y entre los equipos es una de las piedras angulares para hacer que los equipos ágiles tengan éxito. Sin embargo, nos encontramos con que muchos equipos no saben cómo empezar a construir estas relaciones. Te presentamos algunas prácticas que te pueden ayudar en ello.

Dinámica de los 3 amigos

La dinámica de los "tres amigos" es una práctica en el desarrollo ágil de software que se centra en la colaboración entre los miembros del equipo de desarrollo de software para

garantizar la calidad del software. Los "tres amigos" son los roles que colaboran en esta práctica: el desarrollador, el tester y el analista de negocio.

La idea es que los tres amigos trabajen juntos desde el principio del ciclo de vida del software para discutir y definir los requisitos, identificar y abordar posibles problemas de calidad y asegurarse de que el software entregado cumpla con las expectativas del cliente. Deben examinar el incremento del producto antes, durante y después del desarrollo.

El desarrollador aporta su experiencia en el diseño y la construcción del software, el tester se encarga de la calidad del software y de las pruebas necesarias para garantizar su correcto funcionamiento, y el analista de negocio proporciona una comprensión clara de los requisitos y expectativas del cliente.

La dinámica de los tres amigos fomenta la colaboración temprana y frecuente entre los diferentes miembros del equipo, lo que ayuda a identificar y solucionar problemas en una etapa temprana del proceso de desarrollo de software, lo que a su vez ayuda a reducir el tiempo y los costos asociados con la corrección de errores en etapas posteriores del proceso de desarrollo. Además, esta práctica también ayuda a mejorar la calidad y la entrega de software que cumpla con las necesidades del cliente.

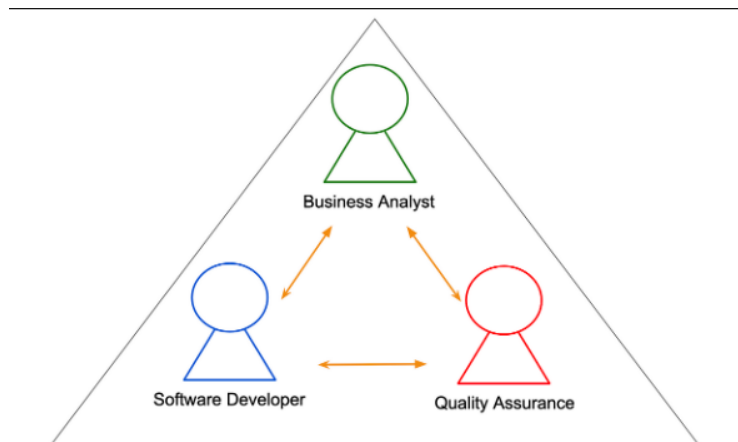


Figura 14 : Dinámica los 3 amigos. Fuente:
<https://colin-but.medium.com/the-3-amigos-in-software-development-fd33ab7bf28c>

Dinámica del Impact Mapping

El Impact Mapping es una técnica de planificación estratégica que se utiliza para definir los objetivos de un proyecto y las acciones necesarias para alcanzarlos. Ayuda a decidir qué features debemos construir y quizás incluso determinar qué prioridad deben tener. La dinámica del Impact Mapping se puede resumir en los siguientes pasos:

1. Identificar el objetivo principal: En primer lugar, se debe identificar el objetivo principal del proyecto. Este objetivo debe ser claro, específico y medible. Es el **WHY**
2. Identificar los actores clave: Se deben identificar los actores clave que pueden afectar o ser afectados por el proyecto. Estos actores pueden ser internos o externos a la organización y pueden incluir clientes, proveedores, reguladores, etc. **WHO**
3. Identificar los impactos: A continuación, se deben identificar los impactos que se esperan alcanzar al lograr el objetivo principal. Estos impactos son los cambios deseados en el comportamiento de los usuarios o las mejoras en la eficiencia del proceso. Por cada “who” preguntamos cómo podrían ayudarnos en alcanzar este objetivo. **HOW**
4. Definir los entregables: Una vez que se han identificado los impactos y los actores clave, se pueden definir las acciones necesarias para alcanzar los impactos. **WHAT**.

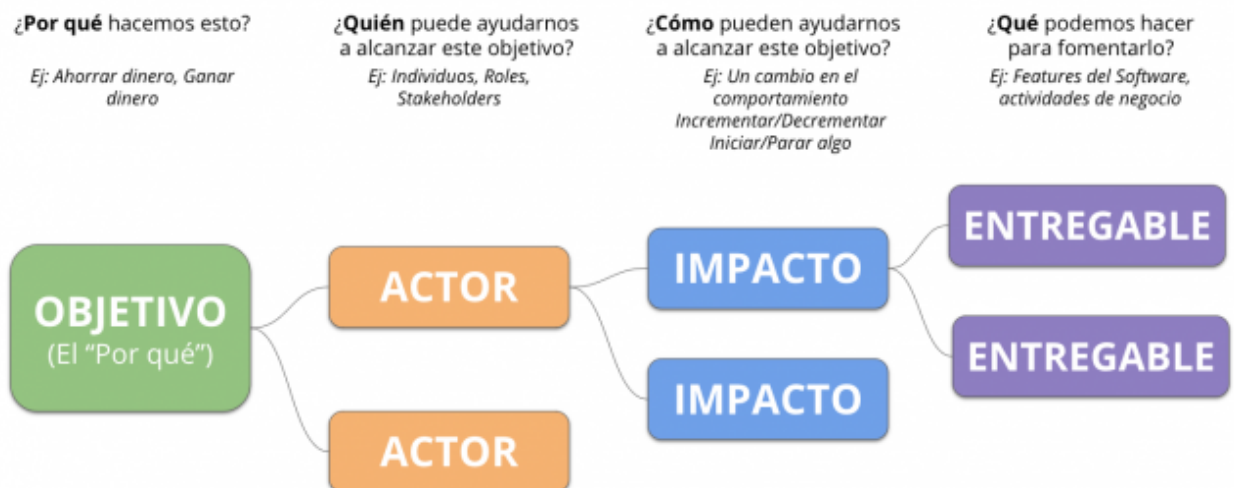


Figura 15: Impact Mapping. Fuente; <https://www.adictosaltrabajo.com/2021/03/02/impact-mapping-creando-productos-y-proyectos-de-gran-impacto/>

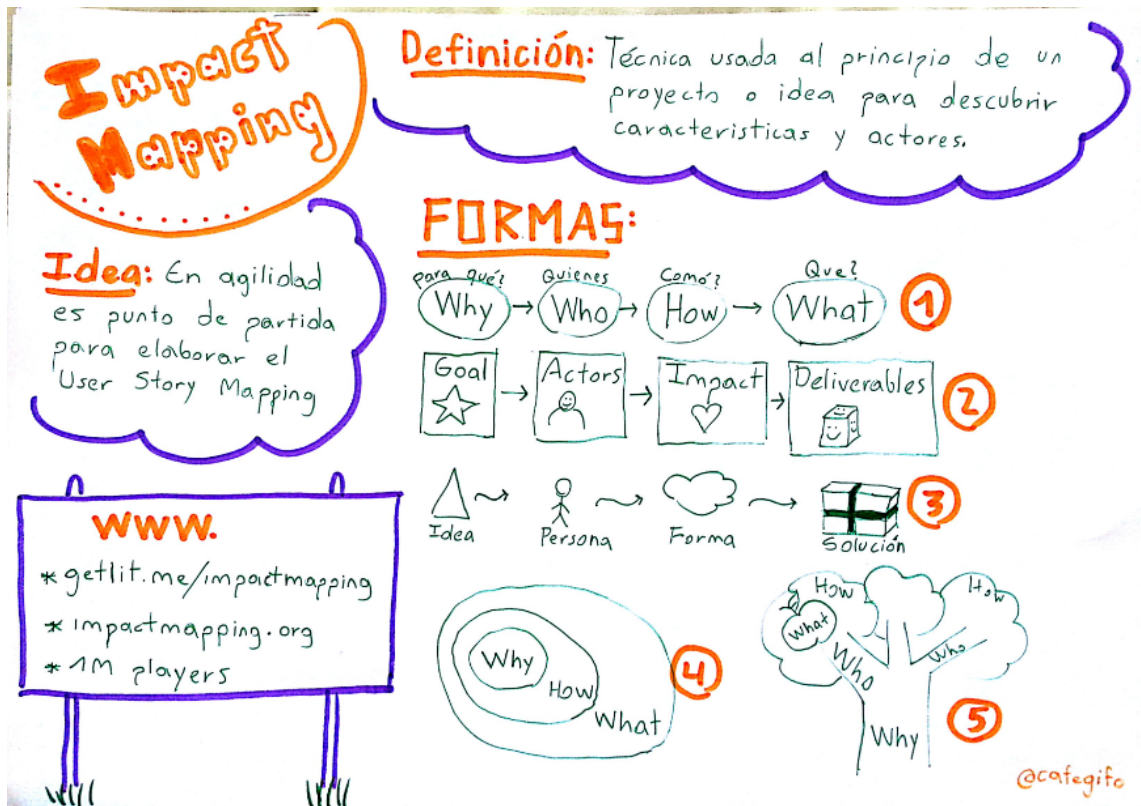


Figura 16: Definición de Impact Mapping. Fuente: https://cafegifo.files.wordpress.com/2015/12/nuevodocumento-7_1.jpg

Un ejemplo es:

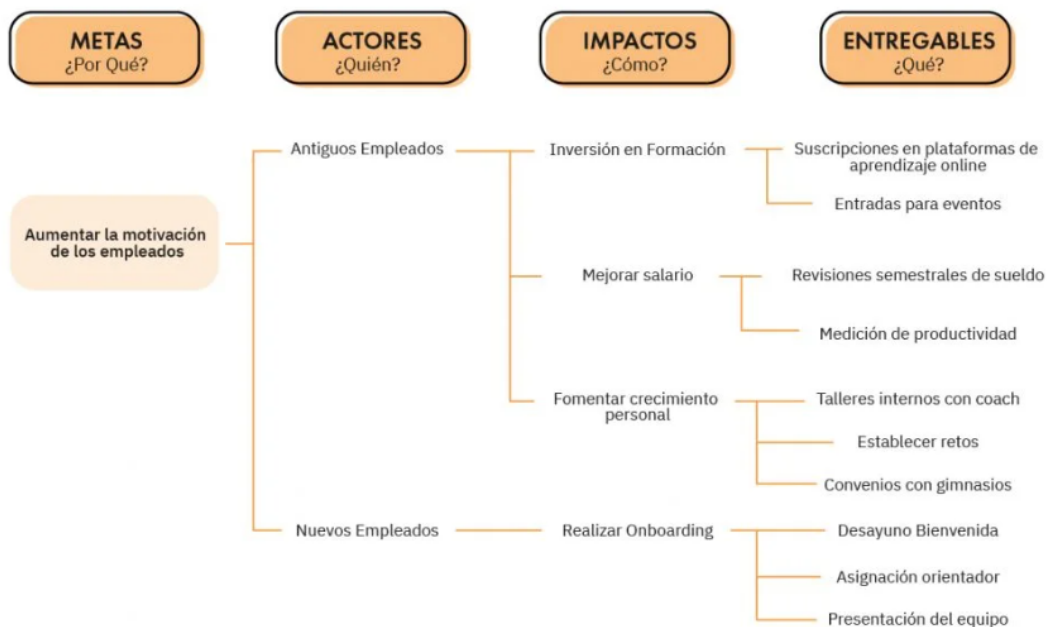


Figura 17: Ejemplo de Impact Mapping. Fuente: <https://formiux.com/mapas-de-impacto/>

Dinámica del Example Mapping

Es una técnica de definición de requerimientos que nos permite explorar soluciones a un problema utilizando ejemplos de reglas y, haciendo las preguntas que nos permitirán cubrir la mayor cantidad de reglas posibles con el fin de diseñar una solución.

Cada regla puede tener más de un ejemplo.

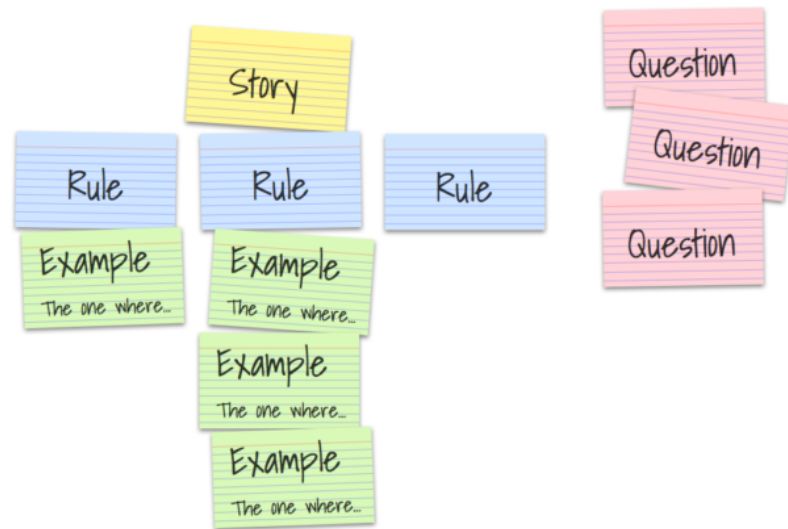


Figura 18. Example Mapping. Fuente: <https://iterative.me/example-mapping-en-espanol/>

- **Story:** Escribimos la historia (o lo que inicialmente pensamos que es la historia, luego lo vamos refinando) en la tarjeta amarilla.
- **Rule:** Son las reglas de negocio. Escribimos los criterios de aceptación del usuario (o los que inicialmente pensamos que son, basado en la conversación con los usuarios, esto también se va refinando). Están representadas en las tarjetas azules.
- **Example:** Escribimos todos los ejemplos posibles que cumplan los criterios de aceptación del usuario. Son las tarjetas verdes y se colocan debajo de cada regla (la que consideremos más acertada según el ejemplo).
- **Questions:** Aquí registramos todas aquellas preguntas que nadie en la sesión puede responder. Requieren mayor investigación o la presencia de otros usuarios o interesados en el proyecto o quizás, de algún experto en la solución que queremos implementar.

Un ejemplo sería:

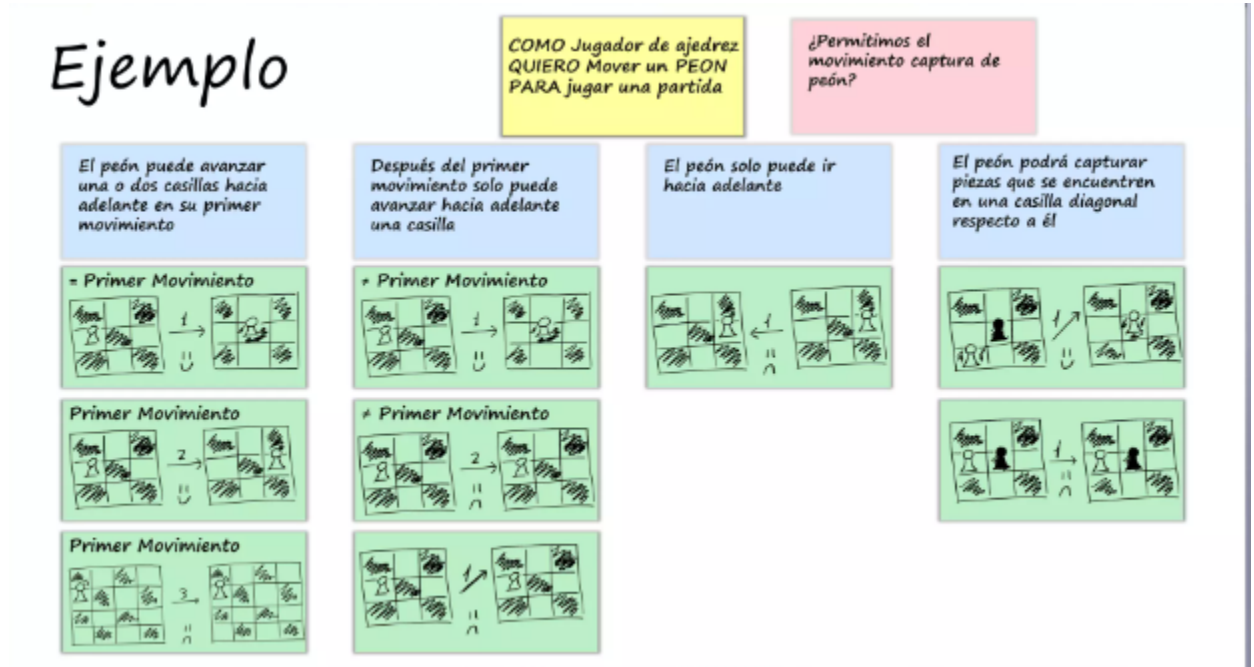


Figura 19: Ejemplo de Example Mapping. Fuente:
<https://www.slideshare.net/JoseNieto1/example-mapping-presentation-125817269>

Pruebas Exploratorias

Las pruebas exploratorias son un enfoque de prueba de software que se basa en la experiencia y habilidad del probador para descubrir y evaluar el software en función de su comportamiento, usabilidad, calidad y rendimiento. Este enfoque se centra en la exploración del software de forma creativa y no estructurada, sin seguir un plan preestablecido, lo que permite descubrir defectos y problemas que podrían no haber sido detectados con otros métodos de prueba.

Existen varias técnicas de pruebas exploratorias que se pueden utilizar para descubrir problemas y defectos en el software. Algunas de estas técnicas son:

- Prueba basada en errores: Esta técnica se enfoca en encontrar defectos específicos en el software. El probador identifica un posible error y luego lo explora para ver si es posible reproducirlo. Esta técnica es particularmente útil en la identificación de problemas complejos y errores no obvios.
- Prueba aleatoria: Esta técnica consiste en realizar pruebas aleatorias del software sin un plan estructurado previo. El probador explora el software de forma espontánea y sin un enfoque específico para descubrir problemas que podrían haber sido pasados por alto por otras técnicas.
- Prueba basada en la experiencia: Esta técnica utiliza el conocimiento y la experiencia del probador para explorar el software en busca de problemas. El

probador utiliza su experiencia previa para identificar áreas del software que podrían ser problemáticas y se enfoca en ellas.

- Prueba exploratoria de flujo de trabajo: Esta técnica se enfoca en probar los flujos de trabajo del software en lugar de sus funciones individuales. El probador sigue los flujos de trabajo del software para identificar problemas en su integridad.
- Prueba de uso del usuario: Esta técnica se enfoca en la experiencia del usuario al utilizar el software. El probador simula el uso del software como si fuera un usuario final y evalúa la facilidad de uso, la accesibilidad y la satisfacción del usuario.
- Prueba de configuración y compatibilidad: Esta técnica se enfoca en la configuración y compatibilidad del software con otros sistemas y entornos. El probador prueba diferentes configuraciones y entornos para identificar problemas de compatibilidad.
- Prueba de rendimiento y carga: Esta técnica se enfoca en la capacidad del software para manejar grandes cantidades de datos y usuarios simultáneamente. El probador simula diferentes cargas y situaciones de rendimiento para evaluar la estabilidad y la capacidad del software.

Atributos de la Calidad del Testing

Un atributo de calidad, también conocido como requisito no funcional, describe las propiedades bajo las cuales una característica debe operar.

Es una especificación que describe las propiedades o características que un software debe tener para satisfacer las necesidades y expectativas del usuario. Estos requisitos no están directamente relacionados con las funciones del software, sino con las cualidades que debe tener para operar correctamente.

Podemos pensar en él como una restricción que el equipo debe considerar con cada Feature o Historia.

Es importante tener en cuenta que estos atributos no son opcionales y deben ser considerados en todo momento durante el ciclo de vida del software.

Existen 2 tipos:

Desarrollo

Son los atributos que representan el CÓMO desarrollamos el código e incluyen:

- Mantenibilidad del Código
- Reusabilidad del Código
- Testabilidad del Código

Operacionales

Disponibilidad	Interoperabilidad	Escalabilidad	Rendimiento
Seguridad	Confiabilidad	Robustez	Recuperabilidad
Instalabilidad	Usabilidad	Protectibilidad	

Figura 20: Atributos de calidad de testing operacionales. Fuente: Autentia-MazosAgile-v2.0

REFERENCIAS

- Mathur, A. P. (2008). Foundations of software testing. Pearson Education India.
- Desikan, S., & Ramesh, G. (2014). Software testing: Principles and practices. Pearson.
- Kaner, C., Falk, J., & Nguyen, H. Q. (1999). Testing computer software. Wiley.
- Dustin, E., Garrett, T., & Gauf, B. (2003). Effective software testing: 50 specific ways to improve your testing. Addison-Wesley Professional.
- Copeland, L. R. (2004). A practitioner's guide to software test design. Artech House.
- International Software Testing Qualifications Board. (2018). Certified Tester Foundation Level (CTFL) Syllabus, Version 2018 v3.1.1. Recuperado de: <https://astqb.org/assets/documents/CTFL-2018-Syllabus.pdf>
- Diario de QA. (s.f.). Blog. Recuperado de <https://www.diariodeqa.com/blog>
- Cypress. (s.f.). Home. Recuperado de <https://www.cypress.io/>
- myservername.com. (s.f.). Home. Recuperado de <https://spa.myservername.com/>
- Tutorialspoint. (s.f.). Recuperado de <https://www.tutorialspoint.com/>
- Izaguirre, A. (2019). Pruebas de software: tipos de pruebas y sus características.
- Beck, K. (2003). Test-driven development: By example. Addison-Wesley Professional.
- Pressman, R. S., & Maxim, B. R. (2015). Ingeniería del software: un enfoque práctico. McGraw-Hill Education.
- Guru99. (s.f.). Guru99. Recuperado de <https://www.guru99.com/>
- Danashby (s.f.). Recuperado de <https://danashby.co.uk/>
- Jose Pablo Sarco (s.f.). Recuperado de <https://josepablosarco.wordpress.com/>

- Iterative. Me (s.f). Recuperado de <https://iterative.me/example-mapping-en-espanol/>