**Pruebas de software**

**¿Qué hace un tester?**

Un tester, también conocido como ingeniero de pruebas o QA (Quality Assurance), es un profesional responsable de **asegurar la calidad del software**. Su trabajo consiste en **encontrar y reportar errores, defectos y anomalías** en el software para que sean corregidos por los desarrolladores.

**Las principales tareas de un tester incluyen:**

* **Planificación y ejecución de pruebas:** Diseñar y ejecutar diferentes tipos de pruebas para verificar que el software cumple con los requisitos y funciona correctamente.
* **Análisis de resultados:** Revisar los resultados de las pruebas e identificar errores, defectos y anomalías.
* **Reportaje de errores:** Documentar los errores de forma clara y concisa para que puedan ser corregidos por los desarrolladores.
* **Comunicación:** Comunicar los resultados de las pruebas a los diferentes desarroladores del proyecto.

**Un tester debe tener las siguientes habilidades:**

* **Conocimiento de software:** Tener un buen conocimiento de los diferentes tipos de software y de las metodologías de desarrollo.
* **Habilidades analíticas:** Ser capaz de analizar los resultados de las pruebas e identificar la causa de los errores.
* **Habilidades de comunicación:** Ser capaz de comunicar los resultados de las pruebas de forma clara y concisa a los diferentes stakeholders del proyecto.

**Anomalías, fallo, defecto y error:**

* **Anomalía:** Cualquier desviación del comportamiento esperado del software.
* **Fallo:** Una anomalía que causa que el software no funcione según lo previsto.
* **Defecto:** Una imperfección en el software que puede causar un fallo.
* **Error:** Un error humano que puede causar un defecto en el software.

**Calidad del producto y del proceso:**

**Calidad del producto:**

La calidad del producto se refiere al grado en que un producto cumple con los requisitos y expectativas de los clientes. Se trata de una medida integral de la excelencia del producto que tiene en cuenta aspectos como:

* **Funcionalidad:** El producto debe cumplir con las funciones para las que fue diseñado.
* **Rendimiento:** El producto debe funcionar de manera eficiente y sin errores.
* **Fiabilidad:** El producto debe ser confiable y funcionar sin interrupciones.
* **Usabilidad:** El producto debe ser fácil de usar y comprender.
* **Seguridad:** El producto debe ser seguro para los usuarios y para el medio ambiente.
* **Estética:** El producto debe ser atractivo y tener una buena presentación.

**Calidad del proceso:**

La calidad del proceso se refiere al grado en que un proceso es eficiente y efectivo para producir un producto de alta calidad. Se trata de una medida de la eficiencia y la eficacia del proceso que tiene en cuenta aspectos como:

* **Planificación:** El proceso debe estar bien planificado y documentado.
* **Ejecución:** El proceso debe ejecutarse de manera eficiente y sin errores.
* **Control:** El proceso debe ser monitoreado y controlado para asegurar que se cumplan los requisitos de calidad.
* **Mejora continua:** El proceso debe ser mejorado continuamente para aumentar la calidad del producto.

**Clasificación de las pruebas:**

**1. Según el nivel de prueba:**

* **Pruebas unitarias:** Prueban unidades individuales de código.
* **Pruebas de integración:** Prueban cómo funcionan juntas las diferentes unidades de código.
* **Pruebas de sistema:** Prueban el sistema en su conjunto.
* **Pruebas de aceptación:** Prueban si el software cumple con los requisitos del cliente.

**2. Según el tipo de prueba:**

* **Pruebas funcionales:** Prueban si el software funciona como se espera.
* **Pruebas no funcionales:** Prueban las características no funcionales del software, como el rendimiento, la seguridad y la usabilidad.

**3. Según la técnica de prueba:**

* **Pruebas de caja blanca:** Prueban el software con conocimiento de su código interno.
* **Pruebas de caja negra:** Prueban el software sin conocer su código interno.

**4. Según el objetivo de la prueba:**

* **Pruebas de humo:** Prueban las funciones básicas del software para asegurar que funciona correctamente.
* **Pruebas de regresión:** Aseguran que los cambios realizados en el software no introducen nuevos errores.
* **Pruebas de aceptación:** Prueban si el software cumple con los requisitos del cliente.

**Nota:** La clasificación de las pruebas no es excluyente, y un mismo caso de prueba puede pertenecer a varias categorías.

**Tipos de pruebas de software:**

**1. Pruebas de rendimiento:**

* **Objetivo:** Medir la velocidad, la capacidad de respuesta y la escalabilidad del software bajo diferentes cargas.
* **Ejemplos:**
  + Probar el tiempo de respuesta de una aplicación web bajo una carga de usuarios.
  + Medir el rendimiento de un servidor bajo una carga de transacciones.

**2. Pruebas de carga:**

* **Objetivo:** Determinar cómo se comporta el software bajo una carga de trabajo extrema.
* **Ejemplos:**
  + Simular un pico de tráfico en un sitio web para ver cómo se comporta.
  + Probar la capacidad de un servidor para manejar una gran cantidad de conexiones simultáneas.

**3. Pruebas de estrés:**

* **Objetivo:** Evaluar la resistencia del software a errores y fallos.
* **Ejemplos:**
  + Inyectar errores en el software para ver cómo se comporta.
  + Probar el software con datos corruptos.

**4. Pruebas de regresión:**

* **Objetivo:** Asegurarse de que los cambios realizados en el software no introducen nuevos errores.
* **Ejemplos:**
  + Volver a ejecutar pruebas de casos de prueba existentes después de realizar un cambio en el software.
  + Probar el software con diferentes versiones de hardware y software.

**5. Pruebas de seguridad:**

* **Objetivo:** Identificar y corregir vulnerabilidades de seguridad en el software.
* **Ejemplos:**
  + Realizar pruebas de penetración para identificar vulnerabilidades.
  + Probar el software contra ataques comunes.

**6. Pruebas de usabilidad:**

* **Objetivo:** Evaluar la facilidad de uso del software para los usuarios.
* **Ejemplos:**
  + Observar a los usuarios mientras usan el software.
  + Realizar pruebas de usuario con diferentes tipos de usuarios.

**7. Pruebas de exploración:**

* **Objetivo:** Encontrar errores y problemas en el software que no se han detectado con otros tipos de pruebas.
* **Ejemplos:**
  + Usar herramientas de prueba de exploración para explorar el software de forma automática.
  + Realizar pruebas de caja negra para probar el software sin conocer su código interno.

**8. Pruebas de humo:**

**Objetivo:** Verificar que las funcionalidades básicas del software funcionan correctamente.

**Ejemplos:**

* **Prueba de humo para una aplicación web de comercio electrónico:**
  + Intentar iniciar sesión con un usuario y contraseña válidos.
  + Navegar por las diferentes categorías de productos.
  + Añadir un producto al carrito.
  + Completar el proceso de pago con una tarjeta de crédito válida.
  + Cerrar la sesión de la aplicación.

**Tester Manual vs. Tester de Automatización**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Casos de prueba que suele ejecutar un tester manual:**

* **Pruebas exploratorias:** Se realizan sin un plan predefinido, explorando la interfaz del software y buscando errores o comportamientos inesperados.
* **Pruebas de usabilidad:** Se enfocan en la facilidad de uso del software para usuarios finales, evaluando la intuición, la navegabilidad y la eficiencia de la interfaz.
* **Pruebas de compatibilidad:** Se realizan para verificar que el software funciona correctamente en diferentes navegadores, sistemas operativos, dispositivos y configuraciones.
* **Pruebas de seguridad:** Se enfocan en identificar vulnerabilidades de seguridad en el software, como ataques de inyección SQL o XSS.
* **Pruebas de regresión:** Se realizan para verificar que los cambios realizados en el software no introducen nuevos errores en funcionalidades ya existentes.

**Casos de prueba que suele ejecutar un tester de automatización:**

* **Pruebas unitarias:** Se ejecutan de forma automática para verificar el comportamiento individual de unidades de código (funciones, métodos).
* **Pruebas de integración:** Se ejecutan de forma automática para verificar cómo funcionan juntas las diferentes unidades de código.
* **Pruebas de API:** Se ejecutan de forma automática para verificar el funcionamiento de las APIs del software.
* **Pruebas de rendimiento:** Se ejecutan de forma automática para medir la velocidad, la capacidad de respuesta y la escalabilidad del software bajo diferentes cargas.
* **Pruebas de humo:** Se ejecutan de forma automática para verificar las funcionalidades básicas del software después de un cambio o actualización.

**Herramientas más comunes:**

**Pruebas exploratorias:**

1. **Zephyr:** Herramienta de gestión de pruebas que permite registrar y gestionar los errores encontrados durante la exploración.
2. **TestRail:** Herramienta de gestión de pruebas que permite organizar y ejecutar las pruebas de exploración.
3. **Xray:** Herramienta de gestión de pruebas que se integra con Jira y permite registrar y gestionar los errores encontrados durante la exploración.

**Pruebas de usabilidad:**

1. **Userlytics:** Plataforma que permite realizar pruebas de usuario en remoto.
2. **Morae:** Herramienta para grabar y analizar las sesiones de pruebas de usuario.
3. **UserZoom:** Plataforma que permite realizar pruebas de usuario en remoto y obtener feedback de los usuarios.

**Pruebas de compatibilidad:**

1. **BrowserStack:** Plataforma que permite probar el software en diferentes navegadores y dispositivos en la nube.
2. **Sauce Labs:** Plataforma que permite probar el software en diferentes navegadores y dispositivos en la nube.
3. **CrossBrowserTesting:** Plataforma que permite probar el software en diferentes navegadores y dispositivos en la nube.

**Pruebas de seguridad:**

1. **Nessus:** Herramienta de escaneo de vulnerabilidades que permite identificar vulnerabilidades de seguridad en el software.
2. **Nmap:** Herramienta de escaneo de puertos que permite identificar posibles vulnerabilidades de seguridad en el software.
3. **Metasploit:** Herramienta de pruebas de penetración que permite simular ataques reales al software para identificar vulnerabilidades.

**Pruebas de regresión:**

1. **PractiTest:** Herramienta de gestión de pruebas que permite organizar y ejecutar las pruebas de regresión.
2. **TestLink:** Herramienta de gestión de pruebas de código abierto que permite organizar y ejecutar las pruebas de regresión.
3. **Zephyr:** Herramienta de gestión de pruebas que se integra con Jira y permite organizar y ejecutar las pruebas de regresión.

**Pruebas unitarias:**

1. **JUnit:** Framework de pruebas unitarias para Java, uno de los más populares y utilizados en el mundo.
2. **NUnit:** Framework de pruebas unitarias para .NET, similar a JUnit y con una amplia comunidad de usuarios.
3. **xUnit:** Framework de pruebas unitarias multiplataforma, compatible con .NET, Java y JavaScript.

**Pruebas de integración:**

1. **Selenium:** Framework de código abierto para la automatización de pruebas web, utilizado por millones de testers en todo el mundo.
2. **Appium:** Framework de código abierto para la automatización de pruebas móviles, compatible con iOS y Android.
3. **Cucumber:** Herramienta de comportamiento para la automatización de pruebas que utiliza lenguaje natural para definir los casos de prueba.

**Pruebas de API:**

1. **Postman:** Herramienta de código abierto para el desarrollo y pruebas de API, ideal para enviar y recibir solicitudes HTTP, inspeccionar respuestas, crear y ejecutar colecciones de pruebas, y automatizar las pruebas de API.
2. **SoapUI:** Herramienta para el testing de APIs SOAP y REST, con una interfaz intuitiva y funcionalidades avanzadas.
3. **Katalon Studio:** Herramienta de automatización de pruebas multiplataforma que permite realizar pruebas de API, web y móviles.

**Pruebas de rendimiento:**

1. **LoadRunner:** Herramienta de pruebas de carga líder en el mercado, utilizada por grandes empresas para evaluar el rendimiento de sus aplicaciones.
2. **JMeter:** Herramienta de código abierto para pruebas de carga y rendimiento, popular entre equipos de todos los tamaños.
3. **Gatling:** Herramienta de código abierto para pruebas de carga y rendimiento, fácil de usar y con una interfaz intuitiva.

**Pruebas de humo:**

1. **Selenium:** Framework de código abierto para la automatización de pruebas web, utilizado por millones de testers en todo el mundo.
2. **Appium:** Framework de código abierto para la automatización de pruebas móviles, compatible con iOS y Android.
3. **Postman:** Herramienta de código abierto para el desarrollo y pruebas de API, ideal para realizar pruebas de humo básicas de API.

Diagrama

Descripción generada automáticamente con confianza baja

**Casos de prueba para cada etapa de desarrollo de software:**

**Definición de necesidades:**

* **Casos de uso:** Describen cómo los usuarios interactuarán con el software para lograr sus objetivos.
* **Historias de usuario:** Narrativas cortas que describen las funcionalidades del software desde la perspectiva del usuario.

**Análisis:**

* **Análisis de requisitos:** Se verifican los requisitos para asegurar que sean completos, consistentes y correctos.
* **Análisis de riesgos:** Se identifican los riesgos potenciales del proyecto y se definen planes de mitigación.

**Diseño:**

* **Diseño de casos de prueba:** Se diseñan casos de prueba para verificar que el diseño del software cumple con los requisitos.
* **Revisiones de diseño:** Se revisan los diseños del software para identificar errores y mejorar la calidad.

**Codificación:**

* **Pruebas unitarias:** Se verifican las unidades de código de forma individual.
* **Pruebas de integración:** Se verifican las interfaces entre las diferentes unidades de código.

**Pruebas:**

* **Pruebas de sistema:** Se verifica que el sistema cumple con los requisitos y funciona correctamente.
* **Pruebas de aceptación:** Se verifica que el sistema es aceptable para los usuarios finales.

**Validación:**

* **Pruebas de instalación:** Se verifica que el software se instala correctamente en diferentes entornos.
* **Pruebas de compatibilidad:** Se verifica que el software funciona correctamente en diferentes plataformas y configuraciones.

**Mantenimiento y evolución:**

* **Pruebas de regresión:** Se verifica que las correcciones de errores no introducen nuevos errores en el software.
* **Pruebas de humo:** Se verifica que las funcionalidades básicas del software funcionan correctamente después de realizar cambios.

Especialidades del testing

Hay muchas especialidades a la hora de hablar de testing y cada una de ellas tiene diferentes funciones y perspectivas que se incluyen en diferentes fases del proceso

**Manual tester**: Nos ayuda a definir los casos de pruebas, establecer estrategias. También ejecuta, pero lleva lo necesario para que todos sepan qué hacer.

* Pensamiento lateral, piensa fuera de la caja con una nueva perspectiva, agrega nuevos casos de usos y entiende muy bien al usuario.

**Automation tester**: Se encarga de agilizar, acelerar el trabajo y actividades que nos quitan la oportunidad de encontrar más casos de usos. frameworks y soluciones. El código crece y las pruebas también, darles ese mantenimiento a las pruebas es un problema común.

**Security tester**: Encargado para el área de seguridad. Debe ser alguien que aporte valor desde la perspectiva de la seguridad.

* Protocolos, estándares, legalizaciones dependiendo de cada país y marca. Está enfocado en prever ataques, virus, problemas de seguridad, acceso no autorizado. Profundizar en las técnicas y prácticas de seguridad.

**Data science tester**: Con la manera en que crecen los datos en un proyecto, se necesita a alguien que los analice, agrupe y limpie estos datos.

* Análisis y Limpieza de datos, omite tener un set de pruebas amplio donde la variedad va a permitir detectar defectos inesperados. Esto puede ser clave para que los resultados no sean falsos positivos.

**SDET (**ingeniero de software interno**)**: Es la combinación de un desarrollador que ya sabe hacer pruebas. Con la diferencia de automatiza y hace uso de herramientas que organizan la operación de la entrega de las pruebas. Esta persona se asegura de que las pruebas se ejecuten antes de liberar el código.

* El programador ahora sabe hacer pruebas y conoce de herramientas que le permite entregarlas de una manera automatizada.

**DevOps**: Conoce todo lo anterior y domina el conocimiento de automatizar el proceso, se asegura de una entrega continua.

* Una automatización de la operación, Entrega Continua. Donde se entregan de forma más rápida las nuevas versiones.

**QA Engineer**: Quality Assurance. Está enfocado en el producto y en el proceso.

* Procesos de Calidad

**QE**: Quality Engineer. Es como un coach, acompaña en las políticas de calidad en la empresa o equipo.

**DevOps** es un conjunto de **prácticas culturales, filosóficas y de herramientas** que unen el desarrollo de software (Dev) y las operaciones de TI (Ops). Su objetivo principal es **aumentar la velocidad y la calidad** con la que se liberan las aplicaciones.

En otras palabras, DevOps intenta romper los silos tradicionales entre desarrolladores y personal de operaciones, fomentando la **colaboración y la comunicación** a lo largo del ciclo de vida del software. Esto permite una entrega más rápida y fiable de las aplicaciones, así como una mayor capacidad de adaptación a los cambios.

**Aspectos clave de DevOps:**

* **Automatización:** DevOps utiliza la automatización para tareas repetitivas en el ciclo de vida del software, liberando a los equipos para centrarse en tareas más estratégicas.
* **Integración e implementación continuas (CI/CD):** DevOps promueve la integración continua del código y la implementación continua de las aplicaciones, lo que permite una entrega más frecuente y con menos riesgos.
* **Monitoreo y registro:** DevOps enfatiza el monitoreo continuo del rendimiento y la salud de las aplicaciones, permitiendo la detección y resolución de problemas de forma proactiva.
* **Infraestructura como código (Infrastructure as Code - IaC):** DevOps trata la infraestructura como código, lo que permite su aprovisionamiento y configuración de forma automatizada y consistente.