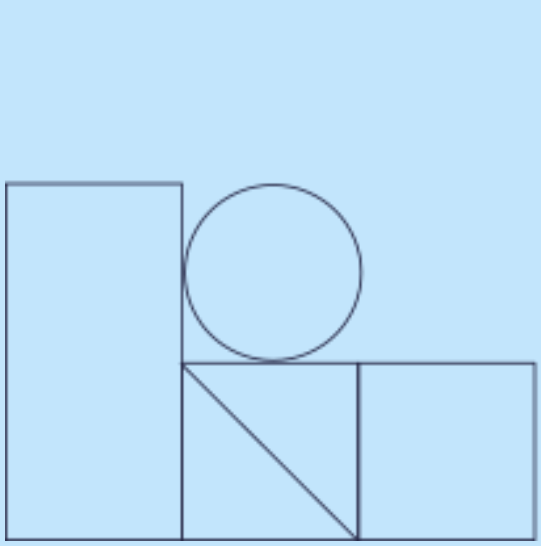
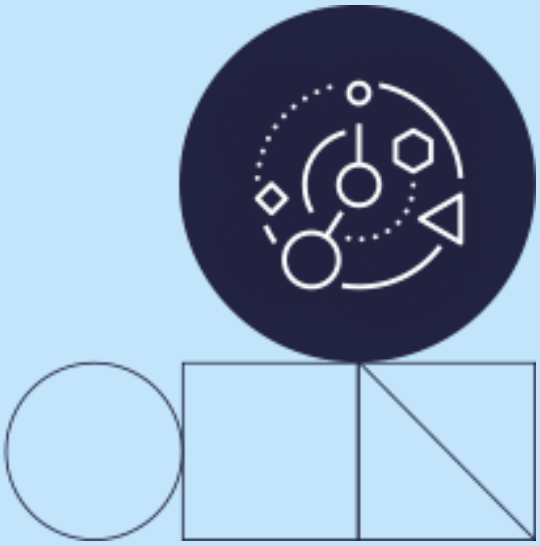
IBM SkillsBuild

IBM

Pruebas con Python

La importancia del testing





Índice

Introducción

Como bien sabemos, el proceso de creación de software se compone de varias fases. Desde su diseño hasta su puesta en producción, debe pasar por varios momentos en los que este software va evolucionando, sin embargo, hay una fase que no se ha potenciado tanto como debe a causa de su naturaleza. Efectivamente, el testing, a pesar de que se reconoce su importancia y necesidad, vemos como en muchos casos se hace mal o simplemente no se hace.

Qué es el testing de software

El testing de software o software QA es una disciplina en la ingeniería de software que permite tener procesos de ejecución de un programa o aplicación y una metodología de trabajo con el objetivo de localizar errores de software. También puede describirse como el proceso de validación y verificación de un programa de software o una aplicación.

Es imprescindible tener en cuenta que el testing es paralelo al proceso de desarrollo del software. A medida que se está construyendo nuestro producto, tenemos que realizar tareas de testing de software para prevenir incidencias de funcionalidad y corregir desviaciones del software antes de su lanzamiento.

Por qué el testing es importante

A un alto nivel, las pruebas de software son necesarias para detectar los errores en el software y para probar si el software cumple con los requisitos del cliente. Esto ayuda al equipo de desarrollo a corregir los errores y entregar un producto de buena calidad.

Hay varios puntos en el proceso de desarrollo de software en los que el error humano puede llevar a un software que no cumple con los requisitos de los clientes. Algunos de ellos se enumeran a continuación.

- El cliente/persona que proporciona los requisitos en nombre de la organización del cliente puede no saber exactamente qué es lo que se requiere o puede olvidarse de proporcionar algunos detalles, lo que puede llevar a que falten características.
- La persona que está recopilando los requisitos puede malinterpretarlos o no cumplirlos por completo al documentarlos.
- Durante la fase de diseño, si hay problemas en el diseño, esto puede conducir a errores en el futuro.
- Los errores pueden ser introducidos durante la fase de desarrollo durante un error humano, falta de experiencia, etc.
- Los probadores pueden perder errores durante la fase de prueba debido a errores humanos, falta de tiempo, experiencia insuficiente, etc.
- Es posible que los clientes no dispongan del ancho de banda necesario para probar todas las funciones del producto y que liberen el producto a sus usuarios finales, lo que puede dar lugar a que los usuarios finales encuentren errores en la aplicación.
- El negocio y la reputación de una organización depende de la calidad de sus productos y en algunos casos incluso los ingresos pueden depender de las ventas de productos de software.

Los usuarios pueden preferir comprar un producto de la competencia en lugar de un producto de baja calidad, lo que puede resultar en una pérdida de ingresos para la organización. En el mundo actual, la calidad es una de las principales prioridades de cualquier organización.

Qué es un tester

Los probadores de software (también conocidos como testers, su denominación en inglés) planifican y llevan a cabo pruebas de software de los ordenadores para comprobar si funcionan correctamente. Identifican el riesgo de sufrir errores de un software, detectan errores y los comunican. Evalúan el funcionamiento general del software y sugieren formas de mejorarlo.

En muchos casos, la fase del testing se ha relegado a una fase final previa a salida a producción y con un tiempo tan limitado que, en muchos casos, no pueden garantizar un testing eficaz.

Hablando de pruebas funcionales, el testing puede ser más valorable, dado a los resultados que esto ofrece (pasado o no pasado), por el contrario, otras pruebas, como rendimiento o seguridad, quedan relegadas a un punto menos cuantificable, ya que no afecta a su funcionalidad directa y por parte de negocio no suelen venir unos requisitos específicos.

Perfilando un tester

En un proyecto, el tester debe ser el segundo que más sepa del proyecto (inmediatamente después del jefe de proyecto), de su arquitectura (después de sus arquitectos), de su diseño (por detrás de sus diseñadores) y de su desarrollo (siguiendo a los desarrolladores). Únicamente hay un caso en el que el tester deba ser el que más sabe de algo, y es del testing. Además de esto, en un perfil de tester de rendimiento, el tester, además de todas las aptitudes citadas anteriormente, añadía el de matemático estadístico, dado que, como los resultados de rendimiento aparecen en gráficas y en estadísticas, el tester debe saber expresar estos términos matemáticos de una forma fácil y entendible.

Cómo implementar un servicio de testing desde cero

Implementar un servicio de testing desde cero es una tarea compleja y de bastante duración. En proyectos contrastados, vemos que se han ido dando pequeños pasos, pero efectivos y sin pausa hacia un servicio de QA en integración continua. Pasos como dedicar personas especializadas en este campo, la implementación de herramientas como Testlink para la gestión de pruebas, SonarQube para evaluar la calidad del código, Jenkins para una integración continua o Selenium para una automatización de pruebas. Dentro del futuro del testing, aparecen horizontes como el testing de Big Data, por lo que el futuro del testing está garantizado.

En nuestro caso usaremos unittest, un marco de pruebas unitarias que se inspiró originalmente en JUnit y tiene un estilo similar al de los principales marcos de pruebas unitarias en otros idiomas. Admite accesorios (fixtures), conjuntos de pruebas y un corredor de pruebas para permitir las pruebas automatizadas.

Los beneficios de las pruebas automatizadas

Realizar y ejecutar casos de prueba es una parte crítica de los servicios de desarrollo de software. Alcanzar altos niveles de calidad no solo beneficia al usuario final, sino que también evita errores en el software que perjudican el propósito del mismo. Dentro de este contexto, el equipo de QA debe tener en cuenta la cantidad de casos de prueba que se deben realizar y el tiempo requerido para ejecutarlos. En ciertas instancias, los testers deben ejecutar pruebas que tardan mucho, como una regresión completa del software; en otras ocasiones, solamente completar los casos de prueba de aceptación puede tomar bastante tiempo. Cualquier que sea el caso, estos tipos de pruebas implican una inversión de recursos que se podrían dedicar a otros aspectos específicos del software que pueden ser más complejos.

Las siguientes son algunas ventajas que dejan clara la importancia de la automatización de pruebas de software:

- **Ahorro de tiempo y esfuerzo.** Por ejemplo, en un sprint se pueden incluir varias funcionalidades nuevas que deben ser probadas, pero además de esto, se debe verificar que éstas no hayan afectado el sistema en general. En estas ocasiones, ejecutar una regresión es siempre muy útil. Si los casos de pruebas están automatizados, estos se ejecutan y los testers solamente deben revisar los resultados, lo cual les permite enfocarse en las pruebas manuales de las nuevas funcionalidades.

- **Verificar que los errores pasados no se reproduzcan.** En caso de que un error haya sido corregido, automatizar el caso de prueba que lo verifica puede asegurar que este error no sea introducido nuevamente por parte de algún cambio en otra funcionalidad.
- **Verificar que el producto sea funcional para el usuario.** Al enfocar los casos de prueba en la funcionalidad mínima del software, el equipo de QA puede asegurar que la nueva versión no vaya a afectar el mínimo útil para el usuario. Automatizar estos casos de prueba es de gran ayuda ya que al estar en etapas cercanas al lanzamiento, los testers se pueden concentrar en la nueva funcionalidad mientras que las pruebas de aceptación se ejecutan de forma automatizada.
- **Verificar que no haya ninguna funcionalidad rota en el 'build'.** Al desplegar una versión del producto, ya sea una nueva funcionalidad o alguna corrección, se debe verificar el 'build'. En este proceso de integración continua/despliegue continuo (CI/CD por sus siglas en inglés), la automatización de casos de prueba enfocados en la verificación del 'build' es de gran ayuda ya que garantizan una funcionalidad mínima.

Automatización de pruebas de software: pautas esenciales

Para poder contar con todos los beneficios de un testing de software automatizado, es crucial seguir las siguientes pautas:

- **Seleccionar cuáles casos de prueba se deben automatizar.** Esta tarea requiere tiempo y esfuerzo, por lo que haber seleccionado los casos de prueba con anterioridad es de gran utilidad cuando llega el momento de ejecutar las pruebas (ya sea en una regresión o pruebas de aceptación). Es clave seleccionar aquellos casos de prueba que permitan cubrir la mayor cantidad de módulos del producto o escoger aquellos que cubran la funcionalidad más crítica del software.
- **Seleccionar la herramienta correcta para crear los casos de prueba.** Es vital tener en cuenta las herramientas que se utilizarán para crear un framework donde implementarlas. De este modo, es más fácil aumentar la cobertura de pruebas en relación a los casos de pruebas automatizados y el tiempo que toma ejecutarlos. Además, se debe considerar todos los aspectos que se cubrirán como GUI, API, validaciones de base de datos y demás.
- **Implementar casos de prueba que se pueden ejecutar en múltiples ambientes de prueba.** Es crucial que los casos de prueba sean útiles para todos los ambientes ya que, en caso de ejecutar pruebas en ambientes similares a producción, solamente sería necesario cambiar los datos usados por cada caso. Esto hace mucho más fácil la actualización de los mismos.
- **Tomar en cuenta la actualización de casos de pruebas.** En caso de cambios en la lógica del software o en el producto en sí, es necesario que los casos de pruebas automatizados sean fáciles de actualizar. Además, estos deben ser fáciles de depurar para encontrar errores, ya que al ser software también puede que sea necesario agregar correcciones.

Es vital que el equipo de QA tome en cuenta las anteriores pautas para optimizar la automatización de testing de software y garantizar que ésta sea una herramienta que facilita su trabajo. De lo contrario, la automatización corre el riesgo de convertirse en una tarea adicional que requiere tiempo y esfuerzo. Al implementarse correctamente, las pruebas automatizadas traen muchos beneficios tanto para el desarrollo de software como para su verificación.

El objetivo

Algunas veces los presupuestos ajustados no permiten incluir un recurso dedicado íntegramente a esta tarea. En todos los casos es recomendable reservar una parte del presupuesto para realizar el testing.

Plantear una mejora cuantificable gracias a la calidad es muy difícil, ya que hablamos de mejoras cualitativas y no cuantitativas, sin embargo, se puede ver la necesidad del testing gracias a errores que ha habido a lo largo de la historia. Es bastante conocido que en jornadas puntuales los servicios pueden caer, como por ejemplo el Black Friday en grandes comercios o el comienzo de la campaña de la renta. También los seguidores de videojuegos conocerán el "parche del día 1", un parche para corregir fallos lanzado el mismo día del estreno del producto. Estos fallos podrían ser previstos gracias a una correcta ejecución de la fase de testing.

Tipos de pruebas

Dentro del mundo del desarrollo de software, se está otorgando un nivel de importancia cada vez mayor a las pruebas y la automatización de pruebas.

A continuación se detallan los principales tipos de pruebas usados a lo largo del ciclo de creación de un programa.

Prueba unitaria

La prueba unitaria es un método de prueba que se centra en examinar "unidades" individuales o fragmentos de código. El objetivo principal de las pruebas unitarias es determinar la integridad lógica: que un fragmento de código haga lo que se supone que debe hacer.

Generalmente, en este tipo de pruebas, se probarán métodos o funciones individuales como unidades y, dependiendo del tamaño y complejidad del código, también clases.

Se prueban de forma aislada y, posteriormente, las dependencias típicas se eliminan o se burlan.

Un ejemplo de esto podría ser una función que mande mensajes a los usuarios de una base de datos. Sin embargo, dado que lo que queremos hacer es una prueba unitaria, no usaríamos una base de datos real: haríamos una llamada a un punto final con stub, que devuelve los datos que normalmente esperaría de una base de datos. De esa forma, la única funcionalidad que se está probando es este fragmento de código o la unidad.

La mayoría de los lenguajes tienen al menos un marco de prueba unitario recomendado para sí mismo (por ejemplo, Java → JUnit , Python → PyUnit o PyTest , JavaScript → Mocha , Jest , Karma , etc.)

Prueba de integración

Las pruebas de integración son un método de prueba centrado en examinar varios componentes juntos.

El objetivo principal de las pruebas de integración es garantizar la integridad de la relación y el flujo de datos entre componentes o unidades. Es decir, si funcionan correctamente una vez que "juntamos" todo el código y lo ponemos en marcha.

Por lo general, se ejecutarán primero pruebas unitarias para probar la integridad lógica de las unidades individuales. Luego, ejecutaremos pruebas de integración para garantizar que la interacción entre estas unidades se comporte como se esperaba. Continuando con el ejemplo anterior, una prueba de integración en este caso sería ejecutar la misma prueba contra una base de datos real. Con bases de datos reales, existen escenarios y comportamientos adicionales a considerar.

"Prueba de integración" es un término amplio y abarca cualquier prueba en la que estén involucrados múltiples componentes. Posteriormente, se puede usar una gran variedad de tecnologías y marcos, incluidos los mismos que se usaron anteriormente en pruebas unitarias, o marcos separados basados en el comportamiento.

Prueba de extremo a extremo (E2E, sistema)

Las pruebas del sistema, o pruebas de un extremo a otro (E2E), se centran en examinar el comportamiento de un sistema de un extremo a otro.

El objetivo principal de las pruebas de extremo a extremo es garantizar que toda la aplicación o el sistema como una unidad se comporte como esperamos, independientemente del funcionamiento interno.

En esencia, las pruebas unitarias y de integración son típicamente de "caja blanca" mientras que las pruebas E2E son típicamente de "caja negra". Un ejemplo de prueba E2E podría ser "Obtener datos de un usuario". La entrada podría ser una simple solicitud GET a una ruta específica, y luego verificamos que la salida devuelta sea la que esperamos. La forma en que el sistema obtuvo esos datos es irrelevante.

Como podemos ver, las pruebas E2E solo pueden verificar el comportamiento general, por eso son necesarias las pruebas unitarias y de integración. Podría ser que, aunque el resultado sea correcto, la forma en que se obtiene el resultado internamente sea incorrecta, y una prueba E2E no lo detectaría.

Para las pruebas E2E, normalmente utilizamos marcos basados en el comportamiento.

Podemos usar marcos como Cucumber, Postman, SoapUI, Karate, Cypress, Katalon, etc. Muchos marcos de prueba de API se utilizan para las pruebas E2E porque una API es típicamente la forma en que interactúan las aplicaciones.

Prueba de aceptación

Las pruebas de aceptación suelen ser una fase del ciclo de desarrollo.

El objetivo principal de las pruebas de aceptación es verificar que un producto o característica determinada se haya desarrollado de acuerdo con las especificaciones establecidas por un cliente o un interesado interno, como un gerente de producto.

Dentro de las pruebas de aceptación, también puede haber múltiples fases, como las pruebas α o las pruebas β. A medida que gran parte del mundo del desarrollo de software avanza hacia los procesos ágiles, las pruebas de aceptación del usuario se han vuelto mucho menos rígidas y más colaborativas.

Es importante tener en cuenta que, si bien las pruebas de aceptación pueden verificar que la aplicación se comporte como el usuario desea, no verifica la integridad del sistema. Otra advertencia de las pruebas de aceptación del usuario es que hay un límite para los casos y escenarios que una persona puede idear; es por eso que los métodos de prueba automatizados anteriores son importantes ya que cada caso de uso y escenario está codificado.

Prueba de caja blanca (estructural, caja transparente)

Las pruebas de caja blanca (también llamadas estructurales o de caja transparente) describen pruebas o métodos en los que se conocen los detalles y el funcionamiento interno del software que se está probando.

Dado que conocemos las funciones, los métodos, las clases, cómo funcionan y cómo se unen, generalmente estas pruebas son las óptimas para examinar la integridad lógica del código.

Por ejemplo, es posible que sepamos que hay una peculiaridad en la forma en que un determinado idioma maneja ciertas operaciones. Podríamos escribir pruebas específicas para eso, que de otro modo no sabríamos escribir en un escenario de caja negra.

Las pruebas unitarias y las pruebas de integración suelen ser de “caja blanca”.

Pruebas de caja negra (funcional, conductual, caja cerrada)

Por el contrario, las pruebas de caja negra (también llamadas funcionales, de comportamiento o de caja cerrada) describen cualquier prueba o método en el que se desconocen los detalles y el funcionamiento interno del software que se está probando.

Dado que no conoce ninguno de los detalles, realmente no podemos crear casos de prueba que se dirijan a escenarios específicos o que hagan hincapié en la lógica específica del sistema.

Lo único que sabemos es que para una solicitud o una entrada determinada, se espera un determinado comportamiento o salida. Por lo tanto, las pruebas de caja negra prueban principalmente el comportamiento de un sistema. Las pruebas de un extremo a otro suelen ser una caja negra.

Prueba de caja gris

La prueba de caja gris es solo una combinación híbrida de caja negra y caja blanca.

La prueba de caja gris toma la facilidad y simplicidad de la prueba de caja negra (por ejemplo, entrada → salida) y apunta a sistemas específicos relacionados con el código de prueba de caja blanca.

La razón por la que existen las pruebas de caja gris es porque las pruebas de caja negra y las pruebas de caja blanca por sí solas pueden perder una funcionalidad importante.

- Las pruebas de caja negra solo prueban que obtiene una determinada salida para una entrada determinada. No prueba la integridad de los componentes internos; podríamos obtener la salida correcta por pura casualidad.
- Las pruebas de caja blanca se centran en la integridad de las unidades individuales y en cómo funcionan juntas, pero a veces son insuficientes para encontrar defectos en todo el sistema o en varios componentes.

Al combinar los dos tipos juntos, las pruebas de caja gris pueden abarcar escenarios más complicados para validar realmente que una aplicación es sólida en estructura y lógica.

Prueba manual

Como su propio nombre indica, las pruebas manuales son pruebas en las que un usuario especifica manualmente la entrada o interactúa con un sistema. También pueden evaluar manualmente los resultados.

Este método de prueba generalmente puede ser lento y propenso a errores. Gran parte de la industria del software se ha movido hacia las pruebas automatizadas junto con la adopción de principios ágiles.

Hoy en día, los usuarios pueden probar manualmente un producto en versión beta para verificar su aceptación, casos extremos y escenarios de nicho.

Prueba estática

La prueba estática describe cualquier método o método de prueba en el que no se ejecuta ningún código real.

En realidad, esto incluye revisar el código junto con otros testers, verificar manualmente la lógica y la integridad de las funciones, clases, etc.

Al igual que las pruebas manuales, las pruebas estáticas pueden ser lentas y propensas a errores, y generalmente las pruebas estáticas se realizan como primera línea de defensa para detectar problemas muy obvios.

Muchas empresas se involucran en revisiones de código antes de que el trabajo de un ingeniero se fusione en la rama principal. Estas revisiones de código son muy útiles para ahorrar tiempo.

Pruebas dinámicas

Las pruebas dinámicas describen cualquier método o método de prueba en el que se esté ejecutando realmente el código.

Generalmente, todos los métodos de prueba mencionados anteriormente son dinámicos, excepto los manuales y, a veces, de aceptación. Por lo general, ejecuta scripts automatizados o utiliza marcos para ejecutar entradas en su sistema.

Pruebas visuales / de interfaz de usuario (pruebas de navegador)

Las pruebas de la interfaz de usuario o del navegador describen pruebas que examinan específicamente la integridad y el comportamiento de los componentes de la interfaz de usuario.

A menudo, cuando se utiliza un sitio web, se espera que determinadas acciones den lugar a determinados estados.

Las pruebas de IU verifican que esto ocurra correctamente. Por ejemplo, la forma en que hemos implementado cierto código CSS puede fallar en Firefox, pero no en Chrome. Las pruebas del navegador pueden comprobarlo.

Hay muchos marcos de prueba de navegadores populares, como Selenium, Cypress, TestCafe, SauceLabs, KatalonStudio, BrowserSync, Robot, etc.

Prueba de humo

Las pruebas de humo solo se refieren a un subconjunto más pequeño de controles para verificar razonablemente que un sistema está funcionando.

Consisten en elegir y ejecutar un conjunto no exhaustivo de pruebas que examinen la funcionalidad principal.

Un ejemplo de esto podría ser probar solo un par de flujos de usuarios, como “Obtener datos de un usuario” de arriba. No es exhaustivo, pero dado que la mayoría de nuestra aplicación incluye un usuario que inicia sesión, realiza una solicitud y obtiene datos de algún lugar, esta prueba o algunas pruebas similares, pueden brindarnos una confianza razonable de que nuestro sistema es funcional y está funcionando.

Por lo general, las pruebas de humo se ejecutan cuando los usuarios esperan que los cambios no hayan tenido ningún impacto significativo en la lógica y la función generales. Puede ser costoso y llevar mucho tiempo ejecutar el conjunto completo de todas las pruebas cada vez, por lo que las pruebas de humo se utilizan como una medida de seguridad económica que se puede ejecutar con más frecuencia.

Prueba de regresión

La prueba de regresión es un método de prueba para verificar si alguna característica previamente funcional se ha roto (o retrocedido) repentinamente.

Esto a menudo incluye ejecutar la totalidad de todas las pruebas de unidad, integración y sistema para garantizar que ninguna funcionalidad haya cambiado inesperadamente.

Las pruebas de regresión a menudo requieren mucho tiempo y pueden ser muy costosas, por lo que a veces las personas realizan pruebas de humo en su lugar, especialmente si no se espera lógicamente que los cambios recientes afecten a todo el sistema.

A menudo, cuando las personas configuran CI / CD, ejecutarán pruebas de humo en casi todas las confirmaciones, mientras que las suites de regresión pueden ejecutarse a intervalos establecidos o en funciones grandes para garantizar una integración continua sin problemas.

Prueba de carga

Las pruebas de carga prueban la respuesta de una aplicación al aumento de la demanda.

Esto incluye probar las afluencias repentinas de solicitudes o usuarios que podrían ejercer una presión inesperada en el sistema. Las pruebas de carga a menudo se realizan como parte de las pruebas de seguridad para garantizar que una aplicación y su sistema no puedan ser DDOS.

Las pruebas de carga también se realizan para verificar la cantidad máxima de datos que un sistema puede manejar en un momento dado. Es fundamental para ayudar a los equipos a determinar fórmulas de escalamiento e implementación de alta disponibilidad (HA) efectivas.

Pruebas de inserción

Las pruebas de inserción (o de penetración) son una forma de prueba de seguridad que implica verificar la solidez y la seguridad de una aplicación.

Se explotan todas las formas en que una aplicación puede verse comprometida (secuencias de comandos entre sitios, entradas no desinfectadas, ataques de desbordamiento de búfer, etc.) para comprobar cómo la maneja el sistema. Las pruebas de penetración son una parte importante para asegurarse de que una empresa no sea víctima de infracciones graves.