Testing

¿Por qué molestarse con la disciplina de pruebas?

Las pruebas son tu primera y mejor línea de defensa contra defectos de software. Tus pruebas son más importantes que el análisis estático y linting (que sólo puede encontrar una subclase de errores, no problemas con la lógica del programa actual).

Las pruebas son tan importantes como la implementación en sí (todo lo que importa es que el código cumple con el requisito - cómo se implementa no importa en absoluto a menos que se implemente mal)..

Cosas que las pruebas pueden hacer por ti

- **Ayuda de diseño**: escribir las pruebas primero te dan una perspectiva más clara del diseño ideal de la API.
- **Documentación de la característica** (para los desarolladores): Las descripciones del test consagran en código cada requisito incorporado de la carcterística.
- **Prueba la comprensión del desarrollador**: ¿El desarrollador entiende el problema lo suficiente para articular en código todos los requisitos de los componentes críticos?
- **Garantía de calidad**: El control de calidad manual es propenso a errores. En mi experiencia, es imposible para un desarrollador recordar todas las características que necesitan pruebas después de hacer un cambio para refactorizar, agregar nuevas características o eliminar características.
- **Asistencia de entrega continua**: El control de calidad automatizado ofrece la oportunidad de prevenir automáticamente que las compilaciones rotas se desplieguen en producción.

Los diferentes tipos de prueba

La primera cosa que necesitas entender sobre los diversos tipos de pruebas es que todas tienen un trabajo que hacer. Todas tienen un papel importante en la entrega continua.
Existen tres tipos de prueba:

- **Las pruebas unitarias:** aseguran que los componentes individuales de la aplicación funcionen como se esperaba. Las aserciones prueban la API del componente.
- **Las pruebas de integración:** garantizan que las colaboraciones de componentes funcionen como se esperaba. Las aserciones pueden probar componentes API, UI o efectos secundarios (como E / S de base de datos, registro, etc ...).
- **Las pruebas funcionales:** Las pruebas funcionales aseguran que la aplicación funcione como se espera desde la perspectiva del usuario. Las aserciones prueban principalmente la interfaz de usuario.

> Debes aislar las pruebas unitarias, las pruebas de integración y las pruebas funcionales entre sí para poder ejecutarlas por separado durante las diferentes fases de desarrollo.

¿Qué son las pruebas unitarias?

Existen pruebas unitarias para probar unidades individuales de funcionalidad de software. Una unidad es un módulo, componente o función. Son partes del programa que pueden funcionar independientemente del resto del programa. La presencia de pruebas unitarias implica que el software está diseñado de manera modular. Usted puede oír de vez en cuando que hay maneras de hacer el software "más testeable".

Si encuentras que es difícil escribir pruebas unitarias para tu programa sin 'mockear' muchas otras cosas, eso es una señal de que tu programa no es lo suficientemente modular. Revela un acoplamiento estrecho (lo opuesto a la modularidad). Este es uno de los muchos papeles importantes que juegan las pruebas unitarias en la creación de software.

> Cuanto más dividas tus problemas en funciones simples y puras, más fácil será probar su código sin mocks.

Las pruebas unitarias deben ser:

- Simples.
- Rápidas.
- Un buen informe de errores.

¿Qué quiero decir con "un buen informe de errores?. Lo explico en los dos siguientes apartados:

La prueba unitaria como un informe de errores

Cuando una prueba falla, ese informe de error de la prueba es a menudo su primera y mejor pista sobre exactamente lo que salió mal - el secreto para rastrear una causa raíz rápidamente es saber dónde empezar a buscar. Ese proceso se hace mucho más fácil cuando se tiene un informe de error realmente claro.

¿Qué hay en un buen informe de error sobre el fallo de la prueba?

1. ¿Qué componente está bajo prueba?
2. ¿Cuál es el resultado esperado (comportamiento esperado)?
3. ¿Cuál fue el resultado real (comportamiento real)?
4. ¿Cómo es el comportamiento reproducido?

Las primeras tres preguntas deben ser visibles en el informe de fallos. La última pregunta debe estar clara desde la implementación de la prueba.

> Cuando una prueba unitaria falla, el mensaje de error es tu informe de error.

Principio FIRST en las pruebas unitarias

FIRST son las siglas de “Fast”, “Independent”, “Repeatable”, “Self-Validating” y “Timely”. Así de primera mano nos dan una pista de a que se refieren en el campo del testing, pero por si acaso vamos con una pequeña explicación de cada término:

- **Fast**: El conjunto de pruebas unitarias del proyecto o del módulo en el que se esté trabajando deberían poder pasarse en su totalidad en cuestión de segundos. Esta simple característica hace que nos cueste menos esfuerzo superar las pruebas. Además, es tan importante que podemos encontrar herramientas y plugins para los distintos entornos de desarrollo. Dichas herramientas nos permiten pasar las pruebas de forma automática según se van detectando los cambios en los archivos. Algunos ejemplos de estas herramientas son wallabyjs para Javascript e Infinitest para Java.
- **Independent**: Las pruebas tienen que ser totalmente independientes entre sí, pudiendo ejecutarlas de forma aislada o en un orden totalmente aleatorio, manteniendo siempre estables los resultados de las mismas.

- **Repeatable**: Implica que las pruebas deben ser siempre repetibles, independientemente del entorno, máquina o usuario que las ejecute. Esto quiere decir que los resultados de las pruebas deben ser siempre los mismos sin importar el entorno o la configuración sobre la que se estén ejecutando.
- **Self-validating**: Este término se refiere a que los test han de ser autoexplicativos, es decir, los resultados de los mismos tienen que aclarar sin lugar a dudas que ha ocurrido sin necesidad de tener que comparar valores o realizar alguna otra operación adicional.
- **Timely**: Este último término quiere reflejar el hecho de que los test unitarios se deben realizar en su justo momento. Se tienen que hacer justo antes de codificar la funcionalidad requerida, ni antes ni después. Tener una batería de pruebas grande antes de empezar a codificar puede ser una mala idea porque lanzándose a la vez podrían mostrar múltiples resultados exitosos a la par que erróneos, pudiendo llevarnos a error a la hora de interpretar dichos resultados. Y peor aún es hacerlos después, ya que en ocasiones tener el código funcionando puede ser una excusa para no hacerlos.

Pruebas de integración

Las pruebas de integración garantizan que varias unidades trabajen juntas correctamente. Por ejemplo, un controlador de ruta de nodo puede tomar un logger como una dependencia. Una prueba de integración podría afectar a esa ruta y probar que la conexión se registró correctamente.

Muchas pruebas de integración prueban las interacciones con los servicios, como las API de terceros, y pueden necesitar conectarse a la red para funcionar. Por esta razón, las pruebas de integración siempre deben mantenerse separadas de las pruebas unitarias, con el fin de mantener las pruebas unitarias corriendo tan rápido como pueden.

> Las pruebas de integración garantizan que varias unidades trabajen juntas correctamente

Pruebas funcionales

> Funcionan simulando las acciones que el usuario final podría tomar para lograr sus objetivos en su aplicación.

Las pruebas funcionales son pruebas automatizadas que aseguran que su aplicación hace lo que se supone que debe hacer desde el punto de vista del usuario. Las pruebas funcionales alimentan la entrada a la interfaz de usuario y realizan afirmaciones sobre la salida que aseguran que el software responda de la manera que debería.

Las pruebas funcionales a veces se llaman pruebas end-to-end (extremo a extremo) porque prueban toda la aplicación y su infraestructura de hardware y de red, desde la parte frontend de la UI hasta el back-end y los sistemas de bases de datos. En este sentido, las pruebas funcionales son también una forma de pruebas de integración, asegurando que las máquinas y las colaboraciones de componentes funcionen como se esperaba.

Las pruebas funcionales normalmente tienen pruebas exhaustivas de "happy paths" (trayectorias felices), lo que garantiza que las capacidades críticas de la aplicación, como los inicios de sesión de los usuarios, los registros, los flujos de trabajo de compra y todos los flujos de trabajo críticos de los usuarios se comporten como se esperaba.

Las pruebas funcionales deberían poder ejecutarse en la nube en servicios como Sauce Labs, que normalmente utilizan el WebDriver API a través de proyectos como Selenium.

Funcionan simulando las acciones que el usuario final podría tomar para lograr sus objetivos en su aplicación. Ellos pueden Hacer clic en botones, ingresar texto, esperar a que las cosas sucedan en la página y hacer afirmaciones mirando el 'output' de la UI.

Pruebas de Humo

Después de desplegar una nueva versión en producción, es importante averiguar de inmediato si funciona o no como esperaba en dicho entorno. Si no deseass que los usuarios encuentren los errores antes de que lo hagan - podría perseguirlos lejos.

Es importante mantener un conjunto de pruebas funcionales automatizadas que actúen como pruebas de humo para su version recién desplegada. Prueba toda la funcionalidad crítica de tu aplicación: Las cosas que la mayoría de los usuarios encontrarán en una típica sesión.

Las Pruebas de humo no son el único uso para las pruebas funcionales, pero en mi opinión, son los más valiosas.

> Se hace la analogía al humo, puesto que en bienes raíces se inyecta humo en las tuberías de agua para validar que no tengan fugas, evitando provocar inundaciones.