Contents

Título: Testing

Las cosas claras, los test no son opcionales. Una verdad como un templo y bien sabida por la comunidad, pero sigue siendo un tema pendiente en el mundo del desarrollo de aplicaciones de software actual.

Aunque parezca mentira, hay muchos compañeros “del metal” que no son consciente que programar sin pruebas no solo es como hacer acrobacias en el trapecio sin red de seguridad, sino además una fuente de errores, malas prácticas y ansiedad.

Así que, vayamos a repasar un poco por encima los conceptos básicos de las pruebas que se deberían aplicar.

¿Por qué hacer pruebas?

Para que lo entienda hasta el más novel, en resumen, asegurarse que lo que queremos que haga el programa, lo haga, y lo haga sin errores.

La construcción de software implica conocimiento, experiencia, talento, capacidad intelectual y un punto de arte. Es un trabajo que requiere que te guste y bien para su buen desempeño porque puede frustrar a más de uno.

> “Las pruebas no son opcionales. Software sin pruebas es igual a dinamita cerca de fuego”

¿A quién no le ha pasado que ha dejado su código medio año sin tocarlo y al volver a toquetearlo tiene la sensación que lo ha escrito otro? No conocemos nuestra propia criatura. Y no hablemos cuando estamos integrados en un equipo, o recibíamos el “regalito” de soportar o evolucionar un código heredado.

Por eso es clave contar con las pruebas adecuadas y bien programadas, para garantizar que las aplicaciones cumplen las funcionalidades que se esperan de ellas y las expectativas de calidad (no solo de código); ayudando a encontrar esos errores o defectos que aún no se han descubierto; reduciendo el costo del desarrollo, el de propiedad para los usuarios; y desarrollar confianza en los clientes al evitar los molestos errores de regresión.

Con todo esto ganamos la sensación de seguridad incremental que se obtiene cuanto más cerca estamos de un despliegue, ya que a más código que tenemos, más pruebas nos aseguran (en forma de una tupida malla) que todo funciona correctamente.

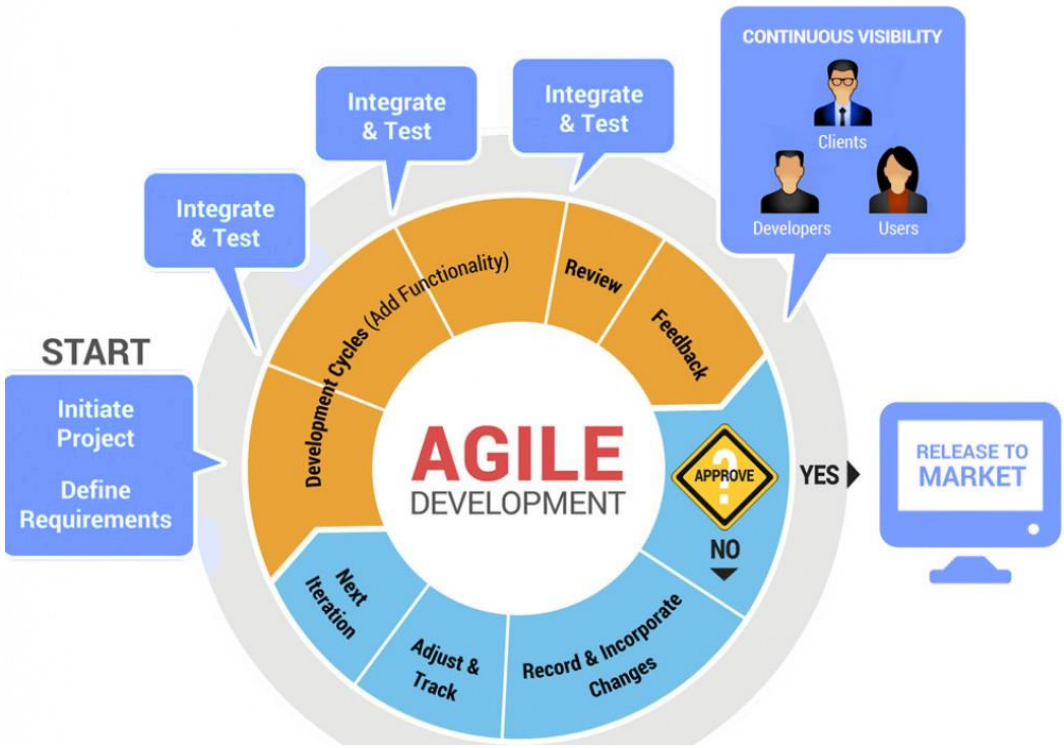
La llegada de DevOps

Las llegadas de las metodologías ágiles han contribuido en esta ardua tarea de crear un software robusto y estable, pero aún queda mucho para que sustituya al humano.

A finales del siglo pasado, llegaron las metodologías agiles, fue un revulsivo en la organización y ejecución de pruebas dentro de procesos Waterfall.

Eran pruebas definidas meticulosamente en voluminosos documentos de planes de pruebas y que solo se realizaban una vez estaba acabado la codificación del software.

Xtreme Programming en cambio, hizo mucho hincapié en la automatización y en el concepto de pruebas orientadas a la prevención de los finales de los años 80; marcando de esta forma, la futura filosofía Agile. Por ello, actualmente utilizamos framework de pruebas que permiten realizar automáticamente la mayoría de los test en todos los ámbitos de la aplicación.

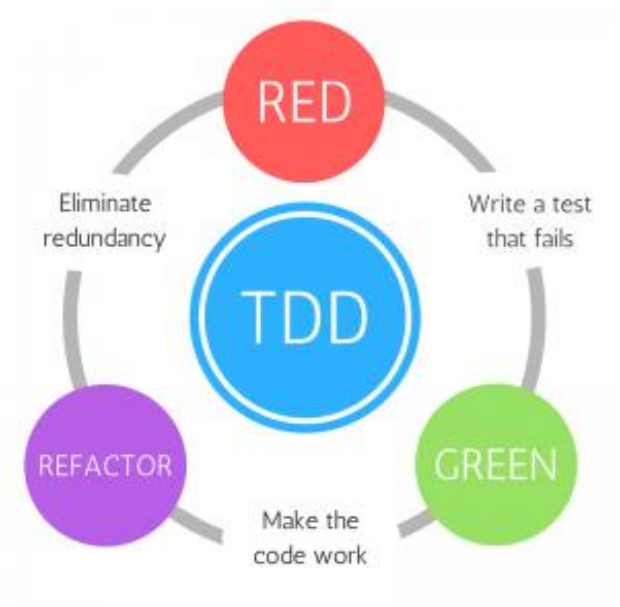


Y más cuando se pretende adoptar el concepto de Integración Continua, como parte imprescindible de DevOps, donde la propia construcción y validación de la Build por medio de todo tipo de pruebas automáticas es parte inherente del proceso.

“Es necesario que todas las pruebas manuales y automatizadas se complementen,”

Siendo esto aún más crítico en niveles altos de madurez en donde llegaríamos a aplicar despliegue automatizado o, incluso, continuo.

La importancia que han ido ganando las pruebas ha sido tal que la propia forma de codificar software también ha sufrido cambios profundos. El nacimiento de TDD (desarrollo orientado a pruebas) y su forma de supeditar el código a los test, implica que hacer software testeable es un requisito imprescindible en el código de calidad.



Y, aunque nunca lleguemos a utilizar tal técnica en nuestro desarrollo (que no es nada fácil), el objetivo de poder probar de forma automática nuestro código, ha reforzado practicas tan importantes en la programación orientada a objetos como es SOLID.

Tipos de pruebas

Como en toda industria, se desarrollan complejos pasos inherentes al proceso de Testing, sufren una sucesión inacabable de tipos, versiones, evoluciones y clases. Pero nos centraremos en las más importantes e imprescindibles, según el caso.

- Prueba Unitaria: Las pruebas unitarias son pruebas automatizadas que verifican la funcionalidad en el componente, clase, método o nivel de propiedad.

Su objetivo principal es tomar las piezas más pequeñas de la aplicación, aislarla del resto del código y determinar cuál es su comportamiento dentro de lo esperado. Cada unidad se prueba por separado antes de integrarlas en los componentes para probar las interfaces entre las unidades.

Las pruebas unitarias deben escribirse antes (o muy poco después) de escribir un método; siendo los desarrolladores que crean la clase o el método, quienes diseñan la prueba.

Así, conseguimos mantener el foco en lo que debe hacer el código, y se convierte en una poderosa herramienta para aplicar KISS, JIT, y mantener el foco en lo que tiene que hacer en vez de en el cómo, evitando introducir complejidad sin valor.

- Prueba de Integración: Se encargan de realizar íntegramente para formar componentes más grandes comprobando la interfaz entre ellas.

  Se centra en probar comunicación entre los componentes y su comunicación.

- Prueba de regresión: Cada vez que se modifica o se realiza cambios en un proyecto, existe la posibilidad de que el código anterior deje de funcionar correctamente o se descubran nuevos fallos.

  Por lo tanto, se evalúa el correcto funcionamiento del software frente a evoluciones o cambios funcionales. Su propósito es asegurar que los casos de pruebas existentes sigan exitosos. Se recomiendan que este tipo de pruebas sean automatizadas para reducir el tiempo y esfuerzo en ejecución.

- Prueba de funcionalidad: Prueba tipo caja negra basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. Dicho de otra forma, son pruebas específicas concretas y exhaustivas para probar y validar que el software hace lo que debe y, sobre todo, lo que se especifica a nivel de usuario.

  En el caso del trabajo manual, el trabajo realizado por un tester desde el punto de vista de un usuario, sigue una serie de pasos establecidos en el plan de pruebas, esperando el resultado esperado.

“Rompe la aplicación, llévala al límite y observa cómo se reconstruye”

- Pruebas de rendimiento: Determinan en condiciones particulares de trabajo, desde perspectiva, la capacidad de respuesta, rendimiento, escalabilidad, fiabilidad y eso de los recursos.

  En resumen, son un subconjunto de la ingeniería de pruebas, que se esfuerza en mejorar el rendimiento, englobando la arquitectura y diseño del software para conocer el estado de salud de la aplicación, obteniendo tendencias y previsiones de funcionamiento.

- Pruebas de estrés: O también conocida como prueba de tortura, es una forma de prueba deliberadamente intensa o exhaustiva que se usa para determinar la estabilidad de un sistema o entidad dado. Su objetivo es llevarla al extremo y ver su punto de ruptura, a fin de observar los resultados.

  Por ejemplo, un volumen alto de usuarios o sobrecarga de datos evaluando su capacidad de resiliencia volviendo a un estado óptimo.

- Pruebas de seguridad: Validar los servicios de seguridad identificando posibles fallos y debilidades a través de ataques, agujeros de seguridad, fallos, exploits…

En resumen y lo de siempre, intentar romperla desde cualquier punto de la red informática a la aplicación utilizando metodologías hacker.

En resumen, nunca se debe de subestimar el plan de pruebas, es importante para ofrecer la calidad de software que espera el cliente bajo los criterios acordados en la definición del proyecto para ofrecer un software robusto libre de errores. Como desarrolladores tenemos la responsabilidad y que procurar que nuestro código esté libre de bugs y cumple con los requisitos funcionales.