Los 7 principios del testing

Como hemos mencionado, los principios o reglas -en cualquier disciplina- orientan, estructuran y guían metodologías de trabajo. Es decir, nos ayudan a orientar la práctica profesional.

Les presentamos los 7 principios del testing:

1	El testing pone de manifiesto la presencia -no la ausencia- de defectos
2	El testing exhaustivo es imposible
3	El testing temprano ahorra tiempo y dinero
4	Defectos agrupados
5	Pesticide paradox
6	El testing es dependiente del contexto
7	La ausencia de errores es una falacia

1. El testing pone de manifiesto la presencia -no la ausencia- de defectos

A través de las pruebas es posible encontrar defectos, visibilizarlos y evidenciarlos. Sin embargo, es IMPOSIBLE confirmar la INEXISTENCIA de algún defecto silencioso, no descubierto en pruebas o validaciones.
Nunca puede determinarse la “ausencia total de fallos”.

💡Información de la industria :

Un sitio web podría fallar por razones relacionadas a la seguridad (fue hackeado). También puede fallar por razones de performance, por ejemplo no soportar la carga/ tráfico de usuarios.

2. El testing exhaustivo es imposible

Mientras usamos un sistema pueden ocurrir diversas situaciones. Algunas serán propias del contexto y/o del espacio en donde vivimos y otras inherentes al software.

¿Necesitas un ejemplo?

Estás comprando un pasaje en micro on line. Luego de seleccionar empresa, fechas de viaje, y reserva de asientos debes completar el formulario de pago para abonar con tu tarjeta de crédito. En ese momento suena el teléfono y mantienes una conversación de 20 minutos. Al finalizar vuelves a la página del sitio y decides terminar de completar el formulario-
¿Qué sucederá?

- ¿Se enviará el formulario correctamente al clickear “enviar”?
- ¿El sistema debería esperar media hora o varios días “abierto”?
- ¿El sistema debería timeoutear[1] y pedirme que vuelva a ingresar todos los datos de cero?

¿Es necesario que el equipo de desarrollo dedique esfuerzos en anticipar ese tipo de situaciones excepcionales?

Veamos otros ejemplos

[1] Es la acción de realizar time out. Se refiere a que una página excede el tiempo de espera.

¿Qué ocurre si me quedo sin conexión a internet mientras intento comprar entradas a un concierto? ¿La transacción debería cancelarse? ¿O almacenar los datos en mi perfil? ¿Y en el caso de una transferencia bancaria?

💡**Información de la industria :**

En algunos casos puede que el comportamiento del sistema frente una situación excepcional ni siquiera esté definido. Sobre todo si la probabilidad de que esa situación excepcional suceda es muy baja.

En la mayoría de los sistemas -a menos que sea extremadamente simple- es imposible testear todos las combinaciones de datos y escenarios. De hacerlo, podríamos poner en riesgo la eficiencia del plan de pruebas.

Para seleccionar qué pruebas testear y cuales no, hay que tener en cuenta los objetivos principales del sistema y los riesgos asociados al fallo de sus diferentes componentes.

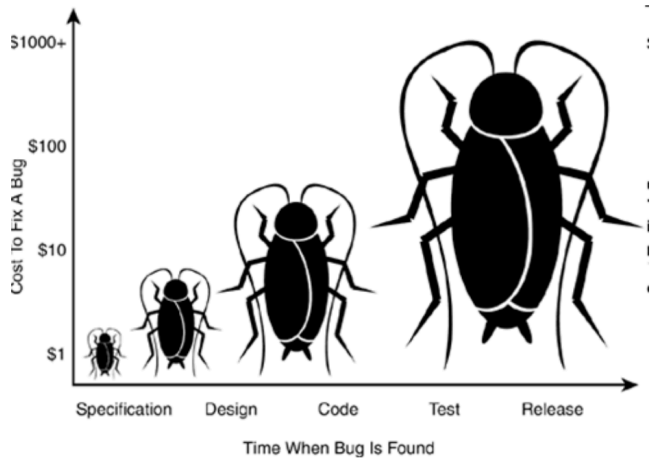
3. El testing temprano ahorra tiempo y dinero

Luego de todo lo analizado, pareciera claro entonces que mientras más temprano comiencen las pruebas de testing, menos costoso será corregir los bugs encontrados.
En ocasiones, si los tester tienen disponible el sistema para la revisión del plan de pruebas pueden encontrar menor cantidad de bugs en el código.

> 💡***¡Pro tip alert!*** *De ser posible, es bueno comunicar al equipo de desarrollo las pruebas que pensamos llevar a cabo. Es probable que los desarrolladores - al tener una idea de los escenarios de prueba que prevemos, los consideren en su trabajo y por ende existan menos bugs.*

Iniciando las tareas de testing y análisis con anticipación, evitamos que los errores que aparecen en las etapas tempranas del proceso de desarrollo migren a etapas más avanzadas.

La siguiente imagen te ayudará a graficar esta situación:



Software Cost-to-fix increasing 10X per project phase. Patton, R. (2005). Software Testing (2nd ed.).

4. Defectos agrupados

Es probable que la mayor cantidad de defectos se concentren en algunas áreas del sistema. Quizás en aquellas que revisten mayor complejidad o que fueron modificadas múltiples veces. También puede deberse al trabajo de desarrolladores con menor experiencia (solo por nombrar algunos factores)

Si bien es importante testear las áreas del sistema más conflictivas o que tiendan a tener defectos, esto no implica dejar de lado la ejecución de pruebas sobre otras partes que -a simple vista- parecen menos complejas.

5. Pesticide paradox/ Paradoja del pesticida

¿Conoces la frase que se le atribuye a Albert Einstein: **"Si buscas resultados distintos, no hagas siempre lo mismo"?**
Trasladamos el espíritu de la misma, a nuestro campo de estudio: repetir una y otra vez las pruebas no servirá para encontrar defectos nuevos.
Las pruebas de regresión tienen por objetivo revisar que los cambios introducidos en el sistema no rompan lo que funcionaba correctamente. Para esto, es necesario revisar estas pruebas y asegurar que sean relevantes para los requerimientos nuevos.
Si el sistema y sus funcionalidades van modificándose con el tiempo, las pruebas deben adaptarse a esos cambios.

6. El testing es dependiente del contexto

Qué tipo de pruebas implementar y cómo llevarlas a cabo dependerá de aquello que se está testeando.
Una aplicación web que permite el ingreso de datos personales requerirá más pruebas y foco en ciertos aspectos que un sitio web que sólo visualiza información.
Un sistema de navegación de un avión va a necesitar pruebas más exhaustivas que los dos sistemas mencionados anteriormente.

7. La ausencia de errores es una falacia[2]

Que no se hayan descubierto errores en un sistema, antes, durante o al finalizar el testeo no implica que el sistema carece de bugs.
El testing busca que el sistema que revisamos alcance niveles de calidad y sea aceptable (según los criterios establecidos) además de cumplir con los requisitos solicitados al inicio del desarrollo del sistema.

[2] Falacia: *argumento que parece válido, pero no lo es.*