Pruebas

Calidad: propiedad de un producto que hace que quien lo usa se sienta conforme. Aparece cuando el producto se construye.

Automatizar una prueba: codificar la prueba y ejecutarla como forma de chequer el funcionamiento del sistema/script. Ventajas:
-> independización del factor humano.
-> sencilléz: es muy fácil repetir las pruebas, y tiene un costo ínfimo.
-> facilita la comunicación.

Las pruebas sirven como control de calidad, pero nunca son suficientes de por si para asegurar la calidad del producto, ya que son sólo para detectar errores.

Hay distintos tipos de pruebas:

* **centradas en verificación**: controlan que se haya construido el producto tal como se pretendía (expectativas de los desarrolladores).

A su vez, se pueden clasificar en:
-> *unitarias*: verifican pequeñas porciones de código, como una postcondición de un método.
-> *de integración*: pruebas varias porciones de código, trabajando en conjunto (aunque no necesarimente de todo el sistema como un todo).
Ambas las desarrollan y ejecutan los programadores y suelen escribirse antes del código y estar automatizadas.

O bien:
-> *de caja negra*: cuando no se visualiza el código que se está probando.
-> *de caja blanca*: cuando se analiza el código durante la prueba.
Está ùltima no se suele usar a menos que no se encuentré la razón de un error y se requiera de un análisis más minucioso (ejemplo: debugging).
Ejemplos:
$ *prueba de escritorio* (en desuso): dar valores a las variables y hacer un seguimiento mental o en papel del código (pero esto esta sesgado por ser el programador el que las hace)
$ revisiones de código (con herramientas):

*__centradas en la validación__: controlan que se haya construído el producto que quería el cliente (expectativas del cliente - ¿funciona como el usuario quiere?).

Ejemplos (todas ellas con el sistema completo):
-> *pruebas de aceptación de usuarios (UAT)*: validaciones de usuarios, las cuales se deben ejecutar en un entorno lo más parecido posible al productivo. En principio las debería escribir los usuarios o los analistas de negocio. Si esto no fuera posible, las pueden escribir los testers o los programadores. Suelen ser automatizadas (salvo las que son explícitamente de experiencia de usuario).
-> *pruebas alfa*: pruebas de usuarios reales en un entorno controlado por el equipo de desarrollo.
-> *pruebas beta*: pruebas de usuarios con el producto listo para ser validado en el entorno del cliente.

Ejemplos (no del sistema completo):
-> prueba de comportamiento: validan que la lógica de la aplicación sea correcta. Pueden ser desarrolladas por programadores, testers o incluso usuarios (es una prueba colaborativa), y suele estar automatizadas.

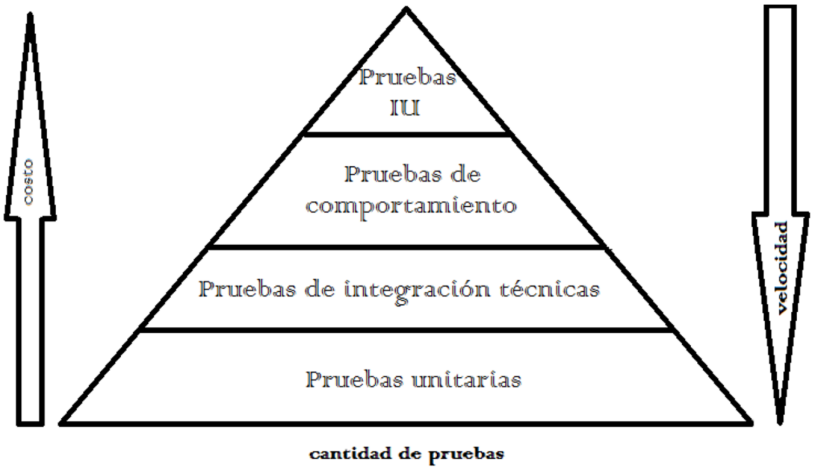
Las pruebas que se suelen automatizar son las funcionales y algunas de atributos de calidad. Las que implican experiencia del usuario (usabilidad, navegabilidad, estética, etc), deben ser probadas manualmente. Estas últimas no deberías ser realizadas por los programadores (tradicionalmente, pueden ser hechas por analistas o testers, pero también se le puede dar participación a los usuarios).

*__centradas en funcionalidades__: controlan lo que el programa puede hacer (pruebas funcionales, es decir, que surgen naturalmente de los requerimientos del usuario).

* __centradas en atributos de calidad__: controlan características no funcionales del sistema (compatibilidad, rendimiento, de estrés, seguridad y recuperación ante una falla, entre muchas otras).

* __de producción__: poner el sistema en un ambiente productivo y esperar a que falle, o a que un usuario reporte un problema (es una opción en aplicaciones que cuyos errores no tienen criticidad alta). Ejemplo: prueba A/B (para ver cómo responden distintos grupos de usuarios a una variable binaria antes de habilitar una característica para todos).

__* pruebas de regresión__: pruebas de todo el sistema ejecutadas a intervalos regulares para evitar que dejen de funcionar cosas que ya habían sido provadas (y eventaulmente entregadas) cada vez que se incorpora una característica nueva al sistema. Casi siempre se automatizan.



Se dice que las pruebas de la punta son más "frágiles" porque están expuestas a mayores cambios, y por ende su mantenimiento es más costoso.

Si falla una prueba en cada nivel, puede ser problema de ese nivel o del anterior (el de abajo), con el caso particular de que si falla una prueba unitaria, el problema podría estar en el mismo código.

Diseño de pruebas

Para las pruebas unitarias, el buen diseño debe contemplar:

- hacer las pruebas antes; utilizar pruebas de verificación de caja negra para no limitar las pruebas a una implementación específica);
- seguir las pautas del diseño por contrato, estableciendo precondiciones (que definen excepciones) y postcondiciones (que definen pruebas);
- casos borde: valores extremos que no suelen considerarse por no ser parte del normal funcionamiento del sistema, pero que deben ser soportados ya que existen como posibilidad. Se debe tener una prueba por cada caso borde (que usualmente se traduce en un caso particular o en una excepción).

Para las pruebas de cliente:

- que el cliente especifique con ejemplos.

- que haya una prueba para: cada escenario típico, para cad aflujo de excepción y para cada flujo alternativo.

Cobertura: el grado en que los casos de prueba de un programa llegan a recorrer dicho programa al ejecutarse (nivel de exhaustividad). La cobertura mide sólo la calidad de las pruebas, indirectamente, y sólo subsidiariamente afecta la calidad del programa (por lo que no debe guiar el desarrollo).

Se utilizan métricas de cobertura para observar tendencias a medida que la aplicación crece, por ejemplo: cobertura de sentencias, de condiciones, de trayectorias, de invocaciones, entre otras. No se pretende abarcar en su totalidad a las métricas ya que esto trae acarreado mayores costos que beneficios; se tiende a un 85% de las métricas cubiertas en general, pudiendo aumentar ese porcentaje en zonas de mayor criticidad, y disminuirlo en zonas de menor criticidad.

Prácticas metodológicas para asegurar la calidad

Enfoque de ciclo de vida de software: se prueba antes de la puesta en producción
Enfoque de procesos iterativos: se prueba después de cada iteración.
Enfoque de métodos ágiles: se prueba continuamente, automatizándolas lo más posible y haciendo que las pruebas guíen el proceso de desarrollo.

Hoy en día el modelo es más parecido al último: las pruebas son continuas a lo largo de todo el desarrollo.

Prácticas para orientar el desarrollo de software

TDD (Test Driven Development): la primera práctica basada en automatización de pruebas (unitarias) que estuvo bien soportada por herramientas (está dentro de la metodología Extreme Programming).
Es una práctica iterativa incremental de desarrollo de software que se base en:

1- Escribir las pruebas antes que el código; esas pruebas deben fallar inicialmente (test-first).
2- Automatizar dichas pruebas (automatización).
3- Refactorización: mejorar la calidad del código sin cambiar el comportamiento.

Ventajas (no sólo de esta metodología, sino en ventajas en general de hacer las pruebas antes):

-> La prueba en código sirve como documentación del uso esperado, sin ambigüedades (mejor que la prosa).
-> Permite entender mejor qué se quiere desarrollar, y logra un conjunto de verificaciones mucho más abarcativa y menos condicionada (en el caso de que sea el programador el que hace las pruebas).
-> Permite especificar el comportamiento sin restringirse a una implementación.
-> Al ser automatizadas, tienen las consecuentes ventajas de la automatización.

Particularmente, la refactorización es una ventaja específica de la práctica TDD.

Desventaja:

-> minimizado la importancia de las pruebas de aceptación para derivar el comportamiento del sistema.

BDD (Behavior Driven Development): variante de TDD que evolucionó hasta derivar en una manera de especificar el comportamiento esperado mediante escenarios concretos que se puedan automatizar como pruebas de aceptación.

STDD (Storytest Driven Development): es una práctica que pretende construir el software basándose en hacer pasar las pruebas de aceptación (automatizadas) que acompañan las "historias de usuario" (representación de un requisito del usuario que sirve como parte de la especificación y el alcance del proyecto).

ATDD (Acceptance Test Driven Development): pareido a STDD; construye el producto en base a pruebas de aceptación de usuarios, con menos énfasis en la automatización de las pruebas y más en el proceso en sí mismo.

SBE (Specification By Example): práctica colaborativa de construcción basada en especificaciones mediante ejemplos que sirven como pruebas de aceptación.

Las cuatro se basan en la misma idea: yo diseño mi programa en base a funcionalidades y casos de acpetación que me define el usuario.

Test Impact Analysis: no ejecutar todas las pruebas en todas las integraciones si llevan mucho tiempo.

Integración continua: realizar la compilación, construcción y pruebas del producto en forma sucesiva y automática como parte de la integración. La automatización termina con una prueba de integración en el ambiente de desarrollo.

Entrega continua: se pretende que el código siempre esté en condiciones de ser desplegado en el ambiente productivo. Como las pruebas de aceptación suelen ser más lentas de ejecutar que las unitarias, se puede ser más laxo en cuanto a la frecuencia de las ejecuciones de las mismas (igual se ejecutan al menos una vez al día).

Despliegue continuo: cada pequeño cambio se despliegue en el ambiente de producción, para lo que se suelen usar ejecuciones sistemáticas de pruebas en producción.

Con estas tres últimas prácticas se disminuye el riesgo de la aparición de errores en las pruebas, porque cualquier problema que surja es atribuible al último tramo de código desarrollado e integrado.