Niveles

Pruebas de Sistemas

UNIVERSIDAD SIGLO 21

Niveles de pruebas

Existen tres preguntas elementales que cualquier metodología nos debe proporcionar para responder:

- **¿Qué se debe probar?** Es decir, ¿cómo elegir un *set* de pruebas apropiado para usar en determinada situación?

- **¿Cuándo deben comenzar las pruebas y cuándo deben terminar?** Enmarca una serie de preguntas como ¿cuándo se considera una prueba exitosa y cuándo no?, o bien ¿cuándo debería comenzarse con el trabajo del diseño y construcción de los casos de pruebas?

- **¿Quién lleva adelante las pruebas?** Esto enmarca no solo a quién hace las pruebas, sino cómo se deberían coordinar con el resto del equipo involucrado en el proceso de desarrollo.

Probar es simplemente seleccionar algo para medir, es decir, un atributo de calidad. Supone desarrollar casos de pruebas con entradas controladas que ejerciten el programa o revelen algo del atributo que se quiera medir, simular situaciones de prueba y observar los resultados en comparación con el comportamiento esperado. Se considera exitoso si los resultados observados dan con lo esperado, y no exitoso cuando sucede lo contrario, es decir, cuando los valores observados no alcanzan los esperados.

El punto central está en seleccionar qué probar.

Por otra parte, encontramos lo que se denomina **economía de las pruebas**. El proceso de pruebas de por sí es un proceso caro, mirado como un costo y no como una inversión. De acuerdo con los resultados obtenidos, puede generar la reescritura del código y eso puede retrasar el proceso de desarrollo.

¿Qué consideras más económico?: ¿probar y encontrar defectos antes de la entrega al cliente o no probar y encontrar defectos una vez que está en producción?

¿Qué involucra encontrar un defecto en producción?

Encontrar un defecto en producción puede generar múltiples consecuencias; dentro de las más importantes, podemos mencionar:

- Cambios de requerimientos, especificaciones y código.
- Cambios en los casos de pruebas y rehacer todos los niveles de pruebas.
- Cambios y reparaciones de lo que está instalado.
- Costo de administración y pérdidas de negocios.

Por esto, cuanto antes se detecta un defecto, más barato resulta corregirlo.

Existen muchas acciones que se pueden utilizar para la prevención de defectos o la detección temprana, y evitar así su multiplicación. Por ejemplo, el análisis de especificaciones, durante la preparación de la prueba, pone de manifiesto errores en las especificaciones que, si no se encontraran en el momento adecuado, implicarían el desarrollo del sistema de manera incorrecta.

Por este motivo, es fundamental probar tempranamente el diseño para prevenir la multiplicación de defectos; probar desde los primeros componentes individuales y no esperar hasta que esté conformado como un todo.

Hetzel (1988) plantea tres niveles de pruebas elementales. Los programas individuales son probados como componentes individuales y se denominan **pruebas de unidad**. Luego, los grupos de programas son probados todos juntos, como un sistema, esto se conoce como **pruebas de sistema** y, por último, una vez que el sistema está completo, se hacen las **pruebas de aceptación**.

Las organizaciones o los equipos de pruebas suelen realizarlas pasando por estos tres niveles. Las pruebas de aceptación pueden ser llevadas a cabo por los usuarios finales o clientes.

Hetzel (1988) también reconoce otros niveles de pruebas utilizados en proyectos más largos y complejos, como podrían ser las **pruebas de integración**. Actualmente, es importante pensar en este tipo de pruebas, debido a la complejidad de los sistemas y la interacción que deben tener con otros.

Pruebas de componente

También son conocidas como **pruebas de unidad**, **pruebas unitarias** o **pruebas de módulos**, según como se denomine al elemento mínimo que se emplee en el lenguaje de programación. Esto es, si se llama función, clase o componente, así se denominará a este nivel de pruebas, que consiste en la prueba de los componentes más pequeños, apenas son generados.

Cuándo terminar de probar, es normalmente una decisión del programador. Además, puede darle tanto un enfoque de perspectiva externa, con casos de pruebas basados en las especificaciones de lo que se supone que hace el sistema, o bien un enfoque de perspectiva interna, con casos de pruebas diseñados para ejercitar y cubrir la lógica interna del programa.

La mayoría de las veces, las pruebas se realizan de manera informal, es decir, que no se mantienen registros de los casos de pruebas corridos ni de los defectos encontrados, por lo que se puede decir que este nivel de pruebas finaliza cuando el programador se siente conforme o a gusto con el programa y ha corregido las deficiencias que descubrió.

Son pocas las organizaciones que realizan el testing unitario de manera formalizada. En el plan de pruebas, se describe qué será testeado y los resultados esperados, los cuales son establecidos tanto por los programadores como por los diseñadores. También se establece un criterio de cobertura, es decir, un porcentaje de código que deberá ser cubierto o alcanzado por las pruebas para decir que estas se completaron de manera fehaciente. Este nivel de pruebas, tan relacionado con el código, tiene como objetivos validar:

- tipos de datos inconsistentes;
- valores de inicialización y default incorrectos;
- correspondencia entre parámetros y argumentos en las llamadas de funciones o módulos;
- precisión en las funciones de cálculo;
- comparación entre tipos de datos diferentes;
- terminaciones de loop impropias o no existentes;

- variables de loop modificadas inapropiadamente;
- manejo de errores inadecuado o inentendible.

Pruebas de integración

Si bien Hetzel (1988) plantea tres niveles elementales de pruebas (componentes, sistemas y aceptación), también reconoce la necesidad de otro nivel, que, como mencionamos, es fundamental en la actualidad. Nos referimos a las **pruebas de integración**.

Pressman (2006) plantea, en la explicación de este nivel, que solo un neófito en el mundo del *software* podría conjeturar: "Si todo funciona bien individualmente, ¿por qué dudan que funcione cuando se une?" (p. 319). Justamente, ahí es donde está el punto. Es en la unión de los componentes donde pueden perderse datos o, al combinarse las funciones, no dar el resultado esperado.

La prueba de integración es la que está orientada a asegurar que las unidades individuales operan correctamente cuando se combinan en la aplicación.

Pueden evaluarse:

- interfaz entre componentes;
- interacciones con diferentes partes de un sistema (SO [sistema operativo], sistema de archivos, hardware);
- interfaz entre sistemas.

No tiene que ser entendido como un nivel que se da siempre luego del nivel de sistemas, como se encuentra en algunas bibliografías, ya que:

- la prueba de integración de componentes se hace una vez que la prueba de componentes se llevó a cabo;
- la prueba de integración de sistemas se hace una vez que la prueba de sistemas se llevó a cabo.

Pero ¿cómo se hace la integración? Puede ser de dos maneras: la llamada ***big bang*** o **incremental**.

Pruebas de integración big bang

Este tipo de integración es la que, una vez probados todos los componentes individuales, se combinan todos de una sola vez y se prueba como un todo.

¿Qué imaginas que puede suceder si, al aplicar la integración big bang, se encuentra una gran cantidad de errores?

Al no poder aislar las causas de esos errores, es decir, conocer en la integración de qué partes se originaron, es difícil corregirlos. Si, una vez corregidos, surgen otros, se entra en un proceso que parece interminable.

Pruebas de integración incremental

Esta modalidad viene a dar solución a lo expuesto anteriormente. Es "la antítesis del enfoque Big Bang" (Pressman, 2006, p. 395). En ella se integran y prueban pequeños incrementos del programa, lo que hace más fácil encontrar la causa de las fallas y así corregir los defectos.

Dentro de esta modalidad de integración, existen diferentes estrategias.

- Descendente (top-down): se integran comenzando por el módulo de control principal y luego se integran los módulos subordinados. Puede ser en profundidad o en anchura (Rodriguez y Soria, 2012).

- Integración ascendente o *bottom-up*: se combinan primero los módulos de bajo nivel que juntos realicen alguna subfunción específica. La mayoría de las veces, es necesario escribir un controlador que coordine la entrada y salida de los casos de pruebas para testear los grupos. Una vez probado el grupo, se eliminan dichos controladores, se integra con otros grupos y se asciende por la estructura. (Uninotas, 2017, https://bit.ly/2Uzfzjz)

¿Qué es un controlador, también llamado *driver*?

Si se tienen los módulos *Y* y *Z* listos, pero el módulo *X* no está listo, y para probar los módulos Y y Z se necesita que el módulo X devuelva valores, entonces se escribe una pieza de código ficticio de X que devuelve valores para Y y Z. Esta pieza de código ficticio es lo que se llama *controlador*.

Los controladores aportan el entorno al proceso del sistema o subsistema, con el objeto de permitir o producir entradas y salidas del sistema (o subsistema) o con el objeto de registrar datos.

Lo que pareciera tan simple en lo que es la integración y el uso de controladores no lo es tanto. Al reemplazar los controladores de prueba por componentes reales, se pueden generar nuevos defectos:

- Pérdida de datos, manipulación errónea de datos o entradas erróneas.
- Los componentes involucrados interpretan los datos de entrada de una manera distinta a los controladores.
- Los datos son transferidos en un momento incorrecto: muy pronto, muy tarde, a una frecuencia distinta de la requerida.

Luego de repasar los diferentes enfoques, ¿cuál es el esquema de integración que recomiendas emplear?

Lo ideal es una combinación de ambos esquemas, según la necesidad que se tenga. Hay que tener en cuenta que los módulos críticos deben ser probados lo más tempranamente posible.

Pruebas de sistema

Las pruebas de sistema comienzan cuando una aplicación funciona como un todo. Tiene por objetivo determinar si el sistema en su globalidad opera satisfactoriamente (recuperación de fallas, seguridad y protección, *stress*, *performance*, otros). Las funciones del sistema son ejercitadas y el programa es estresado para descubrir las limitaciones y medir los topes de capacidades.

La mayoría de las pruebas de sistema se basan en la perspectiva de caja negra, es decir, sin considerar cómo está hecho el programa, sino solo al considerar las funcionalidades solicitadas. El concepto de *cobertura* también se ha extendido a este nivel y solicita que se exprese la cantidad de pruebas realizadas en términos de número de rutinas o funciones

probadas, y se establece un porcentaje de acuerdo con algún estándar que, en comparación, determine cuándo se terminó de probar.

Las pruebas de sistema son mucho más formales que las pruebas de componentes: registran y mantienen los resultados de las pruebas. Es frecuente que se utilicen herramientas como **generadores de datos de pruebas**, que crean baterías de datos para pruebas basadas en ciertos parámetros de entrada, o **comparadores**, que se usan para contrastar dos archivos y encontrar las diferencias.

El ambiente para este nivel de pruebas, donde el sistema funciona como un todo, debe corresponderse, lo más que se pueda, con el ambiente de producción.

Pruebas de aceptación

Cuando las pruebas de sistema se completaron, comienzan las pruebas de aceptación. Para llegar a este nivel de pruebas, también se ha dado el nivel de integración en cualquier punto anterior del proceso: cuando se han realizado las pruebas de sistemas, se hace la integración; o, luego de las pruebas de componentes, se hace la integración.

Las pruebas de aceptación tienen como propósito darle al cliente o al usuario final la confianza y la seguridad de que el sistema está listo para ser utilizado, por lo que encontrar defectos no tiene que ser el foco principal.

Los sets de pruebas se arman con casos de las pruebas de sistemas. Incluyen transacciones y escenarios típicos del negocio. Frecuentemente, se hacen de manera informal, y los registros se mantienen en función de cómo fueron los resultados.

Como la representación del usuario final o del cliente es vital en este nivel, se dice que es este quien dirige las pruebas de este nivel. En otras palabras, la prueba realizada es por el usuario final o cliente para determinar si la aplicación se ajusta a sus necesidades.

Si bien aquí se presenta como un simple nivel de pruebas, según el proceso de desarrollo que se siga no tiene que ser visto como un nivel de pruebas que se hace al final de todo. Por ejemplo, las **pruebas de aceptación** de un componente pueden tener lugar no bien la prueba de componente se haya hecho.

Este nivel de pruebas es popularmente conocido como UAT (*user acceptance testing*) o **prueba de aceptación de usuario**, lo que indica que

el usuario es el actor principal. Sin embargo, además de las pruebas de usuario, en el International Software Testing Qualifications Board (ISTQB) (2011) se plantean otros tipos de pruebas de aceptación, en función del objeto que sea sometido a prueba.

- **Pruebas de aceptación de contrato**: se refiere a las pruebas que demuestran que los criterios de aceptación definidos en el contrato cliente-proveedor se han alcanzado. Demuestra que el sistema adhiere a normativas gubernamentales, definiciones impositivas y reglamentaciones relacionadas. Por ejemplo, si se desarrolla un sistema de facturación, este debe adherir a las normas impositivas de discriminación de IVA (impuesto al valor agregado), cobrar el IVA que corresponde según el tipo de producto, por ejemplo, cobrar el impuesto interno a las bebidas, entre innumerables reglamentaciones que existen.

- **Pruebas de aceptación operacional**: se refiere a lo que implica la operación de un sistema. Por ejemplo, las pruebas de *backups* y *restore*, chequeos periódicos de seguridad ante vulnerabilidades, recupero de desastres, entre otros.

Este nivel de pruebas introduce dos conceptos, según el laboratorio en el que se realicen las pruebas: **pruebas alfa** y **pruebas beta**.

La organización desarrolladora del sistema no puede prever cómo el usuario final utilizará el sistema, ya que este puede llevar a que el usuario malinterprete alguna de sus funciones o que tenga alguna combinación de datos extraña, entre otros ejemplos que pueden surgir.

Ya se vio que, en las pruebas de aceptación, el cliente aplica una serie de pruebas que permiten validar los requisitos.

¿Cómo se hace si el sistema o software que se desarrolla va destinado a muchos clientes diferentes o no tiene un cliente específico?

Esto resulta impráctico. Lo que se hace es tomar clientes que representen el sector de mercado al que se apunta, para que usen el sistema del mismo modo que lo harían si lo compraran. La idea es retroalimentarse con los comentarios, reportes de deficiencias, fallos y depurar el sistema antes de lanzarlo al mercado para su comercialización. Un ejemplo típico es cuando Microsoft lanza una nueva versión del Office y lo deja a disposición de los usuarios. ¿Quién no ha sido alguna vez *beta tester* de este tipo de programas?

***Pruebas beta**: pruebas realizadas por el usuario en ambientes de producción y de trabajo reales.*

Es una práctica muy frecuente en nuestros días, sobre todo para empresas proveedoras de sistemas que no tienen gran reconocimiento.

***Pruebas alfa:** pruebas realizadas por el usuario en ambientes de laboratorio.*

Por su definición, podemos apreciar que, si estamos en las pruebas beta, donde se requiere una versión estable del sistema, ya que el usuario seleccionado lo instalará para llevar adelante sus funciones, de algún modo es una prueba de aceptación de usuario.

De la misma manera, en la definición de que las pruebas alfa son pruebas realizadas por el usuario en ambientes de laboratorio también se indica que se realizan dentro del nivel de aceptación.

Lo que queda claro es que no son tipos de pruebas de aceptación, sino modos de llevar adelante estas pruebas.

Algunos autores llaman pruebas alfa a todas las pruebas desarrolladas, cualquiera sea el nivel al que pertenezcan, mientras sean de laboratorio, y beta a las pruebas que hace un cliente seleccionado al que se le da el sistema para su uso. Pero no es el enfoque más popularmente aceptado, ya que el concepto incluye al cliente o usuario final en ambos modos de prueba (alfa y beta), mientras este enfoque no lo hace.

A continuación, presentamos un cuadro que resume los tres niveles fundamentales dados por Hetzel (1988), con sus características básicas.

Tabla 1: Tres niveles de Hetzel

	Unitario	Sistema	Aceptación
Objetivo	Confirma que el módulo o componente fue probado correctamente	Módulos o componentes ensamblados que trabajan como un sistema	Evaluar su condición para su uso
¿Quién lo hace?	Normalmente, los programadores	Equipo de pruebas	Usuarios finales o cualquier agente que lo represente
¿Qué se prueba?	Funcionalidades. El código puede ser probado. Se exploran límites y extremos	Requerimientos de sistemas y funciones	Principales funciones, documentación y procedimientos

Fuente: elaboración propia a base de Hetzel, 1988, pp. 9-12.

Referencias

Hetzel, B. (1988). *An Introduction. The Complete Guide to Software Testing* (2nd ed). Massachusetts, US: QED Information Science.

International Software Testing Qualifications Board (ISTQB). (2011). *Fundamentals of Testing*. *Certified Tester. Foundation Level Syllabus*. Recuperado de https://www.istqb.org/downloads/send/2-foundation-level-documents/3-foundation-level-syllabus-2011.html

Pressman, R. (2006). *Ingeniería de Software. Un enfoque práctico* (6.ª ed.). México D. F., MX: McGraw-Hill.

Rodriguez, A y Soria, V. (2012). *Clasificación de Herramientas Open Source para Pruebas de Aplicaciones Web – Un caso de estudio.* Recuperado de: http://sedici.unlp.edu.ar/bitstream/handle/10915/63222/Documento_completo__.pdf-PDFA.pdf?sequence=1

Uninotas (2017). *Prueba, implementación y mantenimiento del sistema.* Recuperado de: https://www.uninotas.net/prueba-implementacion-y-mantenimiento-del-sistema-3/