**Pruebas Unitarias**

Uno de los métodos más utilizados para realizar pruebas a nuestro software es el llamado de pruebas unitarias (unit testing). La base de este método es el hacer pruebas en pequeños fragmentos de nuestro programa. Estos fragmentos deben ser unidades estructuradas de nuestro programa encargados de una tarea específica, en programación procedural u orientada a objetos podemos afirmar que estas unidades son los métodos o las funciones que tenemos definidos.  
  
    Tras realizar estas pruebas sobre los elementos unitarios de nuestro programa podremos eliminar gran parte de los errores que podríamos encontrar. Cualquier prueba demuestra no la falta de errores sino que revela la presencia de ellos. Las pruebas unitarias no revelan errores en la integración de las partes unitarias ni tampoco otros problemas como el bajo rendimiento de las aplicaciones o problemas derivados del sistema sobre el que está ejecutándose nuestro programa.  
  
    El objetivo de las pruebas unitarias es el aislamiento de partes del código y la demostración de que estas partes no contienen errores.  
  
    Una vez creados el conjunto de pruebas unitarias sobre un fragmento de código los beneficios obtenidos son múltiples. Tales como:  
  
**Simplificación de la integración**  
    Las pruebas unitarias eliminan las posibles incertidumbres y errores en lo que se espera de cada una de las unidades ayudando a entender la integración de cada una de las partes.  
  
**Refactorización de código**  
    Una vez refactorizado el código; las mismas pruebas unitarias nos pueden servir para probar el nuevo código asegurándonos de que este sigue siendo válido bajo la nueva implementación.  
  
**Documentación**  
    Las pruebas unitarias sirven como método de documentación mismo. Los desarrolladores pueden ver a través de las pruebas unitarias cual es el objetivo de las distintas partes del código de una manera básica.  
  
**Diseño**  
    Cuando se desarrolla el software las pruebas unitarias pueden tomar el lugar del diseño formal. Cada prueba unitaria puede ser visto como un elemento de diseños que especifica las clases, los métodos y el comportamiento observable de la aplicación.

**Pruebas de rendimiento**

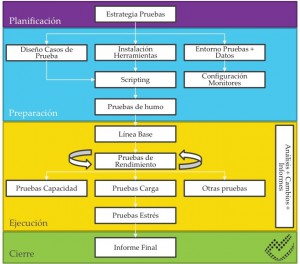
El servicio de pruebas de rendimiento de software se centra en determinar la velocidad con la que el sistema bajo pruebas realiza una tarea en las condiciones particulares del escenario de pruebas. Este servicio ayuda a su organización a detectar los cuellos de botella de su aplicación, antes de que, sus usuarios sufran un mal rendimiento, con la consecuente pérdida económica y frustración de sus clientes o empleados.

La disciplina de las pruebas de rendimiento ejecutan los siguientes tipos de pruebas:

* **Pruebas de rendimiento:** El objetivo es determinar si el usuario estará satisfecho con la velocidad de la aplicación, bajo condiciones de uso esperadas durante el “día a día”.
* **Pruebas de carga:** El objetivo de esta prueba es determinar el rendimiento necesario para que el sistema funcione en hora punta.
* **Pruebas de capacidad**: En este caso, se busca dar una estimación de hasta dónde se puede llegar cargando el sistema antes de que sea inutilizable (por ejemplo, una página web que responde a peticiones en 15 segundos de media, funciona correctamente, no obstante, no es útil al usuario).
* **Pruebas de estrés:** Conocidas como “stress testing”, muchas veces engloban de forma errónea al resto de pruebas mencionadas anteriormente. El objetivo de estas pruebas es obtener datos, sobre la carga del sistema, que ayuden a realizar el dimensionamiento del sistema o “capacity planning”. Esta prueba genera carga en el sistema hasta hacerlo inutilizable. Una vez que la aplicación ha dejado de funcionar, nuestros consultores se centran en distintos objetivos, como por ejemplo: verificar la calidad de los mensajes de error del sistema o establecer alertas para poder anticipar un fallo total del sistema. Las pruebas de estrés son uno de los últimos tipos de pruebas que se deben ejecutar, ya que, por su carácter poco realista, podría darse el caso de que la situación de carga simulada nunca se diera en la vida real.

Las pruebas de rendimiento son ejecutadas por medio de scripts automatizados, éstos se encargan de emular las acciones que realizaría un usuario final sobre la aplicación bajo pruebas. Los scripts se ejecutan en paralelo, cada uno de ellos emulando un “usuario virtual”, de esta forma, anticipando la carga esperada cuando el sistema pase a producción. Durante la ejecución de las pruebas, el encargado debe vigilar el sistema, que recibe la carga por medio de indicadores de rendimiento, esta acción es comúnmente llamada monitorización del sistema. En base a las métricas obtenidas, se sugiere mejoras para optimizar el rendimiento del sistema y, de esta forma, mejorar los tiempos de respuesta de la aplicación.

**Metodologías de pruebas de rendimiento**

[](http://www.globetesting.com/wp-content/uploads/2011/10/Metodolog-Pruebas-Rendimiento-Globe-Testing.jpg)

La noche antes de un paso a producción suele estar caracterizada por pocas horas de sueño y mucha tensión, causada por la incertidumbre de no saber cómo se va a comportar el sistema una vez estén todos los usuarios trabajando en el entorno. Las pruebas de rendimiento tienen como objetivo anticipar los problemas que puedan ocurrir una vez la aplicación esté en producción. Hacer pruebas de rendimiento significa dormir bien, sabiendo que su sistema está preparado para la carga esperada.

No realizar pruebas de rendimiento supone, en muchos casos, una perdida económica, no solo causada por la falta de disponibilidad de sus sistemas y el impacto que esto tiene en su producción, sino también por el impacto que la falta de servicio tiene en el usuario final (ya sea un cliente que no volverá a usar su sistema por falta de confianza, o un empleado que siempre se quejará de la velocidad de su sistema).

**Pruebas de Aceptación**

El uso de cualquier producto de software tiene que estar justificado por las ventajas que ofrece. Sin embargo, antes de empezar a usarlo es muy difícil determinar si sus ventajas realmente justifican su uso. El mejor instrumento para esta determinación es la llamada **prueba de aceptación**. En esta prueba se evalúa el grado de calidad del software con relación a todos los aspectos relevantes para que el uso del producto se justifique.  
  
Para eliminar la influencia de conflictos de intereses, y para que sea lo más objetiva posible, la prueba de aceptación nunca debería ser responsabilidad de los ingenieros de software que han desarrollado el producto.   
Para la preparación, la ejecución y la evaluación de la prueba de aceptación ni siquiera hacen falta conocimientos informáticos. Sin embargo, un conocimiento amplio de métodos y técnicas de prueba y de la gestión de la calidad en general facilita esta labor.   
  
La persona adecuada (o el equipo adecuado) para llevar a cabo la prueba de aceptación dispone de estos conocimientos y además es capaz de interpretar los requerimientos especificados por los futuros usuarios del sistema de software en cuestión.

**Pruebas de Caja Blanca**

La prueba de la caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar los casos de prueba.

Las pruebas de caja blanca se llevan a cabo en primer lugar, sobre un módulo concreto, para luego realizar las de caja negra sobre varios subsistemas (integración).

En los sistemas orientados a objetos, las pruebas de caja blanca pueden aplicarse a los métodos de la clase, pero según varias opiniones, ese esfuerzo debería dedicarse a otro tipo de pruebas más especializadas (un argumento podría ser que los métodos de una clase suelen ser menos complejos que los de una función de programación estructurada). Dentro de las Pruebas de Caja Blanca encontramos las llamadas coberturas (sentencia, decisión, condición y múltiple además de los mencionados caminos ciclomáticos propuestos por McCabe).

Las pruebas de caja blanca intentan garantizar que:

* Se ejecutan al menos una vez todos los caminos independientes de cada módulo.
* Se utilizan las decisiones en su parte verdadera y en su parte falsa.
* Se ejecuten todos los bucles en sus límites.
* Se utilizan todas las estructuras de datos internas.

**Pruebas de Caja Negra**

Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

La prueba de Caja Negra no es una alternativa a las técnicas de prueba de la Caja Blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca. Muchos autores consideran que estas pruebas permiten encontrar:

1. Funciones incorrectas o ausentes.
2. Errores de interfaz.
3. Errores en estructuras de datos o en accesos a las Bases de Datos externas.
4. Errores de rendimiento.
5. Errores de inicialización y terminación.

Para preparar los casos de pruebas hacen falta un número de datos que ayuden a la ejecución de los estos casos y que permitan que el sistema se ejecute en todas sus variantes, pueden ser datos válidos o inválidos para el programa según si lo que se desea es hallar un error o probar una funcionalidad. Los datos se escogen atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa corra bien.

Para desarrollar la prueba de caja negra existen varias técnicas, entre ellas están:

1. Técnica de la Partición de Equivalencia: esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
2. Técnica del Análisis de Valores Límites: esta Técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
3. Técnica de Grafos de Causa-Efecto: es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

Dentro del método de Caja Negra la técnica de la Partición de Equivalencia es una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas existentes en el software, descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así en número de clases de prueba que hay que desarrollar.

**Pruebas de Humo**

Una prueba de humo o smoke test, es un testing rápido que se realiza sobre aspectos funcionales no tanto para encontrar bugs sino para asegurarse que la funcionalidad básica del software o de una parte del software se encuentre estable y responda al comportamiento esperado.

El objetivo es verificar, con pruebas sencillas y que demanden poco tiempo, que ciertos caminos de la aplicación funcionen correctamente. Normalmente se elige un conjunto de funcionalidades significativas, no hace falta que sean todas las de la aplicación.

Es frecuente aplicar este tipo de pruebas previo a una entrega al cliente, de manera que se realice una última comprobación para verificar que a alto nivel todo está bien (antes, como es lógico, se deberían haber realizado pruebas en profundidad del sistema en todos los niveles y no solo pruebas funcionales).

Las pruebas de humo ponen de manifiesto si el software está o no lo suficientemente estable para afrontar un ciclo de pruebas y son un paso previo a la ejecución del plan de pruebas diseñado previamente.

Las características de un smoke test son las siguientes:

* Tests cortos, idealmente un caso para probar por arriba que el feature funcione. Si esta es compleja siempre tratar de tener la mínima cantidad de casos.
* Idealmente no deben encontrarse bugs durante este test o muy pocos. Como dije antes, sirve para quedarnos tranquilos que todo funciona correctamente. Para encontrar bugs estuvieron las pruebas previas, más exhaustivas en cada feature.
* El tiempo de ejecución del smoke test completo debe ser corto.
* Los casos por lo general son positivos. O sea, comportamiento esperado en situaciones en la que el usuario no hace nada raro.
* Debe correrse una vez que se corrieron todos los casos nuevos y regresiones en el producto. Como un chequeo final antes del release.