Prueba unitaria

En programación, una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión.
La idea es escribir casos de prueba para cada función no trivial o método en el módulo de forma que cada caso sea independiente del resto.

Características

Para que una prueba unitaria sea buena se deben cumplir los siguientes requisitos:

- Automatizable: no debería requerirse una intervención manual. Esto es especialmente útil para integración continua.
- Completas: deben cubrir la mayor cantidad de código.
- Repetibles o Reutilizables: no se deben crear pruebas que sólo puedan ser ejecutadas una sola vez. También es útil para integración continua.
- Independientes: la ejecución de una prueba no debe afectar a la ejecución de otra.
- Profesionales: las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc.

Aunque estos requisitos no tienen que ser cumplidos al pie de la letra, se recomienda seguirlos o de lo contrario las pruebas pierden parte de su función.

Ventajas

El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el trozo de código debe satisfacer. Estas pruebas aisladas proporcionan cinco ventajas básicas:

1. Fomentan el cambio: Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura (lo que se ha dado en llamar refactorización), puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
2. Simplifica la integración: Puesto que permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente. De esta manera se facilitan las pruebas de integración.
3. Documenta el código: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.
4. Separación de la interfaz y la implementación: Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro, a veces usando objetos mock (mock object) para simular el comportamiento de objetos complejos.
5. Los errores están más acotados y son más fáciles de localizar: dado que tenemos pruebas unitarias que pueden desenmascararlos.

Limitaciones

Es importante darse cuenta de que las pruebas unitarias no descubrirán todos los errores del código. Por definición, sólo prueban las unidades por sí solas. Por lo tanto, no descubrirán errores de integración, problemas de rendimiento y otros problemas que afectan a todo el sistema en su conjunto. Además, puede no ser trivial anticipar todos los casos especiales de entradas que puede recibir en realidad la unidad de programa bajo estudio. Las pruebas unitarias sólo son efectivas si se usan en conjunto con otras pruebas de software.

Herramientas

- JUnit: Entorno de pruebas para Java creado por Erich Gamma y Kent Beck. Se encuentra basado en SUnit creado originalmente para realizar pruebas unitarias para el lenguaje Smalltalk.
- TestNG: Creado para suplir algunas deficiencias en JUnit.
- JTiger: Basado en anotaciones, como TestNG.
- SimpleTest: Entorno de pruebas para aplicaciones realizadas en PHP.
- CPPUnit: Versión del framework para lenguajes C/C++.
- NUnit: Versión del framework para la plataforma.NET.
- MOQ : Framework para la creacion dinamica de objetos simuladores (mocks). http://code.google.com/p/moq/

Pruebas funcionales

Una prueba funcional es una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. Las pruebas funcionales se hacen mediante el diseño de modelos de prueba que buscan evaluar cada una de las opciones con las que cuenta el paquete informático.

Pruebas de integración

Pruebas integrales o pruebas de integración son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias. Únicamente se refieren a la prueba o pruebas de todos los elementos unitarios que componen un proceso, hecha en conjunto, de una sola vez.
Consiste en realizar pruebas para verificar que un gran conjunto de partes de software funcionan juntos.
Las pruebas de integración (algunas veces llamadas integración y testeo I&t) es la fase del testeo de software en la cual módulos individuales de software son combinados y testeados como un grupo. Son las pruebas posteriores a las pruebas unitarias y preceden el testeo de sistema.

Pruebas de validación

Las pruebas de validación en la ingeniería de software son el proceso de revisión que el sistema de software producido cumple con las especificaciones y que cumple su cometido. Es normalmente una parte del proceso de pruebas de software de un proyecto, que también utiliza técnicas tales como evaluaciones, inspecciones, y tutoriales. La validación es el proceso de comprobar lo que se ha especificado es lo que el usuario realmente quería.
Se trata de evaluar el sistema o parte de este durante o al final del desarrollo para determinar si satisface los requisitos iniciales. La pregunta a realizarse es: ¿Es esto lo que el cliente quiere?.

Enfoques a la verificación

- Dinámica de verificación, también conocido como ensayos o experimentación.
- Estática de verificación, también conocido como análisis.

Tipos

- Pruebas de aceptación: desarrolladas por el cliente.
- Pruebas alfa realizadas por el usuario con el desarrollador como observador en un entorno controlado (simulación de un entorno de producción).
- Pruebas beta: realizadas por el usuario en su entorno de trabajo y sin observadores.

Características

Comprobar que se satisfacen los requisitos:

- Se usan la mismas técnicas, pero con otro objetivo.
- No hay programas de prueba, sino sólo el código final de la aplicación.
- Se prueba el programa completo.
- Uno o varios casos de prueba por cada requisito o caso de uso especificado.
- Se prueba también rendimiento, capacidad, etc. (y no sólo resultados correctos).
- Pruebas alfa (desarrolladores) y beta (usuarios).

Caja blanca (sistemas)

En programación, se denomina cajas blancas a un tipo de pruebas de software que se realiza sobre las funciones internas de un módulo. Así como las pruebas de caja negra ejercitan los requisitos funcionales desde el exterior del módulo, las de caja blanca están dirigidas a las funciones internas. Entre las técnicas usadas se encuentran; la cobertura de caminos (pruebas que hagan que se recorran todos los posibles caminos de ejecución), pruebas sobre las expresiones lógico-aritméticas, pruebas de camino de datos (definición-uso de variables), comprobación de bucles (se verifican los bucles para 0,1 y n iteraciones, y luego para las iteraciones máximas, máximas menos uno y más uno.

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).
Este concepto también es utilizado de manera análoga en la teoría general de sistemas.

Caja negra (sistemas)

En teoría de sistemas y física, se denomina caja negra a aquel elemento que es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno. En otras palabras, de una caja negra nos interesará su forma de interactuar con el medio que le rodea (en ocasiones, otros elementos que también podrían ser cajas negras) entendiendo qué es lo que hace, pero sin dar importancia a cómo lo hace. Por tanto, de una caja negra deben estar muy bien definidas sus entradas y salidas, es decir, su interfaz; en cambio, no se precisa definir ni conocer los detalles internos de su funcionamiento.

Justificación

Un sistema formado por módulos que cumplan las características de caja negra será más fácil de entender ya que permitirá dar una visión más clara del conjunto. El sistema también será más robusto y fácil de mantener, en caso de ocurrir un fallo, éste podrá ser aislado y abordado más ágilmente.

Caja negra y Programación modular

En programación modular, donde un programa (o un algoritmo) es divido en módulos, en la fase de diseño se buscará que cada módulo sea una caja negradentro del sistema global que es el programa que se pretende desarrollar, de esta manera se consigue una independencia entre los módulos que facilita su implementación separada por un equipo de trabajo donde cada miembro va a encargarse de implementar una parte (un módulo) del programa global; el implementador de un módulo concreto deberá conocer como es la comunicación con los otros módulos (la interfaz), pero no necesitará conocer como trabajan esos otros módulos internamente; en otras palabras, para el desarrollador de un módulo, idealmente, el resto de módulos serán cajas negras.