

Pruebas Unitarias
Qué son
Una prueba unitaria es aquella que verifica un aspecto muy determinado de la funcionalidad. Cada prueba unitaria se corresponde con un requerimiento (a bajo nivel) que el código debe satisfacer.
Características
<ul style="list-style-type: none"> - Enfocadas a un aspecto concreto (atomicidad). - De rápida ejecución. - Independencia: <ul style="list-style-type: none"> - de otras pruebas. - del orden de ejecución. - del entorno⁽¹⁾. - Automatizadas. - Calidad de código como el de producción.
No son pruebas unitarias las que...
<ul style="list-style-type: none"> - No operan de forma aislada. - Interactúan con una red. - Acceden a bases de datos. - Acceden al sistema de ficheros. - No pueden ser ejecutadas al mismo tiempo que otras pruebas unitarias. - Deben hacer algo con el entorno (por ejemplo, leer ficheros de configuración) para ser ejecutadas.
Casos de prueba
<ul style="list-style-type: none"> - Un caso de prueba, o <i>test case</i>, determina si el código cumple con los requisitos comprendidos en el mismo. - Los casos de prueba se recogen en una <i>suite</i> de pruebas.
Características de los Test Cases
<ul style="list-style-type: none"> - Cobertura. - Repetibles. - Independientes. - Profesionales.
⁽¹⁾Simulación de entornos
En casos en que las pruebas unitarias deban interactuar con un entorno, debe usarse <i>mocks objects</i> , código que lo simula.
Ventajas
<ul style="list-style-type: none"> - Automatización de las pruebas. - Rapidez de verificación. - El código no es propietario (facilita la incorporación de personas al equipo). - Mayor frecuencia de integración del código. - Documentan el código con las especificaciones del mismo. - Defectos acotados y fácilmente localizables. - Ritmo de trabajo constante. - Facilidad para agregar nuevas funcionalidades al código. - Posibilita los test de regresión. - <i>Refactoring</i> constante, con mejora de la calidad del código.
Desventajas
<ul style="list-style-type: none"> - No descubren todos los defectos del código: determinan la presencia de éstos en el código, no su ausencia. - El código solo es tan bueno como lo sean sus pruebas. - A veces no es trivial determinar todos los casos de prueba.

Pruebas Unitarias
Guía de escritura de pruebas
<ul style="list-style-type: none"> - Rápidas y pequeñas: ya que la suite debe ejecutarse repetidas veces. - Automatizadas, no interactivas: las ejecuta el Framework de pruebas, no una persona. - Fáciles de ejecutar: sin configuraciones previas de la suite. - Comprobar su alcance: Todo el código funcional debe ser sometido a pruebas. - Corregir inmediatamente: Al detectar un error, corregir el mismo para que pasen las pruebas. - Mantener el nivel de unidad: Las pruebas unitarias validan clases, no aplicaciones, o todo un flujo de trabajo. - Simplicidad: Comenzar codificando las pruebas más simples e inmediatas. - Independencia: Unos tests no deben depender de otros. - Cercanía: Las clases de prueba deben estar ubicadas junto a las que están validando. El proyecto de pruebas debe estar bien organizado. - Nombrado: Deben darse nombres asociados a lo que estamos probando. Para la clase Ejemplo, su test asociado debe ser una clase EjemploTest. - Probar las interfaces públicas: Las partes privadas quedan probadas a través de éstas. - Caja negra: Perspectiva del cliente. - Caja blanca: Perspectiva de programador. - Casos triviales: Deben comprobarse siempre. Al fin y al cabo, son los más fáciles de probar. - Comprobar la cobertura: No sólo todo el código debe probarse, también hay que tener en cuenta los parámetros de entrada. - Comprobar casos límite: cadenas no ASCII, valores NaN, infinitos, divisiones por cero... - Generador aleatorio: Cuando sea posible, crear un generador aleatorio de valores para el test. - Probamos una funcionalidad: Un test unitario es, como su nombre indica, atómico. No debemos pretender probar toda nuestra funcionalidad desde un único test. - Usar las aserciones adecuadas: Hay para todos los tipos, usar la más afín. - Pruebas negativas: que deben ser siempre erróneas, y darán robustez a la batería de pruebas. - No conectar al exterior: La prueba no debe interactuar con entornos reales. - Simular bugs: Al informarnos de un bug, intentar recrearlo como un método de prueba. - Realismo: Con todo, pueden escaparse errores. Complementar con otros tipos de pruebas.

Test-Driven Development
Qué es
TDD es una parte de la metodología ágil Extreme Programming . Implica la especificación de los requerimientos como pruebas que debe satisfacer el código. Por tanto, no es sólo un método de desarrollo, sino también de diseño.
Ciclo de TDD
<ul style="list-style-type: none"> - Preparar un listado de casos de prueba. - Ritmo de trabajo según TDD: <ul style="list-style-type: none"> - Seleccionar una prueba. - Implementar la prueba. Fallará. - Escribir el código que supere la prueba. - <i>Refactoring</i> para dar mayor calidad al código.
El ritmo TDD se resume en "Rojo-Verde-Refactoring".
¿Qué incluir en las pruebas?
- Todas aquellas partes del código que puedan "romper".
¿Qué no incluir en las pruebas?
<ul style="list-style-type: none"> - Propiedades de las clases. - Métodos privados de las clases. - Interfaz de usuario.
Selección de la prueba a codificar
Para escoger el siguiente test a codificar debemos guiarnos por aquél que sea el siguiente paso lógico en la construcción de nuestro software.
Tras alcanzar "verde", revisar...
<ul style="list-style-type: none"> - Que no exista duplicación de código. Si es así → Refactoring. - El ámbito de actuación de la operación. - Las posibles entradas que puede recibir.

Buenas prácticas
Lo que debemos hacer con TDD
<ul style="list-style-type: none"> - Escribir primero las pruebas. - Mantener una lista de tareas (<i>tests</i> y <i>refactoring</i> pendientes). - Escribir código nuevo sólo si falla alguna prueba unitaria. - Ejecutar siempre toda la suite de pruebas, no un test aislado. - El código de pruebas debe ser de calidad
Lo que no debemos hacer con TDD
<ul style="list-style-type: none"> - Escribir más de una prueba a la vez. - Escribir nuevas pruebas sin verificar si es necesario el <i>refactoring</i>. - Subir código al repositorio con tests fallidos.

NUnit
Atributos
A nivel de clase
<ul style="list-style-type: none"> - TestFixture: La clase contiene métodos de prueba.
A nivel de método
<ul style="list-style-type: none"> - Test: Es un método de prueba. - ExpectedException: El método va a producir una excepción del tipo indicado. - TestFixtureSetup: Inicializa el entorno de pruebas antes de ejecutarlas. - TestFixtureTearDown: Restablece el entorno tras la ejecución de las pruebas. - Setup: Inicializa el entorno antes de la ejecución de cada prueba. - TearDown: Restablece el entorno tras la ejecución de cada prueba.
A nivel de clase y método
<ul style="list-style-type: none"> - Category: Agrupa las pruebas por categorías, para ejecutarlas en bloque. - Explicit: Obliga a indicar explícitamente que deseamos pasar la prueba así marcada. - Ignore: Ignora la prueba durante la ejecución del plan de pruebas.
Plan de ejecución de pruebas
Asserts
Modelo clásico (versiones NUnit < 2.4)
Distintos métodos según el tipo de objeto y validación: <ul style="list-style-type: none"> - Assert.AreEqual (...) - Assert.AreNotEqual (...) - Assert.AreSame (...)
Modelo por restricciones (NUnit 2.4.x)
Un único prototipo de método, validando según la restricción (IConstraint) dada. <ul style="list-style-type: none"> - Assert.That(object actual, IConstraint constraint) Permite la creación de restricciones personalizadas que implementen el interfaz IConstraint.
Extensión de NUnit
NUnit permite la extensión de su funcionalidad mediante: <ul style="list-style-type: none"> - Asserts personalizados: Creando nuevos métodos de aserción. - NUnit Addins: actualmente sólo es posible hacerlo con extensiones sobre el Core.
El futuro
<ul style="list-style-type: none"> - NUnit está, en este momento, en su versión 2.4.7 - El lanzamiento de Nunit 3.0 se prevé inminente, con importantes cambios de arquitectura respecto a las 2.4.x - xUnit.Net es otro Framework de pruebas similar a lo que se espera de NUnit 3.0, y de sus mismos creadores.

Contacto: Miguel A. Chico (Mithdraug)
mchico@lobosoft.es

Glosario
<ul style="list-style-type: none"> - Test Case: validación de la correcta implementación de una funcionalidad. - Test Suite: conjunto de Test Cases, orientado a validar una serie de características determinadas del código. - Unit Test: Prueba unitaria. Código que implementa un Test Case. - Framework de prueba: Framework que permite "vincular" código de prueba al de nuestra aplicación y ejecutar las suites de prueba. - TDD: Test-Driven Development o Desarrollo Guiado por Pruebas. Ver definición en esta Cheat Sheet. - Refactoring: Reescritura de código, optimizándolo, eliminando repeticiones, mejorando su calidad y legibilidad, pero sin alterar sus interfaces ni su funcionalidad. - Rojo-Verde-Refactoring: Ciclo de pruebas en TDD. Se corresponde con el fallo en las pruebas, superación de las mismas, y etapa de revisión y mejora de código.

Referencias
Bibliografía
<ul style="list-style-type: none"> - Extreme Programming Explained, Kent Beck. - Test Driven Development: By Example, Kent Beck - Test Driven Development in Microsoft .NET, James W. Newkirk and Alexei A. Vorontsov. - Test-Driven Development: A Practical Guide, David Astels.

Enlaces de interés
Metodologías ágiles
<ul style="list-style-type: none"> - http://www.agilemanifesto.org/
Programación Extrema (XP)
<ul style="list-style-type: none"> - http://www.extremeprogramming.org
TDD
<ul style="list-style-type: none"> - http://www.testdriven.com - http://www.agiledata.org/essays/tdd.html - http://epf.eclipse.org/wikis/epfpralib/index.htm
Pruebas unitarias
<ul style="list-style-type: none"> - http://msdn2.microsoft.com/es-es/library/ms182515(VS.80).aspx - http://www.softdevarticles.com/modules/weblinks/viewcat.php?cid=34
Mocks
<ul style="list-style-type: none"> - http://martinfowler.com/articles/mocksArentStubs.html - http://www.mockobjects.com/
Herramientas
<ul style="list-style-type: none"> - http://www.nunit.org - http://www.codeplex.com/xunit
Miscelánea
<ul style="list-style-type: none"> - http://www.methodsandtools.com

