



Eusko Jaurlaritzaren Informatika Elkarte
Sociedad Informática del Gobierno Vasco

NUnit v2.2.7:

Manual de usuario

Fecha:

Referencia:

EJIE S.A.
Mediterráneo, 3
Tel. 945 01 73 00*
Fax. 945 01 73 01
01010 Vitoria-Gasteiz
Posta-kutxatila / Apartado: 809
01080 Vitoria-Gasteiz
www.ejie.es

Control de documentación

Título de documento: NUnit v2.2.7

Histórico de versiones

Código:

Versión:

Fecha:

Resumen de cambios:

Cambios producidos desde la última versión

Primera versión.

Control de difusión

Responsable: Ander Martínez

Aprobado por: Ander Martínez

Firma:

Fecha:

Distribución:

Referencias de archivo

Autor: Consultoría de áreas de conocimiento

Nombre archivo: NUnit. Manual de usuario v1.0doc

Localización:

Contenido

	Capítulo/sección	Página
1	Introducción	4
2	Conceptos básicos	4
3	Funciones elementales	4
3.1	Inicio de la aplicación	4
3.2	Funcionamiento de NUnit v2.2.7	5
3.2.1.	Nunit (línea de comandos)	6
3.2.2.	Nunit (GUI)	7
4	Funcionamiento	9
4.1	Atributos	9
4.2	Aserciones	11
5	Enlaces de referencia	11
6	Utilidad Práctica	12
7	Anexo 1: Prueba de Test	12
7.1	Resolución	12

1 Introducción

El presente documento describe cuales son las tareas que se pueden ejecutar en la explotación de la herramienta NUnit.

2 Conceptos básicos

Frecuentemente cuando desarrollamos siempre realizamos una gran cantidad de pruebas para verificar que nuestro código esté correcto. Estas pruebas normalmente tienen que ser realizadas varias veces y se ven afectadas por los cambios que introducimos conforme vamos desarrollando.

NUnit (<http://www.nunit.org/>) es la alternativa a JUnit para .Net. Permite realizar pruebas unitarias para cualquier lenguaje de .Net. (C#, VB.Net, C++.Net, J#...) Está desarrollada enteramente en C#, su licencia es zlib/libpng (basada en Open Source) y funciona con cualquier versión del framework. También está disponible una versión de NUnit compatible con Mono Project (<http://www.mono-project.com>).

Proporciona una GUI sencilla en donde podemos cargar las DLLs y los ejecutables .Net y ejecutar las pruebas unitarias que hayamos definido en ellos, aunque también es posible hacer lo mismo por línea de comandos.

La definición de pruebas unitarias dentro de los ensamblados es mediante clases destinadas para tal fin, con unos métodos que ejecutan un conjunto de prueba, es decir, una simulación del uso real de los métodos para comprobar su correcto funcionamiento (supongo que como con JUnit).

3 Funciones elementales

3.1 Inicio de la aplicación

Se accederá al menú Inicio → Programas → Nunit 2.2.7 → Nunit-Gui, o bien sobre el acceso directo existente en el escritorio para ejecutar el aplicativo.

3.2 Funcionamiento de NUnit v2.2.7

Test Driven Development trata justamente de desarrollar usando pruebas. Las pruebas son diseñadas por el desarrollador y por los expertos del dominio. Estas pruebas se pueden automatizar usando una herramienta como NUnit. Una ventaja de esta perspectiva es la posibilidad de lanzar las pruebas varias veces. Si hay cambios, normalmente los cambios requieren volver a probar. Con NUnit estas pruebas se llevarían a cabo rápidamente.

Existe una secuencia que hay que seguir a la hora de diseñar nuestras pruebas:

- Escribir una prueba
- Compilarla
- Ejecutarla y hacer que falle
- Ejecutarla bien
- Hacer "Refactoring" al código

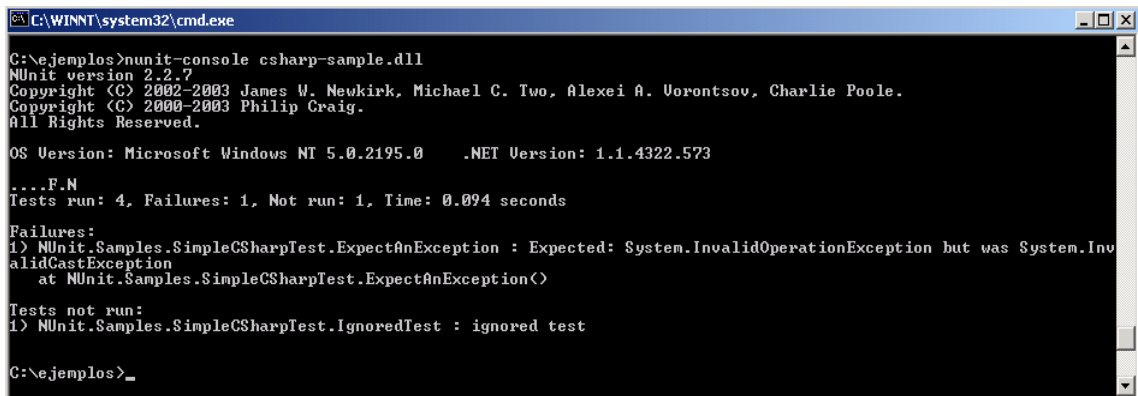
NUnit es un framework de pruebas para .NET y tiene dos características importantes:

- **NUnit** usa "atributos" para identificar las pruebas en .NET. Las pruebas son identificadas por el atributo [**TestFixture**] en la clase y el atributo [**Test**] para indicar las pruebas individuales.
- **NUnit** permite realizar pruebas en cualquier lenguaje de .NET. Se pueden escribir pruebas en Visual Basic .NET o C++. La interoperabilidad de lenguajes es una característica muy importante en .NET.

NUnit tiene dos caminos diferentes de controlar sus pruebas. A través de la línea de comandos ejecutando "nunit-console.exe", es el más rápido para lanzar, pero no es interactivo. O a través de ventanas con "nunit-gui.exe".

3.2.1. Nunit (línea de comandos)

El programa “nunit-console.exe” esta basado en texto y no proporciona una indicación roja/ amarilla/ verde indicativa de éxito o fracaso. En el siguiente ejemplo se muestra el código generado al ejecutar el programa “nunit-console.exe” utilizando el ensamblado “csharp-sample.dll”.



```
C:\WINNT\system32\cmd.exe
C:\ejemplos>nunit-console csharp-sample.dll
NUnit version 2.2.7
Copyright (C) 2002-2003 James W. Newkirk, Michael C. Two, Alexei A. Vorontsov, Charlie Poole.
Copyright (C) 2000-2003 Philip Craig.
All Rights Reserved.

OS Version: Microsoft Windows NT 5.0.2195.0   .NET Version: 1.1.4322.573

...F.N
Tests run: 4, Failures: 1, Not run: 1, Time: 0.094 seconds

Failures:
1) NUnit.Samples.SimpleCSharpTest.ExpectAnException : Expected: System.InvalidOperationException but was System.InvalidCastException
   at NUnit.Samples.SimpleCSharpTest.ExpectAnException()

Tests not run:
1) NUnit.Samples.SimpleCSharpTest.IgnoredTest : ignored test

C:\ejemplos>_
```

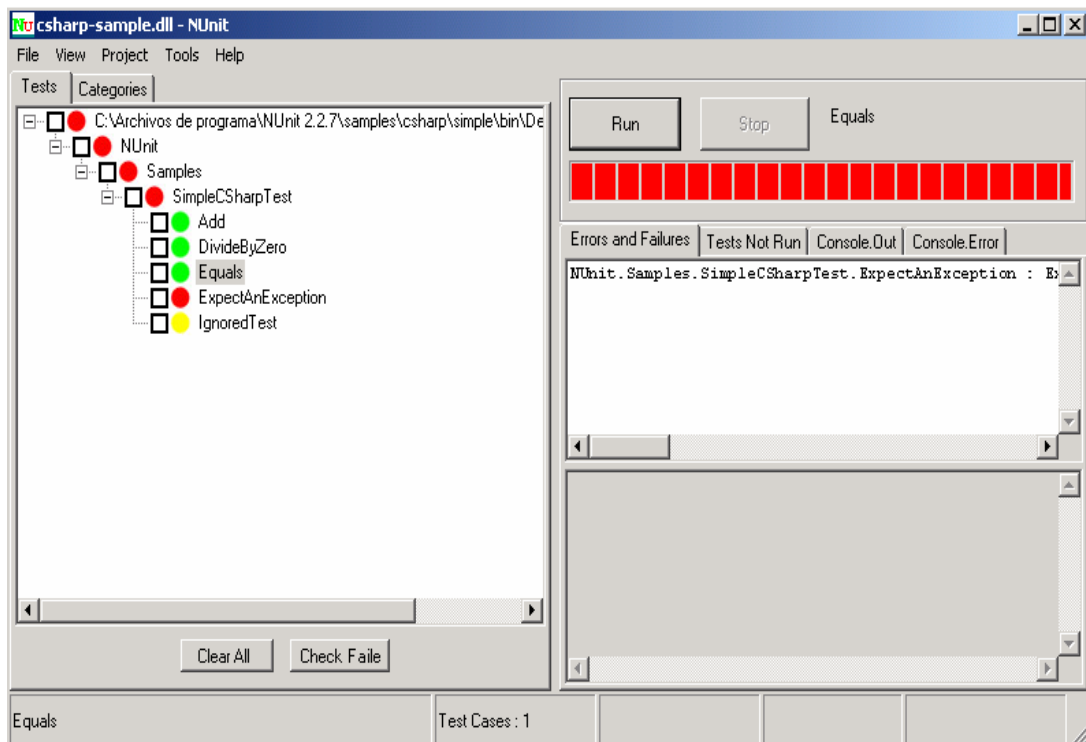
Esto es útil para la automatización de pruebas y la integración en otros sistemas. Esto automáticamente genera sus resultados en el formato de XML, permitiéndole producir informes o si no tratar los resultados.

Existe más información sobre la configuración, en la siguiente dirección:

<http://www.nunit.org/index.php?p=consoleCommandLine&r=2.2.7>

3.2.2. Nunit (GUI)

El programa nunit-gui.exe muestra las pruebas en una ventana parecida a un explorador y proporciona una indicación visual del éxito o el fracaso de las pruebas. Esto le permite con criterio selectivo controlar pruebas solas y recarga automáticamente. Lo siguiente es un ejemplo de nunit-gui utilizando el ensamblado "csharp-sample.dll".



Como se puede observar, las pruebas que no han sido controladas son marcadas con un círculo amarillo, mientras aquellas que fueron controladas satisfactoriamente son coloreadas verdes y las pruebas que han fallado se marcan en rojo.

Menú Principal

File

New Project: Cierra cualquier proyecto abierto, incitando al usuario para salvarlo si ha habido cambios y luego abre un diálogo para permitir seleccionar el nombre y la posición del nuevo proyecto.

Open: Cierra cualquier proyecto abierto, incitando al usuario para salvarlo y luego abre un diálogo para permitir seleccionar el nombre y la posición de un ensamblado.

Close: Cierra cualquier proyecto abierto, incitando al usuario para guardarlo si ha sido cambiado.

Save / Save As: Salva el proyecto actualmente abierto / abre un diálogo para permitir especificar el nombre y la posición a la que el proyecto debería ser guardado.

Reload: Recarga el proyecto actualmente cargado.

Recent files: Muestra una lista de archivos recientemente abiertos de los que el usuario puede seleccionar un para la abrir.

Exit: Cierra la herramienta.

View

Checkboxes: Si la opción esta chequeada, aparecen en cada línea una caja para poder chequear pruebas específicas.

Expand / Expand All: Amplía el nodo de árbol actualmente seleccionado o todo el arbol.

Collapse / Collapse All: Contrae el nodo de árbol actualmente seleccionado o todo el arbol.

Expand Fixtures: Amplía el nodo de árbol actualmente seleccionado a nivel de función.

Collapse Fixtures: Contrae el nodo de árbol actualmente seleccionado a nivel de función.

Properties: Muestra el Diálogo de Propiedades para la prueba actualmente seleccionada.

Project

Configurations: Desde esta opción podemos generar/ eliminar o modificar configuraciones para nuestro ensamblado.

Add Assembly: Muestra un diálogo para permitir añadir un ensamblado a la configuración activa del proyecto actualmente abierto.

Edit: Muestra un diálogo que permite modificar la configuración del proyecto actual.

Tools

Salve Results as XML: Abre un Diálogo para salvar los resultados de prueba como un archivo XML.





Exception Details: Se muestra en detalle la información sobre la última excepción.

Options: Abre un Diálogo con las opciones de la herramienta.

Loaded Test frameworks: Especifica la versión del Nunit cargada.

Menú Contextual

El menú contextual se muestra cuando una de las líneas es pulsada con el botón derecho del ratón.

-  **Run:** ejecuta el test de la línea seleccionada
-  **Expand:** Amplía el nodo seleccionado de prueba.
-  **Collapse:** Contrae el nodo seleccionado de prueba.
-  **Properties:** Muestra las Propiedades para el nodo seleccionado de prueba.

4 Funcionamiento

NUnit basa su funcionamiento en dos aspectos, el primero es la utilización de atributos personalizados. Estos atributos le indican al framework de NUnit que debe hacer con determinado método o clase, es decir, estos atributos le indican a NUnit como interpretar y ejecutar las pruebas implementadas en el método o clase.

El segundo son las aserciones, que no son más que métodos del framework de NUnit utilizados para comprobar y comparar valores.

4.1 Atributos

La versión 1x de NUnit utilizaba las convenciones de nombrado del framework de .NET, pero desde la versión 2 en adelante NUnit usa atributos personalizados. Dado que el framework de NUnit no deriva de otra clase o framework, el desarrollador puede elegir el nombre de la prueba a su antojo.

TestFixture

Este atributo se utiliza para indicar que una clase contiene métodos de prueba. En versiones anteriores para poder utilizar este atributo se debía extender (heredar de) la clase TestCase, a partir de la versión 2 esto no es necesario, situación que hace mucho más flexible el uso del atributo.

Existen algunas restricciones como por ejemplo que la clase debe tener un constructor por defecto y debe ser pública para que el framework NUnit pueda accederla.

Test

Se utiliza para marcar un método como método de prueba, éste no debe tener parámetros de entrada y debe retornar void (en el caso de visual Basic ser un sub) así mismo la clase que lo alberga debe tener marcado el atributo TextFixture.

TestFixtureSetUp / TestFixtureTearDown

El primero de estos atributos se encarga de crear el ambiente de pruebas antes de ejecutarlas y el segundo se encarga de restaurar el ambiente después de que las pruebas han sido ejecutadas. Van de la mano con los atributos Setup y TearDown, Setup es llamado antes de que se ejecute cualquier prueba. TearDown es llamado después de que una prueba se ejecute. Una clase marcada con TestFixture solo puede tener un método marcado con TestFixtureSetUp y un solo método marcado con TestFixtureTearDown, si existe más de un método marcado con estos atributos el proyecto compilara pero no se ejecutaran las pruebas. El orden de ejecución es el siguiente:

ExpectedException

Este atributo como su nombre lo sugiere, tiene como funcionalidad indicar que la ejecución de un método prueba va a lanzar una excepción, el atributo tiene como parámetro el tipo de excepción que se espera que lance el método, el framework ejecuta la prueba y si se genera una excepción del tipo especificado entonces la prueba es exitosa, si por el contrario se genera una excepción de tipo diferente al específico la prueba no lo es. Esto es cierto aun cuando la excepción lanzada herede de la excepción esperada, es decir la excepción debe ser exactamente la especificada en el parámetro del atributo

Suite

El atributo suite es utilizado para definir subconjuntos de pruebas de acuerdo a las preferencias del usuario, sin embargo este atributo ha sido reemplazado desde la versión 2 debido al nuevo mecanismo dinámico de ejecución de pruebas del framework (en modo gráfico se puede seleccionar que pruebas se desean ejecutar utilizando el atributo category, además se pueden agrupar dentro de una estructura marcada como TestFixture). En general es soportada para proveer compatibilidad hacia atrás.

En las nuevas versiones del producto no se puede correr suites.

Category

El atributo category provee una alternativa a las suites para trabajar con grupos de pruebas. Cualquiera, ya sea casos de prueba individuales o Fixtures, pueden ser identificadas como pertenecientes a una categoría de pruebas en particular. Ya sea en modo gráfico o en modo consola se puede especificar que categorías se excluyen o incluyen en la ejecución. Cuando se utilizan categorías, solo las pruebas de la categoría seleccionada son ejecutadas. Las pruebas incluidas en las categorías que se excluyen de la ejecución no son reportadas. Para excluir o incluir determinada categoría en el modo consola incluya el parámetro /exclude o /include seguido del nombre de la categoría. En el modo gráfico existe una pestaña denominada categorías.

Es importante anotar que esta funcionalidad solo se encuentra presente en la versión 2.2 del producto. Este atributo puede utilizarse junto con TestFixture o Test

Explicit

Este atributo ocasiona que una prueba o un Fixture sean ignorados por el programa y no sean

ejecutados a menos que sea especificado lo contrario. La prueba o Fixture será ejecutada si se selecciona directamente en la interfaz grafica de usuario, si su nombre es especificado en la línea de comandos, como el Fixture a ejecutar o si es incluido en una categoría.

Si una prueba o un Fixture con el atributo Explicit es encontrado durante la ejecución, el programa la ignora. El icono de la prueba o el Fixtures se coloca amarillo y es colocado en el reporte de pruebas no ejecutadas.

Esto es útil cuando se desea, por ejemplo ejecutar todas las pruebas menos una o unas. Sin embargo las pruebas marcadas con Explicit pueden ser ejecutadas si explícitamente se le indica al programa que lo haga y esto se hace ya seleccionando la prueba y oprimiendo el botón run en el entorno gráfico o escribiéndola en la ventana de comandos.

Ignore

Utilizado para indicar que se debe ignorar determinada prueba o Fixture, El programa ve el atributo y no ejecuta la prueba o las pruebas, el icono se coloca amarillo y la prueba es reportada como no ejecutada. Esto es útil para inhabilitar pruebas temporalmente, es decir, este atributo es útil para marcar una prueba o Fixture, si no se desea ejecutarla momentáneamente, en vez de comentar el método o la clase, ya que de todas maneras va a ser compilada junto con el resto del código pero no será tenida en cuenta a la hora de ejecutar las pruebas.

4.2 Aserciones

Las aserciones son métodos estáticos que la clase assert provee para realizar comparaciones y condiciones de prueba.

Equality Asserts	Assert.AreEqual
	Assert.AreNotEqual
Identity Asserts	Assert.AreSame
	Assert.AreNotSame
	Assert.Contains
Comparison Asserts	Assert.Greater
	Assert.Less
Type Asserts	Assert.IsInstanceOfType
	Assert.IsNotInstanceOfType
	Assert.IsAssignableFrom
	Assert.IsNotAssignableFrom
Condition tests	Assert.IsTrue
	Assert.IsFalse
	Assert.IsNull
	Assert.IsNotNull
	Assert.IsNaN
	Assert.IsEmpty
Utility methods.	Assert.Fail
	Assert.Ignore

5 Enlaces de referencia

Sitio Web NUNIT

<http://www.nunit.org>

Ejemplo

<http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art150.asp>

Foros sobre NUnit

http://sourceforge.net/forum/?group_id=10749

6 Utilidad Práctica

Nunit, se trata de una herramienta que en la fase de pruebas proporcionara un modo de validar la correcta ejecución de las funcionalidades implementadas en un proyecto.

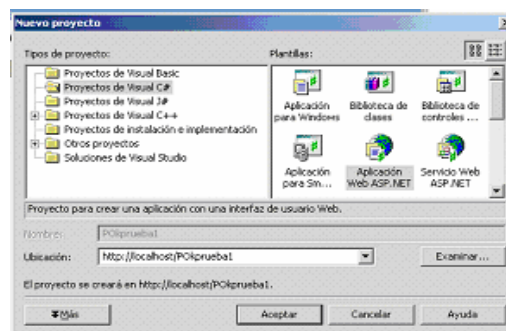
Así mismo, permitirá validar el correcto proceso de paso entre entornos, ya que si los procesos funcionan correctamente en un entorno deberían hacerlo en el siguiente, así como que proporcionara valiosa información en el proceso de realización de pruebas de regresión.

7 Anexo 1: Prueba de Test

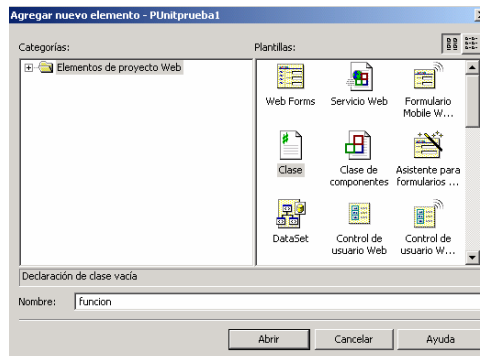
El ejercicio consiste en la realización del proceso de testeo de una clase que realice la suma de dos números.

7.1 Resolución

- En el menú **Archivo** de Visual Studio.NET, seleccione **Nuevo** y, a continuación, **Proyecto de visual C#**. Entre las plantillas seleccionamos la de **Aplicación Web ASP.NET**, insertamos como nombre del proyecto (*PUnitprueba1*)



- Una vez generado el proyecto añadimos dos clases, una primera clase que la denominaremos **funcion** donde añadiremos una función de suma y una segunda clase **funcionTest** que contiene los procedimientos que testan la función de suma. Para añadir las clases seleccionamos con el botón derecho del ratón la opción de **Agregar** → **Agregar nuevo elemento**, seleccionamos el tipo **Clase** y le insertamos el nombre **funcion**, de la misma forma generamos **funcionTest**.



- Añadimos en la clase **funcion** una función **fSuma()** que devuelve la suma de dos números:

```
Static int fSuma(int valA, int valB) {return valA + valB;}
```

- Por cada clase que tenemos vamos a hacer una clase que teste las funciones que hemos generado. Por ejemplo, tenemos la función **fSuma()** que devuelve la suma de dos números, dicha función esta en una clase **funcion**. Para generar la función que testa nuestra función de suma, generamos la función **fSumaTest** dentro de la clase **funcionTest**, para considerarla una función que testa debemos de añadir la etiqueta **[Test]** al inicio de la función. Además, para que se considere una clase de testeo debemos añadir **[TestFixture]** al inicio de la clase, tal y como se muestra en el siguiente código. Todas estas etiquetas se pertenecen a la clase "NUnit.Framework" por lo que debemos de importarla.

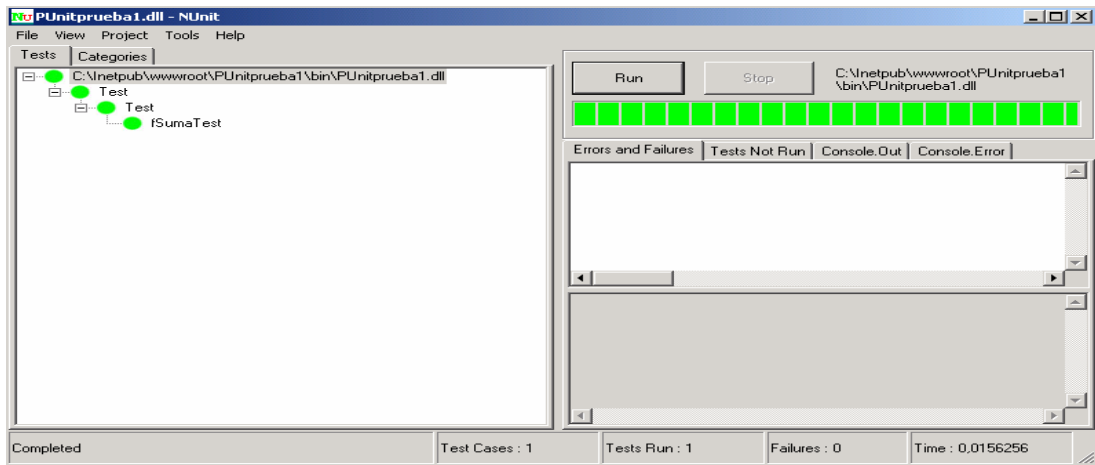
```
using System;
using NUnit.Framework;

namespace funcionTest
{
    [TestFixture] public class funcionTest
    {
        [Test] public void fSumaTest()
        {
        }
    }
}
```

¿Qué insertamos en la función **fSumaTest**? Dentro de la función hacemos diferentes llamadas a la función de **fSuma**, pasándole diferentes combinaciones de los parámetros. Por ejemplo, le pasándole 5 y 6 sabemos que el resultado es 11. Para testar este proceso utilizamos una de las funciones importadas de Nunit, en concreto utilizamos **Assert.AreEqual**, que nos valida que el resultado de la suma sea igual a 11.

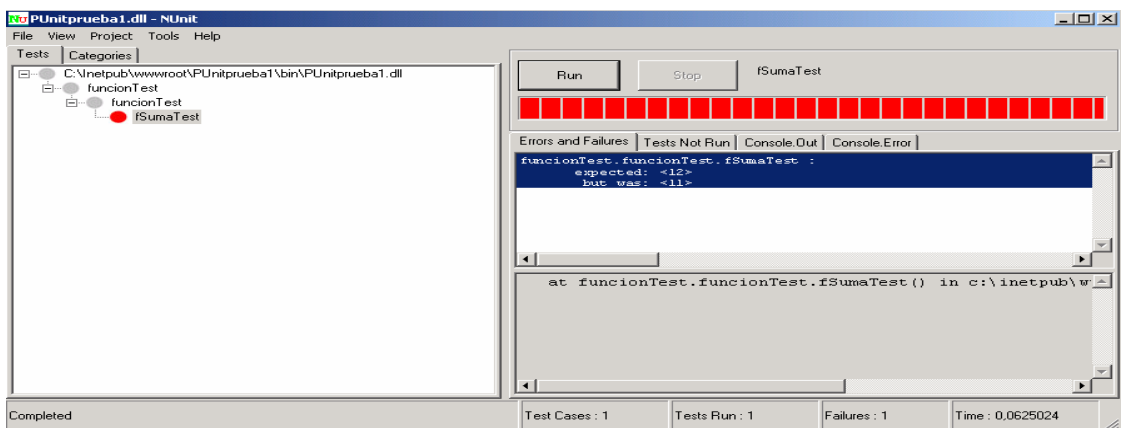
```
Assert.AreEqual(11, PUnitprueba1.funcion.fSuma(6, 5));
```

En este caso el ambos valores son iguales por lo que al testar el ensamblado el color que se nos muestra es el verde. Para verificar que todo es correcto abrimos el ensamblado "PUnitprueba1.dll" con Nunit y después de pulsar Run el resultado es el siguiente:



Si añadimos una segunda función donde el resultado no es correcto, el color que se nos muestra es rojo, ya que el resultado de la suma no es el esperado.

```
Assert.AreEqual(12, PUnitprueba1.funcion.fSuma(6, 5));
```



Al igual que la función **AreEqual** existen otras funciones: