Universitat d'Alacant
Universidad de Alicante

Pruebas Unitarias

Características

- Sin necesidad de intervención manual
- Tienen que poder repetirse tantas veces como uno quiera
- Cubrir casi la totalidad del código
- Ejecutarse independientemente del estado del entorno
- La ejecución de una prueba no puede afectar la ejecución de otra
- Simular las relaciones entre módulos para evitar dependencias
- Conocer claramente cuál es el objetivo

Beneficios

- La vida de desarrollador será mucho más fácil
- Fomentan el cambio y la refactorización
- Se reducen drásticamente los problemas y tiempos dedicados a la integración
- Nos ayudan a entender mejor el código
- Podemos probar o depurar un módulo sin necesidad de disponer del sistema completo

Mitos

- Escribir pruebas unitarias es escribir el doble de código
- Hace que los tiempos de desarrollo se incrementen

Prueba de métodos

```
private int sumar(int a, int b)
    {
        return (a + b);
    }
```

Prueba de métodos

```
/// <summary>
///Una prueba de sumar
///</summary>
[TestMethod()]
public void sumarTest() {
    Program target = new Program(); // TODO: Inicializar en un valor adecuado
    int a = 0; // TODO: Inicializar en un valor adecuado
    int b = 0; // TODO: Inicializar en un valor adecuado
    int expected = 0; // TODO: Inicializar en un valor adecuado
    int actual;
    actual = target.sumar(a, b);
    Assert.AreEqual(expected, actual);
    Assert.Inconclusive("Compruebe la exactitud de este método de prueba.");
 }
```

Prueba de métodos

```
/// <summary>
///Una prueba de sumar
///</summary>
[TestMethod()]
public void sumarTest() {
    Program target = new Program(); // TODO: Inicializar en un valor adecuado
    int a = 1; // TODO: Inicializar en un valor adecuado
    int b = 2; // TODO: Inicializar en un valor adecuado
    int expected = 3; // TODO: Inicializar en un valor adecuado
    int actual;
    actual = target.sumar(a, b);
    Assert.AreEqual(expected, actual);
}
```

Prueba de métodos (excepciones)

```
public int sumar(int a, int b)
{
if (a < 0 || b < 0)
    throw new ArgumentException();
    return (a + b);
}
```

Prueba de métodos (excepciones)

```
/// <summary>
///Otra prueba de sumar
///</summary>
[TestMethod()]
[ExpectedException(typeof(System.ArgumentException))]
public void sumarTest2()
{
    Program target = new Program();
    int a = -1;
    int b = -2;
    int actual;
    actual = target.sumar(a, b);
}
```

Métodos de inicialización

- ClassInitialize
- TestInitialize
- ClassCleanup
- TestCleanup

Pruebas unitarias en web

```
[TestMethod()]
[HostType("ASP.NET")]
[AspNetDevelopmentServerHost("%PathToWebRoot%\\WebSite8", "/WebSite8")]
[UrlToTest("http://localhost/WebSite8")]
public void sumarTest()
{
 Class1_Accessor target = new Class1_Accessor();

 int a = 1;
 int b = 2;
 int expected = 3;
 int actual;
 actual = target.sumar(a, b);
 Assert.AreEqual(expected, actual);
}
```

Orígenes de datos

```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
     "|DataDirectory|\\datosSuma.csv", "datosSuma#csv",
     DataAccessMethod.Sequential), DeploymentItem("Test1\\datosSuma.csv"),
     TestMethod()]
public void sumarTest()
{
    Program target = new Program();
    int a = (int)this.TestContext.DataRow[0];
    int b = (int)this.TestContext.DataRow[1];
    int expected = (int)this.TestContext.DataRow[2];
    int actual;
    actual = target.sumar(a, b);
    Assert.AreEqual(expected, actual);
}
```