



BASTA!
.NET, WINDOWS, VISUAL STUDIO

Malte Clasen

Grundlagen automatisierter Tests

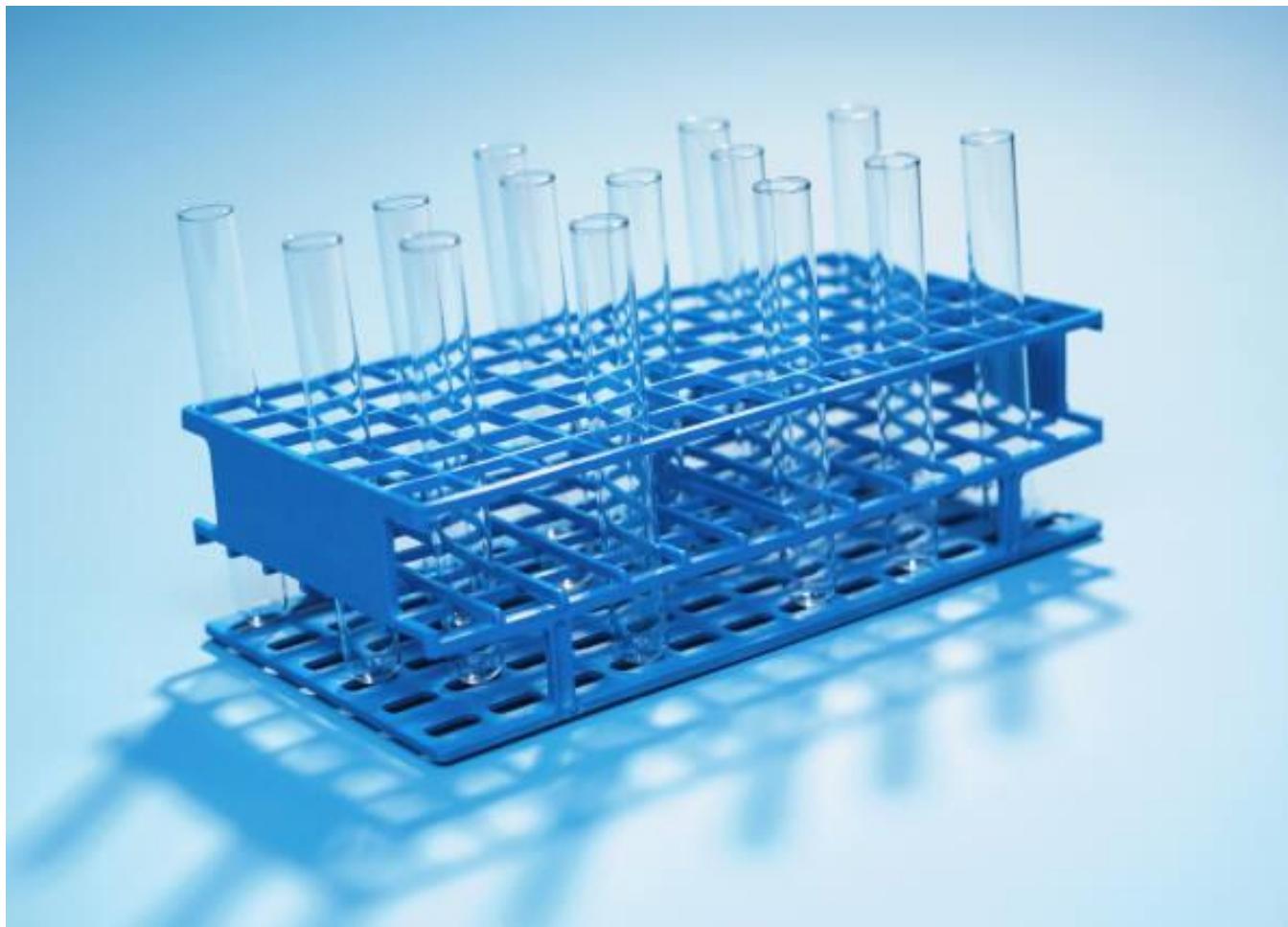
Definition



automatisch



Kontext



Parameter



Messen



Es funktioniert



Es ist fertig



Es ist nutzbar



So funktioniert es



Es funktioniert immer noch



Flow

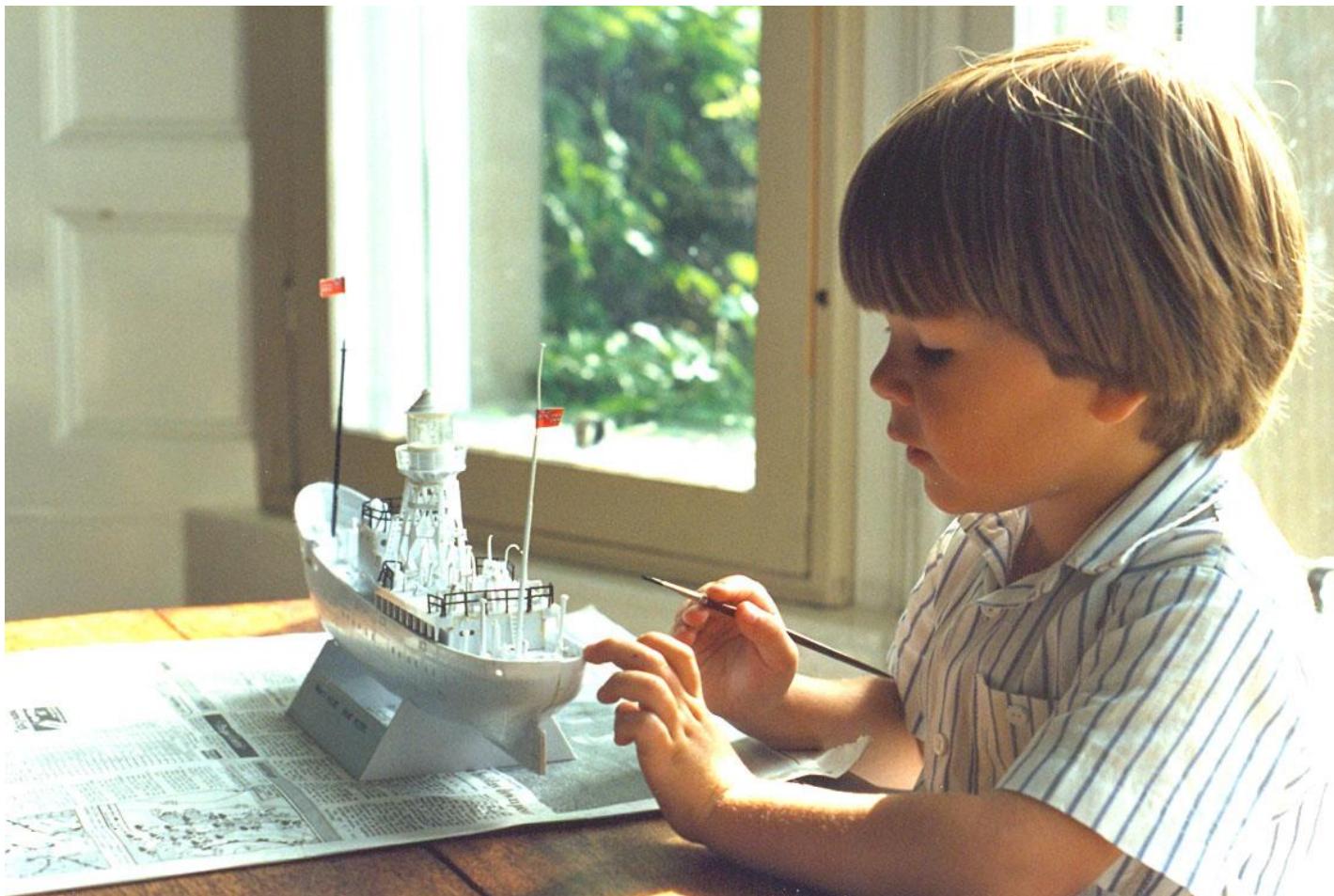
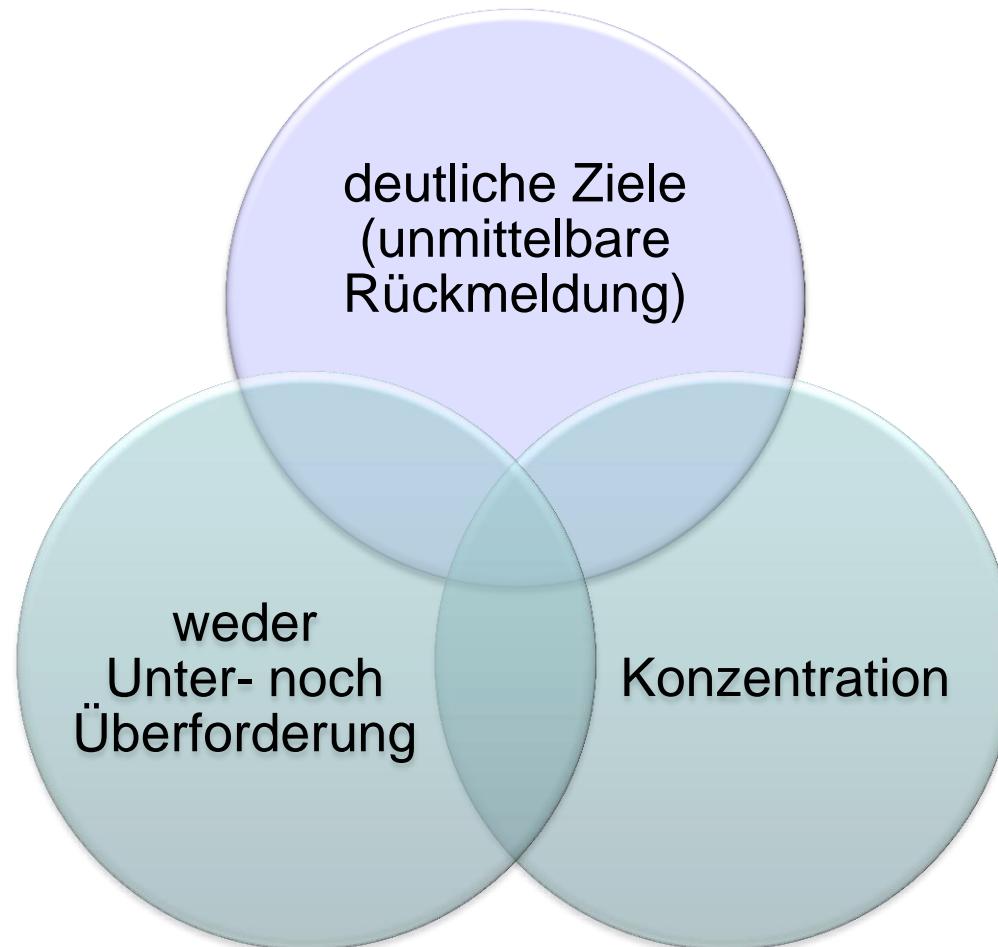


Foto: Charles J Sharp; licensed under the Creative Commons Attribution 2.5 Generic license; <http://commons.wikimedia.org/wiki/File:South-Goodwin.jpg>

Flow



einfacher Test

```
[TestMethod]  
public void OneEqualsOne()  
{  
    Assert.AreEqual(1,1);  
}
```

Given-When-Then

```
[TestMethod]  
public void Addition()  
{  
    var a = 1;  
    var b = 2;  
  
    var result = a + b;  
  
    Assert.AreEqual(result, 3);  
}
```

MsTest

[**TestClass**]

```
public class MsTest
```

```
{
```

[**TestMethod**]

```
public void OneEqualsOne()
```

```
{
```

```
    Assert.AreEqual(1,1);
```

```
}
```

```
}
```

nUnit

[TestFixture]

```
public class NUnitTest
```

```
{
```

[Test]

```
public void OneEqualsOne()
```

```
{
```

```
    Assert.AreEqual(1,1);
```

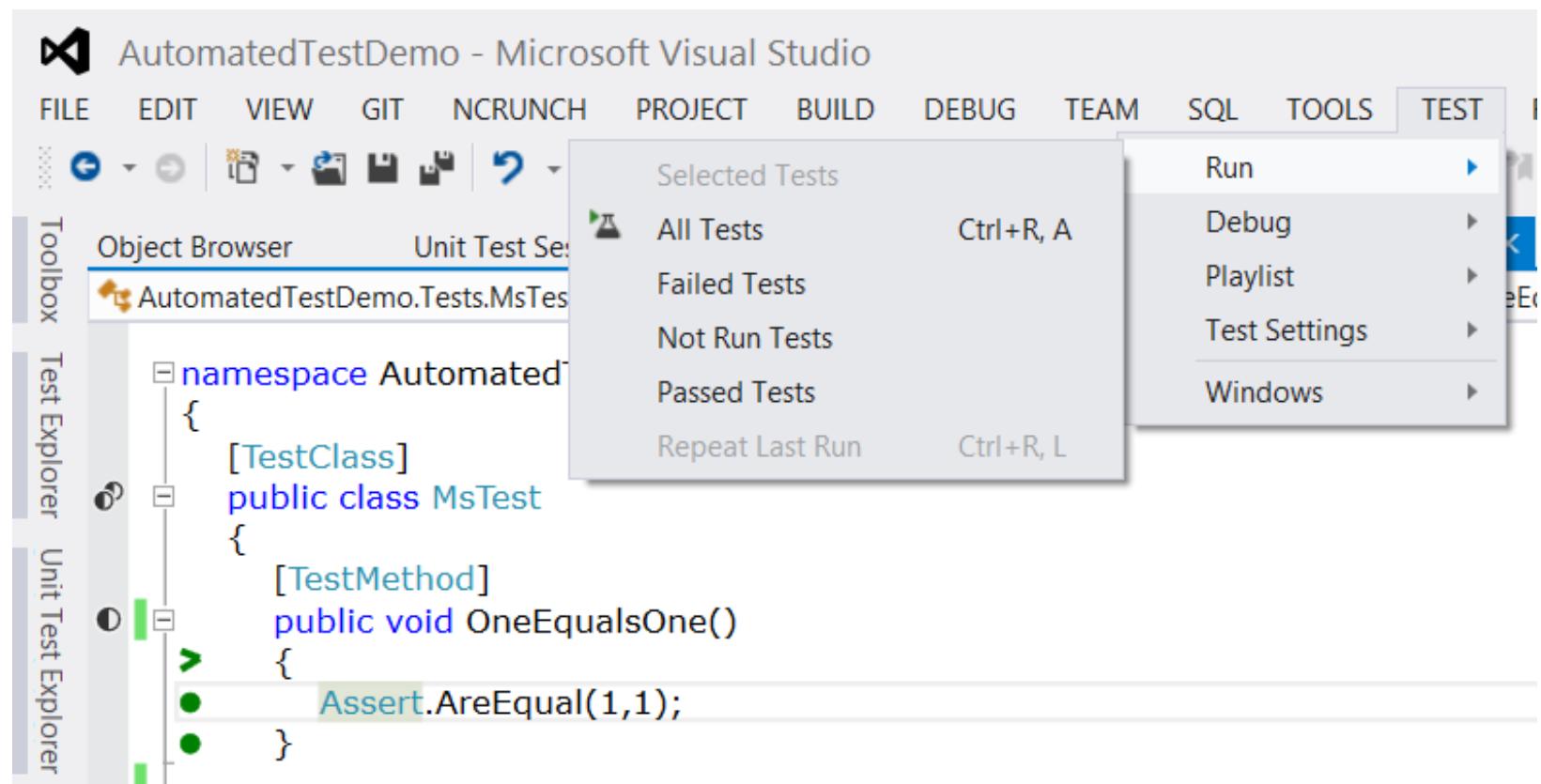
```
}
```

```
}
```

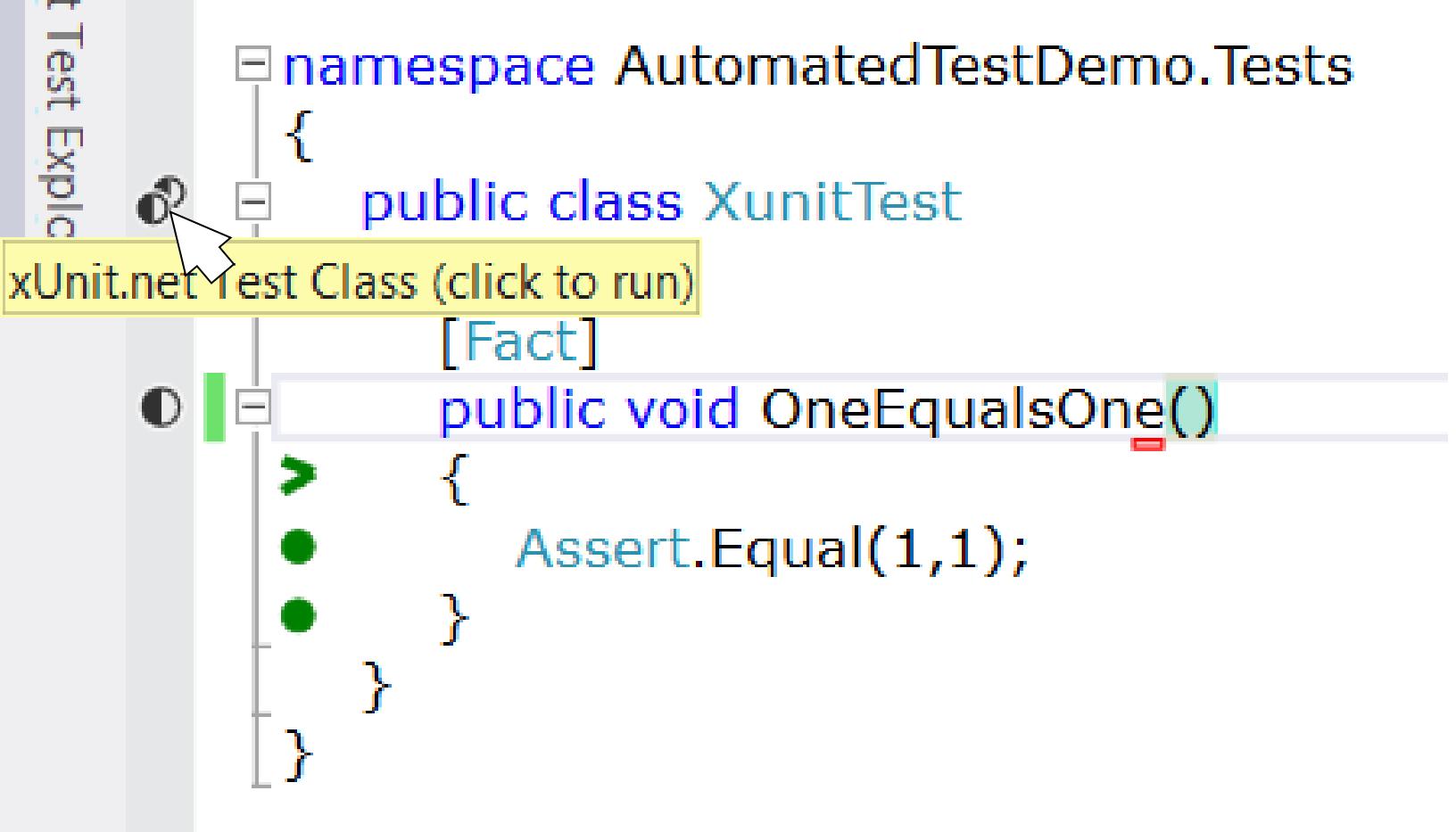
xUnit.net

```
public class XunitTest
{
    [Fact]
    public void OneEqualsOne()
    {
        Assert.Equal(1,1);
    }
}
```

Visual Studio Runner



ReSharper



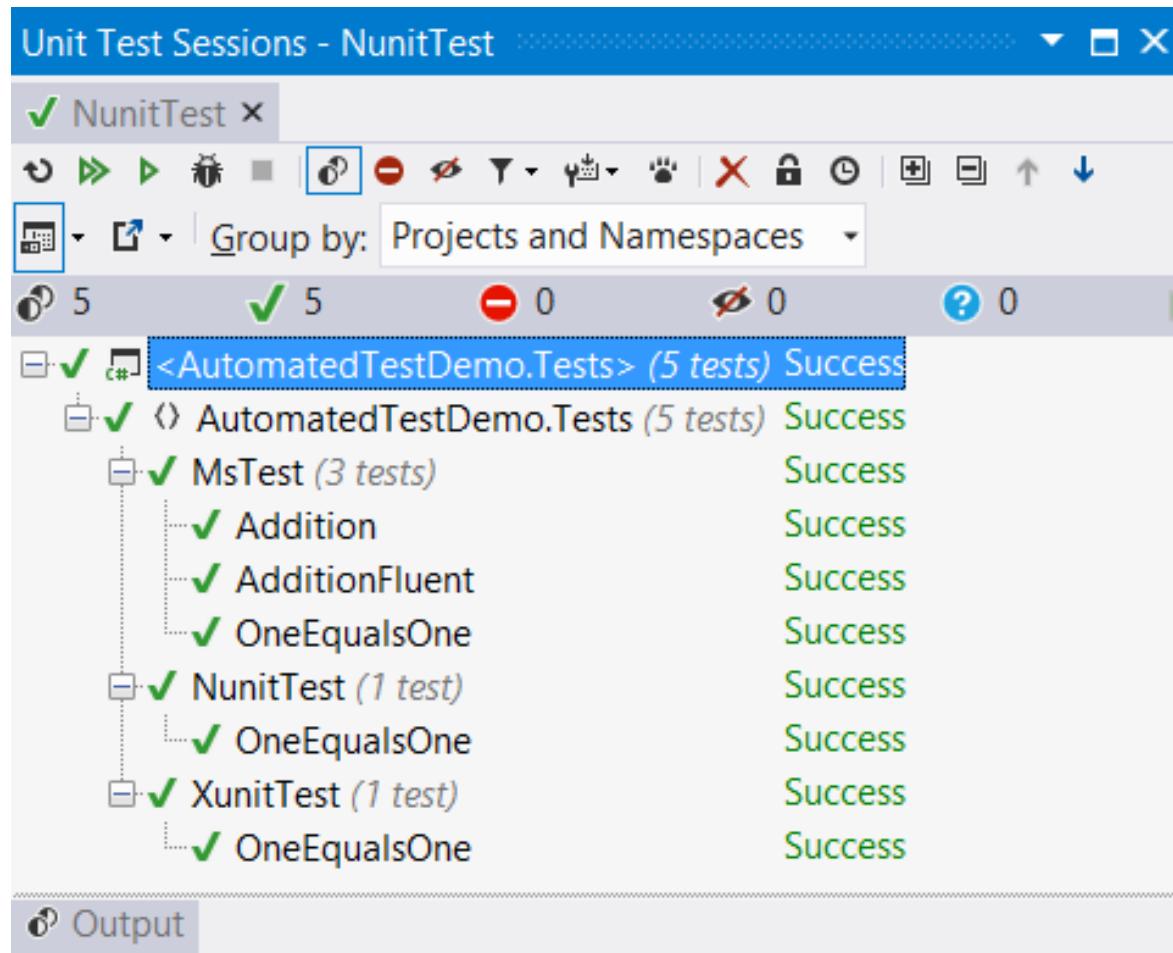
The screenshot shows the ReSharper Test Explorer interface. On the left, there's a tree view with a node labeled "xUnit.net Test Class (click to run)". A mouse cursor is hovering over this node. To the right, the code editor displays the following C# code:

```
namespace AutomatedTestDemo.Tests
{
    public class XunitTest
    {
        [Fact]
        public void OneEqualsOne()
        {
            Assert.Equal(1,1);
        }
    }
}
```

The code editor uses color-coded syntax highlighting: blue for namespaces and classes, red for errors, green for successful tests, and orange for pending tests. The "OneEqualsOne" method is highlighted with a red error squiggle under the closing brace, indicating a syntax error.

ReSharper Test Session

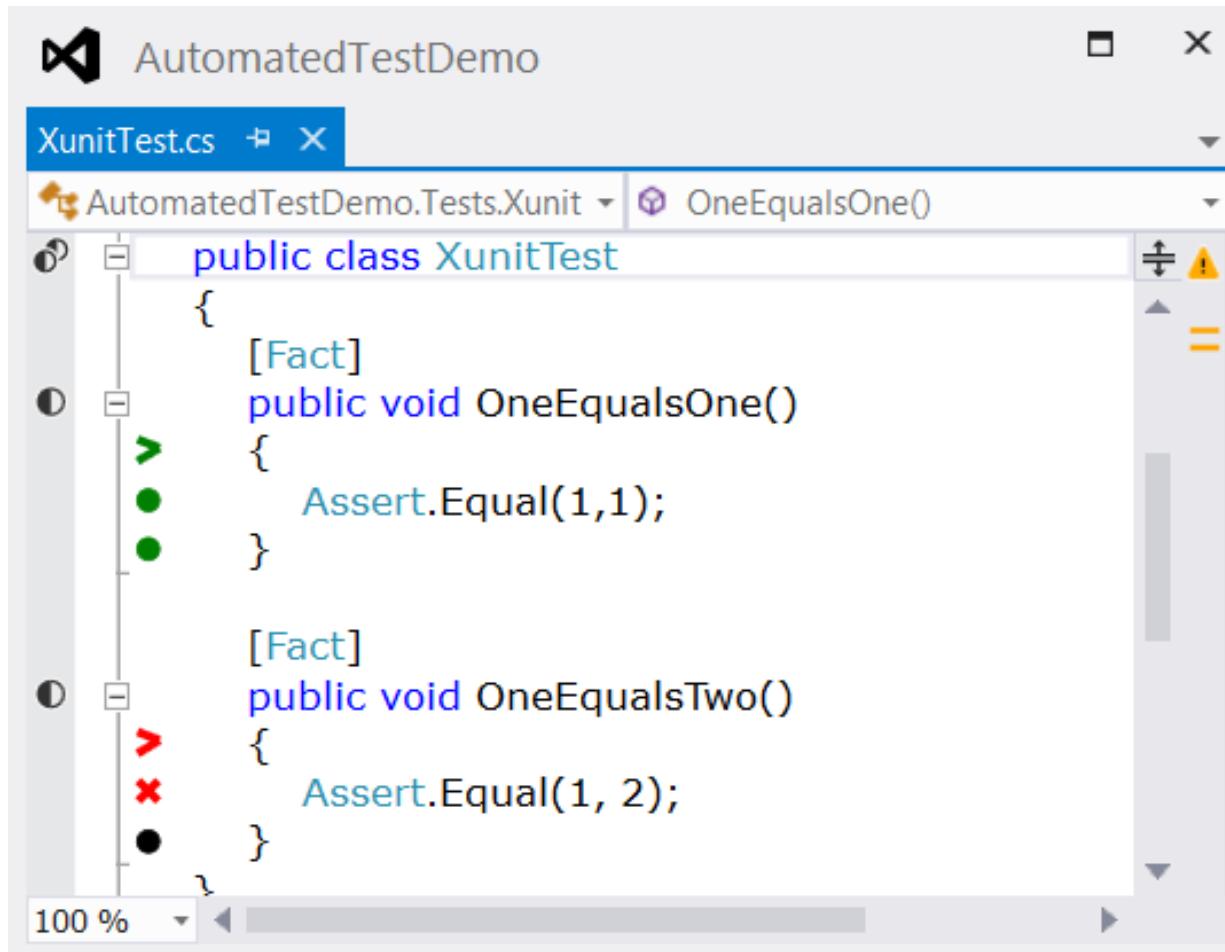
Unit Test Sessions - NunitTest



The screenshot shows the ReSharper Unit Test Sessions tool window titled "Unit Test Sessions - NunitTest". The main interface displays a summary of test results: 5 tests total, all successful (green checkmarks). Below this, a tree view shows the test structure:

- <AutomatedTestDemo.Tests> (5 tests) Success
 - AutomatedTestDemo.Tests (5 tests) Success
 - MsTest (3 tests) Success
 - Addition Success
 - AdditionFluent Success
 - OneEqualsOne Success
 - NunitTest (1 test) Success
 - OneEqualsOne Success
 - XunitTest (1 test) Success
 - OneEqualsOne Success

NCrunch



The screenshot shows the NCrunch interface integrated into a Visual Studio-like environment. The window title is "AutomatedTestDemo". The tab bar shows "XunitTest.cs" is the active file. The status bar at the bottom indicates "100 %".

The code editor displays the following XUnit test class:

```
public class XunitTest
{
    [Fact]
    public void OneEqualsOne()
    {
        Assert.Equal(1,1);
    }

    [Fact]
    public void OneEqualsTwo()
    {
        Assert.Equal(1, 2);
    }
}
```

The test results are shown on the left side of the editor:

- The first test, "OneEqualsOne()", has a green checkmark icon and is marked as successful.
- The second test, "OneEqualsTwo()", has a red X icon and is marked as failed.

FluentAssertions

```
[TestMethod]
```

```
public void AdditionFluent()
```

```
{
```

```
    var a = 1;
```

```
    var b = 2;
```

```
    var result = a + b;
```

```
    result.Should().Be(3);
```

```
}
```

Test-Driven-Development

Write
failing
unit test

Make
unit test
pass



Klasse nutzen

[Fact]

```
public void StartsWithZero()
{
    var calculator = new Calculator();
}
```

Klasse anlegen

```
public class Calculator  
{  
}
```

Eigenschaft nutzen

[Fact]

```
public void StartsWithZero()
{
    var calculator = new Calculator();

    calculator.Value.Should().Be(0);
}
```

Eigenschaft anlegen

```
public class Calculator  
{  
    public double Value { get; set; }  
}
```

Funktion nutzen

[Fact]

```
public void AddsValues()
{
    var calculator = new Calculator { Value = 4 };

    calculator.Add(2);

    calculator.Value.Should().Be(6);
}
```

Funktion anlegen

```
public class Calculator
{
    public double Value { get; set; }

    public void Add(double value)
    {
        Value += value;
    }
}
```

Isolation



Code-Isolation



Assembly



GAC



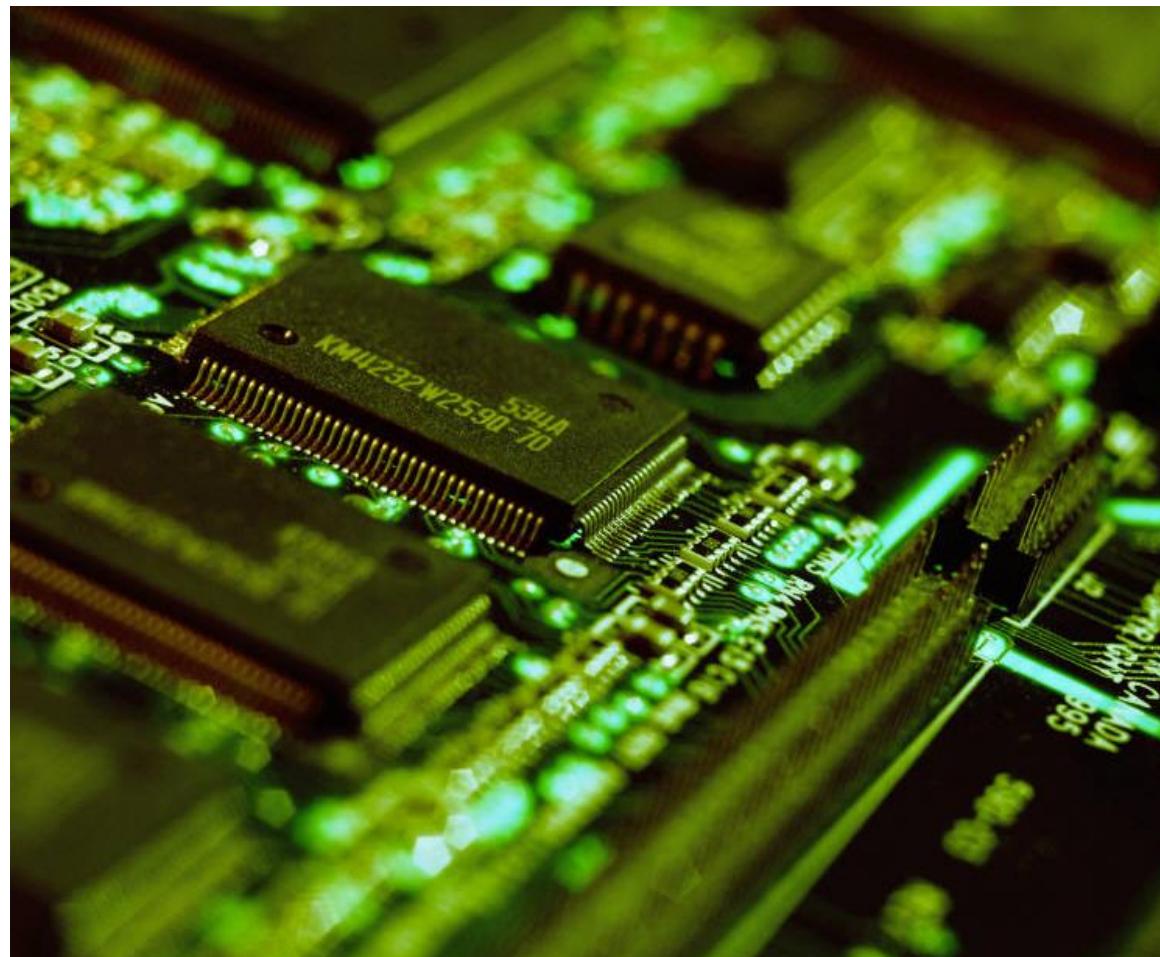
Frameworks



Daten-Isolation



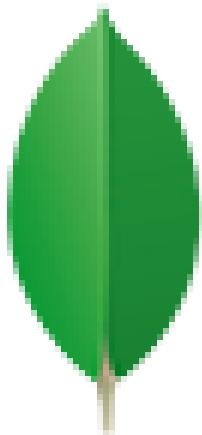
Arbeitsspeicher



Datenbank mit Transaktion



Datenbank ohne Transaktion



mongoDB

Datenbank-ähnliches



weitere Ressourcen



Continuous Integration



Unit Test



Integration Test



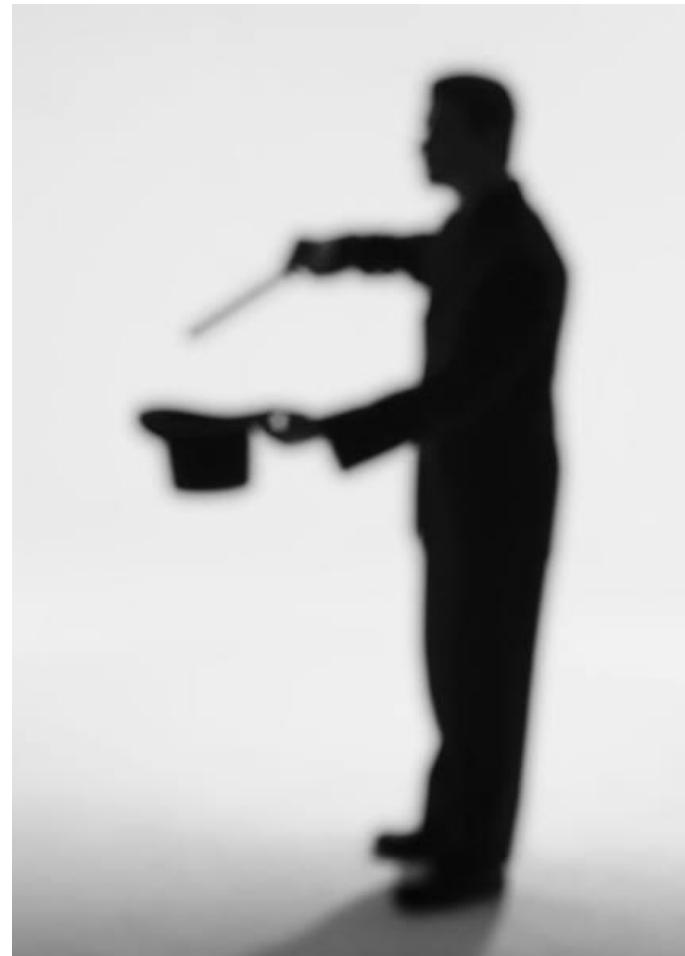
Acceptance Test



Abhängigkeiten



Service Locator



Dependency Injection



Platzhalter



Moq



AutoFixture



Zusammenfassung

- Funktion, Fertigstellung, Qualität, Lehre
 - Unit, Integration, Acceptance
 - Isolation
-
- Folien, Code
<http://malteclasen.de/blog>
 - Beratung, Training
info@malteclasen.de