
10 – Automatisiertes Testen

Agenda

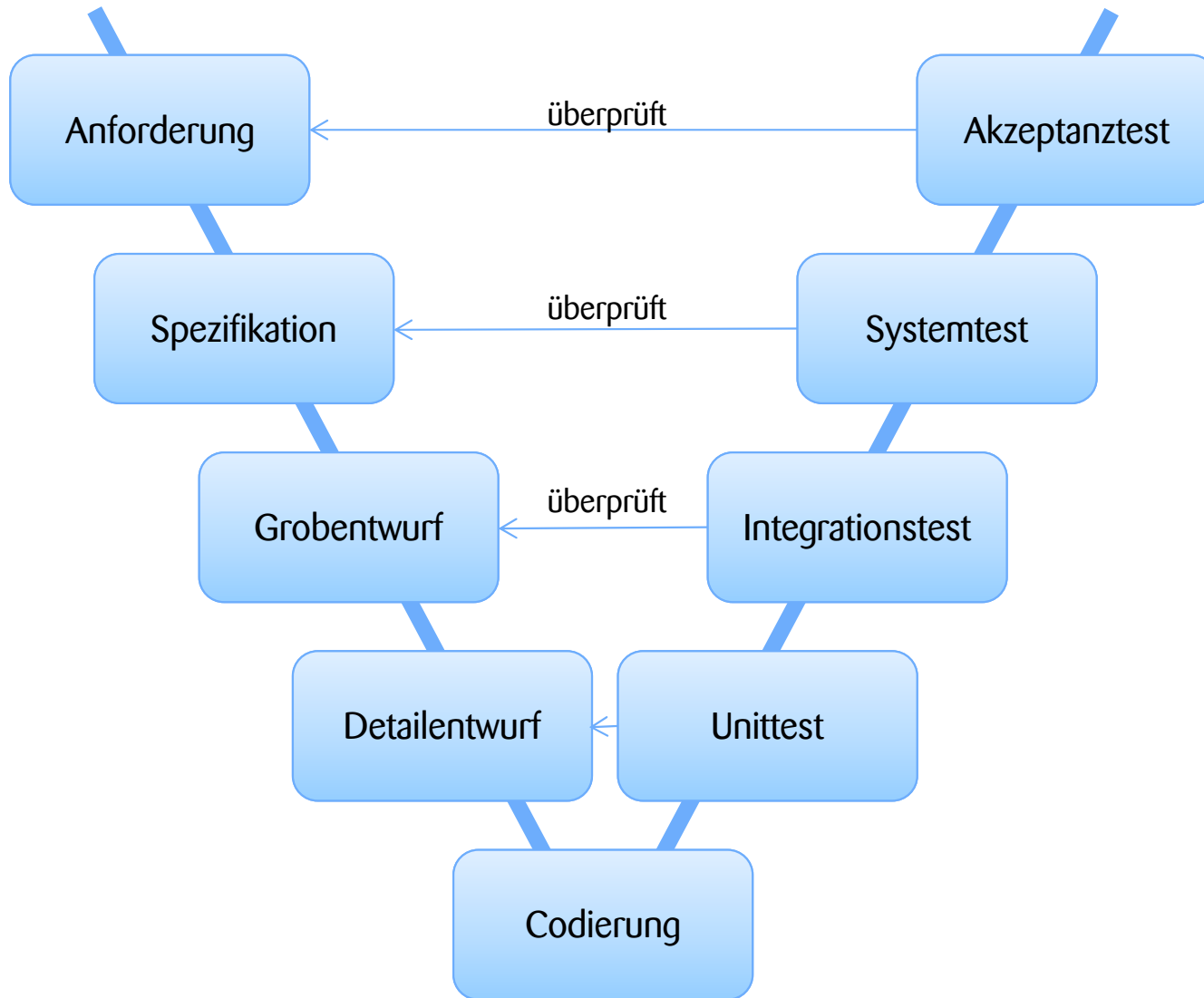
■ V-Modell und Teststufen

- Akzeptanztest
- Systemtest
- Integrationstest
- Unittest

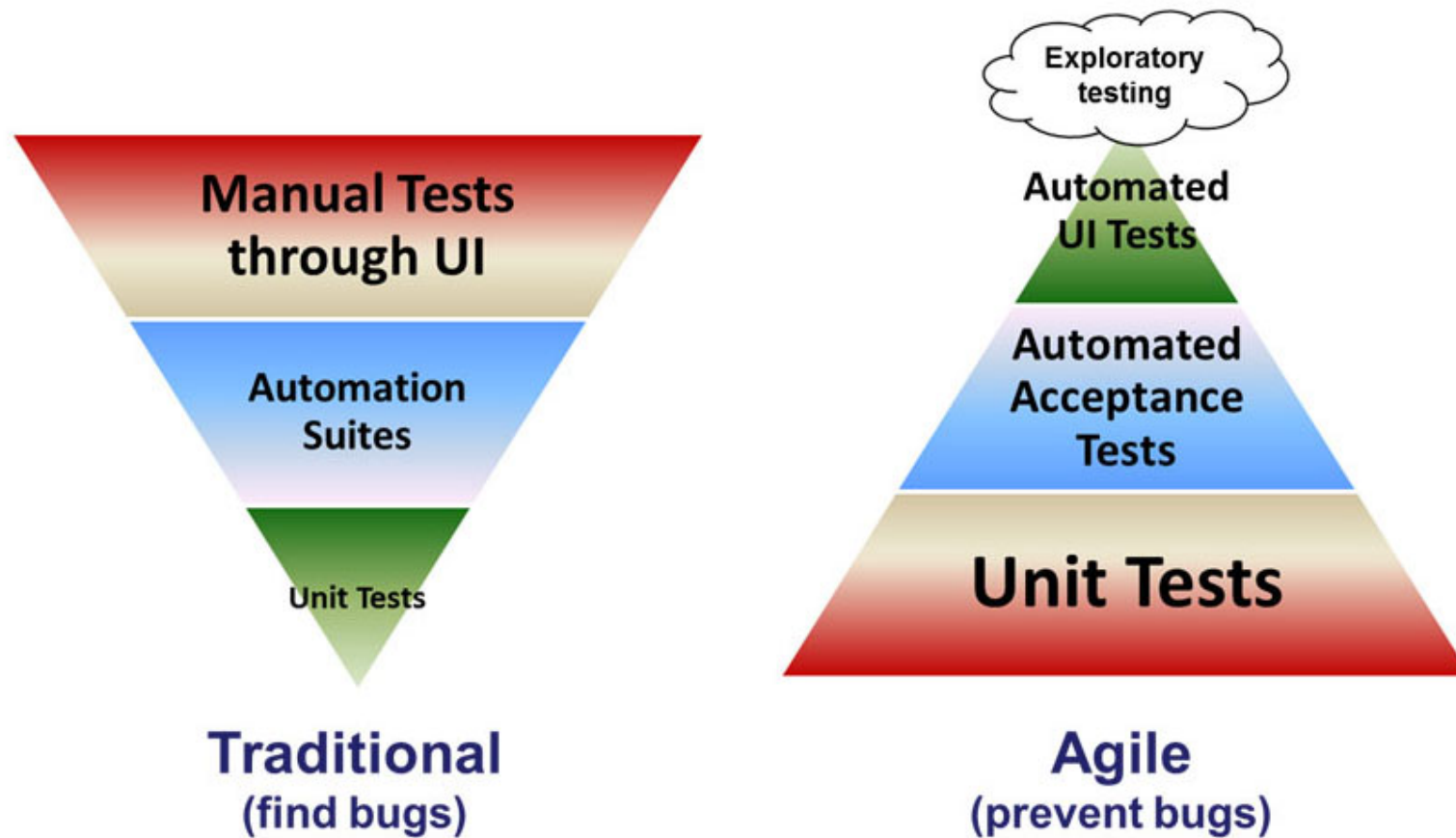
■ Unittests

- Definition
- Nutzen
- Kosten
- Unittests mit Visual Studio
- Aufbau einer Testmethode
- Test Driven Design

V-Modell und Teststufen



Testpyramide



Akzeptanztest

- Der Kunde verfasst (mit Unterstützung) die Anforderungen
- Die Anforderungen werden vom Entwicklungsteam als Funktionalität umgesetzt werden
- Die Funktionalität wird durch den Kunden anhand der Anforderungsliste getestet und als zufriedenstellend umgesetzt quittiert oder als unvollständig bemängelt
- UAT = User Acceptance Testing

Systemtest

- Beim Systemtest wird das System als Ganzes getestet
- End-to-end

Integrationstest

- 2 bis n Komponenten des Softwaresystems werden integriert getestet

- Testkonzept
- Risikoliste
- Teststrategie
- Testumgebung
- Testplan
- Testfälle
- Testprotokoll
- IEEE 829-2008: Standard for Software and System Test Documentation
http://en.wikipedia.org/wiki/IEEE_829

Testfallbeispiel

The screenshot displays the Microsoft Test Center application window. The main title bar shows navigation icons and the text "Testing Center". Below this, a tab bar includes "Plan", "Test", "Track", and "Organize". The "Plan" tab is active, showing a "Contents" pane on the left and a "Properties" pane on the right. The main area is titled "New Test Case 1*: Test Case for functionality 1".

Test Case Details:

- Title:** Test Case for functionality 1
- Iteration:** <Required>
- Status:**
 - Assigned To:** Reijnen, Clemens
 - State:** Design
 - Priority:** 1
- Details:**
 - Automation Status:** Not Automated
 - Area:** <Required>

Steps:

The "Steps" tab is selected, showing a list of actions and their expected results:

Action	Expected Result
1. Open Application	
2. Click button 1	show screen
3. Add @Value1	
4. Add @Value2	

Parameter Values:

Below the steps, there is a section for parameter values. It includes a table with three columns: Value1, Value2, and Value3.

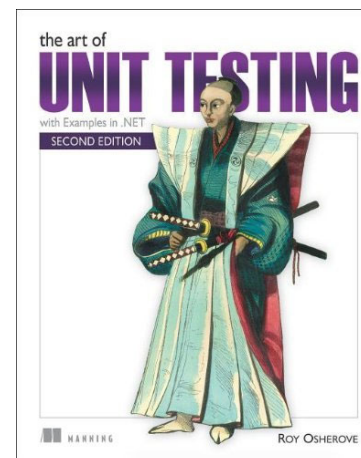
Value1	Value2	Value3
1	1	2
2	2	4

Definition von Unittests

- “A unit test is an automated piece of code that invokes a unit of work in the system and then checks a single assumption about the behavior of that unit of work”

- The Art of Unit Testing

- Unit of work = ~~eine Klasse oder eine Gruppe von wenigen Klassen~~ = eine Funktionalität, unabhängig davon aus wievielen Klassen sie besteht



Nutzen von Unittests

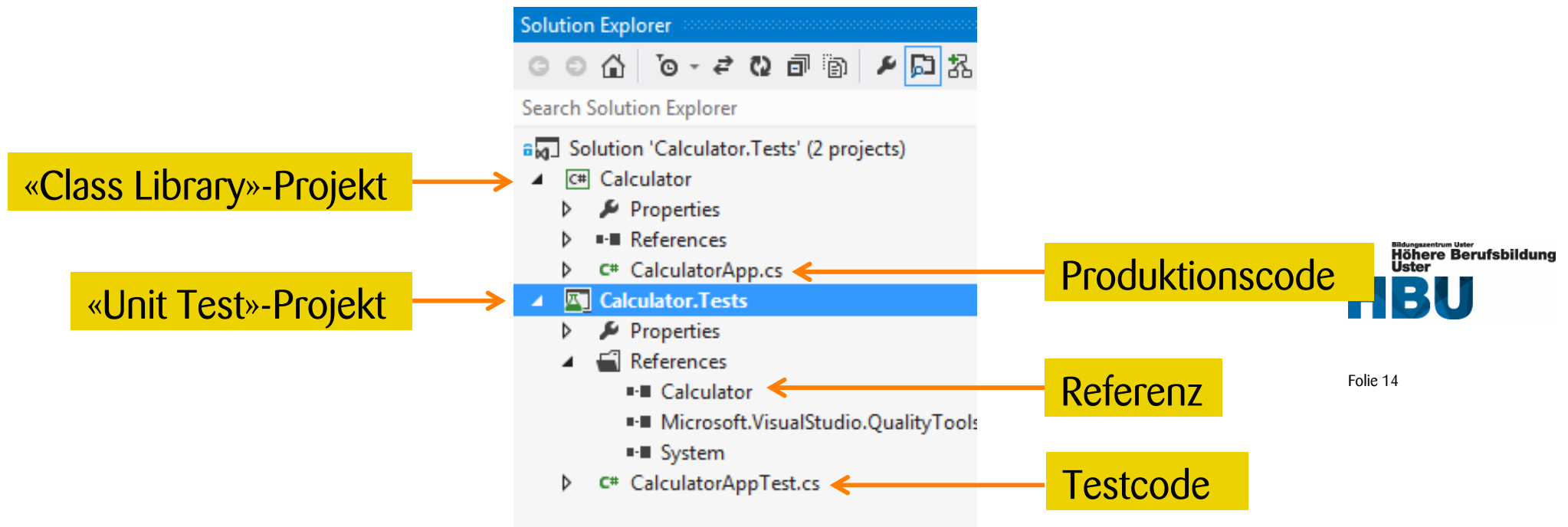
- Regressionstests sind rasch durchführbar
- Design verbessert sich (meistens)
- Fehler werden früh entdeckt
- Code wird durch Tests dokumentiert

- Zeitaufwand Tests zu schreiben
- Testcode muss auch gewartet werden
- Unittesting-Techniken und -Werkzeuge müssen erlernt werden

- Automatisierte Testen nur genau das, wofür sie geschrieben wurden
- Fehler rechts und links der Tests werde nicht gefunden
- Automatisierte Tests sind kein Ersatz für exploratives Testen

Unittests mit Visual Studio

1. Ein «Unit Test»-Projekt erstellen
2. Vom «Unit Test»-Projekt das «Class Library»-Projekt referenzieren



Folie 14

Unittests mit Visual Studio

3. Testklasse erstellen
4. Testmethode schreiben

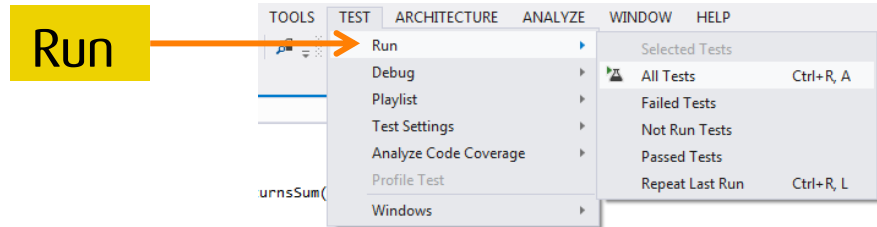
The image shows a side-by-side comparison of two C# files in Visual Studio. The left file, `CalculatorAppTest.cs`, contains a test class `CalculatorAppTest` with a test method `Add_PositiveNumbers_ReturnsSum()`. The right file, `CalculatorApp.cs`, contains the `CalculatorApp` class with an `Add()` method. Two yellow boxes on the left label the components: 'Testklasse' points to the `CalculatorAppTest` class, and 'Testmethode' points to the `Add_PositiveNumbers_ReturnsSum()` method. An orange arrow points from the `calculator.Add(1, 2);` line in the test method to the `Add()` method in the `CalculatorApp` class.

```
CalculatorAppTest.cs
1 using Microsoft.VisualStudio.TestTools.UnitTesting;
2
3 namespace Calculator.Tests
4 {
5     [TestClass]
6     public class CalculatorAppTest
7     {
8         [TestMethod]
9         public void Add_PositiveNumbers_ReturnsSum()
10        {
11            // Arrange
12            var calculator = new CalculatorApp();
13            int expected = 3;
14
15            // Act
16            int actual = calculator.Add(1, 2);
17
18            // Assert
19            Assert.AreEqual(expected, actual);
20        }
21    }
22 }
```

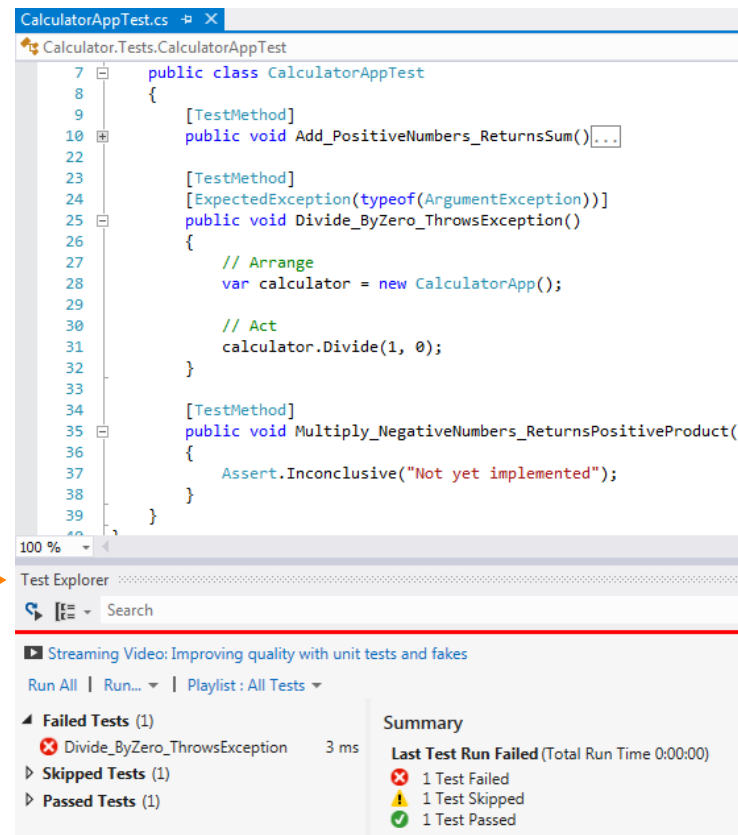
```
CalculatorApp.cs
1 namespace Calculator
2 {
3     public class CalculatorApp
4     {
5         public int Add(int x, int y)
6         {
7             return x + y;
8         }
9     }
10 }
```

Unittests mit Visual Studio

5. Testmethode laufen lassen
6. Testergebnis überprüfen

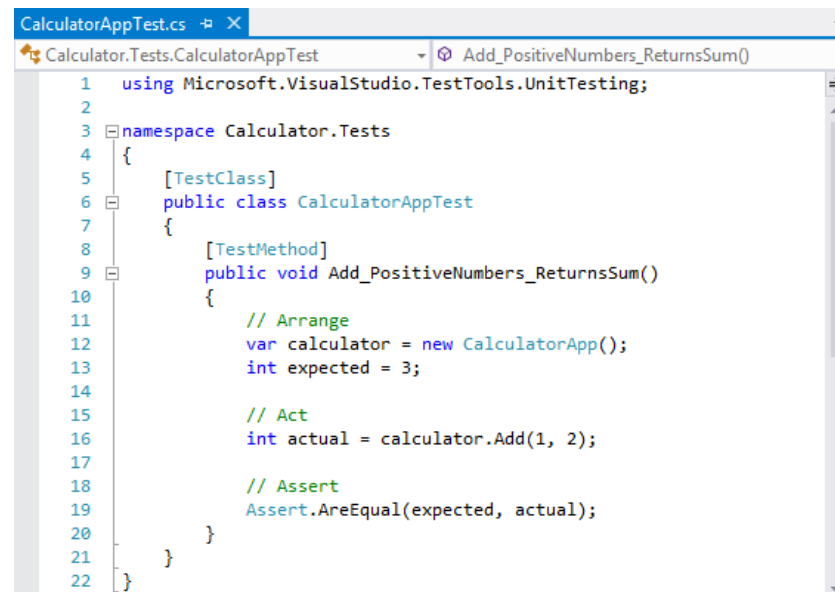


Test Explorer



Aufbau einer Testmethode

- Arrange
 - Stellt den Zustand vor der Testausführung her
- Act
 - Führt die zu testenden Methode aus
- Assert
 - Überprüft, ob das erwartete Ergebnis zurückgeliefert wurde



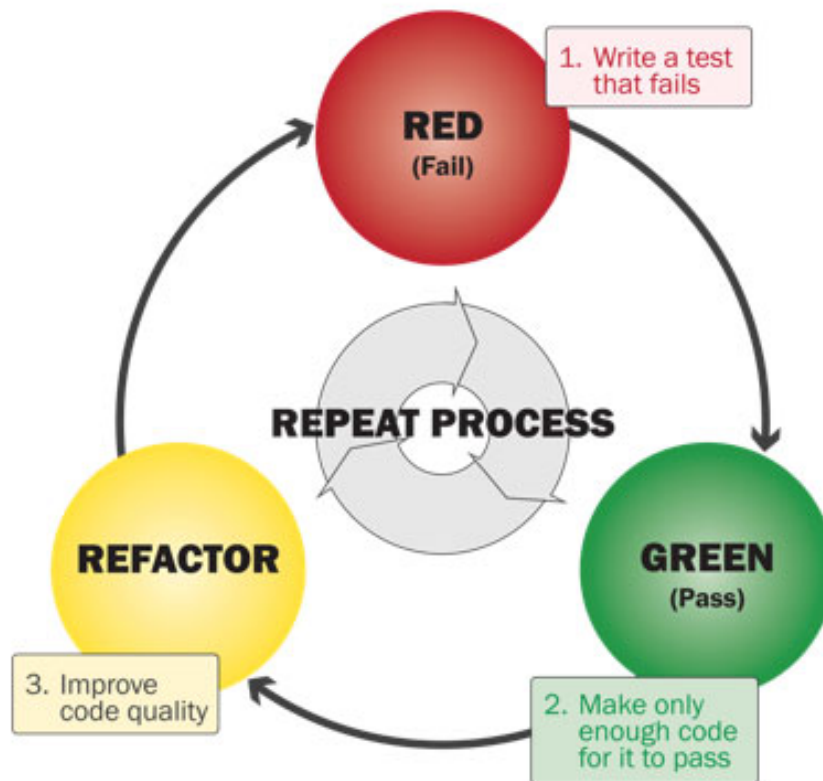
```
CalculatorAppTest.cs
Calculator.Tests.CalculatorAppTest
Add_PositiveNumbers_ReturnsSum()

1  using Microsoft.VisualStudio.TestTools.UnitTesting;
2
3  namespace Calculator.Tests
4  {
5      [TestClass]
6      public class CalculatorAppTest
7      {
8          [TestMethod]
9          public void Add_PositiveNumbers_ReturnsSum()
10         {
11             // Arrange
12             var calculator = new CalculatorApp();
13             int expected = 3;
14
15             // Act
16             int actual = calculator.Add(1, 2);
17
18             // Assert
19             Assert.AreEqual(expected, actual);
20         }
21     }
22 }
```

Assertion-Klassen

Klasse	Beschreibung
Assert	Überprüft Bedingungen mit Hilfe boolescher Aussagen.
CollectionAssert	Wird verwendet, um Objektaufstellungen zu überprüfen.

Test Driven Development (TDD)



Übung 10.1



Unitest Projekt erstellen und Calculator testen

- Erstelle ein Projekt Calculator und ein Testprojekt Calculator.Tests
- Gehe testgetrieben vor und erstelle zuerst eine Testklasse Calculator mit einer Testmethode für die Additionsmethode (red)
- Schreibe den Produktionscode, damit der Test für die Additionsmethode durchläuft (green)
- Verbessere die Codestruktur (refactor)
- Erstelle weitere Tests für Division und Multiplikation
- Erstelle einen Test für die Division durch 0