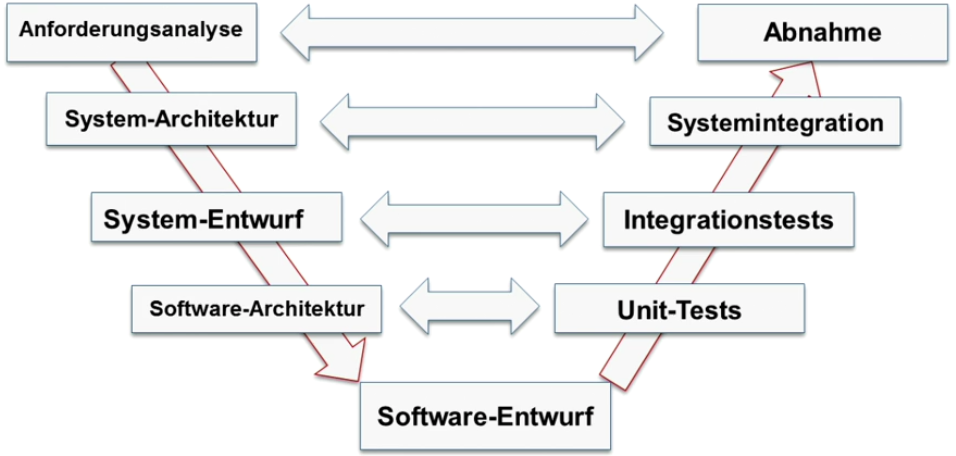
**Modul 320**

***- Block 04 (Unittest)-***

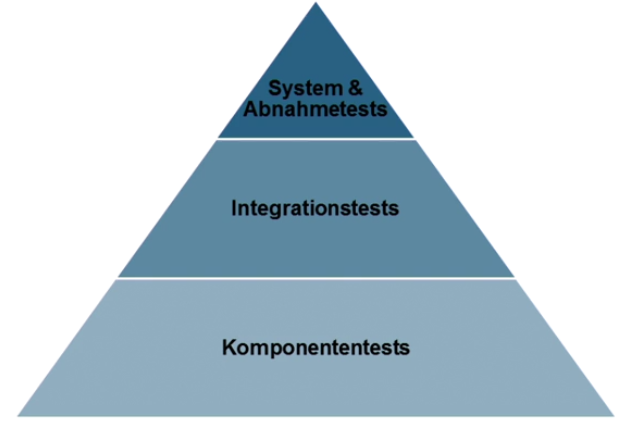
# **Was sind Unittests**

Die in der Anforderungsanalyse erfassten Anforderungen werden durch einen Abnahmetest durch den Endkunden überprüft. Die Eigentliche Entwicklung der Software läuft durch mehrere Phasen, die durch weitere Tests überprüft werden.

Unit-Tests überprüfen die kleinsten Einheiten innerhalb des Softwaresystems. Integrationstests überprüfen, wie die einzelnen Bestandteile miteinander kommunizieren. Der Systemtest prüft das ganze System.



Dieser Sachverhalt kann auch durch die Testpyramide dargestellt werden:



System-und Abnahmetests werden auf einer identischen Zielplattform wie beim Kunden (oder direkt beim Kunden) durchgeführt. Sie benötigen am meisten Ressourcen und lassen sich am wenigsten gut automatisieren.

Integrationstests testen z.B. die Kommunikation zwischen einem Client und einem Server. Diese Tests sind viel Aufwändiger, da hierfür z.B. Datenbanken und Webservices, etc. ausgeführt werden müssen. Integrationstests können nicht mehr auf jedem Computer überprüft werden

Komponenten (auch Unittests genannt) testen z.B. einzelne Methoden einer Klasse. Diese Tests lassen sich sehr gut automatisieren.

**Softwareentwickler befassen sich in der Regel mit Komponenten und Integrationstests**. Tester befassen sich in der Regel mit System & Abnahmetests und manchmal auch mit Integrationstests.

# **Unit Tests mit Nunit durchführen**

**NUnit** ist ein Software-Framework, mit dem sich Unit-Test für alle .NET-Sprachen durchführen lassen. NUnit ist eine Umsetzung des xUnit-Konzepts und wurde ursprünglich von JUnit portiert. NUnit wird oft zusammen mit NMock (open source Bibliothek von mock-Objekten für TDD TestDrivenDevelopment) verwendet. VisualStudio Unit Testing Framework, das Konkur­renzprodukt von Microsoft, ist mittlerweile in VisualStudio enthalten.

## **Grundkonfiguration für Verwendung von NUnit in Microsoft VisualStudio 2022**

Um NUNit in einem Projekt verwenden zu können, brauchen sie:

* NUnit (= das Framework)
* Den NUnit TestAdapter (Fenster, welche Test und Ausführungserfolg anzeigen)

Beide Elemente können als NuGet-Packete innerhalb des Projektes installiert werden. Wenn Sie die beiden Elemente separat installieren, müssen sie darauf achten, dass sie zueinander kompatibel sind, andernfalls werden die Tests nicht korrekt angezeigt.

Ich empfehle ihnen deshalb, gleich ein NuGet-Packet zu installieren, welches beide Elemente beinhaltet.

1) Erstellen sie zuerst die zu testende Applikation. Wählen sie als Projekttyp eine KonsolenApp (.NET Framework) und Nennen sie die App «MyApp». Fügen sie der Applikation die folgenden Klassen mit ihren jeweils beiden Methoden hinzu:

public class TextHandler {

public string MakeUpperCase(string value) {

return value.ToUpper();

}

public string MakeLowerCase(string value) {

throw new NotImplementedException();

}

}

public class MathsHandler {

public int Addition(int valueA, int valueB) {

return valueA + valueB;

}

public int Subtraktion(int valueA, int valueB) {

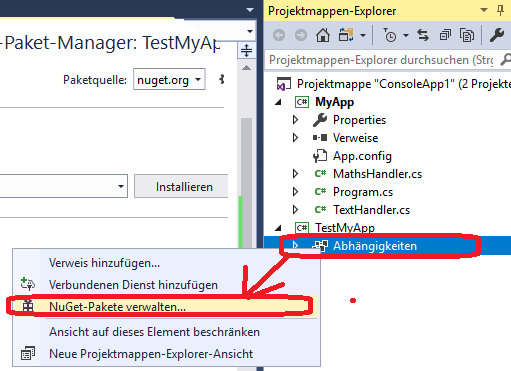
return valueA - valueB;

}

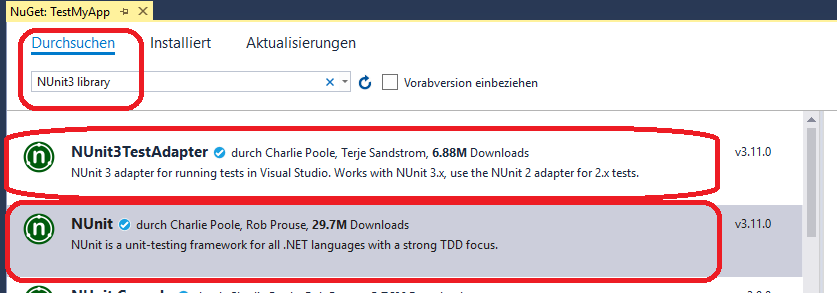
}

2) Fügen Sie der selben Projektmappe ein weiteres Projekt vom Typ Klassenbibliothek (.NET Framework) hinzu. Nennen sie das Projekt TestMyApp.

3) Klicken sie im Klassenbibliotheksprojekt mit der rechten Maustaste auf Abhängigkeiten und wählen sie im Kontextmenü den Eintrag “NuGet-Packete verwalten“ (siehe Bild unten).



Suchen sie nach “NUnit3 Library“ und installieren sie die Pakete NUnit (= Framework) und NUNit3TestAdapter (Visualisierung für VisualStudio).



4) Erstellen sie in der Bibliothek «TestMyApp» für die beiden zu testenden Klasse je eine TestKlasse. Für jede zu überprüfende Methode/Funktionalität erstellen wir eine TestMethode. Dazu verwenden wir die für das Testframework wichtigen Beschreibungen **[TestFixture]** und **[Test].**

Fügen Sie dazu dem Namespace einen Verwies auf das NUnit-Framework hinzu (**using NUnit.Framework;**). Sonst werden diese Tags nicht erkannt. Konfigurieren sie jeden Test mit der Struktur //arrange (=Vorbereitung) //act (=Durchführung) und //assert (=(Auswertung)

**[TestFixture]**

public class **Test**TextHandler {

**[Test]**

public void **Test**MakeUpperCase(){

// arrange

// act

// assert

}

**[Test]**

public void **Test**MakeLowerCase(){

// arrange

// act

// assert

}

}

**[TestFixture]**

public class **Test**MathsHandler {

**[Test]**

public void **Test**Addition() {

// arrange

// act

// assert

}

**[Test]**

public void **Test**Subtraktion() {

// arrange

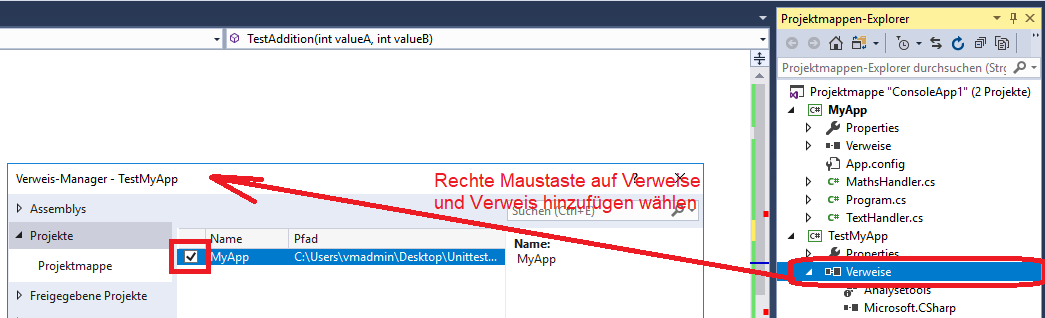
// act

// assert

}

}

4) Damit wir auf die zu testenden Elemente des eigentlichen Projektes zugreifen können, müssen wir der Testbibliothek und dem Namespace einen Verweis hinzufügen (**using TestAppA;)**



5) Schreiben sie die vier Unittests:

**using MyApp;**

**using NUnit.Framework;**

namespace TestMyApp {

[TestFixture]

public class TestMathsHandler {

[Test]

public void TestAddition() {

// arrange

**MathsHandler mh = new MathsHandler();**

// act

**int result = mh.Addition(50, -30);**

// assert

**Assert.That(result, Is.EqualTo(20));**

}

[Test]

public void TestSubtraktion() {

// arrange

**MathsHandler mh = new MathsHandler();**

// act

**int result = mh.Subtraktion(-20, 45);**

// assert

**Assert.AreEqual(-65, result);**

}

}

}

**using MyApp;**

**using NUnit.Framework;**

namespace TestMyApp {

[TestFixture]

public class TestTextHandler {

[Test]

public void TestMakeUpperCase() {

// arrange

TextHandler th = new TextHandler();

// act

string result = th.MakeUpperCase("bbzw");

// assert

Assert.AreEqual("BBZW", result); }

[Test]

public void TestMakeLowerCase() {

// arrange

TextHandler th = new TextHandler();

// act

string result = th.MakeLowerCase("BBZW");

// assert

Assert.AreEqual("bbzw", result);

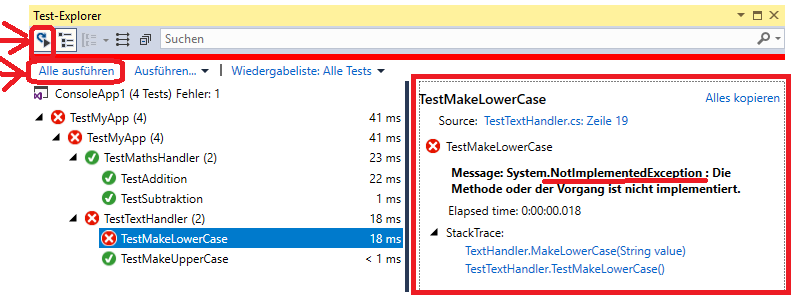
}

}

}

6) Führen sie die Unittests aus:

Öffnen sie im VisualStudio via Menüpunkt «Test🡪Test-Explorer» das Fenster zur Über­wachung der Tests.



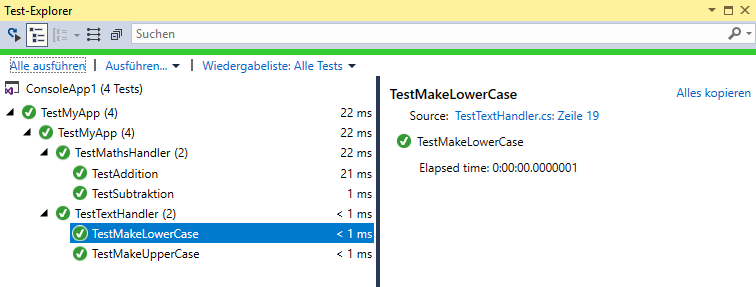
Mit dem oben rot markierten Button können sie wählen, ob nach jedem Kompilationsvorgang die Tests automatisch ausgeführt werden sollen oder nicht.

Führen sie alle Test aus in dem sie auf «Alle ausführen» klicken.

Anhand der grünen und roten Symbole sehen sie, welche Test erfolgreich/nicht erfolgreich waren.

Sie sehen, dass der Test «TestMakeLowerCase» nicht erfolgreich war. In der Beschreibung erfahren sie, dass eine Exception ausgeführt wurde, das diese Methode noch nicht implementiert wurde.

Implementieren Sie die Methode und führen sie die Tests erneut aus. Es sollte nun alles «im Grünen» liegen.



6) Ändern sie den Test «TestAddition» so, dass mehrere Testfälle überprüft werden. Sie sehen, dies ist mit Parameter und Rückgabewerten möglich. Wir bauen bewusst einen Fehler ein (-5 + 50 ist nicht gleich 40!!. Prüfen sie, ob das Testframework dies bemerkt:

**[TestCase (10, 20, ExpectedResult = 30)]**

**[TestCase(-5, 50, ExpectedResult = 40)]**

**[TestCase(-30, -40, ExpectedResult =-70)]**

**[TestCase(80, -20, ExpectedResult = 60)]**

public int TestAddition(**int numberA, int numberB**) {

// arrange

MathsHandler mh = new MathsHandler();

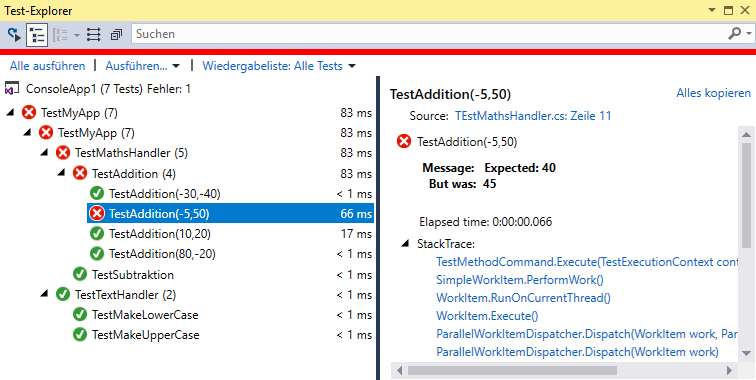
// act

int result = mh.Addition(numberA, numberB);

// assert

**return result;**

}



# **Aufgaben**

**Aufgabe 01 [mehr Details zu Nunit erfahren]: (30 Min)**

Öffnen sie die Webseite <http://dotnetpattern.com/nunit-example-test-case> und überfliegen (schnell lesen) sie die vier NUnit Tutorials «First TestCase», «TestCase Attributes», «Arrays and Order» and «Assert».