Walkthrough: Unit tests voor managed C# code

Inleiding

We doorlopen als begeleide oefening de creatie en de uitvoering van een verzameling unit tests met behulp van het Microsoft unit test framework MSTestv2 en Visual Studio 2019 **Test Explorer**.

We volgen **"best practices" TDD en AAA**.

Je begint met een C# project dat onder ontwikkeling is, maakt testen aan die toelaten de code op de proef te stellen, voert deze testen uit en onderzoekt de resultaten. Daarna pas je de project code aan en voer je de testen opnieuw uit.

Creeer een project

1. Open Visual Studio.
2. In het start venster, kies **Create a new project**.
3. Zoek en selecteer de C# **Class library** (A project for creating a class library that targets .NET Standard or .NET Core) project template en klik **Next**.
4. Noem het project **Bank** en klik vervolgens op **Next**. Selecteer als Target Framework .NET Core 3.1 (Long-term support). Klik op **Create**.

   Het Bank project wordt aangemaakt en getoond in **Solution Explorer** met het *Class1.cs* bestand open in de code editor.

   > **NOOT**
   > Indien *Class1.cs* niet geopend is in de editor, dubbel-klik op bestand *Class.cs* in **Solution Explorer** om het te openen.

5. Hernoem Class1 naar Bank: dit heeft als voordeel dat de naam van het bestand overeenkomstig aangepast wordt. Vervang de inhoud van het bestanddoor de volgende C# code die een class *BankAccount* definieert:

```
using System;

namespace BankAccountNS
{
    /// <summary>
    /// Bank account demo class.
    /// </summary>
    public class BankAccount
    {
        private readonly string m_customerName;
        private double m_balance;

        private BankAccount() { }

        public BankAccount(string customerName, double balance)
        {
            m_customerName = customerName;
            m_balance = balance;
```

```
        }

        public string CustomerName
        {
            get { return m_customerName; }
        }

        public double Balance
        {
            get { return m_balance; }
        }
    }
}
```

6. In het **Build** menu, klik **Build Solution**.

Je hebt nu een project om mee te starten. In deze walkthrough focuseren we onze testen op de `Debet` method. De `Debet` method wordt opgeroepen wanneer geld van een rekening gehaald wordt.

Creeer een unit test project

1. In het **File** menu selecteer **Add** > **New Project**.

   > **TIP**
   > Je kan ook rechts klikken op de "solution" in **Solution Explorer** and kiezen voor **Add** > **New Project**.

2. Zoek en kies de C# **Unit Test Project** *(A project that contains unit tests that can run on .NET Core on Windows, Linux and maxOS) project template*, en klik op **Next**. Selecteer als Target Framework *.NET Core 3.1 (Long-term support)* en klik op **Create**.
3. Noem het project **BankTests**.
4. Klik op **Create**.

   Het **BankTests** project wordt toegevoegd aan de **Bank** solution.

5. In het **BankTests** project, voeg een referentie toe naar het **Bank** project.

   In **Solution Explorer**, selecteer **Dependencies** onder het **BankTests** project en kies **Add Reference** uit het rechterklik-menu.

6. In de **Reference Manager** dialog box, expandeer **Projects**, selecteer **Solution** en klik het **Bank** item aan.
7. Klik op **OK**.

Creeer de test class

Creeer een test class om class `BankAccount` te testen. Je kan bestand *UnitTest1.cs* gebruiken dat aangemaakt is door de project template, maar je moet het bestand en de class een betere naam geven.

Hernoem bestand en class

1. Om het bestand te hernoemen, selecteer in **Solution Explorer**, het *UnitTest1.cs* bestand in het BankTests project. Klik **Rename** in het rechterklikmenu en hernoem het bestand naar *BankAccountTests.cs*.

2. Om de class te hernoemen class, positioneer de cursus op `UnitTest1` in je code editor, rechterklik en kies **Rename**. Tik in **BankAccountTests** en druk op **Enter**.

Het *BankAccountTests.cs* bestand bevat de volgende code:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace BankTests
{
    [TestClass]
    public class BankAccountTests
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

Voeg een using statement toe

Voeg een using statement toe aan de test class om code te kunnen oproepen van het project dat je test zonder een volledig gekwalificeerde naam te moeten gebruiken. Voeg bovenaan in het bestand toe:

```
using BankAccountNS;
```

Test class vereisten

Een test class moet:

- Een attribuut `[TestClass]` hebben om uitgevoerd te kunnen worden in Test Explorer.
- Een test method hebben die herkend kan worden door Test Explorer aan de hand van een `[TestMethod]` attribuut.

Je kan in een test project classes opnemen die geen `[TestClass]` attribuut dragen en je kan methods hebben zonder `[TestMethod]` attribuut. Deze methods en classes kunnen opgeroepen worden door je test methods.

Creeer een eerste test method

In deze procedure zal je unit test methods schrijven om het gedrag van de `Debet` method van de `BankAccount` class te testen.

Minstens drie gedragingen moeten gecontroleerd worden:

- De method werpt een `System.ArgumentOutOfRangeException` op indien het debet bedrag groter is dan het bedrag op rekening.
- De method werpt een `System.ArgumentOutOfRangeException` op indien het debet bedrag kleiner is dan 0.
- Indien het debet bedrag geldig is, telt de method het bedrag af van het bedrag op rekening.

> [!TIP]
> Je mag de default `TestMethod1` method verwijderen aangezien we deze niet gebruiken.

Creeer een test method

De eerste test verifieert dat een geldig bedrag (minder of gelijk aan het bedrag op rekening en groter dan 0) afgehaald wordt. Voeg de volgende method toe aan de `BankAccountTests` class:

```
[TestMethod]
public void Debet_WithValidAmount_UpdatesBalance()
{
    // Arrange
    double beginningBalance = 11.99;
    double debetAmount = 4.55;
    double expected = 7.44;
    BankAccount account = new BankAccount("Mr. Tom Vervoort", beginningBalance);

    // Act
    account.Debet(debetAmount);

    // Assert
    double actual = account.Balance;
    Assert.AreEqual(expected, actual, 0.001, "Account not debeted correctly");
}
```

Deze method spreekt voor zich: er wordt een nieuw `BankAccount` object aangemaakt met een startbedrag en hiervan wordt een geldig bedrag afgehouden. De `Microsoft.VisualStudio.TestTools.UnitTesting.Assert.AreEqual()` method wordt gebruikt om na te gaan of de eindbalans naar verwachting is.

Test method vereisten

Een test method moet aan volgende vereisten voldoen:

- de test method is voorzien van het `[TestMethod]` attribuut.
- `void` wordt teruggegeven.
- Parameters zijn niet toegelaten.

Bouw je test applicatie en breid je code uit

Wanneer je je applicatie bouwt, verschijnt er een rode krullijn onder .Debet. Beweeg je je muis hierover, dan zie je staan CS1061: 'BankAccount' does not contain a definition for 'Debet' and no accessible extension method 'Debet' accepting a first argument of type 'BankAccount' could be found. Je kan dan klikken op de link eronder: Show potention fixes (ALT+Enter or CTRL+.). vJe kiest Generate method 'BankAccount.Debet'. Op dat moment wordt je class Bank uitgebreid met:

```
        public void Debet(double debetAmount)
        {
            throw new NotImplementedException();
        }
```

Zo werkt TDD: je schrijft in je test code wat je van je class verwacht en breidt deze uit. Vergeet niet je method van **commentaar te voorzien**. Dat doe je door boven de method drie slashes in te geven: er wordt dan automatisch een xml-commentaarsectie aan je code toegevoegd. Een dergelijke xml-commentaar kan gebruikt worden om automatisch documentatie uit je code te genereren.

De gegenereerde method Debet() werpt enkel een uitzondering op en doet nog niets. Je voorziet een implementatie:

```
        public void Debet(double amount)
        {
            if (amount > m_balance)
            {
                throw new ArgumentOutOfRangeException("amount");
            }

            if (amount < 0)
            {
                throw new ArgumentOutOfRangeException("amount");
            }

            m_balance += amount; // intentionally incorrect code
        }
```

Bouw de test en voer deze uit

1. In het **Build** menu kies **Build Solution**.
2. Indien **Test Explorer** nog niet geopend is, open deze via **Test** > **Windows** > **Test Explorer** in de top menu.
3. Kies **Run All** om de test uit te voeren.

Terwijl de test loopt, wordt de status bovenaan het **Test Explorer** venster geanimeerd. Wanneer de test uitgevoerd is, zal de status groen kleuren indien alle testen succesvol uitgevoerd konden worden of rood kleuren wanneer er een test faalde. In dit geval faalt de test.

4. Select the method in **Test Explorer** to view the details at the bottom of the window.

Verbeter je code en voer de testen opnieuw uit

Het testresultaat bevat een boodschap die de fout beschrijft. Voor de `AreEqual` method toont de boodschap wat er verwacht werd en wat er eigenlijk vastgesteld werd. Je verwachte dat het bedrag op rekening afnam, maar het bedrag dat eraf moest, kwam erbij.

De unit test bracht een bug boven: het afgehaalde bedrag werd *toegevoegd* aan het bedrag op rekening in plaats van *afgetrokken*.

Verbeter de bug

Om de fout te verbeteren pas in bestand *BankAccount.cs* de volgende lijn aan:

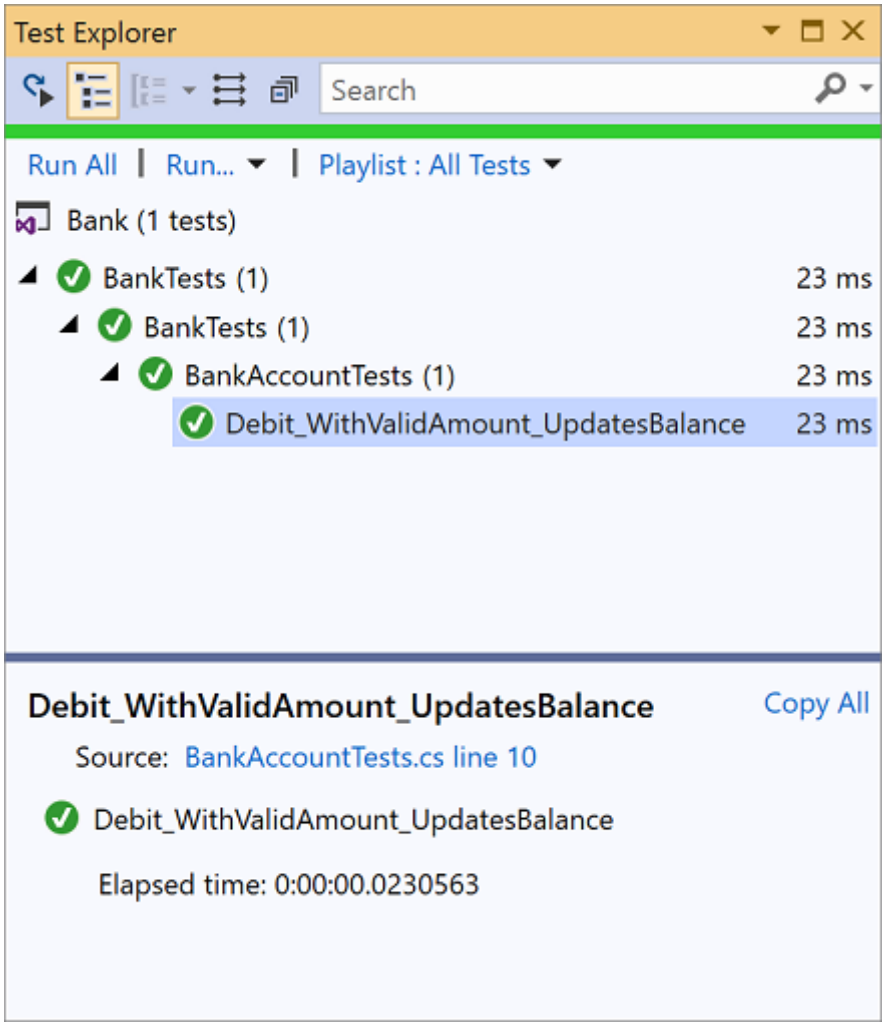
```
m_balance += amount;
```

with:

```
m_balance -= amount;
```

Voer de test opnieuw uit

In **Test Explorer**, kies **Run All** om de test opnieuw uit te voeren. De rode status wordt groen om aan te tonen dat de test lukte.



Gebruik unit testen om je code te verbeteren

Deze sectie beschrijft hoe het iteratieve proces van analyse, unit test ontwikkeling en refactoring kan helpen om je productiecode robuuster en effectiever te maken.

Analyzeer de problemen

Je hebt reeds een test method aangemaakt om te controleren dat een geldig bedrag correct werd afgeteld in de `Debet` method. Verifieer nu of de method een `System.ArgumentOutOfRangeException` oproept indien het debet bedrag:

- groter is dan het bedrag op rekening
- kleiner is dan 0.

Maak nieuwe unit tests aan en voer deze uit

Creeer een method om na te gaan of alles goed verloopt wanneer het debet bedrag kleiner is dan 0:

```
[TestMethod]
public void Debet_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRange()
{
    // Arrange
    double beginningBalance = 11.99;
    double debetAmount = -100.00;
    BankAccount account = new BankAccount("Mr. Tom Vervoort", beginningBalance);

    // Act and assert
    Assert.ThrowsException<System.ArgumentOutOfRangeException>(() =>
account.Debet(debetAmount));
}
```

Gebruik de `Microsoft.VisualStudio.TestTools.UnitTesting.Assert.ThrowsException` method om na te gaan of de juiste uitzondering werd opgeworpen. Deze method doet de test falen tenzij een `System.ArgumentOutOfRangeException` uitzondering wordt opgeworpen. Je kan de code tijdelijk aanpassen om een meer generieke uitzondering `System.ApplicationException` op te werpen: dit werkt ook.

Om het geval te testen waarbij het debet bedrag groter is dan het bedrag op rekening, doe het volgende:

1. Maak een nieuwe test method aan met de naam `Debet_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRange`.
2. Copieer de zogenaamde *method body* van method `Debet_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRange` naar de nieuwe method.
3. Wijs een getal groter dan het bedrag op rekening toe aan `debetAmount`.

Voer de twee testen uit en kijk na of ze slagen.

Vervolg de analyse.

De method die getest wordt, kan verder verbeterd worden. Met de huidige implementatie kunnen we niet weten welke conditie tot de uitzondering leidde (`amount > m_balance` of `amount < 0`) die opgeworpen werd tijdens het uitvoeren van de test. We weten enkel dat `ArgumentOutOfRangeException` opgeroepen werd. Veel beter is dat we weten welke conditie in `BankAccount.Debet` tot de uitzondering leidde (`amount > m_balance` of `amount < 0`) zodat we er zeker van kunnen zijn dat de method zijn argumenten correct nakijkt.

Bestudeer de method opnieuw (`BankAccount.Debet`) en merk op dat in beide gevallen dezelfde constructor van `ArgumentOutOfRangeException` gebruikt wordt:

```
throw new ArgumentOutOfRangeException("amount");
```

Er is nog een constructor die toelaat veel meer informatie mee te geven: `System.ArgumentOutOfRangeException` kan de argumentnaam, de argumentwaarde en een boodschap die je zelf kiest, als argument meekrijgen. Je kan de code die getest wordt, nu aanpassen zodat deze de bewuste constructor gebruikt. Beter nog, je kan publieke type members gebruiken om de fouten te specificeren.

Refactor de code die getest wordt

Definieer twee constanten voor de foutboodschappen op niveau van je class. Plaats deze onder de class `BankAccount` die getest wordt:

```
public const string DebetAmountExceedsBalanceMessage = "Debet amount exceeds
balance";
public const string DebetAmountLessThanZeroMessage = "Debet amount is less than
zero";
```

Wijzig vervolgens de conditionele statements in de `Debet` method:

```
if (amount > m_balance)
{
    throw new System.ArgumentOutOfRangeException("amount", amount,
DebetAmountExceedsBalanceMessage);
}

if (amount < 0)
{
    throw new System.ArgumentOutOfRangeException("amount", amount,
DebetAmountLessThanZeroMessage);
}
```

Refactor de test methods

Refactor de test methods door de oproep van `Microsoft.VisualStudio.TestTools.UnitTesting.Assert.ThrowsException` te verwijderen. Wrap het oproepen van `Debet()` in een `try/catch` blok, vang de verwachte uitzondering op, and verifieer de geassocieerde boodschap. `Microsoft.VisualStudio.TestTools.UnitTesting.StringAssert.Contains()` method biedt de mogelijkheid om twee strings te vergelijken.

`Debet_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRange` kan er bijgevolg volgende vorm aannemen:

```
[TestMethod]
public void Debet_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRange()
{
    // Arrange
    double beginningBalance = 11.99;
    double debetAmount = 20.0;
    BankAccount account = new BankAccount("Mr. Tom Vervoort", beginningBalance);

    // Act
    try
    {
        account.Debet(debetAmount);
    }
    catch (System.ArgumentOutOfRangeException e)
    {
        // Assert
        StringAssert.Contains(e.Message,
BankAccount.DebetAmountExceedsBalanceMessage);
    }
```

```
}
```

Hertest, herschrijf en heranalyseer

Neem aan dat er een bug zit in de method die we testen. De `Debet` method werpt niet eens een uitzondering op (System.ArgumentOutOfRangeException). Zoals de code momenteel is, wordt het geval niet eens afgehandeld. Indien de `debetAmount` waarde geldig is (dus minder dan de balans en groter dan nul), wordt geen uitzondering opgevangen en de assert voert nooit uit. Toch slaag de test method. Dit is niet goed want je wenst dat de test method faalt wanneer er geen uitzondering opgeworpen wordt.

Dit is een bug in de test method. Om dit op te lossen voeg je een `Microsoft.VisualStudio.TestTools.UnitTesting.Assert.Fail` assert toe aan het einde van de test method om het geval op te vangen waarin geen uitzondering opgeworpen wordt.

Wanneer je de test opnieuw uitvoert, zie je dat de test nu *faalt* wanneer de correcte uitzondering wordt opgevangen. Het `catch` block vangt de uitzondering op, maar de method blijft uitvoeren en faalt bij de nieuwe `Microsoft.VisualStudio.TestTools.UnitTesting.Assert.Fail` assert. Om dit op te lossen voeg je een `return` statement toe na `StringAssert` in de `catch` block. Voer de test opnieuw uit om na te gaan dat je het probleem hebt opgelost. De definitieve versie van `Debet_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRange` ziet er zo uit:

```
[TestMethod]
public void Debet_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRange()
{
    // Arrange
    double beginningBalance = 11.99;
    double debetAmount = 20.0;
    BankAccount account = new BankAccount("Mr. Tom Vervoort", beginningBalance);

    // Act
    try
    {
        account.Debet(debetAmount);
    }
    catch (System.ArgumentOutOfRangeException e)
    {
        // Assert
        StringAssert.Contains(e.Message,
BankAccount.DebetAmountExceedsBalanceMessage);
        return;
    }

    Assert.Fail("The expected exception was not thrown.");
}
```

Implementeer een test voor method Credit() en voorzie een implementatie

```
    public void Credit(double amount)
    {
        if (amount < 0)
        {
            throw new ArgumentOutOfRangeException("amount");
        }

        m_balance += amount;
    }
```

Tot slot

De verbeteringen aan de test code leidden tot meer robuuste en informatieve test methods. Belangrijker: de code die getest wordt, werd ook verbeterd.

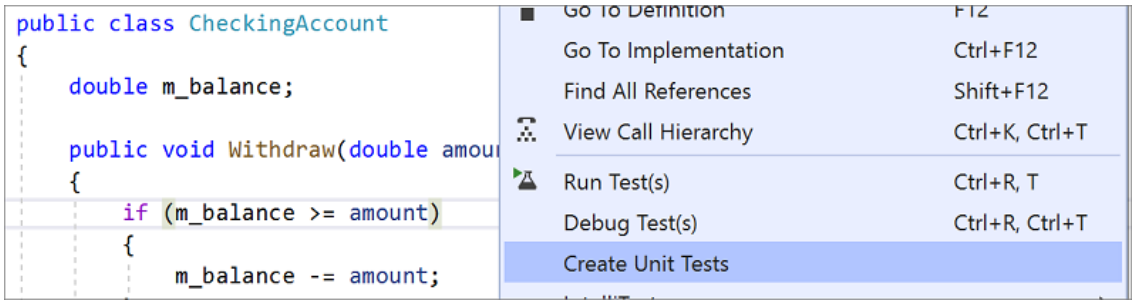
> [!TIP]
> Deze walkthrough gebruikt het Microsoft unit test framework voor managed code. **Test Explorer** kan ook testen van third-party unit test frameworks uitvoeren die adapters hebben voor **Test Explorer**.

Maak unit test projects en test methods aan

Vaak kan je sneller test projecten en test stubs aanmaken op een automatische manier, maar manueel gaat uiteraard ook, zoals we hierboven zagen.

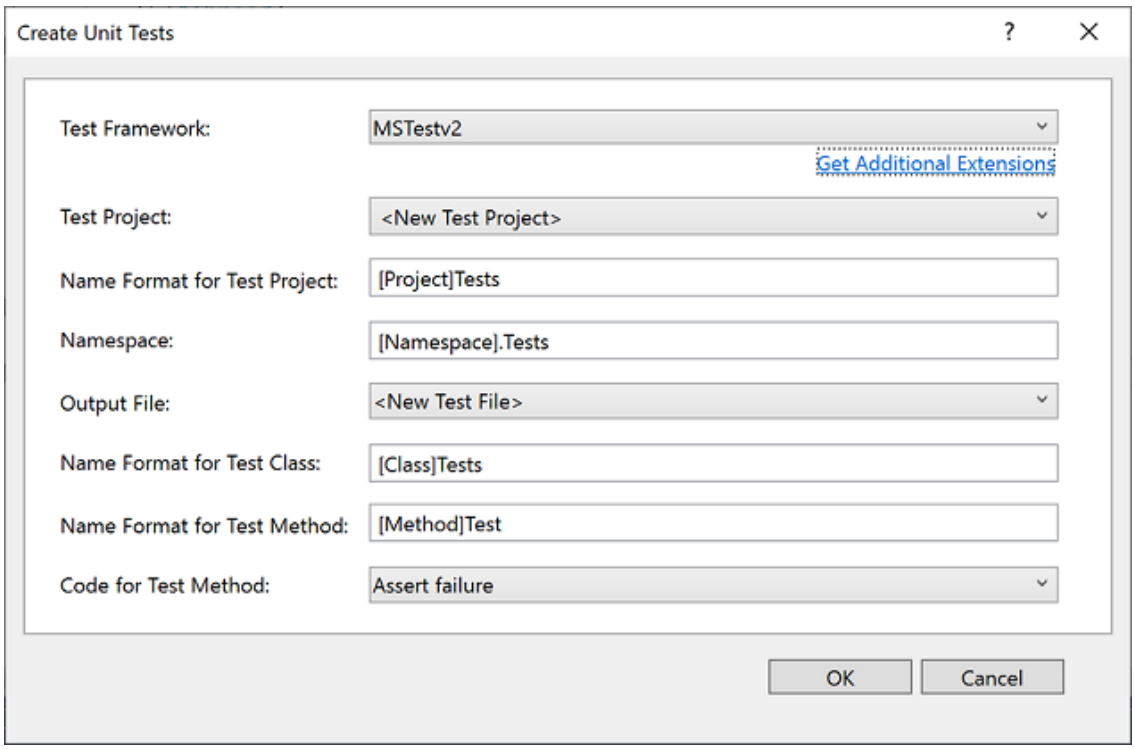
Genereer een unit test project en unit test stubs

1. vanuit het code editor venster rechter-klik en kies **Create Unit Tests**.

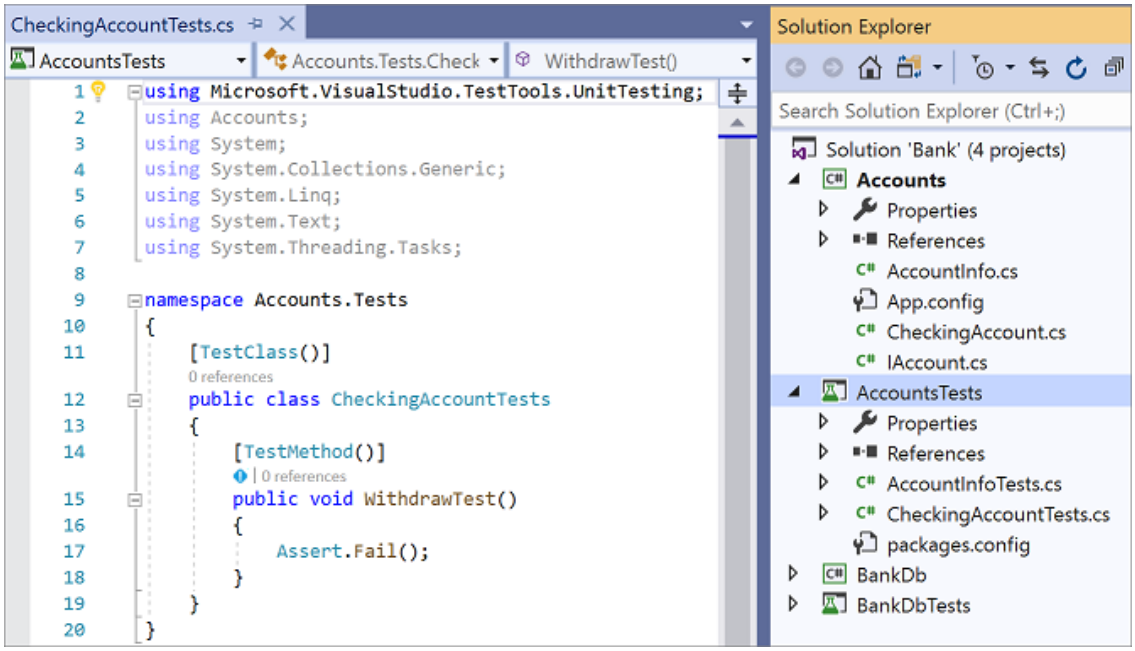


> [!NOOT]
> De **Create Unit Tests** menuopdracht is enkel beschikbaar voor managed code.

2. Klik **OK** om de default waarden te aanvaarden bij het aanmaken van je unit tests, of wijzig eventueel de namen van project en testen.



3. De unit test stubs worden aangemaakt in een nieuw unit test project voor alle methods in de class.



Schrijf je tests

Het unit test framework dat je gebruikt en Visual Studio IntelliSense zullen je begeleiden bij het schrijven van de testen.

Zet timeouts voor je unit tests

Als je het MSTestv2 framework gebruikt, kan je een attribuut gebruiken om een timeout in te stellen bij een specifieke individuele test:

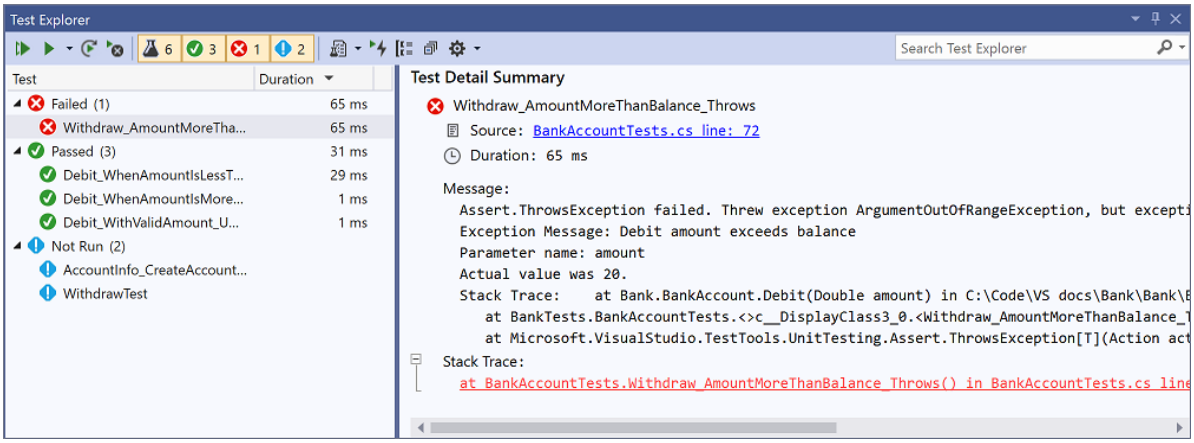
```
[TestMethod]
[Timeout(2000)]  // Milliseconds
public void My_Test()
{ ...
}
```

Je kan de timeout ook instellen op "oneindig":

```
[TestMethod]
[Timeout(TestTimeout.Infinite)]  // Milliseconds
public void My_Test ()
{ ...
}
```

Voer testen uit in Test Explorer

Wanneer je een test project bouwt, verschijnen de testen onder **Test Explorer**. Indien **Test Explorer** niet zichtbaar is, kies dan onder **Test** in het Visual Studio menu **Windows** > **Test Explorer**.

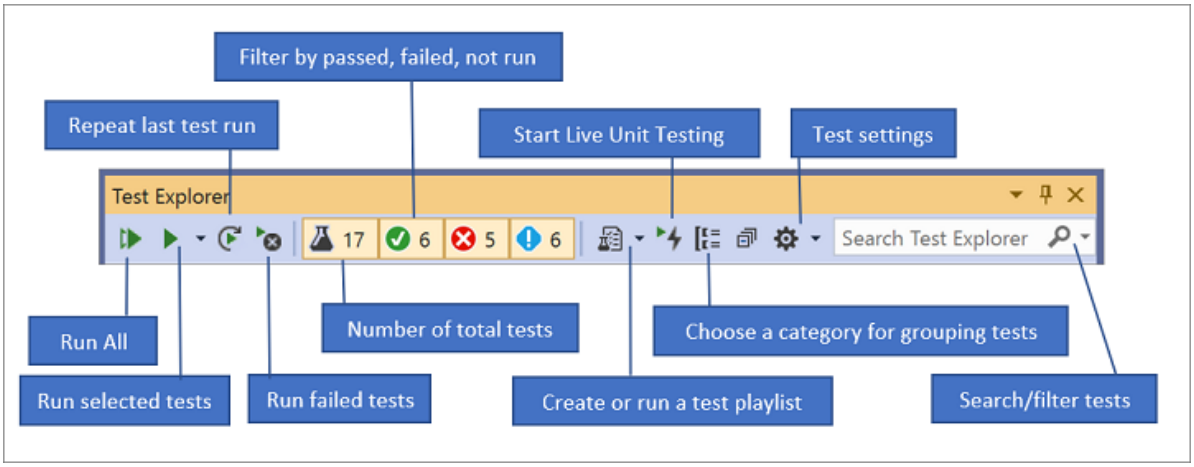


Terwijl je testen uitvoert, schrijft en opnieuw uitvoert, toont **Test Explorer** de resultaten in groepen **Failed Tests**, **Passed Tests**, **Skipped Tests** en **Not Run Tests**. De toolbar laat je toe een andere groep te kiezen.

Je kan ook filters toepassen op de testen door gebruik te maken van tekst in de globale search box of door een van de voorgedefinieerde filters te kiezen. Je kan eender welke selectie van testen steeds opnieuw uitvoeren. Details van een test method worden getoond wanneer je een specifieke method aanklikt.

Voer testen uit en bekijk deze

De **Test Explorer** toolbar helpt je testen organiseren.



Je kan opteren voor **Run All** indien je alle testen wenst uit te voeren of voor **Run** indien je een subset wenst uit te voeren. Kies **Open Test** vanuit het rechterklikmenu (Keyboard: **F12**) om de source code van de geselecteerde test te zien.

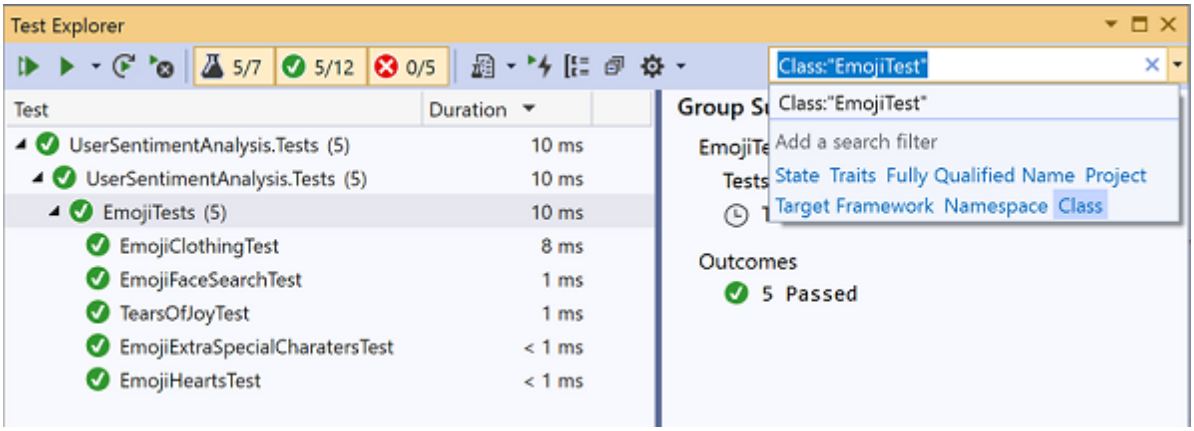
Indien individuele testen geen onderlinge afhankelijkheden hebben die een orde opleggen onderling kan je opteren voor *parallel test execution* bij de instellingen (toolbar). Dit kan de tijd benodigd voor het uitvoeren van de testen gevoelig inkorten.

Voer testen uit na elke build

Om je testen uit te voeren na elke locale build, open het settings (instellingen) icoon in de Test Explorer toolbar en selecteer **Run Tests After Build**.

Filter en groepeer de lijst van testen

Wanneer je veel testen hebt, kan de **Test Explorer** search box van nut zijn om te filteren. Je kan ook een voorgedefinieerde filter kiezen.



Gebruik Live unit testing

Zoals we reeds tijdens vorige les zagen, kan je in Test Explorer het "Live Unit Testing Window" oproepen: klik op het bliksem-icoon in de toolbar. Wanneer je "Live Unit Testing" start, zal je applicatie bij elke codewijziging gebouwd worden en de unit test suite zal uitgevoerd worden, zodat je onmiddellijk ziet wat het resultaat van je codewijzigingen is.

Controleer je code coverage

Je zal opmerken dat je, ondanks het feit dat je een test voorzag voor method *Credit()*, je assembly *bank.dll* nog niet voor 80% getest werd. Werk verder testen uit tot je deze 80% wel bereikt.

Controleer de metrics

In principe zal je hier zien dat je applicatie vrij eenvoudig is.

Breng je oplossing onder bij GitHub

Je kan zo de github link bij je opdracht indienen.