Unit Tests

Ingo Köster

Diplom Informatiker (FH)

Überblick

- Durch UnitTests (auch Modul- oder Komponententests) wird in der Softwareentwicklung die Funktionalität von Modulen (meist Funktion/Methode) getestet
- Modultests testen ein Modul isoliert, d. h. weitgehend ohne Interaktion mit anderen Modulen
- Modultests zählen zu den sog. White-Box-Tests
  - bedeutet, dass bei der Definition der Testfälle der zu testende Quellcode bekannt ist
- Die Spezifikation der Software wird für die Bestimmung der Soll-Ergebnisse benutzt

Zu testendes Beispiel

› Code für die Verwaltung einfacher Konten

› Jedes Konto soll folgendes Interface erfüllen:

```
public interface IBankAccount
{
    double Balance { get; set; }

    double Withdraw(double amount);
    void Deposit(double amount);
}
```

Was soll getestet werden?

› Implementierung:

```
public class CheckingAccount : IBankAccount
{
    public CheckingAccount() { this.Balance = 0; }
    public CheckingAccount(double start_amount) { this.Balance = start_amount; }

    private double balance;

    public double Balance
    {
        get { return this.balance; }
        set { this.balance = value; }
    }

    public void Deposit(double amount)
    {
        this.Balance += amount;
    }
} CheckingAccount
```

```
public double Withdraw(double amount)
{
    if (this.Balance >= amount)
    {
        this.Balance -= amount;
        return amount;
    }

    throw new ArgumentException("Konto überzogen!");
}
```

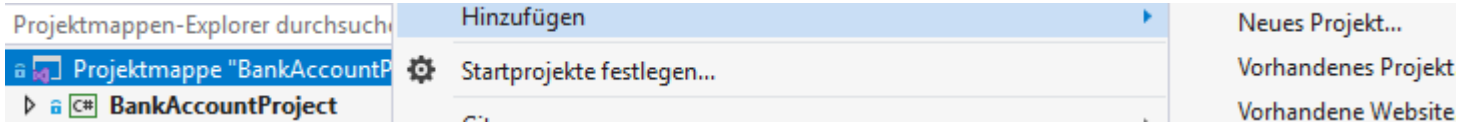
Warum testen?

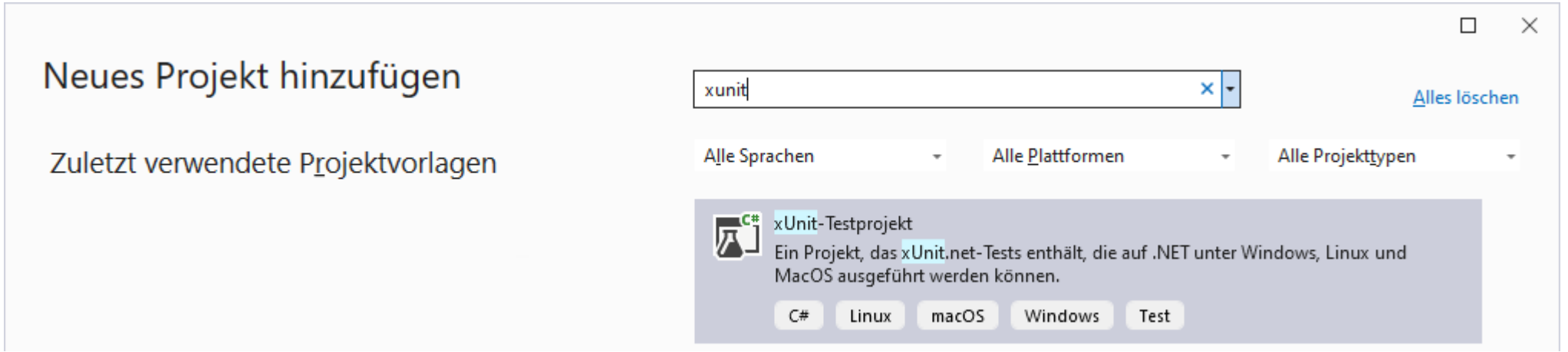
- Wird bereits in der Entwicklungsphase der Code einzelner Klassen getestet, dann
  - Werden Fehler frühzeitig erkannt
  - Kann die vollständige Implementierung der Spezifikationen überprüft werden
  - Werden fehlende Funktionen der Klassen schneller erkannt

- Komponententests (UnitTests) helfen robusteren Code zu schreiben

Der Projektmappe ein UnitTest Projekt hinzufügen

› Der Projektmappe ein neues xUnit-Projekt hinzufügen



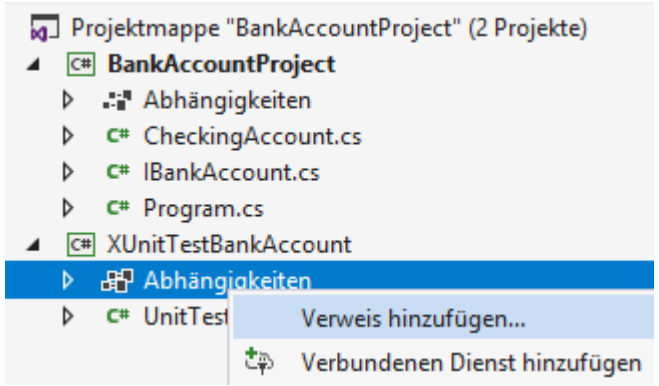


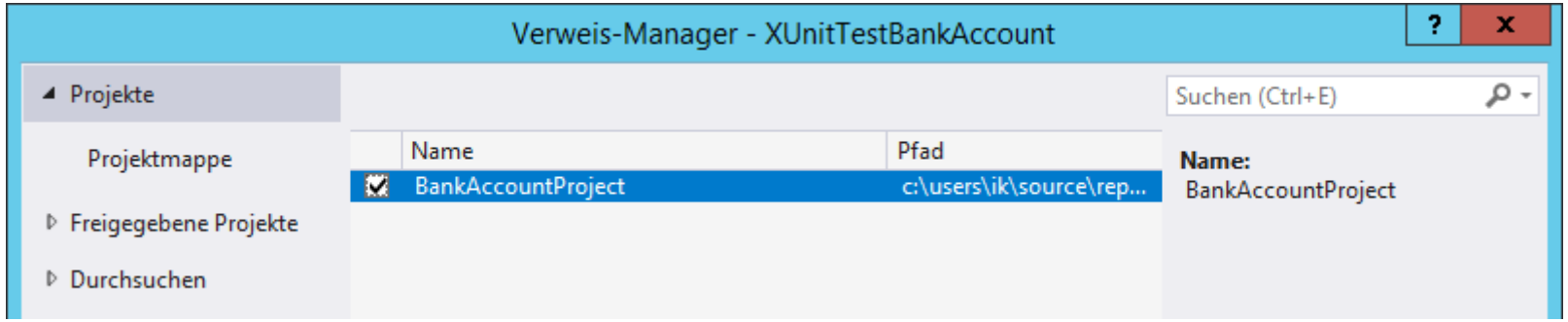
Projekt für einen UnitTest

› Verweis im Testprojekt auf das zu testende Projekt hinzufügen

› Namensraum des zu testenden Projektes in der Testklasse angeben
  › `using TestDemo_Accounts;`

› Zu testende Klassen müssen `public` sein





Erstellen einer Test-Klasse

- Für jede zu testende Komponente/Klasse wird eine Test-Klasse erstellt
- Für jede Methode der zu testenden Klasse wird eine oder mehrere Test-Methoden erstellt
  - je nach Komplexität
- Jede Test-Methode sollte nach dem AAA-Muster auf gebaut sein
  - Arrange
  - Act
  - Assert

Beispiel für einen Test

```
[Fact]
public void DepositTest()
{
    // Arrange
    int current_balance = 10;
    int deposit = 10;
    int expected_balance = 20;
    CheckingAccount account = new CheckingAccount(current_balance);

    // Act
    account.Deposit(deposit);

    // Assert
    Assert.Equal(expected_balance, account.Balance);
}
```

Wichtige Assert-Methoden

Methode	Überprüft …
`Equal<T>(T, T)`	…mithilfe des Gleichheitsoperators, ob zwei angegebene generische Typdaten gleich sind. Schlägt fehl, wenn sie nicht gleich sind
`NotEqual<T>(T, T)`	… ob zwei angegebene generische Typdaten ungleich sind. Schlägt fehl, wenn sie gleich sind
`IsNull(object)`	… ob das angegebene Objekt den Wert null hat. Schlägt fehl, wenn es nicht den Wert null hat
`IsNotNull(object)`	… ob das angegebene Objekt nicht den Wert null hat. Schlägt fehl, wenn es den Wert null hat

Wichtige Assert-Methoden

Methode	Überprüft …
`True(result)` `False(result)`	… ob das Ergebnis true bzw. false ist
`IsType()`	… ob das angegebene Objekt eine Instanz des angegebenen Typs ist. Schlägt fehl, wenn der Typ nicht in der Vererbungshierarchie des Objekts festgestellt wird

Assert-Methoden

- to assert -> feststellen
- Assert-Methoden bieten verschiedene Überladungen
- Beispiel:
  - `Assert.True(x > 1, "Das Ergebnis ist falsch!");`
- Meldung wird ausgegeben, wenn der Test fehlschlägt
  - Hinweis: Nicht alle Methoden bieten eine Überladung für eine Textmeldung

Auf Ausnahmen prüfen

› Wird min. eine Ausnahme ausgelöst ist der Test erfolgreich

› Pro Test kann daher nur auf eine Ausnahme geprüft werden

```
[Fact]
public void InvalidWithdrawalTest()
{
    // Arrange
    int current_balance = 20;
    int withdraw = 50;
    CheckingAccount account = new CheckingAccount(current_balance);

    // Act / Assert
    Assert.Throws<ArgumentException>(() => account.Withdraw(withdraw));
}
```

Ausführen der Tests

› Die Ausführung der Tests wird durch den Test-Explorer von Visual Studio gesteuert