Unit Test

Erklärung

Unit-Tests stellen die unterste Testebene dar. Díese Tests werden auch als Komponententests bezeichnet. Das Testobjekt ist eine Klasse oder sogar nur eine Methode einer Klasse. Bei der nicht-objektorientierten Programmierung kann ein solcher Komponententest auch nur eine einzelne Funktion umfassen. Erst wenn durch den Test die fehlerfreie Funktion nachgewiesen wurde, kann diese in das System integriert werden. Unit-Tests sind das Basismerkmal der Testgetriebenen Entwicklung. Hier werden erst. Viele kleinere Testeinheiten geschrieben, auf deren Basis dann die eigentliche Entwicklung folgt.
Wir halten also fest: Unit-Tests sind eine Methode, das Verhalten einer Komponente isoliert ohne ihre Abhändigkeiten von anderen Komponenten zu überprüfen.

Achtung: Unit-Tests erleichtern Fehlerfindung, sind aber kein Beweis für komplette Fehlerfreiheit!

AAA-Prinzip

Mit dem AAA-Prinzip wird der Aufbau jedes Tests beschrieben. AAA ist ein Akronym und steht für Arrange, Act, Assert.

- Arrange
  - Einstellungen, die für die Test-Ausführungen benötigt werden
- Act
  - Ausführung der zu testenden Komponente
- Assert
  - Prüfung, ob der Test erfolgreich abgelaufen ist.

FIRST-Prinzip

Akronym für: Fast, Isolated, Repeatable, Self-Validating, Timely

- Fast
  - Der Test soll schnell sein. Große Softwarekomponenten können eine Vielzahl von Tests umfassen. Damit die Motivation zum regelmäßigen Testen bleibt, sollte ein einzelner Unit-Test maximal wenige Sekunden Laufzeit haben.
- Isolated
  - Alle Tests müssen unabhängig voneinander angestoßen werden können
- Repeatable
  - Unit-Test müssen jederzeit wiederholt werden können. Wenn Daten während einem Test geändert werden, müssen diese am Schluss wieder zurückgeändert werden
- Self-Validating
  - Der Test kann selbst bestimmen, ob er erfolgreich absolviert wurde. Menschliche Interpretation ist nicht notwendig
- Timely
  - Source-Code und Unit-Test sollten gleichzeitig entwickelt werden. Gegebenfalls kann der Test auch vor der Methode geschrieben werden

Beispiele

Die Methode soll testen, ob die überschriebene ToString-Methode das richtige Ergebnis zurückliefert.

```
public override string ToString()
{
    string done = (Done == true ? "1" : "0");
    string date = Date.ToString("yyyy-MM-dd");
    return Convert.ToString(Id) + ";" + Name + ";" + date + ";" +
Convert.ToString(Type.getInt()) + ";" + done;
}
```

```
[TestMethod()]
public void ToDoItemToString()
{
    ToDoItem item = new ToDoItem(new string[] { "1", "ToDo1", "2018-11-03", "0", "1"
});
    string expected = "1;Test1;2018-11-03;0;1";
    Assert.AreEqual(expected, item.ToString());
}
```

Warum ist in dieser Situation zu beanstanden?
Ganz einfach. Wenn im Text „ToDo1“ ein Strichpunkt vorkommt, werden die Daten falsch abgespeichert.
Jetzt gibt es mehrere Möglichkeiten:

- Strickpunkte verhindern
- Strichpunkte ersetzen

Aus UX-Sicht ist es definitiv besser, die Strichpunkte in andere Zeichen zu verwandeln. Das ist natürlich mehr Programmieraufwand. Wir müssen nicht nur die echte Methode ändern sondern auch den Testfall.