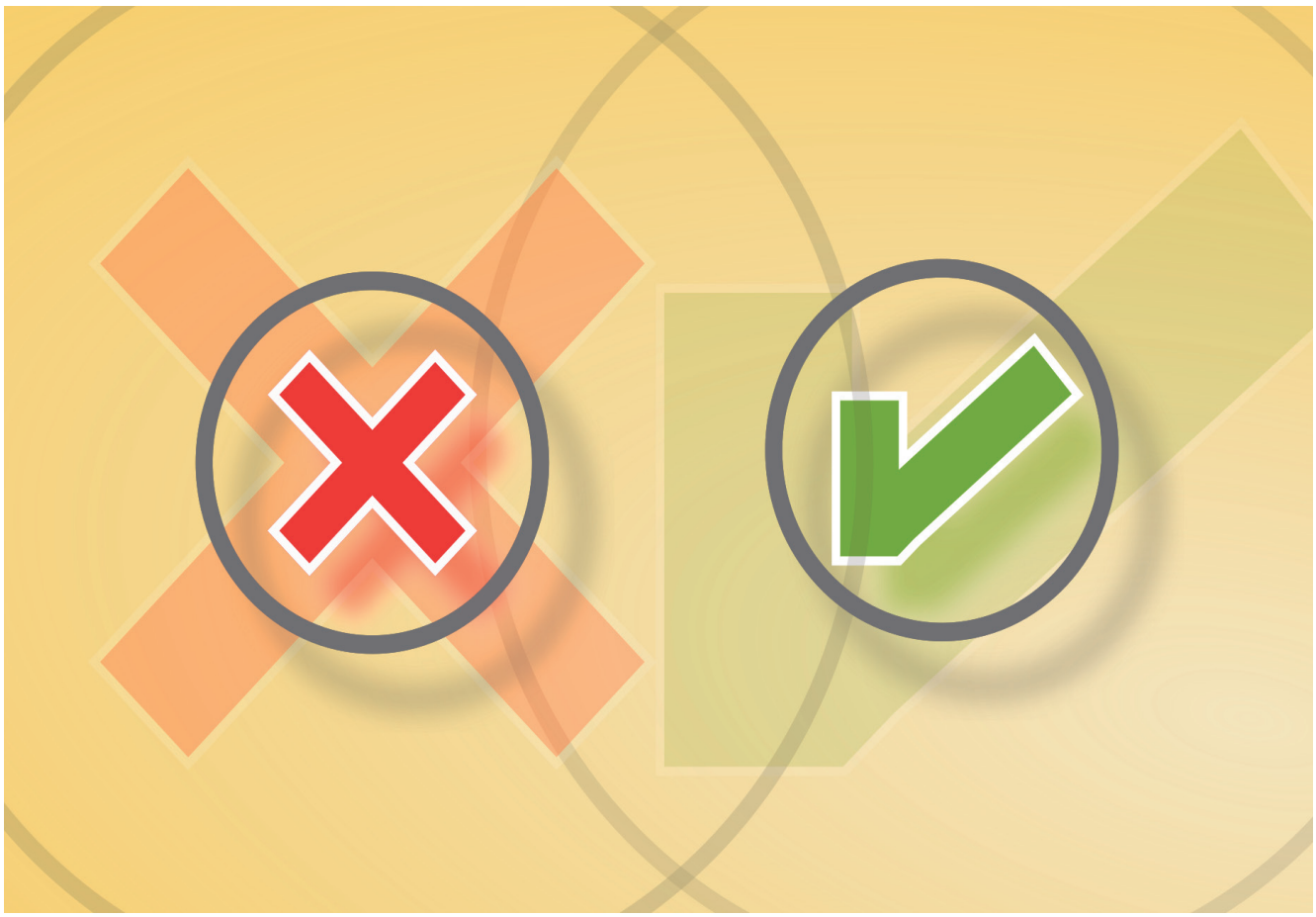

DAS ASSERTION-FRAMEWORK SHOULDLY

Entscheidend ist, was hinten rauskommt

Beim Schreiben von Tests kommt es unter anderem auf die Lesbarkeit und gute Fehlermeldungen an. Die Bibliothek Shouldly verspricht beides.

Tests zu schreiben ist nicht unbedingt die spannendste Aufgabe. Damit ist nicht die Überlegung gemeint, wie diese inhaltlich aufgebaut sein sollten. Das kann im Einzelfall durchaus sehr spannend sein und jede Menge Hirnschmalz erfordern.

Gemeint ist vielmehr die reine Implementierung der Vergleiche, zum Beispiel, ob ein berechneter Wert einem erwarteten entspricht. Das artet oft in viele, sehr ähnliche Anweisungen aus, deren Implementierung trotzdem Konzentration erfordert. Denn Fehler an dieser Stelle können den eigentlichen Test ad absurdum führen.

Aus dem Grund haben sich in der Vergangenheit zahlreiche Bibliotheken um diesen Anwendungsfall herum gebildet. Sie versprechen Verbesserungen beim Schreiben von Tests. Oftmals liegt der Fokus auf einer verbesserten Schreibweise, die einem natürlichsprachigen Text näherkommen soll. Das geht in der Regel mit verbesserten Fehlermeldungen einher.

Was ist Shouldly?

An dieser Stelle tritt Shouldly auf. Die Bibliothek soll das Schreiben von Tests vereinfachen. Konkret geht es um die Vergleiche von berechneten und erwarteten Werten.

Diese Vergleiche können, je nach Anwendungsfall, beliebig komplex und umfangreich sein. Gleichzeitig ist es nicht sonderlich spannend, den für die Vergleiche notwendigen Code zu schreiben, da oftmals in die Vorbereitung und die Durchführung der Tests wesentlich mehr Zeit investiert werden muss.

Jedoch kommt dem Code für Vergleiche eine große Bedeutung zu. Shouldly bietet zur Fehlervermeidung an der Stelle eine vereinfachte Syntax an. Des Weiteren nutzt die Bibliothek ihre Andersartigkeit dafür, bessere Meldungen auszugeben – sowohl im Erfolgsfall als auch im Fehlerfall, um die betreffende Stelle und die Ursache im Code schneller ausfindig machen zu können.

Shouldly ist Open Source und steht unter der BSD-Lizenz. Den Quelltext gibt es in einem Repository [1] auf GitHub. Die letzten Änderungen liegen schon etwas zurück, allerdings ist das bei einer Bibliothek wie Shouldly nicht allzu schlimm. Steht das API erst einmal, ergeben sich nur in unregelmäßigen Abständen Änderungen.

Installation

Die Installation von Shouldly ist denkbar einfach. Erfreulicherweise gibt es ein NuGet-Paket [2] unter dem gleichen Namen. Das Einbinden in eigene Projekte stellt damit keine Hürde dar. Aktuell ist Version 3.0.0 aus dem Januar 2018. Lauffähig ist die Bibliothek unter dem .NET Framework 4.0 und 4.5.1 sowie .NET Standard 1.3. Abhängigkeiten sind bei den beiden Erstgenannten keine vorhanden. Bei .NET Standard sind wenige notwendig.

Gibt es so etwas nicht schon?

Beim Durchsehen der Features von Shouldly fragt man sich unwillkürlich, ob es so etwas nicht schon gibt. Ein berechtigter Gedanke, da Funktionen wie die von Shouldly selten neu erdacht, sondern von anderer Seite beeinflusst werden. Die Bibliothek FluentAssertions geht in eine ähnliche Richtung. Sie wurde ebenfalls im Rahmen dieser Serie vorgestellt [3].

Die beiden Bibliotheken gehen die Sache allerdings etwas unterschiedlich an, was bereits beim Blick in die Dokumentation ersichtlich wird. FluentAssertions nähert sich der Thematik eher aus der Sicht der Datentypen.

Es werden zum Beispiel Operationen für Zeichenketten, numerische Datentypen, Booleans und Nullables beschrieben. Bei Shouldly erfolgt die Aufteilung nach der Art des Vergleichs.

Denn wie bei FluentAssertions auch, geht es um das Schreiben der Bedingungen in Tests, die vereinfacht werden sollen. In der Dokumentation gibt es eine Kategorie für Equality, Enumerable, Dictionary und so weiter. Primär geht es daher um die Art der Vergleiche. Auf Datentypen bezogene Kategorien gibt es aber dennoch.

Der Aufbau von Shouldly

Shouldly bietet zahlreiche Operationen für Vergleiche an, die bei Tests erforderlich sind. Darunter fallen insbesondere die Kategorien:

- Equality
- Enumerable
- String
- Dictionary
- Exceptions
- Tasks/Async
- Dynamic

In diesen jeweiligen Kategorien gibt es verschiedene Methoden. Bei Equality sind das unter anderem *ShouldBeGreaterThan* und *ShouldBeInRange* inklusive der jeweiligen Verneinungen. Bei Zeichenketten, um eine weitere Kategorie zu bemühen, gibt es Methoden wie *ShouldEndWith*, *ShouldContain* oder *ShouldBeNullOrEmpty*.

An den Beispielen wird deutlich, dass alle Operationen mit dem Wort *Should* beginnen – das Markenzeichen der Bibliothek Shouldly.

Bei FluentAssertions gibt es das nicht so explizit. Dort ist ein Vergleich eher so aufgebaut, wie er gesprochen werden würde. Allerdings gibt es dadurch ebenfalls das Wort *Should*, da ein Satz ansonsten keinen Sinn ergeben würde, wie im folgenden Beispiel zu FluentAssertions:

```
theString.Should().NotBeNull();
```

In Shouldly sieht diese Anweisung wie folgt aus:

```
theString.ShouldNotBeNullOrEmpty();
```

Da es keine einzelne Methode für die Abfrage auf nicht Null gibt, wird bei Shouldly gleichzeitig immer auch auf einen Leerstring verglichen. Welche Variante besser gefällt, ist Geschmackssache. Das Resultat unterscheidet sich nicht.

Einige Beispiele

In der Regel erklären sich die Operationen von Shouldly selbst. Ein klassisches Beispiel ist die folgende Anweisung:

```
1.ShouldBeGreaterThan(2);
```

Auf diese Weise ergibt der Aufruf zwar nicht viel Sinn, da die linke Seite auf einen festen Wert gesetzt ist, sie zeigt aber, wie Shouldly zum Einsatz kommt.

Der Kern sind Erweiterungsmethoden für die verschiedensten Datentypen. Dadurch können die Operationen ohne große Mühen direkt verwendet werden. Im obigen Beispiel ist das Ergebnis eindeutig, da eins definitiv nicht größer als zwei ist.

Shouldly sorgt, wie bereits weiter oben erwähnt, für ordentlich lesbare Meldungen. In diesem Fall sieht die Fehlermeldung wie folgt aus:

```
Message: Test method Twainsoft.Articles.DNP.Shouldly-
Tests.Tests.ShouldBeGreaterThan threw exception:
Shouldly.ShouldAssertException: 1
    should be greater than
2
    but was not
```

Die Formatierung ist nicht verändert, sondern so von Shouldly gewählt. Eine weitere Operation, ebenfalls aus der Kategorie Equality, ist die Methode *ShouldBeOneOf*, die wie folgt aufgerufen wird:

```
objectA.ShouldBeOneOf(listOfObjects.ToArray());
```

Es wird geprüft, ob das Objekt auf der linken Seite, in diesem Fall durch die Variable *objectA* repräsentiert, in dem Array vorhanden ist, das als Parameter übergeben wird.

Diesem Vorgehen folgen alle weiteren Operationen beziehungsweise Methoden von Shouldly. Interessant ist in diesem Zusammenhang noch die Methode *ShouldBeAssignableTo*, die einen Typcheck ausführt.

Ob eine Exception geworfen wird oder nicht, lässt sich ebenfalls überprüfen. Ersterer Fall über die *ShouldThrow*-Methode, die wie folgt zum Einsatz kommt:

```
Should.Throw<DivideByZeroException>(() => {
    var denominator = 0;
    var y = 100 / denominator;
});
```

In diesem einfachen Fall stimmt die Behauptung, sodass ein so aufgebauter Test positiv verläuft, denn die *DivideByZeroException* wird tatsächlich geworfen.

Eine weitere interessante Methode ist *CompleteIn*. Mit ihr kann geprüft werden, ob ein Stück Code in einer vorgegebenen Zeit ausgeführt wurde:

```
Should.CompleteIn(() =>
{
    Thread.Sleep(2000);
    // ...
}, TimeSpan.FromSeconds(1));
```

Bei dem oben gezeigten Code ist das ganz offensichtlich nicht der Fall, da ein *Thread.Sleep* zu Demonstrationszwecken verbaut ist. Beim Ausführen eines so präparierten Tests erscheint auch prompt die korrekte Fehlermeldung:

```
Message: Test method Twainsoft.Articles.DNP.Shouldly-
Tests.Tests.CompleteIn threw exception:
Shouldly.ShouldCompleteInException:
Task
    should complete in
00:00:01
    but did not ---> Shouldly.ShouldlyTimeoutException:
Timeout für den Vorgang wurde überschritten.
```

Mehr lässt sich über die Shouldly-Methoden kaum schreiben. In der überwiegenden Anzahl der Anwendungsfälle ergibt es sich ganz automatisch, wie vorzugehen ist. Wenn doch einmal Fragen offenbleiben, hilft die Dokumentation zuverlässig weiter.

Die Dokumentation

Die Dokumentation von Shouldly ist ganz allgemein in einem guten Zustand. Sie wirkt aufgeräumt, erklärt die vorhandenen Operationen/Methoden und verdeutlicht diese mit Beispielen in C#.

Zusätzlich werden sehr übersichtlich die Fehlermeldungen angegeben, die eine Test-Methode verursacht, wenn einmal etwas schiefläuft.

Das gilt zumindest für die Dokumentation, die unter [4] zu finden ist. Einem Issue-Eintrag [5] auf GitHub ist zu entnehmen, dass die Dokumentation zu readthedocs.org [6] migriert werden soll.

Allerdings stammt der Eintrag aus dem September 2015 und der letzte Kommentar vom April 2017. Die Liste, die bereits migrierte Teile der Dokumentation aufführt, scheint nicht mehr aktuell zu sein, da bereits mehr Teile migriert wurden, als es laut der Liste den Anschein hat.

Das ist der Grund, warum die Suche nach Informationen manchmal gestückelt abläuft, da die Dokumentation aufgeteilt ist. Aus Sicht der aktuellen Qualität ist die alte Version der Dokumentation etwas ausführlicher.

Die Erklärung der Fehlermeldungen fehlt auf der neuen Webseite. Bleibt zu hoffen, dass diese ebenfalls mit übernommen werden.

Fazit

Shouldly ist eine der kleineren Bibliotheken, die von außen sehr unscheinbar daherkommen, die Arbeit aber trotzdem enorm erleichtern können. In diesem Fall das Schreiben von Bedingungen beziehungsweise Vergleichen innerhalb von Tests. Der Funktionsumfang ist passend, die Dokumentation verständlich, wenn auch etwas verteilt, und die Beispiele helfen, sich schnell in die Thematik einzuarbeiten.

Ob nun FluentAssertions oder Shouldly die bessere Wahl ist, muss jeder für sich selbst entscheiden. Das Ziel ist bei beiden Bibliotheken identisch, lediglich die Herangehensweise ist eine andere.

Alles in allem verdient sich Shouldly ein „Sehr gut“ inklusive Empfehlung. Die kleineren Makel in der Dokumentation können das eindeutig positive Gesamtbild definitiv nicht mehr trüben. ■

[1] Shouldly auf GitHub, www.dotnetpro.de/SL1807Frameworks1
[2] Shouldly in der NuGet-Galerie, www.dotnetpro.de/SL1807Frameworks2
[3] Lass es fließen, Fabian Deitelhoff, dotnetpro 3/2015, Seite 78 ff., www.dotnetpro.de/A1503Frameworks
[4] Die ältere Dokumentation von Shouldly, www.dotnetpro.de/SL1807Frameworks3
[5] GitHub-Issue-Eintrag zur Migration der Dokumentation, www.dotnetpro.de/SL1807Frameworks4
[6] Die neuere Dokumentation von Shouldly, www.dotnetpro.de/SL1807Frameworks5

Fabian Deitelhoff
promoviert am Graduiertenkolleg „User-Centred Social Media“ im Themenumfeld der Code Comprehension. Zudem ist er Autor, Entwickler, Trainer und Gründer von www.brickobotik.de. Erreichbar über www.fabiandeitelhoff.de oder **@FDeitelhoff**

dnpCode A1807Frameworks