SYP - Handout

Table of Contents

User Stories

Was ist einer User-Story?

Eine Softwareanforderung, welche in der Alltagssprache formuliert wird aus Sicht des Endnutzers. User-Stories werden oft mit den Kunden gemeinsam verfasst. Sie werden kurz gehalten und umfassen normalerweise nicht mehr als zwei Sätze.

Warum User-Story basiertes Testen?

In großen Firmen mit vielen Programmierern, wurde die vom Kunden gewünschte Software oftmals zu sehr ausgebaut, sodass die Software viel zu groß wurde, nicht mehr den Anforderungen entsprach oder sogar gar nicht mehr brauchbar war.

Durch die Verwendung von User-Stories, werden die Grundfunktionalitäten der Software definiert und die Software wird so klein wie möglich gehalten.

Beispiel einer User-Story

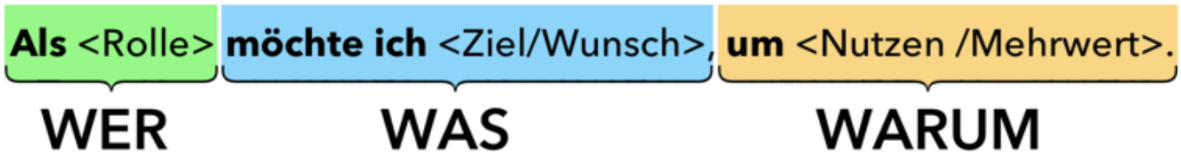

Figure 1. Beispiel einer User-Story

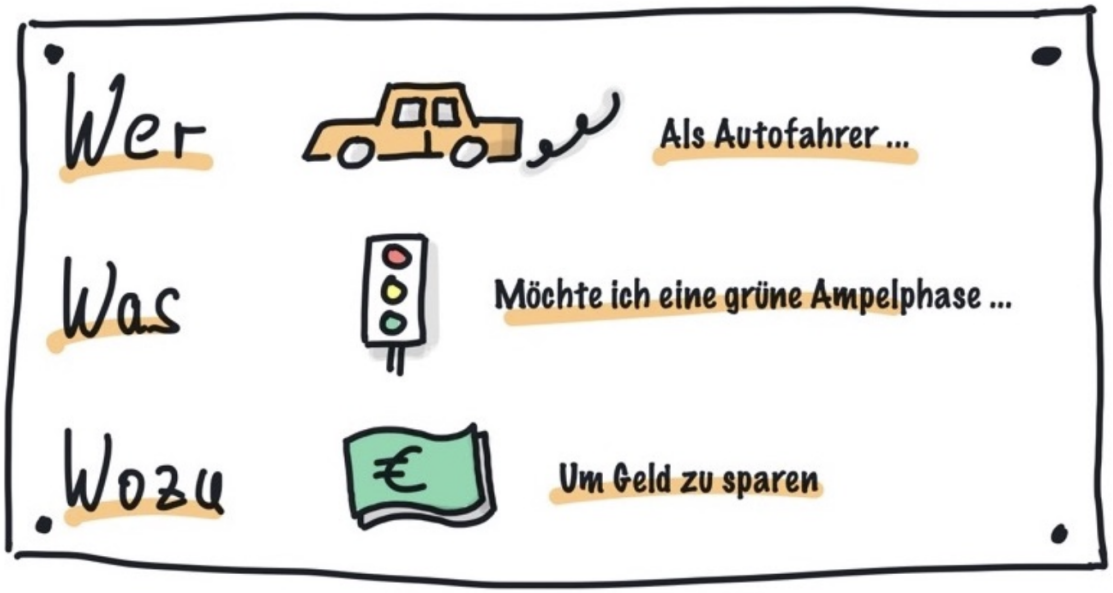

Figure 2. Beispiel einer User-Story

Gherkin

Ist eine Beschreibungssprache. Sie wird in natürlicher Sprache formuliert. Es gibt lediglich ein paar wichtige Schlüsselwörter. Wird von Testing Frameworks wie Karate und Cucumber. Dabei werden Aufgaben, Ziele, Ergebnisse werden in Textform gebracht und können später als automatisierte Tests ausgeführt werden.

Feature vs Szenario

Ein Feature stellt eine User Story dar.

- Ein Feature ist eine Sammlung von Szenarien.
- Ein Szenario ist eine Sammlung von Schritten und beschreibt das Ergebnis des Testfalls.
- Ein Schritt ist eine Aktion, die ausgeführt werden soll.

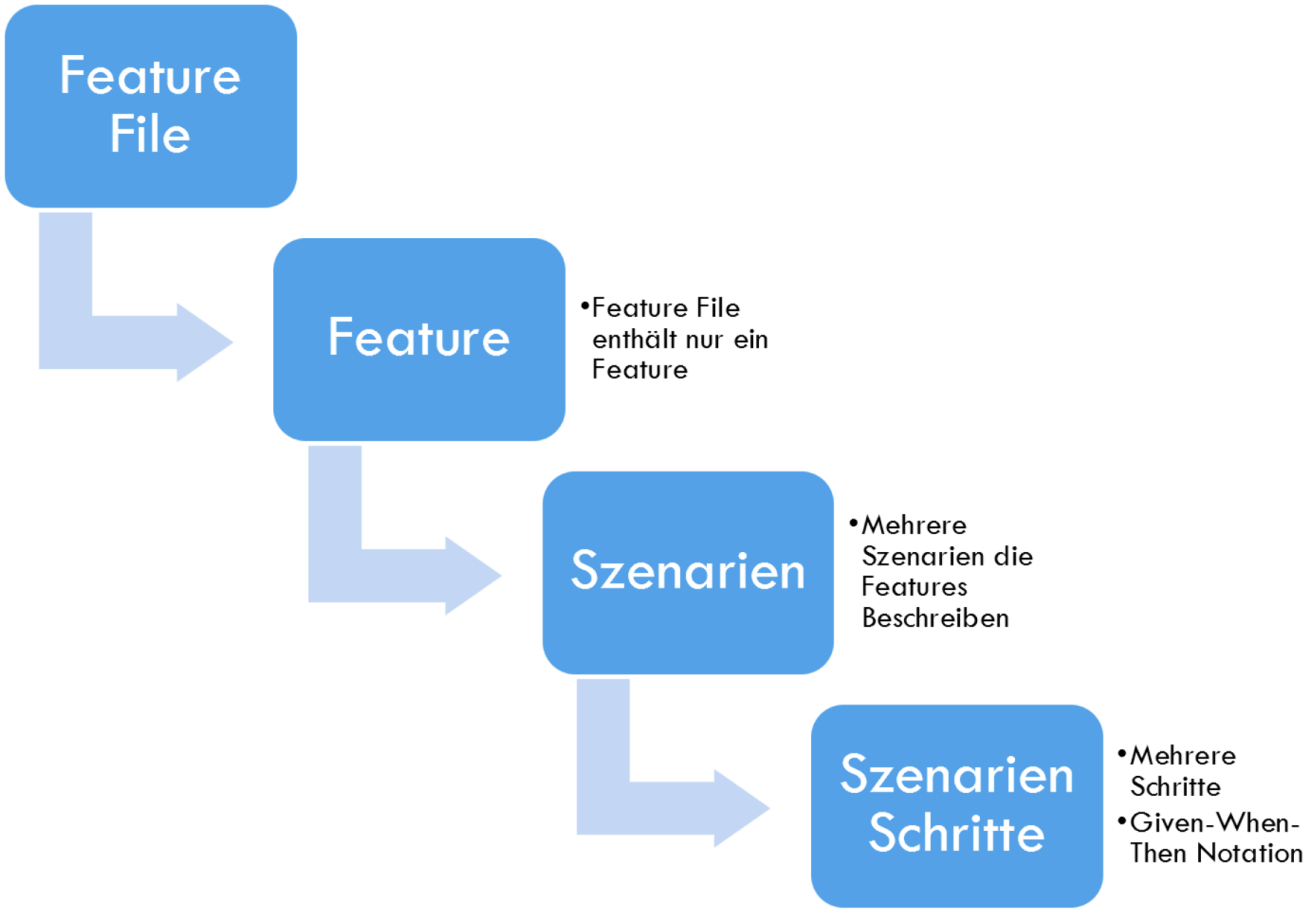

Figure 3. Grundlegende Struktur

Feature: **Standard-Rechenoperationen**
Als Benutzer möchte ich die Standard-Rechenoperationen: Addition, Subtraktion, Multiplikation und Division verwenden können

Szenario: **Addieren zweier Zahlen**
Given: **Eine Addition**
When: **Ich wähle als erste Zahl 3**
And: **Ich wähle als zweite Zahl 9**
Then: **Ich sollte das Ergebnis 12 erhalten**

Szenario: **Subtrahieren zweier Zahlen**
Given: **Eine Subtraktion**
When: **Ich wähle als erste Zahl 9**
And: **Ich wähle als zweite Zahl 3**
Then: **Ich sollte das Ergebnis 6 erhalten**

Figure 4. Beispiel eines Features, Szenarios und Schritte

Testing pattern (AAA)

Beim Verwenden des AAA-Patterns werden die Tests in drei Abschnitte unterteilt:

- Arrange: Hier werden alle notwendigen Daten und Objekte initialisiert.
- Act: Hier wird die Aktion ausgeführt, die getestet werden soll.
- Assert: Hier wird überprüft, ob das Ergebnis des Tests mit dem erwarteten Ergebnis übereinstimmt.

Dieses Muster ist auch unter dem Given-When-Then-Pattern bekannt. Dieses wird von Gherkin verwendet.

Given-When-Then

- Given: Vorbedingung eines Beispiels
- When: Aktion, die ausgeführt wird
- Then: Ergebnis, das erwartet wird
- And/But: Ergänzen andere Schlüsselwörter

Kann in über 70 (natürlichen) Sprachen beschrieben werden

Beispiel

Gherkin

```
Feature: Standard arithmetic operations
    As a user I would like to be able to use the standard arithmetic operations:
addition, subtraction, multiplication and division

  Background:
    Given I initialise a calculator

  Scenario: Adding two integer numbers
    Given an integer operation '+'
    When I enter the first number 4
    And I enter the second number 5
    Then the operation results 9

  Scenario: Subtracting two integer numbers
      Given an integer operation '-'
      When I enter the first number 7
      And I enter the second number 5
      Then the operation results 2
```

Um die Tests auszuführen wird Cucumber verwendet.

Cucumber: Die obigen Testfälle können damit getestet werden

```
@Given("I initialise a calculator")
    public void givenIInitialiseACalculator() {
        c = new Calculator();
    }

    @Given("an integer operation {string}")
    public void givenAnIntegerOperation(String op) {
        switch (op) {
```

```
            case "+"    ->  operation = "+";
            case "-"    ->  operation = "-";
            case "*"    ->  operation = "*";
            case "/"    ->  operation = "/";
            default     ->  fail();
        }
    }

    @When("I enter the first number {int}")
    public void whenIProvideFirstNumber(int number1) {
        this.number1 = number1;
    }
    @And("I enter the second number {int}")
    public void whenIProvideSecondNumber(int number2) {
        this.number2 = number2;
    }

    @When("the second number cannot be {int}")
    public void the_second_number_cannot_be(int zero) {
        if(number2 == zero) {
            Assertions.fail();
        }
    }

    @Then("the operation results {int}")
    public void thenTheOperationEvaluatesTo(int expected) {
        int result = 0;
        switch (operation) {
            case "+"    ->  result = c.addition(number1, number2);
            case "-"    ->  result = c.subtraction(number1, number2);
            case "*"    ->  result = c.multiplication(number1, number2);
            case "/"    ->  result = c.division(number1, number2);
            default     ->  fail();
        }
        assertEquals(expected, result);
    }
```

In den Assoziationen muss dasselbe stehen wie in den Schritten der Szenarien im .feature-file

Naming Conventions

Es gibt einige Empfehlungen für die Namensgebung von Tests:

- Der Testname sollte eine bestimmte Anforderung ausdrücken
- Der Testname könnte die erwartete Eingabe oder den erwarteten Zustand und das erwartete Ergebnis für diese Eingabe oder diesen Zustand enthalten
- Der Testname sollte als Aussage oder Tatsache dargestellt werden, die Arbeitsabläufe und

Ergebnisse ausdrückt

- Der Testname könnte den Namen der getesteten Methode oder Klasse enthalten
- Testnamen sind situationsabhängig

Test Method Naming

- MethodName_StateUnderTest_ExpectedBehavior
  - cons: should be renamed if method change name
  - example: isAdult_AgeLessThan18_False
- MethodName_ExpectedBehavior_StateUnderTest
  - cons: should be renamed if method change name
  - example: isAdult_False_AgeLessThan18
- testFeatureBeingTested
  - cons: “test” prefix is redundant
  - example: testIsNotAnAdultIfAgeLessThan18
- FeatureToBeTested
  - cons: no clue what result is expected from name
  - example: IsNotAnAdultIfAgeLessThan18
- Should_ExpectedBehavior_When_StateUnderTest
  - cons: duplicates `should` and `when`, long name
  - example: Should_ThrowException_When_AgeLessThan18
- When_StateUnderTest_Expect_ExpectedBehavior
  - cons: duplicates `when` and `expect`
  - example: When_AgeLessThan18_Expect_isAdultAsFalse
- Given_Preconditions_When_StateUnderTest_Then_ExpectedBehavior — Behavior-Driven Development (BDD)
  - cons: duplicates `given`, `when`, `then`; really long names
  - example: Given_UserIsAuthenticated_When_InvalidAccountNumberIsUsedToWithdrawMoney_Then_TransactionsWillFail
- Prof. Bauers Favorite
  - It_Should_Return_1_Given_1

Link zum Artikel: https://medium.com/@stefanovskyi/unit-test-naming-conventions-dd9208eadbea