



# Protokoll

## Software Testing mit JUnit

**SEW**
**3AHIT 2015/16**

**Markus Reichl**

**Version 0.2s**

## Inhaltsverzeichnis



# 1 Einführung

Software Tests dienen zur Fehlerfindung innerhalb von Software Projekten.
Es gibt verschiedene Arten von SW-Tests, wir haben besprochen:

- Blackbox
- White
- Greybox

Zusätzlich haben wir noch einen Schreibtischtest angesprochen welcher jedoch mehr als Konzept dient.

## 1.1 Ziele

Ziel unserer Aufgabe war es sich die Grundlagen von Unit-Testing mittels JUnit anzueignen.

## 1.2 Voraussetzungen

Unit-Tests werden genutzt um einzelne Teile einer Software zu testen. Ziel ist es Erwartungen zu überprüfen und dadurch Fehler oder Unklarheiten zu finden.

## 1.3 Aufgabenstellung

a) Implementieren Sie den angegebenen Testfall mit JUnit, so dass kein weiterer
manueller Vergleich mehr notwendig ist. Nennen Sie die Testklasse (Klasse die den zu implementierenden Testfall enthält) StringTokenizerTest, und legen Sie diese Klasse im Package example1 ab.

b) Implementieren Sie einen weiteren Testfall, der das Verhalten der Methode nextToken() zeigt, wenn keine weiteren Token vorhanden sind.
Hinweis: Das erwartete Verhalten ist – wie in der Javadoc-Dokumentation angegeben
– eine NoSuchElementException. D.h. der Testfall geht gut, wenn
genau diese Exception geworfen wird.

OPTIONAL
c) Implementieren Sie drei weitere Testfälle für die Klasse StringTokenizer, die zusätzliche (Sonder-)Fälle (z.B. ein String der ausschließlich aus Trennzeichen besteht) und andere Methoden (z.B. countTokens()) als Ziel haben.

Wie Sie sehen sind weit mehr als die insgesamt fünf Testfälle notwendig, um sämtliche Kombinationen von Methoden und Sonderfällen vollständig zu testen. Treffen Sie daher eine möglichst sinnvolle Auswahl an Testfällen die Sie implementieren.

# 2 Theorie

## 2.1 Black-Box Test

Eine Test-Methode bei welcher der Tester ohne Wissen über Struktur, Design oder Implementierung, die Funktionsweise eines Programms testet.

- **Erkennung von:**
  - Falschen oder fehlenden Methoden
  - Fehlern im (User-) Interface
  - Fehlern in der Datenstruktur
  - Fehlern beim Zugriff auf externe Ressourcen
  - Fehlern in der Umsetzung der (vorgegebenen) Funktion
  - Fehlern in der Performance und Antwortzeit
  - Fehlern bei der Initialisierung und Beendigung der Methode
- **Techniken**
  - Äquivalenz-Prüfung (Gültig vs. Ungültig)

    Bei dieser Technik wird der Wertebereich für den Input in gültige und ungültige Werte geteilt. Anschließend werden repräsentative Werte für beide Bereiche ausgewählt und getestet.
  - Grenzwert-Analyse (Grenzen überschreiten)

    Bei dieser Technik werden die Grenzen für Inputwerte betrachtet. Hier werden Werte auf der Grenze und knapp innerhalb bzw. außerhalb der Grenze liegende Werte für den Test herangezogen.
  - Ursache/Wirkungs-Analyse (Ursache vs. Wirkung)

    Hier wird aus der Ursache (Konditionen der Eingabe) und den Wirkungen (Konditionen der Ausgabe) ein Ursache-Wirkungsdiagramm erstellt. Daraus werden die notwendigen Test-Klassen erstellt.

- **Vorteile**
    - Die Tests werden aus User-Sicht durchgeführt und helfen bei der Aufdeckung von Diskrepanzen zur Spezifikation.
    - Die Tester müssen die Programmiersprache nicht beherrschen.
    - Die Tester kennen die Implementierung nicht.
    - Die Tests können von Personen umgesetzt werden, die selber nicht an der Erstellung beteiligt waren um eine Befangenheit zu vermeiden.
    - Die Test-Cases können bereits mit der Spezifikation (Pflichtenheft) erstellt werden.
- **Nachteile**
    - Ein großer Teil der Software bleibt ungetestet (Software-Coverage)
    - Ohne eine genaue Spezifikation ist es schwierig aussagekräftige Test-Cases zu erstellen.
    - Tests könnten redundant sein.

## *2.2* White-Box Test

Eine Test-Methode bei welcher der Tester über Struktur, Design und Implementierung eines Programms Bescheid weiß.

- **Erkennung von**
    - Falschen oder fehlenden Methoden
    - Fehler im (User-) Interface
    - Fehler in der Datenstruktur
    - Fehler beim Zugriff auf externe Ressourcen
    - Fehler in der Umsetzung der (vorgegebenen) Funktion
    - Fehler in der Performance und Antwortzeit
    - Fehler bei der Initialisierung und Beendigung der Methode
- **Techniken**
    - Ein Tester – meist der Entwickler – untersucht den implementierten Code legt alle möglichen Werte (gültig und ungültig) für den Input fest und vergleicht den Output mit den erwarteten Werten.

- **Vorteile**
    - Die Tests können bereits frühzeitig begonnen werden, ohne Rücksicht auf vorhandene GUIs.
    - Die Tests können den Ablauf des Source Codes gründlicher testen.
    - Es ist möglich den gesamten Pfad zu testen. (Software-Coverage).
- **Nachteile**
    - Tests können sehr komplex werden und es werden teure Spezialisten benötigt, welche über Wissen zur Implementierung und der Programmiersprache verfügen.
    - Änderungen in den Test-Scripts können durch häufige Änderungen in der Implementierung aufwendig werden.
    - Es kann vorkommen, dass Tests aufgrund noch Fehlender Programmteile nicht durchführbar sind.
    - Es können weitläufige Fehler auftreten welche durch Tiefentests nicht aufgefallen sind.

# 3 Vorgehen

Zu Beginn habe ich mich mithilfe des „Getting Started" Bereichs auf junit.org über die Grundlagen von Unit Testing informiert. Diese habe ich dann anhand des Beispiels aus dem Tutorial angewandt und somit das Grundgerüst für die Aufgabe a) gestellt. Der Test war erfolgreich und ich habe das Prinzip an einer weiteren Methode getestet.

Als nächstes überlegte ich mir wie auf eine Exception zu testen wäre, wenn die unser erwartetes Ergebnis ist. Hierzu bediente ich mich einer „try {} catch" welche im Falle einer „NoSuchElementException" ein return gibt (den Test erfolgreich beendet) und in einem anderen Fall ein „fail()" ausführt.

Zuletzt wurden die oben angewandten Vorgehensweisen auf 3 weitere Fälle angewandt und das Projekt dokumentiert.

# 4 Ergebnisse

Die Tests werden erfolgreich durchgeführt und tuen was erwartet wird.

Das hierzu genutzt Eclipse Project findet sich nun in meinem SEW Github Repository unter: *https://github.com/mreichl-tgm/SEW_15-16.git*