EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Wilhelm-Schickard-Institut für Informatik
Programmiersprachen und Übersetzer

End-to-End-Testen von Web-Applikationen mit Webtests

Intro

Matthias Hirzel

22.02.2016

Unit Testing - Was ist das?

"Unit-Tests (=Komponententests) überprüfen, ob die von den Entwicklern geschriebenen Komponenten so arbeiten, wie diese es beabsichtigen. In agilen Methoden wird zur Qualitätssicherung eine sehr häufige Ausführung der Komponententests angestrebt. [. . .]“ [1]

"Im Unit Test werden kleinere Programmteile in Isolation von anderen Programmteilen getestet. Die Granularität der unabhängig getesteten Einheit kann dabei von einzelnen Methoden über Klassen bis hin zu Komponenten reichen. Unit Tests sind in den überwiegenden Fällen White-Box-Tests. Das bedeutet, der Test kennt die Implementierungsdetails seines Prüflings und macht von diesem Wissen Gebrauch."[4]

Guideline für gute JUnit-Tests [1]

- isoliert/unabhängig
- testen eine fest definierte Eigenschaft
- kurz und verständlich
- laufen automatisiert $\Rightarrow$ Framework
- sollten nie redundant sein
- testen nie getter/setter oder (private) Hilfsmethoden
- sind um eine Menge von Testobjekten (Testfixture) aufgebaut
- sollten gemäß TDD zuerst geschrieben werden (test-first)

Unit-Tests mit Java: JUnit

Wichtige Elemente:

- Annotationen für Test:
  `@Test, @Before, @After, @BeforeClass, @AfterClass`
- Assertions:
  `assertEquals(<expectedValue>, <actual>);`
  `assertTrue(<actual>); assertFalse(<actual>);`
- Exceptions: `@Test(expected = XYException.class)`

Code-Beispiel

Advanced Unit Testing: Parametrisierter Test

Code-Beispiel

End-to-End-Testen mit Webtests - Was ist das?

Unter Web Testing versteht man das Testen der korrekten Funktionalität von Web-Applikationen.

> *"Web application testing, a software testing technique exclusively adopted to test the applications that are hosted on web in which the application interfaces and other functionalities are tested."*

Weitere Elemente

- Load Testen

  Security Testen

  ...

End-to-End-Testen mit Webtests - Wofür?

- Inhalte, Elemente, Komponenten von Webseiten/ Web-Applikationen
- Browser Compatibility $\Rightarrow$ bei uns: nicht relevant da Fokus auf Firefox
- Browsing/Naviagation Tests
- Festlegen von User Stories
- Sicherstellen einer bestimmten Funktionsweise aus Sicht des Benutzers

$\Rightarrow$ Abnahmetests

$\Rightarrow$ Archivieren von Bugs

Idenitifikation und Zugriff auf Elemente einer Webseite

- gute Selektoren für Elelemnte:
  - ✓ id, class, name
  - ✓ css
- schlechte Selektoren:
  - ✗ xPath, absolute Pfadangaben
  - ✗ Position über Pixel

$\Rightarrow$ verwende Firebug (Plugin für Firefox)

Guidelines für gute Webtests - Übersicht

Für Webtests gelten ähnliche nicht-funktioale Eigenschaften wie für normalen Programm-Code:

- adaptability, maintainability, reusability
- understandability
- stability

Konsequenzen:

- "DRY": Don’t repeat yourself [2]
- kurz, einfach (max 3 Minuten Laufzeit)
- decken eine User Story ab
- sind modular [2] $\Rightarrow$ Wiederverwendung als Subtests

$\Rightarrow$ Ein Test hat eine klare Aufgabe, die in einem Satz definiert werden kann. Ist dies nicht möglich, muss der Test aufgeteilt werden. [3]

Detail: "DRY" - Don’t repeat yourself [3]

- Vermeide doppelten Code
- Teste nicht "nebenbei"

$\Rightarrow$ fokusiere eigentliche Aufgabe

✓ modulare Tests
✓ (möglichst) kein duplizierter Code
✓ Anpassbarkeit, Wiederverwenbarkeit, Wartbarkeit
✗ Abhängkeiten von anderen Tests erschwert Verständlichkeit
✗ Teammitglieder benötigen guten Überblick über Testbase

$\Rightarrow$ Realisierung im Code mit

- `setUp()`-/ `tearDown()`-Methoden (`@Before`/`@After`)
- Hilfsmethoden

Detail: Unabhängige Tests (Hermetic test) [3]

- keine Abhängigkeit vom Erfolg/Misserfolg anderer Tests
- keine Abhängigkeit vom Zustand des Vorgängertests

$\Rightarrow$ `setUp()`-/ `tearDown()`-Methoden (`@Before`/`@After`)

$\Rightarrow$ Zuverlässigleit, schnelles Finden des eigentlichen Fehlers

✓ Anwendung ist in sauberem Zustand

✓ Zustand der Web-Applikaiton wird selbst bestimmt vom Test

✓ Änderbarkeit der Ausführungsreihenfolge

✓ Paralleles Testen

✗ Wiederherstellen eines festgelegten Zustands zu Beginn

- $\Rightarrow$ Zeitaufwand
- $\Rightarrow$ Aufwand für Entwickler

Beispiel: Guter vs. schlechter Teststil

Code-Beispiel

Anlegen von Tests auf Github

- ✓ Vermeiden von doppelten Tests
- ✓ Tests mit klarer Aufgabe
- ✓ Leichtes Zusammenarbeiten im Team außerhalb der Uni
- ✓ Diskussion über Tests im Team und zwischen den Teams
- ✓ Test sharing für Reuse als Subtests

Hinweis:

- kein collective Code-Ownership

$\Rightarrow$ nur Requests stellen

$\Rightarrow$ nur Autoren bearbeiten Tests

Link zu Github folgt per mail

Thank you.

Contact:

Wilhelm-Schickard-Institut für Informatik
Programmiersprachen und Übersetzer
Sand 13, 72076 Tübingen
Telefon: +49 7071 29-75467
hirzel@informatik.uni-tuebingen.de

Bibliography I

[1] it-agile GmbH. Unit-Tests. http://www.it-agile.de/wissen/praktiken/agiles-testen/unit-tests/.

[2] Justin Klemm. 5 Best Practices for Automated Browser Testing. https://ghostinspector.com/blog/5-best-practices-automated-browser-testing/, June 2015.

[3] Dima Kovalenko. *Selenium Design Patterns and Best Practices*. Packt Publishing, 2014.

[4] Frank Westphal. *Testgetriebene Entwicklung mit JUnit-Tests & Fit*. dpunkt.verlag, 2006. Last access: 22. February 2016.

