Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Seminararbeit

im Bachelor-Studiengang Computer Science

Javascript Testing Frameworks

von

Per Bernhardt

Betreuer:	Christoph Tornau
Eingereicht am:	7. Dezember 2013

Eidesstattliche Erklärung

Per Bernhardt
Schumannstr. 115
53113 Bonn

Ich versichere an Eides statt, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

.....	.....
Ort, Datum	Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

GSM Global System for Mobile communication
NUA Not Used Acronym
UA Used Acronym
BUT Block Under Test

1 Einleitung

1.1 Problemstellung

Die Scriptsprache Javascript, die im Jahre 1995 von Netscape entwickelt wurde, erfreute sich in den letzten Jahren enormer Popularität im Bereich der Webseiten- Entwicklung. Dort wird sie vor allem zur Verbesserung der Interaktivität von Webseiten genutzt. In jüngster Vergangenheit beginnt Javascript nun auch bei der serverseitigen Entwicklung eine Rolle zu spielen. In beiden Bereichen hat sich die Komplexität von Javascript-Programmen so weit gesteigert, dass sie mit jeder anderen Programmiersprache vergleichbar geworden ist. Damit werden natürlich verschiedene Software-Engineering-Methodiken, die sich im Laufe der Zeit über alle Sprachen hinweg etabliert haben, auch für Javascript interessant. Dabei müssen die Lösungen für Javascript allerdings einige Besonderheiten abdecken, die mit der Sprache einhergehen: So dient Javascript nach wie vor vor allem für die Erstellung interaktiver Webseiten, die in verschiedensten Browsern und Plattformen funktionieren sollen. In all diesen Browsern gibt es verschiedene Javascript-Umgebungen, die sich zum Teil syntaktisch unterscheiden. Auch bei der serverseitigen Verwendung von Javascript, also mit Javascript-Interpretern oder Headless-Browsern, gibt es verschiedene Javascript-Engines, die nicht in allen Punkten identisch sind und somit beim Testen berücksichtigt werden müssen.

1.2 Zielsetzung

Ziel dieser Seminararbeit ist es hierbei, auf den Bereich des Testens einzugehen und einen Überblick über existierende Testframeworks zu geben.

1.3 Vorgehensweise

Zunächst werden dafür allgemeine Aspekte des Testens aufgezeigt. Anschließend werden diese genutzt, um verschiedene Tools und Ansätze für Javascript mit einander zu vergleichen. Anhand eines Fallbeispiels werden einige dieser Tools dann angewendet und veranschaulicht. Abschließend werden alle Ergebnisse zusammengefasst und ein Fazit gezogen.

1.4 Zitate

Google (2013) sagen hier steht ein Text.
Hidayat (2013) sagen hier steht ein Text.
Yahoo (2013) sagen hier steht ein Text.
Trostler (2013) sagen hier steht ein Text.
Wikipedia (2013c) sagen hier steht ein Text.
Wikipedia (2013b) sagen hier steht ein Text.
Wikipedia (2013a) sagen hier steht ein Text.
„Hier steht ein Text.“ (Johansen 2010)
Hier steht ein Text. (Vgl. Kleivane 2011)
Hier steht ein Text. (Koch 2001, S. 200)
Hier steht ein Text. (Pivotal-Labs 2013, S. 200)
Hier steht ein Text. (Nguyen 2013; Selenium-Project 2013)

1.4.1 Abkürzungen

Global System for Mobile communication (GSM)
NUA
Block Under Test

Used Acronyms (UAs)

2 Stand der Forschnung

Das Testen von Software dient vor allem der Vermeidung von Fehlern innerhalb des Programmablaufs. Man unterscheidet dabei unter anderem verschiedene Testarten und verschiedene Testmethodiken, von denen einige im Folgenden näher erläutert werden.

2.1 Testarten

Es gibt verschiedene Arten von Softwaretests. Diese unterscheiden sich vor allem darin, was getestet wird, aber auch darin, wer den Test durchführt. Neben den Modul- und Funktionstests, die vom Entwicklungsteam durchgeführt werden, gibt es noch System- und Abnahmetests, bei denen das gesamte Softwareprodukt vom Kunden bzw. späteren Benutzer entweder auf einem Testsystem oder auf dem Produktivsystem getestet wird.

2.1.1 Modultests

Modultests (im Englischen auch Unit Tests genannt) sind Tests, die einzelne Module der Software testen. Für gewöhnlich versucht man hier möglichst kleine Module zu testen, um die Stabilität der einzelnen Tests zu steigern. Es wird also vor allem eine einzelne Methode bzw. Funktion getestet. Da der Test somit direkt mit einem Programmcode-Stück interagiert, ist klar, dass er selbst in der gleichen Programmiersprache verfasst wird. Er wird somit von einem Entwickler geschrieben. Der Test sollte sich dabei auf das gewünschte Verhalten der Methode konzentrieren und nicht auf deren konkrete Implementierung abstellen (Design-by-contract). Bei den Modultests hat sich im Laufe der Zeit eine bestimmte Definitionsweise etabliert, der die meisten Modultest-Frameworks folgen. Es gibt sogar eine Reihe von Frameworks (xUnit), die von Sprache zu Sprache portiert werden und alle auf das von Kent Beck entworfene SUnit für die Sprache Smalltalk zurückzuführen sind. Von ihm stammt auch die erste Portierung in die Sprache Java namens JUnit. Typisch für diese Frameworks ist, dass die eigentlichen Tests ihrerseits Methoden innerhalb einer Testklasse (Test- Case) sind. Mehrere dieser Testklassen können dann in Gruppen organisiert werden (Test-Suite). Außerdem gibt es fast immer die Möglichkeit, innerhalb einer Testklasse Methoden für die Initialisierung bzw. Deinitialsierung der Testumgebung zu definieren, die vor und nach jeder einzelnen Testmethode ausgeführt weren (Set-Up- und Tear- Down-Methoden). Innerhalb der eigentlichen Testmethode stehen dann weitere Hilfsmethoden zur Verfügung, mit denen einfach Behauptungen über das Testsubjekt formuliert werden können (Assertions), die dann während der eigentlichen Testausführung überprüft werden. Ein Modultest-Framework bietet neben einer einfachen Test-Definitions-Syntax auch ein Programm, Test-Runner genannt, welches die Auswahl bestimmter Tests erlaubt und nach deren Ausführung ein Testergebnis darstellt.

2.1.2 Funktionstests

Der Funktionstest dient dazu, das Zusammenspiel mehrerer Programmbausteine zu testen. Gerne wird er auch als Schnittstellentest bezeichnet, da er gegen eine bestimmte Schnittstelle des Programms arbeitet. Bei Webanwendungen bietet es sich hierbei an, die HTTP-Schnittstelle der Anwendung zu testen. Man spezifiziert also Aufrufe an den Webserver und überprüft dabei, ob die Antwort bestimmten Erwartungen entspricht. Meist werden auch die Funktionstest vom Entwickler selbst geschrieben und damit auch gerne wieder in der gleichen Programmiersprache wie das zu testende Programm selbst.

2.1.3 System- und Abnahmetests

Da bei den System- und Abnahmetests das fertige Softwareprodukt durch den Kunden bzw. späteren Benutzer getestet wird, ist für die Implementierung dieser Tests die der

Software zugrunde liegende Programmiersprache unerheblich. Es gibt hier vor allem keine sprachspezifischen Frameworks.

2.2 Testmethodiken

Testmethodiken dienen dazu, das Schreiben der Tests in den eigentlichen Software- Entwicklungsprozess zu integrieren. In den meisten Softwareprojekten haben sich dabei vor allem zwei Methodiken etabliert: Das Test Driven Development (zu Deutsch „Testgetriebene Entwicklung“) und die Continuous Integration (zu Deutsch „Kontinuierliche Integration“).

2.2.1 Test Driven Development

Beim Test Driven Development handelt es sich um ein Vorgehen, dass der agilen Softwareentwicklung zugeschrieben wird. Im Gegensatz zur klassischen Softwareentwicklung, bei der Tests bei oder sogar erst nach der fertigen Implementierung des Programmcodes geschrieben werden, kann man vereinfacht sagen, dass der Entwickler beim Test Driven Development erst den Test und dann den eigentlichen Programmcode schreibt. Dies soll folgende Vorteile mit sich bringen:

- Der Ansatz garantiert eine sehr hohe Testabdeckung.
- Es wird verhindert, dass nicht testbarer Code entsteht.
- Zu Projektende werden Ressourcen (Zeit, Budget) typischerweise knapp und verhindert oft die Fertigstellung nachträglicher Tests.
- Das vorherige Schreiben des Tests begünstigt, dass der Tests auf das gewünschte Verhalten abstellt, da die Implementierung ja noch nicht vorliegt (Design-by-contract).

Da der Entwickler die Tests bei diesem Vorgehen sehr oft ausführt, nämlich vor der Implementierung und während der Implementierung so oft, bis der Test bestanden ist, ist es sehr wichtig, dass das Ausführen der Tests sehr einfach vorgenommen werden kann. Man spricht gerne davon, dass die Tests „auf Knopfdruck“ durchgeführt werden können müssen. Ein gutes Test-Framework bietet hierfür eine Möglichkeit, den Test- Runner direkt aus der IDE (Integrated Development Environment) des Entwicklers aufzurufen.

2.2.2 Behavior Driven Development

Es haben sich aber auch Verfahren etabliert, bei denen die Tests als Prosa-Text verfasst werden. Dieser muss aber einer sehr eingeschränkten Syntax unterliegen, da er zum Ausführen des Tests in entsprechenden Programmcode übersetzt wird. Ziel dieses Ansatzes ist es, dass die Tests auch von Nicht-Entwicklern (z.B. dem Kunden) gelesen und verstanden werden können.

2.2.3 Continous Integration

Bei der Continuous Integration geht es darum, die Test-Ergebnisse und andere Metriken teamübergreifend zu erfassen und kontinuierlich zu verfolgen, um so im Lauf der Zeit positive oder negative Tendenzen in der Entwicklung des Softwareprodukts erkennen zu können. Typischerweise werden die Testergebnisse und andere Metriken in einem sogenannten Continuous-Integration-Server archiviert, der sich darum kümmert, diese von Zeit zu Zeit abzufragen bzw. selbständig zu generieren. Oft tut der Server dies zum Beispiel jede Nacht (Nightly Builds). Viele solcher CI-Server bieten aber sogar die Möglichkeit, sie über jede neue Version der Software zu informieren, die in der Versionskontrolle abgelegt wird und daraufhin ein neues Ausführen der Tests usw. auszulösen. In Bezug auf die Tests ist es dazu notwendig, dass diese nicht durch einen Entwickler sondern programmatisch

ausgeführt werden können und ihre Ergebnisse anschließend auf maschinenlesbare Art und Weise zur Verfügung stellen.

3 Javascript Testing

Nachdem eingangs verschiedene allgemeine Testarten und -methodiken aufgeführt wurden, sollen im folgenden konkrete Frameworks für die Programmiersprache Javascript aufgeführt und in Bezug auf diese allgemeinen Ansätze miteinander verglichen werden. Wie in der Einleitung erläutert, wird dabei auch untersucht, ob und wie sich die Tests in Browsern bzw. von einem eigenständigen Javascript-Interpreter ausführen lassen.

3.1 Modultest-Frameworks

Zunächst folgen Javascript-Frameworks, mit denen man Modultests schreiben kann.

3.1.1 JsUnit

Abbildung 1: Logo von JS-Unit

JsUnit ist das Javascript-Framework der xUnit-Reihe. Es handelt sich also um eine Portierung des von Kent Beck entworfenen Modultest-Frameworks SUnit. Demzufolge bietet sie sowohl die Möglichkeit, Set-Up- und Tear-Down-Methoden, als auch Behauptungen zu formulieren:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript" src="jsUnitCore.js"></script>
        <script type="text/javascript" src="myCode.js"></script>
        <script type="text/javascript">

            var counter;

            function setUp() {
                counter = 1
            }

            function testAddition() {
                assertEquals(2, counter + 1);
            }

            function testFalseAddition() {
                assertNotEquals(3, counter + 1);
            }

            function tearDown() {
                counter = null;
            }

        </script>
    </head>
    <body></body>
</html>
```

Um den Test-Case auszuführen, bietet JsUnit einen Test-Runner, den man als HTML-Seite in einem beliebigen Browser öffnen kann. In diesem kann man über ein Text-Feld die oben gezeigte Test-Case-Datei referenzieren und ausführen. JsUnit bietet von Haus aus

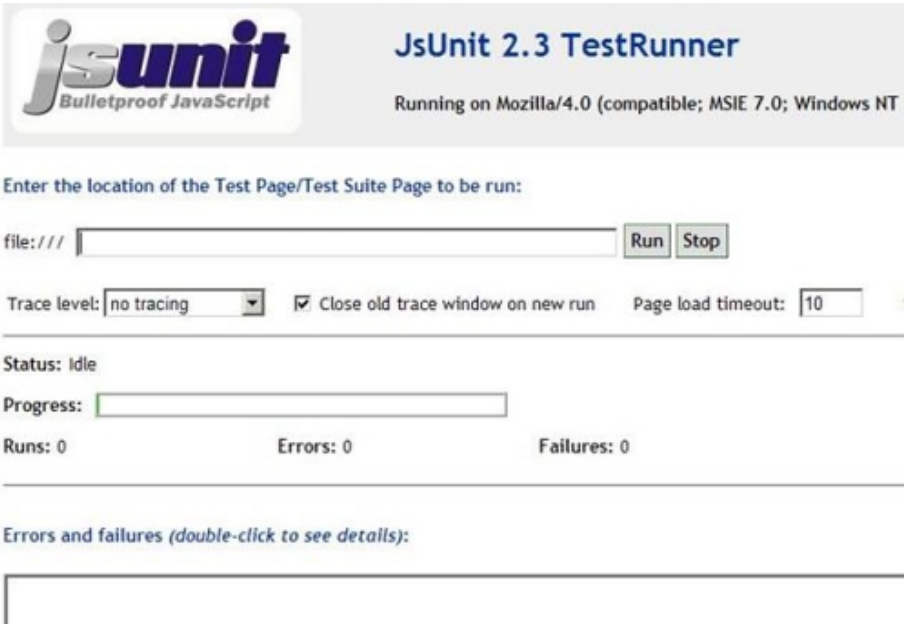

Abbildung 2: JS-Unit Testrunner

keine Möglichkeit, den Test-Case in einem Headless- Javascript-Interpreter auszuführen. Dadurch ist eine einfache, automatisierte Ausführung der Tests nicht möglich. Zudem bietet das Framework keine Möglichkeit, die Test-Cases in Test-Suiten zu organisieren. Die Entwicklung von JsUnit wurde inzwischen eingestellt.

3.1.2 YUI Test

3.1.3 Jasmine

3.2 Funktionstest-Frameworks

3.2.1 Phantom-JS

3.2.2 Selenium-Driver

3.3 Test Driven Development

3.3.1 Browserfernsteuerung

3.3.2 Browser-Mocking

3.4 Behaviour Driven Development

3.5 Continous Integration

3.5.1 Karma

3.5.2 SauceLabs

4 Fallbeispiel

5 Fazit

6 Literaturverzeichnis

Google 2013
GOOGLE: *Karma - Configuration File*. `http://karma-runner.github.io/0.10/config/configuration-file.html`. Version: 2013. – [Online; Stand 07. Dezember 2013]

Hidayat 2013
HIDAYAT, Ariya: *Quick Start | PhantomJS*. `http://phantomjs.org/quick-start.html`. Version: 2013. – [Online; Stand 07. Dezember 2013]

Johansen 2010
JOHANSEN, Christian: *Test Driven JavaScript Development*. Addison-Wesley Professional, 2010

Kleivane 2011
KLEIVANE, Tine F.: *Unit Testing with TDD in JavaScript*, Norwegian University of Science and Technology, Diplomarbeit, 2011

Koch 2001
KOCH, Stefan: *JavaScript. Einführung - Programmierung - Referenz*. Dpunkt Verlag, 2001

Nguyen 2013
NGUYEN, Lauren: *Sauce Labs Adds Expanded JavaScript Unit Testing Capabilities to its Cloud Testing Platform*. `http://sauceio.com/index.php/2013/09/sauce-labs-adds/`. Version: September 2013. – [Online; Stand 14. November 2013]

Pivotal-Labs 2013
PIVOTAL-LABS: *Jasmine*. `http://pivotal.github.io/jasmine/`. Version: 2013. – [Online; Stand 07. Dezember 2013]

Selenium-Project 2013
SELENIUM-PROJECT: *Selenium Webdriver*. `http://www.seleniumhq.org/docs/03_webdriver.jsp`. Version: 2013. – [Online; Stand 07. Dezember 2013]

Trostler 2013
TROSTLER, Mark E.: *Testable JavaScript*. O'Reilly Media, 2013

Wikipedia 2013a
WIKIPEDIA: *Funktionstest*. `http://de.wikipedia.org/wiki/Funktionstest`. Version: 2013. – [Online; Stand 07. Dezember 2013]

Wikipedia 2013b
WIKIPEDIA: *JavaScript*. `http://de.wikipedia.org/wiki/JavaScript`. Version: 2013. – [Online; Stand 07. Dezember 2013]

Wikipedia 2013c
WIKIPEDIA: *Modultest*. `http://de.wikipedia.org/wiki/Modultest`. Version: 2013. – [Online; Stand 07. Dezember 2013]

Yahoo 2013
YAHOO: *Test - YUI Library*. `http://yuilibrary.com/yui/docs/test/`. Version: 2013. – [Online; Stand 07. Dezember 2013]

7 Anhang