FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Evaluieren von automatisierten Tests bei Web-Applikationen

MAGISTERARBEIT

zur Erlangung des akademischen Grades

Magister
der Sozial- und Wirtschaftswissenschaften

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Serkan Avci, Bakk. rer. soc. oec.
Matrikelnummer 9925155

an der

Fakultät für Informatik,
Institut für Medizinisches Informationsmanagement und Bildverarbeitung
Medizinische Universität Wien

Betreuung
Betreuer: Univ.-Prof. Dipl.-Ing. Dr. Ernst Schuster
Mitwirkung:

Wien, 04.05.2010

(Unterschrift Verfasser) (Unterschrift Betreuer)

Serkan Avci Bakk.

1020 Wien, Vorgartenstraße 140-142/8/4

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien, 04.05.2010

Danksagung

An dieser Stelle möchte ich mich herzlichst bei Herrn Univ. Prof. DI Dr. Ernst Schuster für seine fachliche und freundliche Betreuung meiner Magisterarbeit bedanken.

Weiter möchte ich mich bei meiner lieben Familie und ganz besonders bei meiner Ehefrau für ihre jahrelange Unterstützung bedanken.

Kurzbeschreibung

Evaluierung von automatisierten Tests bei Web-Applikationen

Für Unternehmen und Organisationen, die Informationssysteme entwickeln und verwenden, ist es sehr kritisch, Web-Applikationen und Web-Informationssysteme ungetestet zu veröffentlichen. Die Komplexität und Fülle an Daten sorgen oft dafür, dass Webseiten nicht die gewünschte Funktionalität oder Performance liefern, die an sie gestellt werden. Die automatisierten Webtests sollen garantieren, dass Web-Applikationen und Informationssysteme richtig arbeiten und auch unter verschiedenen Browser die gleiche Funktionalität und Performance liefern sowie Plattformunabhängig verwendet werden können. Diese Tests sollen dem Betreiber der Web- Applikation auch Auskunft über die Skalierbarkeit ihrer Web-Applikation geben, und ob diese auch unter starker Last, Resultate in annehmbarer Zeit liefern.

Die vorliegende Arbeit setzt sich mit den Grundlagen des automatisierten Testens von Web-Applikationen auseinander. Das Hauptziel dieser Arbeit ist es, die Selenium Tests mit Hilfe von Linux Shell-Skripts zu automatisieren und somit präzise Testergebnisse in kürzeren Testläufen zu erreichen. Außerdem zeigt die Arbeit das Zusammenspiel von Selenium Framework und Linux Shell-Skripts bei automatisierten Webtest.

Selenium HQ WATS ist ein Test-Framework zum Erstellen und Ausführen automatischer Tests für „Web-Applikation“. Das Selenium Framework wird oft mit Selenium IDE gleichgesetzt, jedoch ist die IDE nur eine Anwendung in einem großen Paket. Dieses Paket beinhaltet Selenium Core, Selenium IDE und Selenium Remote Control, das sogenannte RC. Diese Produkte können entweder alleine oder gemeinsam eingesetzt werden. Die Kern-Einheit dieses Frameworks bildet Selenium Core.

Fragestellung:

- Wie kann man Selenium Tests automatisieren?
- Wie generiert man tägliche Selenium Test-Reports?
- Welche Werkzeuge und Infrastruktur ist dazu notwendig?

Schlüsselwörter: Web-Applikation, automatisierte Tests, Webtest, Selenium, Web-Element, Scripts, Shell, Linux

Evaluation of automated tests with Web-Applications

For enterprises and organizations, which develop and use information systems, it is critically to publish untested Web applications and Web information systems. The complexity and abundance of data make sure, that web pages do not supply the desired functionality or performance. Automated tests should guarantee that Web applications and information systems work correctly under various browsers and platform-independently. These tests should also give the operator of the Web application information about the scalability and how it reacts under heavy load to supply the intend results.

This master thesis explains the automated testing fundamentals of Web applications. The main goal of this work is to automate Selenium test by Linux Shell scripts to get precise test results in shorter test runs. Furthermore, the work shows interactions between Selenium Framework and Linux Shell script of automated Web tests.

Selenium HQ WATS is a test Framework for constructing and running automatic tests for „Web application “. The Selenium Framework is often equated with Selenium IDE, however the IDE is only an application within the Selenium package. This package contains Selenium core, Selenium IDE and Selenium remote control „RC“. These products can be used either alone or together. Selenium core is the core-unit of Selenium Framework.

Inhaltsverzeichnis

“The difference between a good programmer and a bad programmer is that the good programmer uses tests to detect his mistakes as soon as possible.”

[Ber10]

1. Einleitung

Die Softwaretechnologien haben sich mittlerweile so schnell und umfangreich entwickelt, dass Entwickler keine Zeit mehr haben, sich zusätzlich um die Softwarequalität zu kümmern.

Zu weitläufige Spezifikation und immer neue Kundenwünsche führen dazu, dass Entwickler laufend neue und komplexere Funktionen in die Software einbauen müssen. Dieser steigende Umfang lässt den Entwicklern meist keine Zeit für ausgiebiges Testen übrig. Aus diesem Umstand entstand eine neue Berufsparte, das Softwaretesten, da man auch in der Softwareentwicklung dem Kunden ein Produkt nach aktuellen Qualitätsstandards anbieten wollte. Im technischen und kommerziellen Software-Bereich spielt die Qualität der Software für den Erfolg des Produktes und somit der Unternehmen eine große Rolle. Viele Unternehmen haben diese Wichtigkeit erkannt und streben eine verbesserte Qualität ihrer Softwareprodukte sowie ihres Softwareentwicklungsprozesses an.

Die Softwarequalität umfasst deshalb viele Aspekte in der Softwareentwicklung. In dieser Arbeit werde ich einige davon aufgreifen und näher betrachten.

In meiner Diplomarbeit setze ich mich mit den Grundlagen des automatisierten Testens von Web-Applikationen auseinander. Das Hauptziel dieser Arbeit ist es, die Selenium Tests mit Hilfe von Linux Sell-Skripts zu automatisieren und somit präzise Testergebnisse in kürzeren Testläufen zu erreichen. Außerdem zeigt die Arbeit das Zusammenspiel von Selenium Framework und Linux Shell-Skripts bei automatisierten Webtest. Das Fallbeispiel dient zur Unterstützung dieses Zieles und zeigt, wie mit Hilfe eines Testwerkzeuges und einiger frei zur Verfügung stehender Programme, eine automatisierte Testumgebung geschaffen werden kann.

In dieser Arbeit werde ich folgende Fragen beantworten.

- Wie kann man Selenium Tests automatisieren?
- Wie generiert man tägliche Selenium Test-Reports?
- Welche Werkzeuge und Infrastruktur ist dazu notwendig?

2. Grundlagen

2.1 Definition

Zu Anfang dieser Magisterarbeit werde ich die wichtigsten Grundbegriffen des Software-Testens erklären. Dazu werde ich verschiedene Definitionen zum Thema „Testen“ betrachten und anhand dessen die Sichtweisen erklären. Im Zusammenhang mit Software-Testen fallen oft Wörter wie Qualität, Testen, Softwaretest und Tester. Was bedeuten diese Wörter, wie werden diese definiert und woher stammen diese. Hierzu muss man sich mehrere Quellen anschauen, da eine einzige Quelle meistens für die Beantwortung dieser Fragen unzureichend ist.

2.1.1 Qualität

Das Wort *„Qualität“* kommt vom lateinischen Wort *„qualitas“* und bedeutet *„Beschaffenheit“*.

Im Langenscheidt Online-Wörterbuch wird das Wort *„Qualität“* mit folgenden Worten erklärt:

***„Qua·li'tät**, die; -,-en **1.**Güte, Wert, Beschaffenheit **2.** Eignung, Eigenschaft, Fähigkeit **3.** WIRTSCHAFT Ware von einer bestimmten Beschaffenheit oder Güte **4.** Wertunterschied der Figuren beim Schach “* [Wir10]

Das gebundene Brockhaus-Lexikon geht bei seiner Erklärung auf das Wort *„Qualität“* etwas genauer ein und unterteilt diese nach zwei Merkmalen.

„Eigenschaft, Beschaffenheit, Güte, Wertstufe. Die Beschaffenheit einer Ware nach ihren Unterscheidungsmerkmalen gegenüber anderen Waren im Bezug auf ihre Fähigkeit, Nutzen zu stiften. Der Begriff Qualität wird sowohl objektiv auf messbare Eigenschaften (z.B. Reinheit chemischer Erzeugnisse) als auch subjektiv angewendet, wenn er die Abstufung des Eignungswertes gleichwertiger Güter für die Befriedung bestimmter Bedürfnisse ausdrückt.“ [Bro08]

Im Brockhaus-Lexikon wird die Qualität als *„Eignungswerte für die Befriedung bestimmten Bedürfnisse“* bezeichnet, welches die Ziele der Anwender in zwei Merkmale einteilt. Zum einen wird Qualität objektiv (das zum Beispiel bestimmten Regeln und Standards folgt und diese erfüllt) zum anderen subjektiv angewendet (diese wären Benutzerfreundlichkeit, Effizienz, usw.).

In der Umgangssprache versteht man darunter vor allem zwei Aspekte. Der erste Aspekt ist Fehlerlosigkeit, der ein Produkt ohne Fehler als qualitativ hochwertig sieht und der zweite Aspekt ist Gebrauchstauglichkeit bzw. Gebrauchswert, der das Produkt als einwandfrei ansieht, wenn es alle unsere Ansprüche erfüllt und gut zu handhaben ist.

Für eine etwas genauere Beschreibung im Zusammenhang zur Technik insbesondere der Softwaretechnik muss man sich die Definition der Internationalen Organisation für Standardisierung nach der ISO[1]-Norm ansehen.

„Qualität: *von lat. qualitas = Beschaffenheit, nach DIN ISO 8402 „die Gesamtheit von Merkmalen einer Einheit bezüglich ihrer Eignung, festgelegte und vorausgesetzte Erfordernisse zu erfüllen. (Qualität entsteht mit Hilfe der Technik auf der Grundlage einer entsprechenden Geisteshaltung)“* [Duh10]

Die Definition besagt, dass es nicht das eine Merkmal gibt, das die Qualität eines Produktes beschreibt, sondern eine ganze Reihe von Kriterien, die sich mit der Software-Qualität in direkter Weise verbinden lassen.

Die wesentlichen Merkmale für die Beurteilung der Software-Qualität:

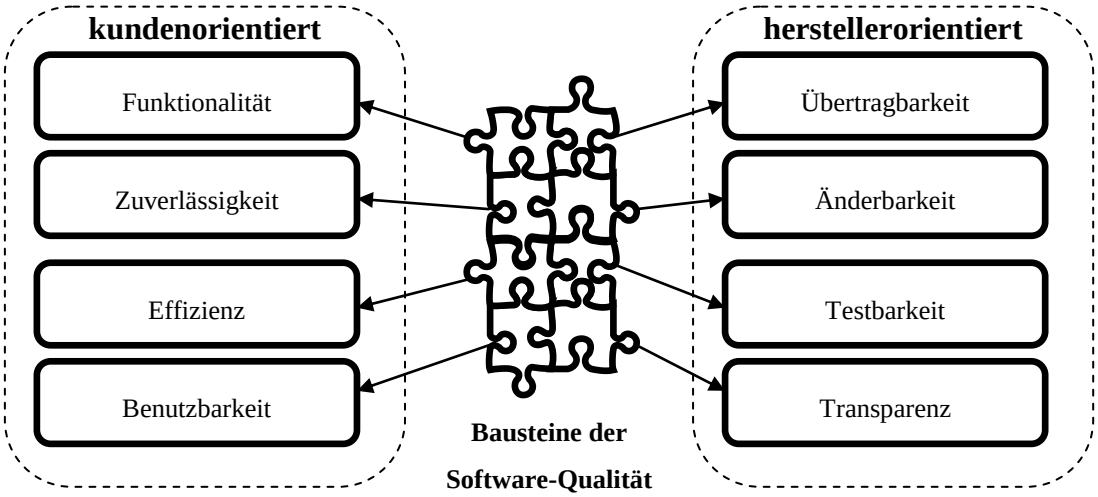

Abbildung 1: Qualitätsmerkmale eines Software-Produkts [Hof08]

[1] ISO: Internationale Organisation für Normung – kurz ISO, engl. International Organization for Standardization [IOS10]

2.1.2 ISO

Die Internationale Organisation zur Standardisierung (ISO) brachte unter der Bezeichnung ISO 9000 eine Reihe von Normen mit dem Ziel heraus, die Nomenklatur und das Vorgehen der industriellen Qualitätssicherung zu regeln.

1979 setzte sich ein aus 20 ISO-Mitgliedsländern gegründetes Technisches Komitee zusammen, um die Arbeit an der Entwicklung internationalen Qualitätsstandards aufzunehmen.

1987 wurde die erste Ausgabe der Qualitätsnormen (Tabelle 1) veröffentlicht, welche anfangs nur Empfehlungen waren und sich inzwischen durchgesetzt haben. Die Qualitätsnormen sind in fünf Gruppen unterteilt.

ISO 9000	Qualitätsmanagement- und Qualitätssicherungsnormen – Leitfaden zur Auswahl und Anwendung
ISO 9001	Qualitätsmanagementsysteme – Modell zur Qualitätssicherung/QM-Darlegung in Design, Entwicklung, Produktion, Montage und Wartung
ISO 9002	Qualitätsmanagementsysteme – Modell zur Qualitätssicherung/QM-Darlegung in Produktion, Montage und Wartung
ISO 9003	Qualitätsmanagementsysteme – Modell zur Qualitätssicherung/QM-Darlegung bei der Endprüfung
ISO 9004	Qualitätsmanagement und Elemente eines Qualitätsmanagementsystems – Teil 1: Leitfaden

Tabelle 1: Qualitätsnormen ISO-9000er Reihe

„DIN EN ISO 9000 Qualität. Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt Definition aus der früher gültigen DIN EN ISO 8402 Qualität. Gesamtheit von Merkmalen einer Einheit bezüglich ihrer Eignung, festgelegte und vorausgesetzte Erfordernisse zu erfüllen“ [Pro10]

Darüber hinaus sind in der ISO-Norm 8402 (*„Qualitätsbegriffe“*) aus dem Jahr 1986[2] die Begriffsdefinitionen festgelegt. ISO 9001 bezieht sich im speziellen auf die Softwareherstellung.

[2] 1994 erschien eine überarbeitete Fassung dieser Normen, die nun Bestandteil der 9000-Reihe sind. Daneben befassen sich noch einige andere Normen mit dem Qualitätsmanagement.

2.1.3 Testen

Das Wort *„Test“* stammt ursprünglich aus dem altfranzösischen *„Tiegel“* und bedeutet Topf für alchemistische Versuche. In Frankreich wurde ein Test als Versuch gesehen, der Sicherheit darüber geben sollte, ob ein technischer Apparat innerhalb der geplanten Rahmenbedingungen funktioniert oder nicht. Der Unterschied zwischen einem Test und einem Experiment ist, dass beim Test zwei erwartete Ergebnisse vorliegen, während beim Experiment das Ergebnis offen ist oder nur vermutet werden kann.

Beim Testen wird ein Programm mit dem Ziel ausgeführt Fehler zu finden, da das erwartete Ergebnis schon bekannt ist. Dabei unterscheidet man drei Arten von Tests, die Black-Box-Tests, White-Box-Tests und die Grey-Box-Tests. Darüber hinausgehende Teststufen wie Integrationstests oder Systemtests werden von eigenen Testgruppen durchgeführt.

2.1.3.1 Probleme beim Testen

„Testen bedeutet nicht Herumprobieren.“ [Alp94]

Testen ist ein komplexer Prozess, der geplant und dokumentiert gehört. Ein grundsätzliches Problem des Testens liegt darin, dass nur die Anwesenheit von Fehlern gezeigt werden kann, nicht aber die Abwesenheit der Fehler. Außerdem sollten erschöpfende Tests vermieden werden, da das Ausmaß der möglichen Eingaben und Programmzustände bei größeren Programmen zu umfangreich ist.

- Fehlende oder fehlerhafte Dokumentation.
- Keine oder unzureichende Testplanung während der Softwareentwicklung.
- Schlechte Testskripts führen zu falschen Ergebnissen.
- Unzureichend ausgebildete Tester.
- Nichterfüllen der Voraussetzungen.

Tabelle 2: Probleme beim Testen

Probleme beim Testen (Tabelle 2) lassen sich nur dann frühzeitig erkennen, wenn eine effiziente Zusammenarbeit von Management, Entwicklung und Testteam stattfindet.

2.1.4 Softwaretesten

*"Ein **Softwaretest** ist ein Test während der Softwareentwicklung, um die Funktionalität einer Software an den Anforderungen und ihre Qualität zu messen, und Softwarefehler zu ermitteln."* [Wik01]

Das Testen ist in der modernen Software-Entwicklung eine wesentliche Methode zur Qualitätssicherung. Im weiteren Sinn ist das Testen eine Ausführung mit dem Ziel, einen Fehler, eine Anomalie oder einen Mangel zu finden. Das Software-Qualitätsmanagement hat seine Wurzeln und Ideen aus der Industrialisierung, welche nach und nach an Bedeutung dazu gewonnen haben. Diese wurden in den darauf folgenden Jahrzehnten weiter entwickelt und verfeinert. Der Begriff *„Qualität“* ist heutzutage im Informations- und Internetzeitalter allgegenwärtig. Es steht für Fehlerlosigkeit und gebrauchstaugliche Produkte.

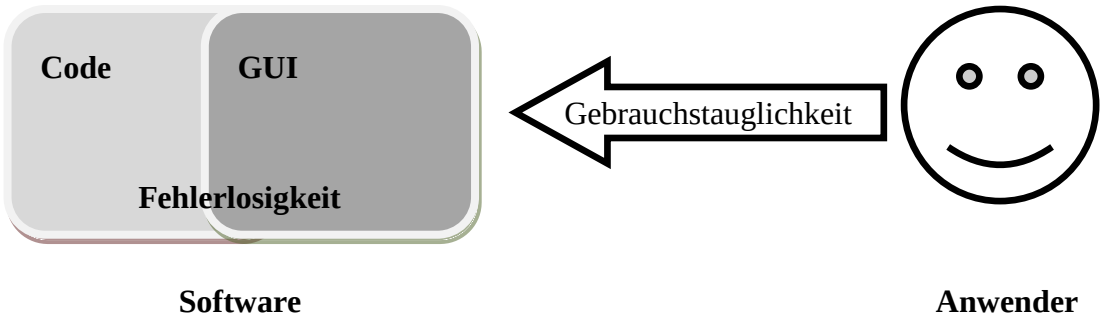

Abbildung 2: Qualität

- **Fehlerlosigkeit:** ein Produkt ohne Fehlern ist qualitativ hochwertig. Das gilt sowohl für den Code als auch für die Benutzeroberfläche (GUI[3]).
- **Gebrauchstauglichkeit oder Gebrauchswert:** ein Produkt, das alle unsere Ansprüche im GUI erfüllt und gut zu handhaben ist, ist qualitativ einwandfrei.

Tabelle 3: Testdefinition

[3] GUI : Graphische Benutzeroberfläche (engl. Graphical User Interface)

2.1.5 Tester

Tester sind Personen, welche die Tests durchführen. Diese Personen müssen fachliche sowie spezielle Fertigkeiten besitzen, da jede Software für einen bestimmten Zweck entwickelt wird. Zum Beispiel sollten Tester jederzeit Zugang zu allen Softwarespezifikationen und Entwürfen haben. Eine Testperson kann einem speziellen Personenkreis (Entwickler) angehören, welcher die Tests durchführt. Hier kann es z.B. Entwicklertests oder Anwendertests geben.

2.1.6 Webanwendung

„Eine Webanwendung oder Webapplikation ist ein Computer-Programm, das auf einem Webserver ausgeführt wird, wobei eine Interaktion mit dem Benutzer ausschließlich über einen Webbrowser erfolgt. Hierzu sind der Computer des Benutzers (Client) und der Server über ein Netzwerk, wie das Internet oder über ein Intranet miteinander verbunden, so dass die räumliche Entfernung zwischen Client und Server unerheblich ist.“ [Wik02]

Eine Web-Applikationen muss heutzutage vielen Anforderungen entsprechen, die teils technischer und teils fachlicher Art sind. Diese Applikationen können zu großen Gebilden (Abbildung 3) anwachsen, die viele Fehler in sich beherbergen können. Um diese Fehler schnell, effizient und mit geringen Kosten zu finden, muss man neue Testtechniken und Prozesse entwickeln.

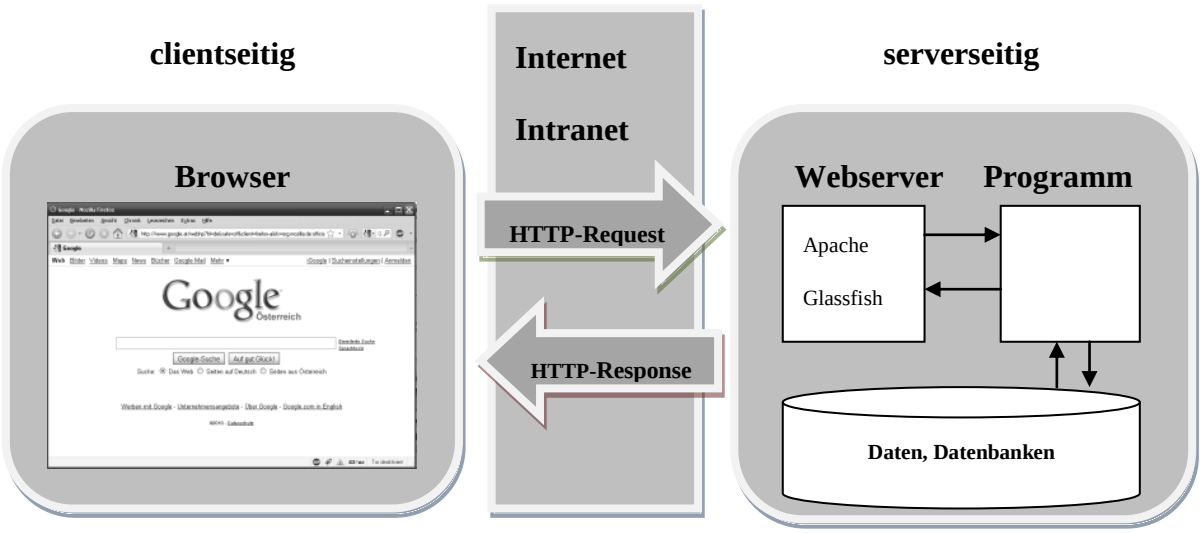

Abbildung 3: Webanwendung, Server-Client

Die klassischen Testmethoden und Testtechniken sind bei Web-Applikationen zwar auch notwendig, aber nicht ausreichend, um die Qualität der Applikation sicherzustellen und wurden deshalb durch neue Automatisierungs-Techniken ergänzt. Diese neuen Techniken werde ich hier in meiner Arbeit erläutern und erklären.

2.1.7 Automatisiert

Der Wortstamm von „Automat“ ist aus zwei altgriechischen Stämmen zusammengesetzt (αὐ τόματος) und bedeutet etwa „von selbst tun“, „sich selbst bewegend“.

„Au·to'mat, der; -en,-en 1.TECHNIK selbsttätig arbeitende Maschine 2. TECHNIK selbsttätig arbeitender Verkaufsapparat 3. INFORMATIONSTECHNOLOGIE Informationen aufnehmendes, speicherndes, verarbeitendes und ausgebendes System 4. seelenloser Mensch“ [Wir10]

Bei einem Automaten handelt es sich um eine Vorrichtung, die nach dem Einstellen und Auffüllen eine vordefinierte Handlung selbstständig ohne äußere Einflüsse durchführt.

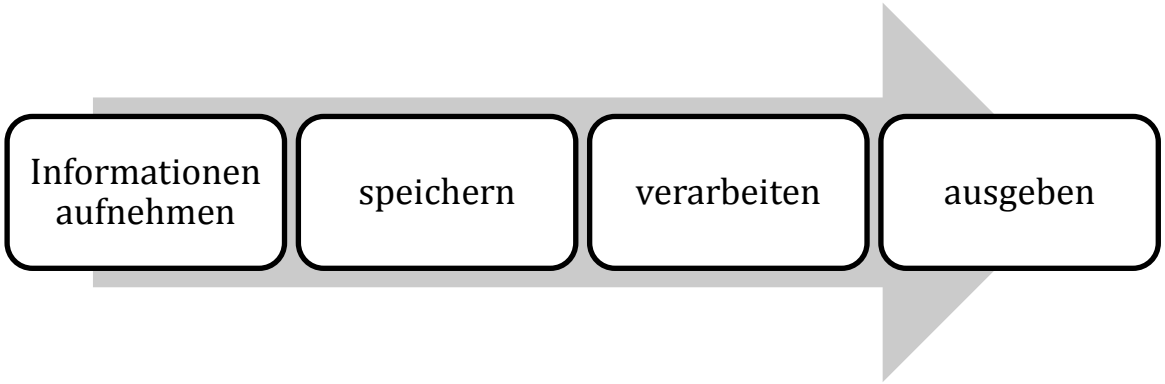

Abbildung 4: Automat in der Informationstechnologie

2.2 Testen

Testen ist das Vergleichen von definierten Soll-Zuständen mit den gemessenen Ist-Zuständen. Die *"Soll"*-Zustände werden in der Produktspezifikation beschrieben. Sind diese Zustände identisch, hat man das Testziel erreicht und das Testobjekt besteht den Test. Ein bestandener Test hat zur Folge, dass das zu testende Objekt als gut bewertet wird und seinen Sinn und Zweck erfüllt. Heutzutage wird alles einer Testprozedur unterzogen. Diese Prozedur ist jedes Mal identisch und wird bei veränderten Vorbedingungen angepasst. Eine wiederkehrende Testprozedur nennt man *Regression*, welche zum Ziel hat, die vorher getesteten Funktionen nach einer zeitlichen Folge noch einmal einem Test zu unterziehen. Die Regression wird im Kapitel „Testmethoden“ behandelt.

"Testen ist der Schlüssel zu einer besseren Softwarequalität." [Ruf05]

Testen gehört in der Softwareentwicklung eigentlich zu den ganz selbstverständlichen Prozessen. Jeder in der Softwareentwicklung bestätigt einstimmig, dass Testen sehr wichtig ist, jedoch kaum einer testet wirklich. In der Softwareentwicklungsplanung wird schon reflexartig in jede Iteration auch ein Testprozess eingeplant. Obwohl diese immer in die Planung einfließen, fehlt dem Entwicklungspersonal oft die Motivation und die Zeit zum Testen. Hinzu kommen auch noch unzählige fadenscheinige Begründungen der Softwareentwickler. Einige dieser Begründungen klingen wie die Beispiele in Tabelle 4.

- Testen von Software ist langweilig und stupide.
- Mein Code ist praktisch fehlerfrei.
- Die Testabteilung testet. Die können das viel besser.
- Ich habe keine Zeit zum Testen.

Tabelle 4: Ausreden von Entwicklern gegen das Testen

Heutzutage wird das Testen hauptsächlich von professionellen Testmitarbeitern und Qualitätssicherungs-Personal wahrgenommen, denn ohne Tests wird eine Web-Applikation kaum eine ausreichende Qualität und somit die Abnahme durch die Auftraggeber erreichen. Im Allgemeinen sagt man, dass das Testen etwas kostet aber auch das Nicht-Testen kostet. Das Zitat *„Testing is not free, but it pays off."*[4] bringt das Ganze auf den Punkt.

„Testing is not free, but it pays off." [Ruf05]

Die Arbeit soll überprüft werden, ob diese auch dem gesetzten Qualitätsstandard entspricht. Hierzu benötigt man Werkzeuge, die genau das dem Tester bestätigt.

Antworten, die nach dem Testen die Qualität der Software bestätigen.

- Ja, wir haben es richtig gemacht.
- Ja, es funktioniert.
- Ja, wir sind mit diesem Teil fertig.
- Nein, durch unsere Änderung wurde keine alte Funktionalität ungewollt verändert.

Tabelle 5: Testbestätigungen

[4] Übersetzung: Testen ist nicht gratis, aber es zahlt sich aus.

Wenn alle diese Fragen positiv beantwortet werden können, ist die Software richtig getestet worden. Testen ist der Schlüssel zu einer besseren Softwarequalität. Es ist für den Entwickler eine Bestätigung, dass er seine Arbeit vollständig erledigt hat. Softwaretests helfen bei der Entscheidung, ob eine Anwendung in die Produktion überführt werden kann. Softwaretests haben einen dokumentarischen Charakter. Die Produkt-Dokumentation wird dadurch auf den aktuellen Stand gebracht. Softwaretests sind immer aktuell zu halten und sagen präzise aus, welche Funkionen die Anwendung unterstützt.

2.2.1 Testdefinitionen

"Test: (1) An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component." [IEE90]

Das bedeutet, ein System oder eine Komponente wird unter bestimmten Rahmenbedingungen ausgeführt, die dabei entstehenden Resultate werden beobachtet oder aufgezeichnet. Danach wird eine Evaluierung von allen oder nur bestimmten Teilen des Systems oder einer Komponente vorgenommen.

2.2.1.1 Testen in der Softwareindustrie und Softwareentwicklung

In der Softwareindustrie und Softwareentwicklung sind klassische Software-Tests die meist verbreitete Methode zum Identifizieren von Fehlern in Programmen. Anhand technischer Spezifikationen werden für das zu testende Programm einzelne Testfälle erstellt und die dabei erwarteten Resultate gespeichert. Nach Durchführung dieser Testfälle werden diese erwarteten Resultate mit den tatsächlichen Ergebnissen verglichen. Diese Reihenfolge bildet die klassische Testmethode. Das zu testende Programm wird dazu anhand korrekter, fiktiver oder falscher Datensätze (Eingaben) ausgeführt oder gestartet wobei das Ist-Ergebnis mit dem vorher verifizierten Soll-Ergebnis verglichen wird.

2.2.1.2 Testfall

Ein Testfall beschreibt einen Softwaretest, der zur Überprüfung einer in der Testspezifikation zugesicherten Eigenschaft eines Testobjektes dient. Ein Testfall wird mittels Anwendung der Testmethoden spezifiziert. Die während der Planung ausgewählten Testmethoden werden dann zur Erstellung der Testfälle beigezogen.

Die wichtigsten Bestandteile der Beschreibung eines Testfalls sind in Abbildung 5 zu sehen.

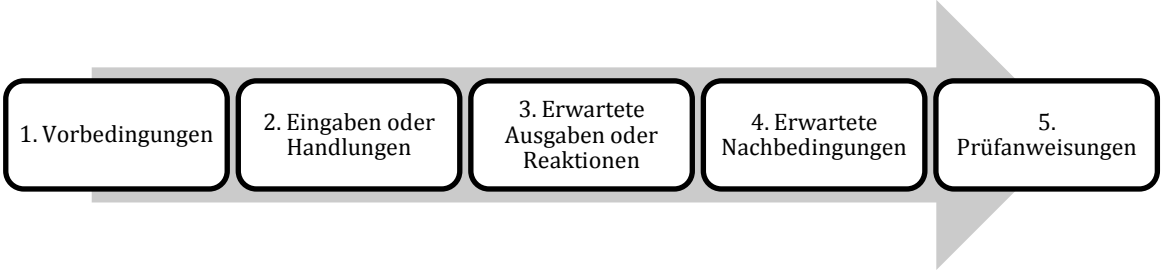

Abbildung 5: Bestandteile eines Testfalls

Ein Testfall wird in fünf Einzelbereiche unterteilt, die hintereinander ausgeführt werden. Zuerst müssen die Vorbedingungen vor der eigentlichen Testausführung hergestellt werden. Danach erfolgen die eigentlichen Testeingaben und Handlungen, die zur Durchführung des Testfalls notwendig sind. Im dritten Schritt werden die erwarteten Ausgaben und Reaktionen des Testobjekts auf die vorhin getätigten Eingaben aufgezeichnet. Im vorletzten Schritt des Testfalls werden die erwarteten Nachbedingungen ermittelt, die als Testergebnis der Durchführung erzielt wurden. Die Prüfungsanweisungen stehen im letzten Teil des Testfalls, das den Tester über sein Vorgehen aufklärt. Der Tester wird angewiesen, wie Eingaben an das Testobjekt übergeben werden und wie die Sollwerte abzulesen sind.

Beispiel für einen Testfall: Manueller Browsertest

1. Den Webbrowser (z.B. Firefox 3.5) öffnen.
2. Den Cache leeren.
3. 1. Einen Eintrag in die Adressleiste vornehmen. (z.B. www.orf.at)
   2. Eingabe-Taste betätigen
4. Webseite sollte (nach Betätigen der Eingabe-Taste) geladen und richtig dargestellt werden.
5. Vollständige und korrekte Darstellung der Webseite.
6. Die URL in die Adressleiste manuell eingeben und keine aus den Favoriten wählen. Die Webseite vorher in einem korrekt funktionierenden Browser laden und die beiden Ergebnisse vergleichen.

Tabelle 6: Testfall Beispiel

Ein Fehler, Mangel oder Anomalie liegt dann vor, wenn die Ausgaben oder die Nachbedingungen, die während der Testdurchführung ermittelt worden sind, von den erwarteten Werten abweichen.

2.2.1.3 Testszenario

In einem Testszenario (auch Testsequenz oder Testsuite genannt) werden mehrere fachlich oder technisch zusammengehörende Testfälle zusammengefasst. Dieses Szenario wird allgemein als Zusammenstellung von Testsequenzen beschrieben. Bei der Aneinanderreihung der einzelnen Testfälle, dient jeweils die Nachbedingungen des einen Tests als Vorbedingung des darauf folgenden Tests.

2.2.1.4 Fehler

„Fehl (niederhochdeutsch) – aus dem altfranzösischem faille entlehnt; es kommt heute selbstständig nur in der Fügung „ohne Fehl“, d.h. Fehler vor. Erst um 1500 erscheint Fehler, zunächst in der Bedeutung „Fehlschuß“, seit dem 18. Jahrhundert wie heute als „Versehen“ (Schreib-Rechenfehler) und „bleibender Mangel“. [Alp94]

Die Erklärung „Fehlschuss“ bringt es im Bereich der Software auf dem Punkt und bedeutet, die Anforderungen nicht zu erfüllen. Ein Fehler muss weder dokumentiert noch wiederholbar sein, um als solcher bezeichnet zu werden.

- Funktionale Fehler (failure)
- Usability Mängel
- Schlechte Wartbarkeit
- Mangelnde Robustheit
- Fehler von Schnittstellen

Tabelle 7: Fehler

Ein funktionaler Fehler, eine Fehlwirkung oder Fehlfunktion tritt erst bei der Ausführung eines Testobjekts nach außen in Erscheinung. Es ist die Abweichung zwischen dem spezifizierten Sollwert und dem beobachteten oder erhaltenen Ist-Wert. Man spricht hier auch vom Soll- und Ist-Verhalten.

Ein Usability-Mangel besteht dann, wenn die zu testende Software sich nur schwer bedienen lässt. Eine mangelhafte Usability wird meistens zwar wahrgenommen, ohne dass dann allerdings die genaue Ursache benannt wird.

Die Wartbarkeit einer Software wird durch einige Punkte in der Softwareentwicklung erreicht. Diese sind z.B. gute Spezifikation, die verwendeten Technologien, eingespieltes und zertifiziertes Team, gut lesbarer einfacher Code, Kommentare im Softwarecode, einfache Schnittstellen, gute Fehlerbehandlung. Mit diesen Punkten kann die Softwarewartbarkeit und somit auch die Qualität gesteigert werden.

• Das Softwaresystem leistet nicht das, was es laut Spezifikation leisten soll. • Ein Programm verhält sich nicht so, wie es in der Dokumentation oder in sonstigen Papieren, die der Anwender erhält, angekündigt ist. • Ein Programm verhält sich nicht so, wie es im Vertrag oder im Anforderungskatalog des Anwenders vorgesehen ist. • Ein Softwaresystem leistet nicht das, was der Benutzer vernünftigerweise erwartet.

Tabelle 8: Softwarefehler [Mic04]

2.2.1.5 Testspezifikation

Eine Testspezifikation ist ein Dokument, welches die Testfälle sowie die Testszenarien begründet und beschreibt, die in der Testdurchführung zu berücksichtigen sind. In der Testspezifikation werden die zur Testdurchführung notwendigen Aktivitäten festgelegt.

2.2.1.6 Resultate

Resultate oder Testergebnisse werden hauptsächlich vom Tester oder Entwickler definiert und evaluiert. Beim Testen werden Resultate produziert, die mit den vorher evaluierten Ergebnissen verglichen werden und somit das Testendergebnis bilden. Das Resultat bildet das Ende eines Testfalls und dieses kann entweder positiv oder negativ ausfallen.

2.2.2 Testdokumentation

Die Struktur und Inhalt einer Testdokumentation wird von der IEEE-Norm 829 (Abbildung 6) vorgegeben. Die Testdokumente sollten nach dieser Norm verwendet werden.

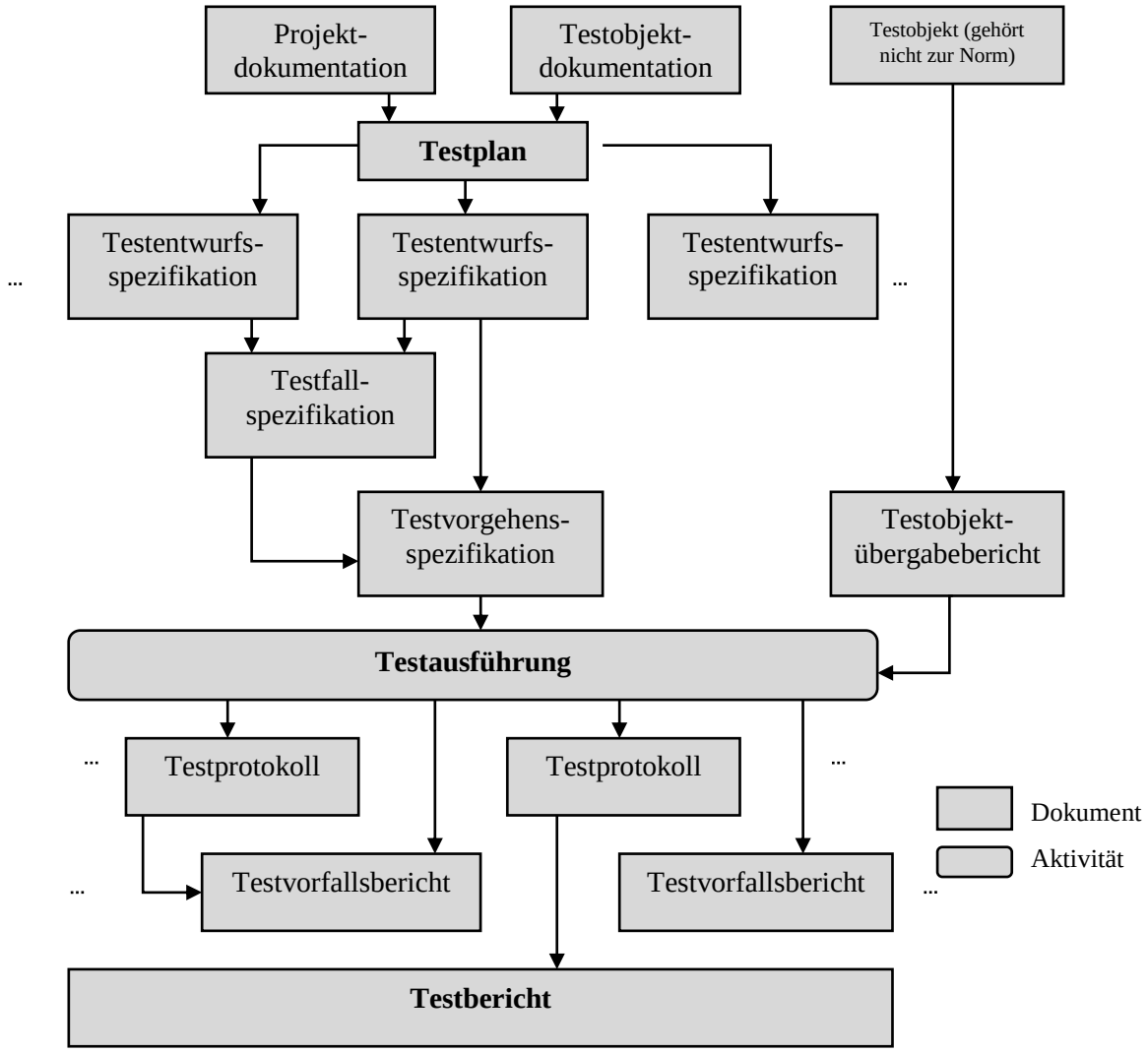

Abbildung 6: Testdokumente nach der IEEE-Norm 829 [Wal01]

2.2.2.1 Testplan

Beim Software-Testen spielt der Testplan die zentrale Rolle. Ein Testplan ist ein Dokument, das die Ziele, den Umfang, die Methoden, die Ressourcen, einen Zeitplan und die Verantwortlichkeiten für die beabsichtigte Testaufgabe beinhaltet. Ein Testplan sollte möglichst früh erstellt, benutzt und ggf. geändert werden. Der Testplan muss auch im Projektplan referenziert sein. Die Infrastruktur des Testplans sollte wie folgt aussehen.

- Testplanidentifikation
- Einleitung
- Testobjekte
- Zu testende Merkmale
- Nicht zu testende Merkmale
- Testvorgehen
- Bestanden-/Nichtbestanden-Kriterien für jedes Testobjekt
- Unterbrechungs- und Wiederaufnahmekriterien für Tests
- Ergebnisse (Dokumente) des Testprozesses
- Testaufgaben
- Testumgebung
- Verantwortlichkeiten
- Personelle Anforderungen und Schulungsanforderungen
- Zeitplan mit Testmeilensteinen
- Risikoabschätzung der Testplanung
- Verantwortliche für die Genehmigung des Testplans

Tabelle 9: Testplan Infrastruktur

2.2.2.2 Testentwurfsspezifikation

Die Testentwurfsspezifikation ist für die Teststrategie, die Funktionen und die Merkmale der Testfälle notwendig. Hier werden die einzelnen Teststrategien für einzelne Testbereiche präzisiert und definiert.

2.2.2.3 Testfallspezifikation

In diesem Dokument werden die Testfälle definiert, die vorher in der Testentwurfsspezifikation identifiziert wurden. Zu einer Testfallspezifikation gehören im Allgemeinen die Testfallkennzeichnung, ein Hinweis auf das Testobjekt, eine Ein-Ausgabespezifikation, eventuelle Anforderungen an die Testumgebung und die Reihenfolge der Testfälle.

2.2.2.4 Testvorgehensspezifikation

Das Testvorgehensspezifikations-Dokument befasst sich mit den eigentlichen Testverfahren, wodurch der genaue Ablauf der Testdurchführung für die jeweiligen Einzeltests festgelegt wird.

2.2.2.5 Testobjektübergabebericht

Im Testobjektübergabebericht werden die Testobjekte gemäß der Konfigurationsliste im Detail aufgelistet. Dieses Dokument ist dann von Bedeutung, wenn das Entwicklungsteam nicht mit dem Testteam identisch ist und das Abnahmetesten vertraglich geregelt wird.

2.2.2.6 Testprotokoll (*„Test-log“*)

Das Testprotokoll beschreibt in zeitlicher Reihenfolge alle Details jeden Tests. Dazu gehören genaue Fehlermeldungen, alle Abstürze und die Anforderungen eines Operatoreinsatzes.

2.2.2.7 Testvorfallsbericht

Im Testvorfallsbericht werden alle aufgetretenen Abweichungen zusammengefasst und stellt den Bezug zu den Testspezifikationen sowie Testprotokollen her.

2.2.2.8 Testbericht

Der Testbericht fasst die Ergebnisse der Testaktivität zusammen. Es wird auch eine Bewertung der in den Anforderungen formulierten Ziele durchgeführt.

2.2.3 Testen

Testen ist keine eindimensionale Tätigkeit, sondern erfordert Kreativität und Ehrgeiz, sowie etwas Bosheit. Denn ohne diese Eigenschaften wird man es kaum lange aushalten in dieser Tätigkeit. Testen ist jener Prozess, der alle Testaktivitäten beinhaltet, die dem Ziel dienen, für ein Software-Produkt die korrekte und vollständige Umsetzung der Anforderungen sowie das Erreichen der festgelegten Qualitätsanforderungen nachzuweisen. Zum Testen gehören in diesem Sinne auch Aktivitäten zur Planung, Steuerung, Vorbereitung und Bewertung, welche der Erreichung der genannten Ziele dienen. Der Softwaretest wird oft als analytische Maßnahme definiert, die erst nach Erstellung des Prüfgegenstandes durchgeführt werden kann. Weiter unterteilt werden kann nach

- dynamische Maßnahmen, die eine Ausführung der Software erfordern (Bsp. Überdeckungstest)
- statische Maßnahmen, die auf eine Ausführung der Software verzichten können (Bsp. Verifikation, Code-Reviews)

Tabelle 10: Test-Maßnahmen

Als Abgrenzung zu den analytischen Maßnahmen sind konstruktive Maßnahmen zu sehen, die bereits bei der Erstellung vorliegen bzw. die Erstellung begleiten.

Es gibt mehrere Definitionen für den Begriff „Softwaretest“:

Nach ANSI/IEEE Std. 610.12-1990 ist ein Software-Test *„the process of operating a system or component under specified conditions, observing or recording the results and making an evaluation of some aspects of the system or component.“*

Eine andere Definition liefert Ernst Denert, wonach der *„Test […] der überprüfbare und jederzeit wiederholbare Nachweis der Korrektheit eines Softwarebausteines relativ zu vorher festgelegten Anforderungen“* ist. [Den92]

2.2.4 Webtest

Bei einem Webtest handelt es sich um Tests von Web-Applikationen. Eine Web-Applikation ist ein Computer-Programm, das auf einem Webserver ausgeführt wird. Eine Interaktion mit dem Benutzer findet ausschließlich über einen Webbrowser statt. Der Computer des Benutzers *(Client)* und der des Dienstanbieters *(Server)* sind hierzu über ein Netzwerk wie das Internet oder über ein Intranet (geschlossenes Firmennetz) miteinander verbunden, so dass die räumliche Entfernung zwischen Client und Server unerheblich ist.

2.2.5 Box-Tests

Beim Testen gibt es unterschiedliche Sichten auf bzw. in das Testobjekt, die durch die Box-Tests (Black-Box, White-Box, Grey-Box) beschrieben werden. Gelegentlich werden die Box-Tests mit den Box-Verfahren verwechselt, weil die gleichnamigen Tests und Verfahren in engem Bezug zueinander stehen. Der folgende Abschnitt erläutert die Unterschiede.

2.2.5.1 Black-Box-Test

Der Black-Box Test bezieht sich hauptsächlich auf die Person des Testers. Black-Box-Test bedeutet, dass der Tester den Programmcode (Softwarecode) nur nach den Funktionen beurteilt und die inneren Vorgänge nicht zu berücksichtigen hat. Der Tester weiß normalerweise nicht, wie die Funktion im Programmcode implementiert wurde. Auch wenn für eine Funktion oder für ein Unterprogramm zwei unterschiedliche Lösungen möglich sind, macht das für den Tester keinen Unterschied. Es kommt auf das Ergebnis an, das der Tester bei seinem Test erhält. Die Mitglieder der Testgruppe sollten sich auf den Standpunkt eines Kunden oder Benutzers des zu testenden Programms stellen. Von den Testern sollte kein Vorwissen vorausgesetzt werden, das die Entwickler haben.

Bei Black-Box Tests werden nur öffentlich zugängliche Schnittstellen (wie zum Beispiel die Benutzerschnittstelle oder die veröffentlichte Schnittstelle für Anwendungsprogrammierung[5]) verwendet. Die Black-Box Testfälle werden mit Black-Box Verfahren erstellt. Black-Box-Verfahren sind Testmethoden zur Erstellung von Black-Box-Tests.

Für die Nutzung von Black-Box-Tests sind Informationen über die internen Vorgänge nicht notwendig, in diesem Test werden nur die fehlerfreie und vollständige Umsetzung der Anforderungen überprüft. Es werden nur Tests durchgeführt, welche die internen Strukturen des Programmes nicht betrachten. Black-Box-Tests können zu jeder Zeit und in jedem Stadium der Entwicklung durchgeführt werden. Zu den Black-Box-Tests gehören der Zufallstest und die funktionalen Äquivalenzklassenbildung. Der Zufallstest ist eine dynamische Black-Box Testtechnik, bei dem die Testfälle nach statistischen Kriterien gewählt werden. Die funktionale Äquivalenzklassenbildung ist wie der Zufallstest eine dynamische funktionsorientierte Testtechnik. Die Äquivalenzklassen-Testfälle werden aus den Spezifikationen abgeleitet.

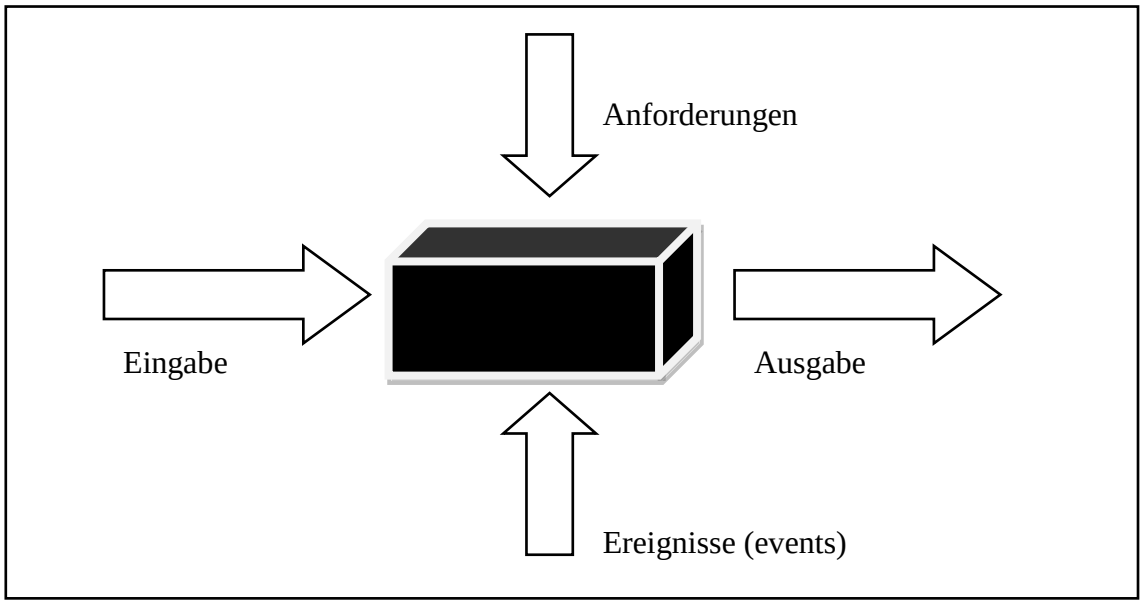

Abbildung 7: Black-Box-Test

Ein Black-Box Testfall benötigt folgende Schritte. Die Eingabe erfolgt bei diesem Test über eine externe Eingabequelle (z.B. Tastatur oder Maus). Es werden nur solche Elemente getestet, die eine Interaktion oder eine Ausgabe ermöglichen. Black-Box Testen kann man somit in fünf Fehlerklassen unterteilen (Tabelle 11).

[5] Anwendungsprogrammierung Schnittstelle: API aus dem engl. application programming interface, ist eine Schnittstelle, die von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird. [Wik03]

- Inkorrekte oder fehlende Funktionen
- Schnittstellenfehler
- Fehler in Datenstrukturen oder externem Datenbankzugriff
- Performancefehler
- Fehler bei Initialisierung und Terminierung von Abläufen

Tabelle 11: Fehlerklassen beim Black-Box Testen

2.2.5.2 White-Box-Test

Der White-Box-Test gehört in der Software-Technik zu den Software-Tests. Es ist eine Methode, mit dem ein Programm teilweise validiert und verifiziert wird. Es ist das Gegenstück zum Black-Box-Test und gehört auch zu den Box-Tests.

Ein White-Box-Test prüft die innere Struktur eines Testobjektes oder Programmcodes auf Korrektheit und Vollständigkeit. Die Systemstrukturen und Programmabläufe sind bekannt und der Programmcode wird bei einem White-Box-Test analysiert. Dieser Test wird daher auch Strukturtest genannt. Wie beim Black-Box gibt es auch hier Verfahren (White-Box Verfahren), die zur Ermittlung von Testfällen eingesetzt werden. In Abbildung 8 wird schematisch der Aufbau eines White-Box Testfalls gezeigt. Um einen Testfall überhaupt erstellen zu können, benötigt der Entwickler genaue Kenntnisse über die internen Strukturen und Abläufe des Programmes. White-Box-Tests werden hauptsächlich von Entwicklern geschrieben, die auch den Programmcode verfasst haben.

Das Ziel des White-Box-Tests ist es, alle Anweisungen, Bedingungen und Pfade mindestens einmal auszuführen. Somit kann sichergestellt werden, dass es auch eine 100-prozentige Testabdeckung vorhanden ist.

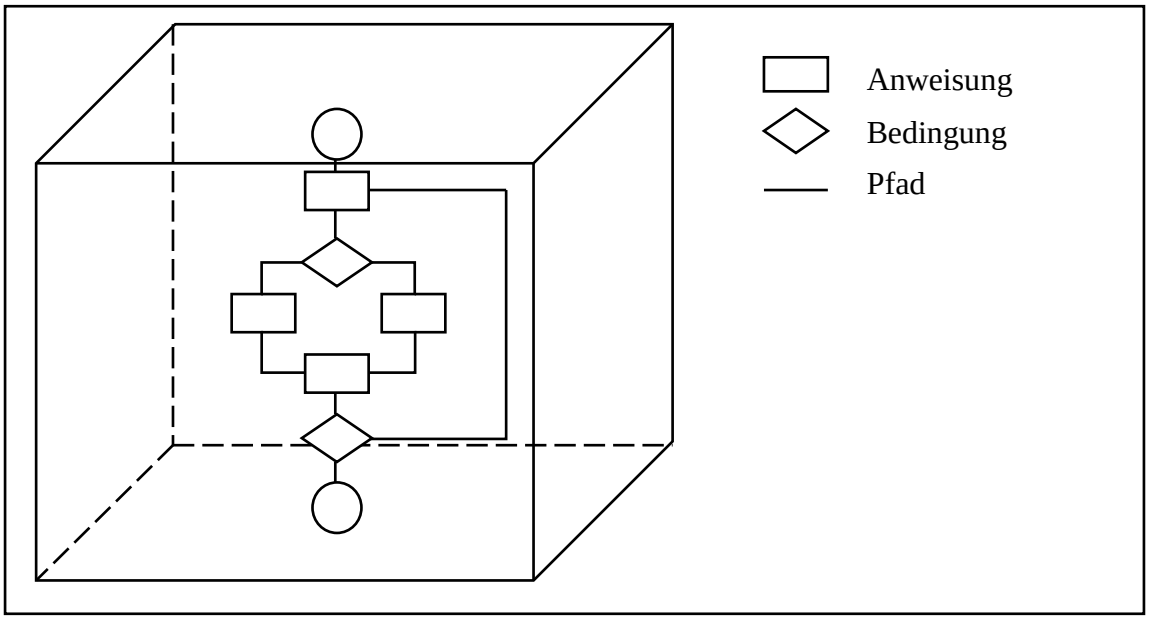

Abbildung 8: White Box-Test

2.2.5.3 Black-Box-Tests vs. White-Box-Tests

Beide Box-Tests haben in der Praxis ihre Vorteile und Nachteile. White-Box-Tests sind allgemein aufwendiger in der Durchführung als Black-Box-Tests, da sie ein größeres Testumfeld benötigen. Die Unterschiede zwischen Black-Box und White-Box-Tests sind in der Tabelle 12 erklärt.

Black-Box	**White-Box**
• Testfälle gehen von der Spezifikation aus. • Struktur und Programminterna sind bei der Erstellung der Testfälle nicht bekannt. • Testüberdeckung wird vom spezifizierten Ein-Ausgabeverhalten gemessen.	• Testfälle gehen von der Struktur aus. • Testfälle werden vom Entwickler beschrieben. • Testüberdeckung wird an Hand des Codes gemessen.

Tabelle 12: Unterschiede zwischen Black-Box und White-Box-Tests

2.2.5.4 Grey-Box-Test

Im Grey-Box-Test werden die beiden vorher erwähnten Box-Tests (Black- und White-Box-Tests) vereint. Grey-Box ist eine Komponente im Programm, die über ihre Schnittstellen (Interfaces) nach außen definiert und beobachtet werden kann. Da der Grey-Box-Test keine eigenen Methoden vorgibt, können ihm weder spezifische Tools noch eigene Qualitätsanforderungen zugeordnet werden. Der Entwickler eines Grey-Box-Tests muss sich im Aufbau des Programmes (also im speziellen mit dem Source Code) auskennen, um dafür eine Komponente programmieren zu können. Der Tester benutzt diese Komponente, um seine Black-Box-Tests darauf anzuwenden. Grey-Box-Tests sind in der Regel automatisierte Tests, die spezielle Interfaces vom Programm benutzen, um bestimmte Zustände im Programm auszulösen.

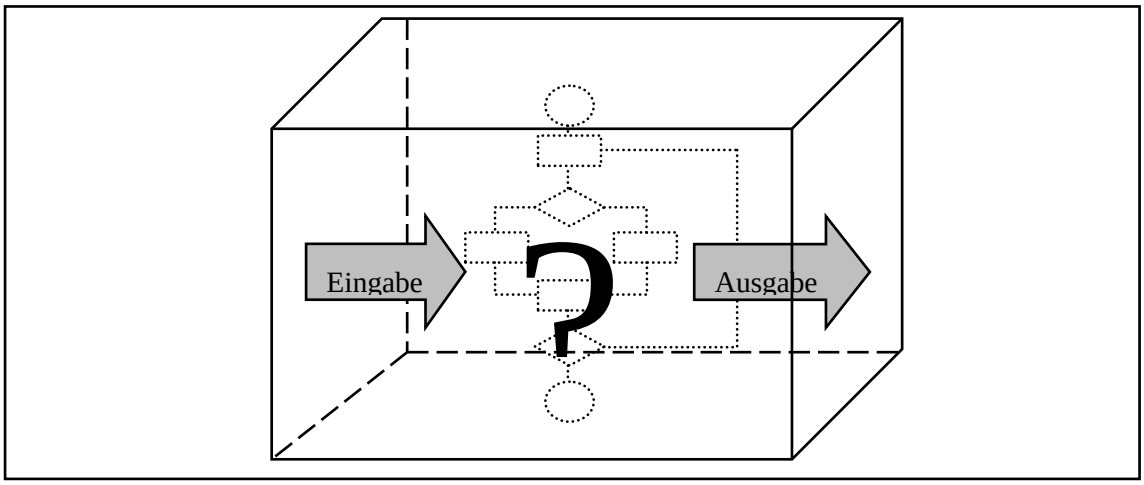

Abbildung 9: Grey-Box-Test

2.3 Testziele

Eine Software oder Web-Anwendung soll ein bestimmtes Ziel erfüllen oder eine bestimmte Leistung erbringen. An diesen Vorgaben orientiert sich das *„Testing“*, also jene Person oder auch Personen, die das Testen durchführen. Allgemein bedeutet das, dass das, was die Anwendung leisten soll, auch getestet werden muss. Deshalb muss das Testvolumen und die Testbereiche klar definieren werden. Nur so lassen sich realistische und innerhalb des Projektzeitrahmens erreichbare Testziele festlegen. Die Definition der Testziele kann und sollte vor der Testdurchführung durch entsprechende Projektbeteiligte erfolgen, da die zu verifizierenden Kennzahlen unabhängig vom eingesetzten Tool sind. Man beachte, dass sich ein Testziel von einem Testfall dadurch unterscheidet, dass ein Testziel nicht ausgeführt werden kann, außer wenn es mit Testwerten und erwarteten Resultaten angereichert wird. Bei jedem Test ist die vollständige, korrekte Funktionalität und die Robustheit der Anwendung zu prüfen. Besonders die Ausnahmebedingungen, Sonderfälle, Fehlersituationen und Antwortzeitverhalten sollen dabei getestet werden.

Die Ergebnisse der Tests werden in einem schriftlichen Protokoll dokumentiert und als Arbeitsergebnis geliefert. Je nach Anforderung werden Testprotokolle auch als Zwischenergebnisse bereits vor Ende der gesamten Tests erstellt. Die Testziele nennt man auch die Testendekriterien[6].

2.3.1 Wann ist das Testen beendet?

Da es unmöglich ist, alle Tests durchzuführen, ergibt sich die Frage, wann das Testen enden soll?

Wird zu wenig getestet, zeigt sich dies sofort in Systemdefekten. Das Programm funktioniert also nicht wie gewünscht.

Wird hingegen zu viel getestet, ist die Gefahr sehr groß, dass Ressourcen verschwendet werden und die Testkosten den Nutzen (bei weitem) überschreiten, da keine oder nur noch ganz wenige Fehler gefunden werden.

Um das Testende zu bestimmen, gibt es sogenannte Testendekriterien. Diese können einen ökonomischen oder statistischen Hintergrund besitzen.

Ökonomische Testendekriterien sind z.B. Finanzen und Zeit. Das Testen endet in diesen Fällen, wenn die finanziellen Mittel zur Neige gehen oder die Zeit knapp wird.

Bei statistischen Testendekriterien wird hauptsächlich die Überdeckung als Kriterium verwendet. Dabei kann sie aber verschieden definiert werden. Z.B. kann das Testende erreicht sein, wenn jede Funktion (bzgl. Anforderungen) einmal getestet wurde. Die Überdeckung kann sich aber auch auf den Programmcode beziehen. In diesem Fall hört man auf, wenn z.B. 95% vom Code getestet wurde, oder wenn jede Methode einmal beim Testen aufgerufen wurde. Es ist auch möglich aufzuhören, wenn jeder Zweig oder Pfad einmal durchlaufen wurde.

Eine Pfadüberdeckung ist allerdings so gut wie nie zu erreichen, da manche Schleifen beliebig oft durchlaufen werden können. Bei der Pfadüberdeckung müsste die Schleife dann in den Tests zuerst kein Mal, dann 1mal, dann 2mal, usw. durchlaufen werden. Eine vollständige Pfadüberdeckung kann in diesem Fall nie erreicht werden.

2.4 Testgrundsatz

Testen soll die Anwesenheit von Fehlern zeigen und nicht das Gegenteil davon. Ein vollständiges Testen ist nicht möglich, da eine 100-prozentige Abdeckung nie erreicht werden kann. Es sollte so früh wie möglich mit dem Testen begonnen und definierte Ziele verfolgt werden. Die Fehlerfreiheit von Applikationen kann mit Testen nicht nachgewiesen werden. Testen ist abhängig vom Umfeld und keine zwei Systeme sind auf die exakt gleiche Art und Weise zu testen. Es entstehen auch zunehmende Testresistenzen, d.h. Testwiederholungen zeigen keine neuen Fehlerwirkungen mehr auf.

---

[6] Engl. Exitcriteria

1. Testen zeigt die Anwesenheit von Fehlern
2. Vollständiges Testen ist nicht möglich
3. Mit dem Testen frühzeitig beginnen
4. Häufung von Fehlern
5. Zunehmende Testresistenz
6. Testen ist abhängig vom Umfeld
7. Trugschluss: Keine Fehler bedeutet ein brauchbares System

Tabelle 13: Grundsätze zum Testen [Spi07]

2.5 Automatisierte Tests

Zu Beginn werden Tests in der Regel hauptsächlich manuell durchgeführt, da man damit ohne großen zeitlichen Aufwand viele Probleme und Programmfehler ausfindig machen kann. Dies hat leider den Nachteil, dass man bei kleinen Änderungen in einem sehr umfangreichen Programm viele einzelne Tests durchführen muss und dafür oft nicht genügend Ressourcen zur Verfügung hat. In diesem Zusammenhang kommen dann die sogenannten Regressionstests zur Verwendung. Darunter versteht man in der Softwareentwicklung eine ein- oder mehrmalige Wiederholung aller Testfälle, die auch nur einen Teil der Testfälle beinhalten kann. Regressionstests sollen unbeabsichtigte Reaktionen von Änderungen in bereits getesteten Teilen vor einer Auslieferung der Software aufdecken. Solche Änderungen kommen während der Pflege, Korrektur oder Änderung einer Software vor.

2.5.1 Was sind automatisierte Tests

Testautomatisierung ist die Verwendung eines Testwerkzeugs mit dem Ziel, das Testen zu unterstützen.

„Ein Testtool ist ein automatisiertes Hilfsmittel, das bei einer oder mehreren Testaktivitäten, beispielsweise Planung und Verwaltung, Spezifikation, Aufbau von Ausgangsdateien, Testdurchführung und Beurteilung, Unterstützung leistet.“ [PKS00]

Vorteile	Nachteile
• Eine umfangreiche Menge von Tests kann ohne Kontrolle und automatisch durchgeführt werden, z.B. über Nacht. Dies gilt besonders für Regressionstests. • Automatisierung von sich wiederholenden und oft lästigen Testaktivitäten führt zu höherer Zuverlässigkeit dieser Aktivitäten und zu höherer Zufriedenheit im Testteam, was wiederum zu höherer Produktivität führt.	• Die meisten kommerziell erhältlichen Testwerkzeuge sind sehr teuer. • Viele Testwerkzeuge erfordern viel Aufwand für den erfolgreichen Einsatz und sind daher nur für große Projekte oder für große Firmen mit vielen Projekten profitabel.

Tabelle 14: Vor und Nachteile von automatisierten Tests [Win04]

Die Testwerkzeuge für automatisierte Tests sind im Kapitel 3.3.5 auf Seite 63 genauer beschrieben.

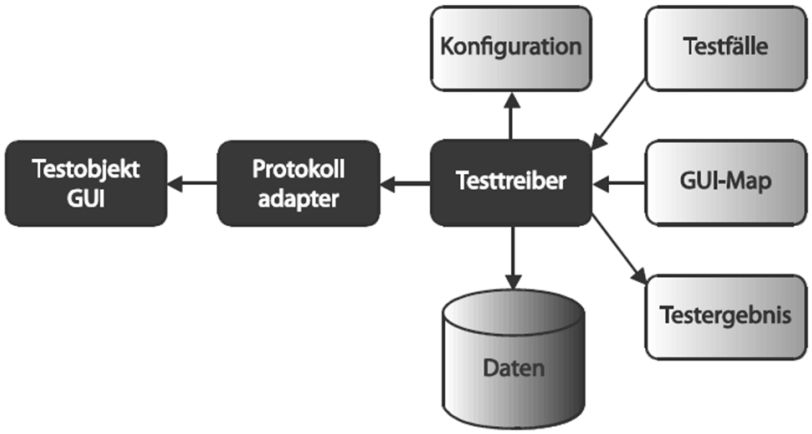

Abbildung 10: Schema für automatisierte GUI-Tests [INS10]

2.5.2 Warum Testautomatisierung

Aufgrund der langwierigen und Ressourcenhungrigen manuellen Tests fand man schnell den Weg zu automatisierten Tests. Testautomatisierung wurde vor einigen Jahren als ein echtes Schlagwort schlechthin im Projektalltag verwendet. IT-Projektleiter wollten so viel und so schnell wie möglich ganze Tests automatisieren, um den Gesamtaufwand für Tests zu minimieren. Einmal eingestellt wollen die IT-Projektleiter schnell und effizient jeden Tag die automatisch erstellten Resultate der nächtlichen Testergebnisse sehen. Was im ersten Augenblick auf dem Papier sehr gut aussah erfüllte sich später nur zu einem gewissen Teil, da eine viel zu frühe Testautomatisierung das Projektbudget erheblich zusätzlich belastet hat.

Die Wirtschaftlichkeit der Testautomatisierung zeigt die Studie von Dustin aus dem Jahre 1999 beim Vergleich des Aufwands von manuellen und automatisierten GUI-Softwaretests.

	Testvorbereitung			Testdurchführung (pro Testlauf)		
	Durchschn.	Min.	Max.	Durchschn.	Min.	Max.
Automatisiert	19.2 h	10.6 h	56.0 h	0.21 h	0.1 h	1.0 h
Manuell	11.6 h	10.0 h	20.0 h	3.93 h	0.5 h	24.0 h

Abbildung 11: Studie von Dustin [INS10]

Die Testvorbereitung für automatisierte Tests ist durchschnittlich in etwa doppelt so hoch wie für manuelle Tests, diese Zeit reduziert sich aber während der Testdurchführung pro Testlauf auf ein Minimum.

1. Testfälle erzeugen
2. Testfälle abarbeiten
3. Ergebnisse überprüfen
4. Resultat protokollieren und ggf. Versagen anzeigen

Tabelle 15: Vorgehensweise beim Testen

Testautomatisierung bedeutet, einen oder mehrere dieser Schritte (Tabelle 15) programmgesteuert auszuführen. Siehe auch Regressionstests in diesem Zusammenhang auf Seite 43 die „Diversifizierende Testmethoden“.

2.5.3 Infrastruktur

Beim automatisierten Test ist die Infrastruktur die ausschlaggebende Komponente. Diese umfasst sämtliche Ressourcen wie Arbeitsrechner, Server, Netzwerke, Netzwerkdienste, Dateiserver, Programme, Testtools, Betriebssysteme, Schnittstellen, Internet, Intranet und Datenbanken, die beim automatisierten Tests verwendet werden können. Ein Beispiel, wie ein automatisierter Test ausgeführt werden kann, zeigt Abbildung 12. Die Testumgebung auf dem Server 1 testet die Anwendung auf dem Server 2. Beide Server kommunizieren mit einer Datenbank, die als Speicher für die Anwendungsdaten sowie für die Testdaten benötigt wird. Die Kommunikation der Informationen findet über das Intranet (internes Netz) oder das Internet (World Wide Web) statt.

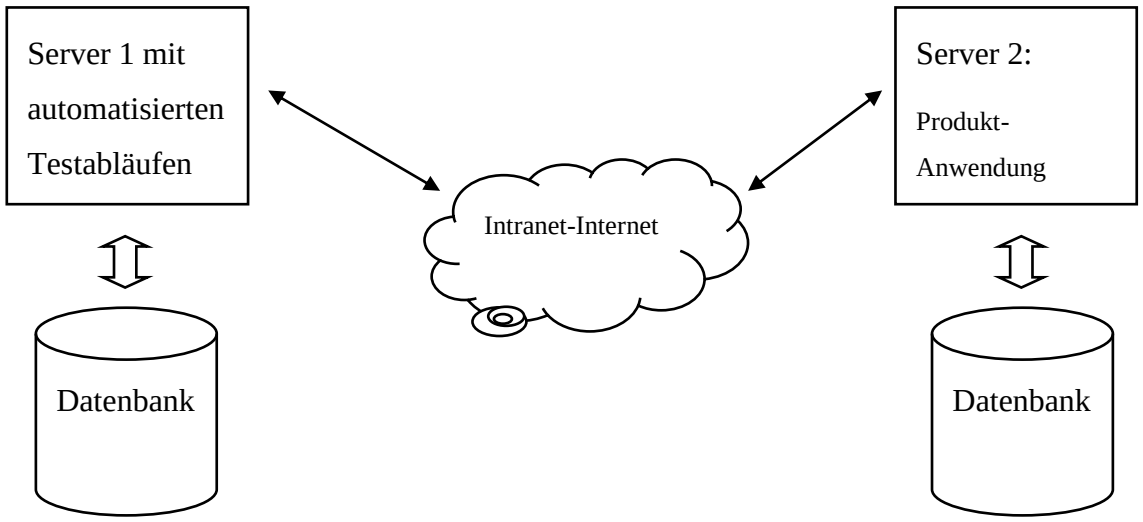

Abbildung 12: Infrastruktur einer einfachen Testumgebung

2.5.4 Vergleiche zu manuellen Tests

Automatisierte Tests sind wie manuelle Tests ein Teil des Testplans. Automatisierte Tests können sowohl die Black Box Tests der manuellen Tests als auch die White Box Tests umfassen.

	Automatisierte Tests	**Manuelle Tests**
Vorteil	• Black Box Tests • White Box Tests • Regressionstests über Nacht möglich • Verwendung von Testtools • Kostenersparnis • Zeitersparnis	• Black Box Tests • Geringer Aufwand • Kaum Schulung notwendig
Nachteil	• Findet nur Fehler innerhalb der Methode, Prozedur oder Bedingung • Teure Testtools • Hoher Aufwand bei der Erstellung der Tests	• Keine White Box Tests • Keine Regressionstests • Ermüdend • Langwierig • Keine Verwendung von Testtools

Tabelle 16: Vergleich zu manuellen Tests

3. Aspekte von Webtest

3.1 Phasen des Testens

1. Planungsphase: Sie legt die Rahmenbedingungen des Tests fest.

2. Durchführungsphase: Sie besteht aus der eigentlichen Durchführung des Tests.

3. Auswertungsphase: Sie besteht aus dem Schreiben eines Ergebnisreports.

3.1.1 Modultest

Ein Modultest befasst sich mit der untersten Ebene einer Software, der sogenannten Modulebene. Modultests werden von den Entwicklern selbst erstellt, die auch normalerweise die Module erstellen. Zur Spezifikation der Testfälle dient die Modulschnittstellenspezifikation. Anhand dieser Modulschnittstellenspezifikation werden die einzelnen Modultests erstellt. Im Modultest kommen Black-Box und White-Box Tests zum Erstellen der Testfälle zum Einsatz. Die Unit Tests befassen sich mit den Modultests. (siehe 4.1 Unit Tests auf Seite 66.)

3.1.2 Integrationstest

Im Integrationstest wird das Zusammenspiel zwischen den einzelnen Modulen getestet. Dazu werden diese zu sogenannte Subsysteme zusammengesetzt und getestet. Die vorher gewählte Teststrategie legt den genauen Zeitpunkt des Tests fest und wann welche Module im speziellen verwendet werden. Die Verfügbarkeit der einzelnen Module spielt dabei eine große Rolle. Die Grundlage für den Integrationstest legen zwei Entwürfe (der Systementwurf sowie der Programmstrukturentwurf) fest. Auch hier sind Black-Box- sowie White-Box-Tests wie beim Modultest zum Erstellen der Testfälle notwendig.

Durch den mit wachsender Komponentenanzahl steigenden Zeitaufwand werden angrenzende Komponenten zusammengelegt. Die zusammengeführten Komponenten werden im Integrationstest als eine eigene Komponente betrachtet. Dieser Test wird Bottom-Up-Integrationstest genannt. Es wird so lange getestet, bis die Testläufe der Testumgebung mit dem späteren Produktivsystem identisch sind. Im Top-Down-Integrationstest beginnt man mit der ranghöchsten Komponente und schreitet in Richtung der rangniederen Komponente. Der Vorteil vom Top-Down-Integrationstest ist, dass schon zu Beginn ein Produkt mit einem groben Ablauf zu erkennen ist und die noch nicht fertiggestellten Komponenten durch Dummies (Stub[7]) ersetzt werden.

[7] Stub: Wird ein Programmcode genannt, der anstelle eines anderen Programmcodes steht.

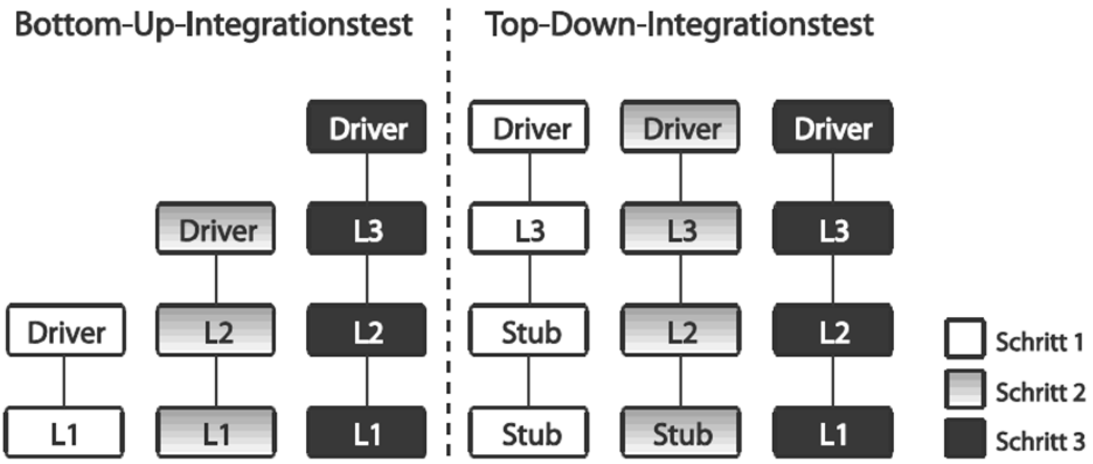

Abbildung 13: Integrationstests [INS10]

Die Nachteile eines Top-Down-Integrationstests sind:

- Dummies sind erforderlich.
- Erzeugung gewünschter Testsituationen wird mit zunehmender Integrationstiefe immer schwieriger.
- Am Ende des Integrationstests ist die Zeit für Systemtests oft knapp bemessen.

3.1.3 Systemtest

Beim Systemtest wird das gesamte Testobjekt mit der Leistungsbeschreibung verglichen. Dieser Test ist sehr kritisch, da neben den Testfallspezifikation und der Leistungsbeschreibung auch auf die Benutzerdokumentation zurückgegriffen werden muss. Mit Hilfe dieser wird dann eine genaue Definition der Schnittstellen des Testobjekts erstellt, daher kommen in dieser Teststufe ausschließlich Black-Box-Tests in Frage. Das ist auch der größte Unterschied zu den vorhin erwähnten Modul- und Integrationstests. Beim Erstellen der Testfälle sollte man eine große Abdeckung der Kategorien bedenken.

Solche Kategorien können in Sicherheit, Leistung, Handhabung, Zuverlässigkeit, Wartbarkeit, Dokumentation und Regressionstest unterteilt werden (Tabelle 17).

Dabei ist die Durchführung von Regressionstests besonders wichtig. Die fortschreitenden Versionsentwicklungen machen Regressionstests unverzichtbar. Eine sinnvolle Regressionstest-Durchführung kann nur mit einer leistungsfähigen Werkzeugunterstützung stattfinden.

- Sicherheit
- Leistung
- Handhabung
- Zuverlässigkeit
- Wartbarkeit
- Dokumentation
- Regressionstest

Tabelle 17: Systemtest-Kategorien

3.1.4 Abnahmetest

Der letzte Test ist der Abnahmetest. Dieser Test hat eine große rechtliche Bedeutung und dient hauptsächlich als vertrauensbildende Maßnahme für den Kunden. Dabei wird das Testobjekt mit den ursprünglichen Anforderungen verglichen. Diese Anforderungen können jederzeit aktualisiert werden und müssen auch vom Kunden bestätigt werden. Als Basis für den Abnahmetest dient der Vertrag zwischen dem Unternehmen und dem Kunden. In diesem Test kommen nur Black-Box-Tests zur Anwendung, da der Kunde keinen Zugriff auf den Source-Code des Projekts besitzt.

3.2 Testmethoden

Während statische Software-Testmethoden die Applikation nicht ausführen und sich mit Reviews und Inspektionen begnügen, benötigt die dynamische Testmethode eine ausführbare Applikation. Die Applikation wird dabei mit systematisch festgelegten Eingabewerten ausgeführt. Die Testmethoden werden in drei Kategorien unterteilt: Funktionsorientierte Testmethoden, Strukturorientierte Testmethoden und Diversifizierende Testmethoden.

3.2.1 Funktionsorientierte Testmethoden

Funktionsorientierte Testmethoden besitzen geeignete Voraussetzungen für die Anwendung auf objektorientierte Applikation. Mithilfe der Testmethoden werden Testfälle bestimmt und damit die Erfüllung der Systemspezifikationen überprüft. Das nennt man „gegen die Spezifikation Testen“.

3.2.1.1 Äquivalenzklassenbildung

Dabei werden mögliche Werte der Eingabe in Klassen eingeteilt. Bei der Verarbeitung einer Klasse nimmt man an, dass alle anderen Elemente der Klasse nicht zu Fehler führen. Mit diesen Informationen werden dann die Testfälle erstellt.

3.2.1.2 Grenzwertanalyse

Ein Spezialfall der Aquivalenzklassenbildung ist die Grenzwertanalyse. Die Grenzwertanalyse ist aus der Betrachtung entstanden, dass Fehler sehr oft an den *„Rändern“* der Äquivalenzklassen erscheinen. Hier werden nicht beliebige Werte wie sonst üblich getestet, sondern *„Grenzwerte“*.

3.2.1.3 Zustandsbasierte Testmethoden

Zustandsbasierte Testmethoden basieren auf Zustandsautomaten. Die Zustandsautomaten werden oft als UML[8]-Diagramme dargestellt. Die Zustände sind mit den Werten der lokalen Daten der Klasse bezeichnet. Fehlerfälle müssen dazu extra spezifiziert werden, da diese im UML-Diagramm nicht vorgesehen sind.

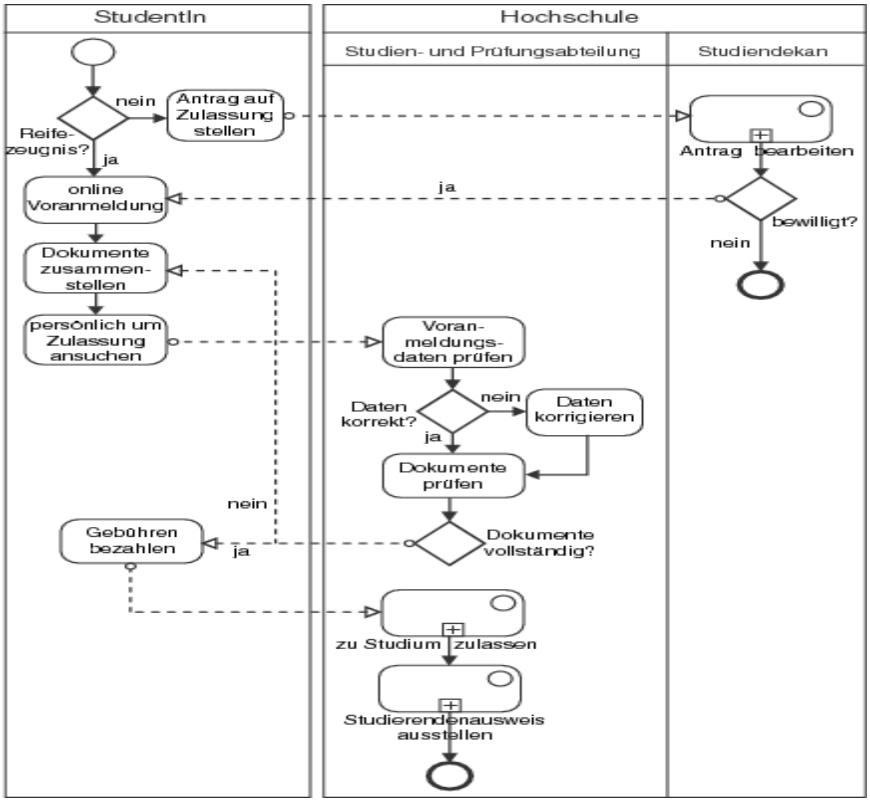

Abbildung 14: UML-Diagramm [INS10]

Eine vollständige Testfallermittlung kann nur beurteilt werden, wenn alle Zustände und Zustandsübergänge durchlaufen werden.

[8] UML: Unified Modeling Language (Vereinheitlichte Modellierungssprache)

3.2.2 Strukturorientierte Testmethoden

Strukturorientierte Testmethoden beeinflussen Testfälle auf Basis von Modellen der inneren Struktur (Applikation-Quellcodes). Das Modell einer Applikation ist der Kontrollflussgraph zur Darstellung der Kontrollstruktur. Diese Testmethode ist ein White-Box-Test. Neben dem *Kontrollflussorientierten Test* gibt es noch den *Datenflussorientierten Test*. Kontrollflussorientierte Tests beziehen sich auf logische Ausdrücke der Implementierung und Datenflussorientierte Merkmale konzentrieren sich auf den Datenfluss der Implementierung.

3.2.2.1 Kontrollflussorientierter Test

Kontrollflussorientierte strukturelle Testverfahren fordern die Überdeckung der Elemente der Kontrollstruktur. Dieses Testverfahren orientiert sich am Kontrollflussgraphen (Abbildung 15) der Applikation.

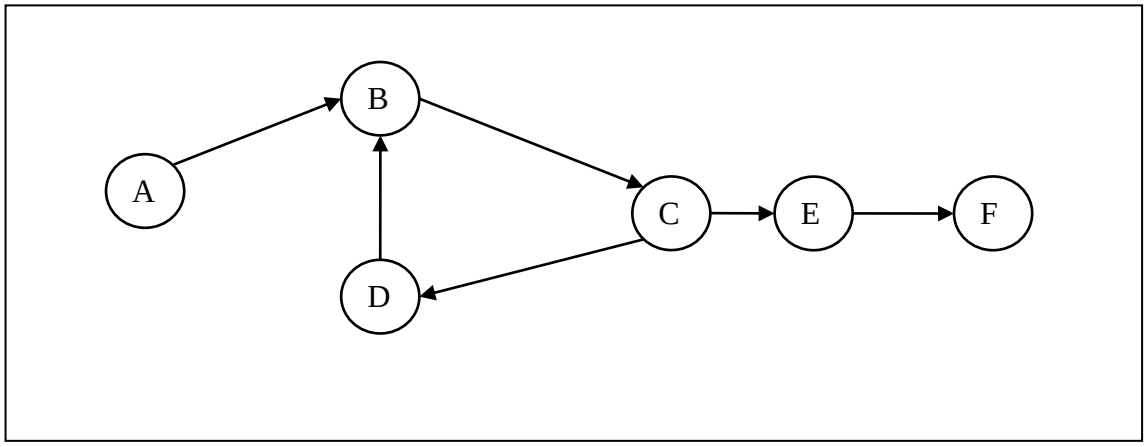

Abbildung 15: Kontrollflussgraph

Man unterscheidet folgende Arten des Kontrollflussorientieren Test:

- Anweisungsüberdeckung (C0):

  Alle Knoten werden mindestens einmal in einem Testfall abgedeckt und jede Anweisung mindestens einmal ausgeführt.

- Kantenüberdeckung (C1):

  Alle Kanten müssen mindestens einmal durchlaufen werden.

- Bedingungsüberdeckung (C2, C3):

  Die Testfälle müssen so erstellt werden, dass jede atomare und auch jede zusammengesetzte Teilentscheidung (Bedingung) einmal den Wert wahr und einmal den Wert falsch ergibt.

- Pfadüberdeckung (C4):

  Alle unterschiedlichen Pfade müssen in mindestens einem Testfall durchlaufen werden.

3.2.2.2 Datenflussorientierter Test

Datenflussorientierte Tests basieren auf dem Datenfluss, also dem Zugriff auf Variable. Sie können auch für den Test von Interaktionen von Operationen genutzt werden und sind daher für objektorientierte Applikationen besser geeignet als kontrollflussorientierte Techniken. Auch hier gibt es mehrere Kriterien.

- All defs-Kriterium:

  Der All defs-Test verlangt, dass jede Definition mindestens einmal referenziert wird ohne zuvor redefiniert zu werden. Die Fehlererkennungsrate liegt bei diesem Kriterium bei ca. 24%.

- All p-uses-Kriterium:

  Die „p-uses“ dienen zur Bildung von Wahrheitswerten innerhalb eines Prädikates (predicate-uses). Die Fehlererkennungsrate liegt bei diesem Kriterium bei ca. 34%. Es werden hauptsächlich Kontrollflussfehler erkannt.

- All c-uses-Kriterium:

  Unter dem „c-uses-Kriterium“ wird die Berechnung von Werten innerhalb eines Ausdrucks verstanden. Dieses Kriterium deckt ca. 48% aller c-uses-Fehler auf. Das c-uses-Kriterium erkennt speziell Berechnungsfehler.

Ein Schreibzugriff ist eine Definition (def) und ein lesender Zugriff wird als Referenz (c-use bzw. p-use) ausdrückt. Bei der Ausführung der Operation eines Objekts wird auf die Attribute schreibend und lesend zugegriffen.

3.2.3 Diversifizierende Testmethoden

Diversifizierende Testmethoden testen verschiedene Versionen einer Applikation gegeneinander. Dabei findet keine Gegenüberstellung zwischen den Testergebnissen und der Applikation-Spezifikation statt, sondern ein Vergleich der Testergebnisse verschiedener Versionen.

3.2.3.1 Regressionstests

Regressionstests kommt aus dem lateinischen „regredior“ (regressus sum = zurückschreiten) und man versteht darunter in der Softwaretechnik die Wiederholung aller oder einer Teilmenge aller Testfälle, um Nebenwirkungen von Veränderungen oder Neukonstruktionen in bereits getesteten Teilen der Software aufzuspüren. Solche Veränderungen oder Neukonstruktionen entstehen regelmäßig z. B. aufgrund der Pflege, Änderung und Korrektur von Software. Korrekturen können zum Beispiel die Bereinigung von Bugs sein. Der Regressionstest ist ein Teil der dynamischen Testtechniken. Weitere dynamische Testtechniken sind unter anderem Back-to-Back-Tests und Mutationen-Tests.

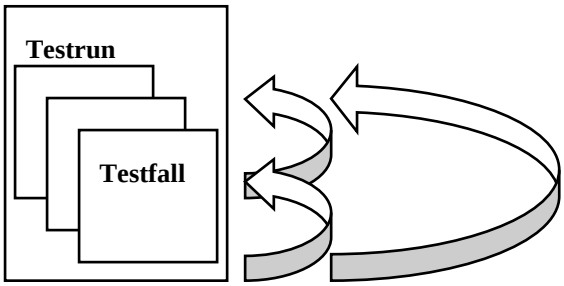

Abbildung 16: Regressionstests

In der Testpraxis steht der allgemeine Begriff der Regression für eine reine Wiederholung von Testfällen und Testgruppen (engl. test groups). Für Regressionstests werden in der Regel keine neuen Testfälle entwickelt, sondern alle schon vorher bekannten und vorhandenen Testfälle verwendet. Dieser Test ist somit eine wiederkehrende Aufgabe, wobei die Tests nicht neu sind. Diese Tests haben den Nachteil, dass sie durch die Anzahl der Testfälle sehr zeitintensiv und dadurch auch sehr kostspielig sind. Regressionstests sollten mit automatisiertem Test durchgeführt werden, da dies aufgrund des Wiederholungscharakters und der Häufigkeit dieser Wiederholungen sehr sinnvoll ist. Die Testfälle selbst werden anhand anderer Techniken spezifiziert und mit einem Soll-Ergebnis versehen, welches mit dem Ist-Ergebnis eines Testfalls verglichen wird. Die Testergebnisse eines aktuellen Tests haben keinen direkten Bezug zu den vorherigen Testläufen. Ein Testfall gilt beim Regressionstest als erfolgreich absolviert, wenn die Ausgaben identisch sind.

3.2.3.2 Regressionstests in Echtzeitsystemen

Der Regressionstest stellt bei Echtzeitsystemen ein großes Problem dar, da bei Echtzeitsystemen eine Wiederholung (also Regression) des Tests im strengeren Sinn nicht möglich ist. Bereits geringe Änderungen an der Hardware des Systems führen zu einem veränderten Verhalten. Eine Lösung dieses Problems liegt in der Implementierung eines automatischen Testsystems. Der Aufwand für die Implementierung automatischer Tests wird aus folgenden Gründen meistens gescheut (Tabelle 18).

- das automatische Testsystem muss alle Funktionen des Prüflings abdecken
- das automatische Testsystem muss parallel zum Prüfling entwickelt werden
- das automatische Testsystem muss parallel zum Prüfling angepasst werden
- eine Hardware-Änderung führt zu einem Neu-Aufsetzen der Testergebnisse, gegen die verglichen werden soll.

Tabelle 18: Aufwand für automatische Tests

Die Vorteile einer Automatisierung der Tests zeigt Abbildung 17 an einem kleinen Beispiel. Die Erstellung automatisierter Testfälle führt in diesem Beispiel dazu, dass in der zweiten Regression die Personentage um 80% und die Ausführungszeit um 8 Tage reduziert wird. Dies hat zur Folge, dass man die überschüssige Fachkraft in neue Tests einsetzten kann.

	Regression 1	**Regression 2**
Manual Test Cases	675	18
Automated Test Cases	454	1011
Total	1029	1029
% Automation	44,1	98,3
Execution Time (Days)	13	5
Execution Effort (Persondays)	**130**	**26**

Abbildung 17: Aufwand bei automatisierten Regressionstests [HCL10]

3.2.3.3 Back-to-Back-Test

Beim Back-to-Back-Test entstehen verschiedene gegeneinander zu testende Versionen aus der n-Versionen-Programmierung, d. h. die Programmierung verschiedener Versionen einer Software nach der gleichen Spezifikation. Die Unabhängigkeit der Programmierteams ist dabei eine Grundvoraussetzung.

Dieses Verfahren ist sehr teuer und nur bei entsprechend hohen Sicherheits-Anforderungen gerechtfertigt. [Wik06]

Back-to-Back-Tests werden zwischen Simulations-Modell und automatisch erzeugtem Code empfohlen. [Ele10]

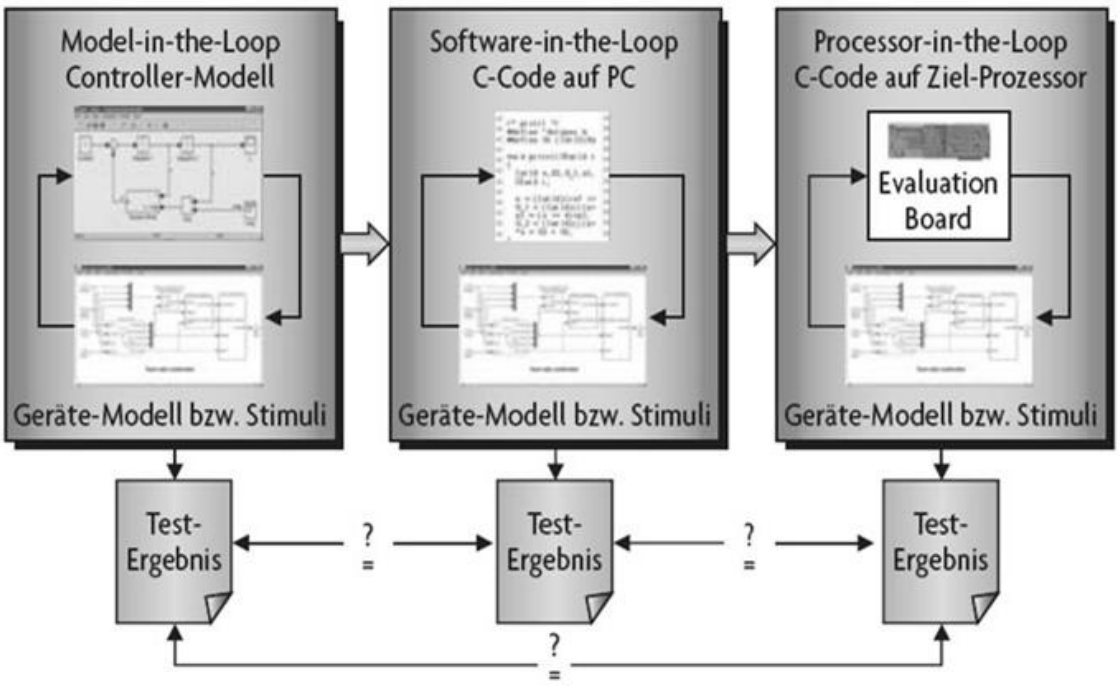

Abbildung 18: Back-to-Back Test [Ele10]

3.3 Web-Elemente

In diesem Abschnitt werde ich die Web-Elemente auf einer HTML-Seite etwas näher erläutern und wie diese realisiert werden. HTML[9] ist eine textbasierte Auszeichnungssprache zur Strukturierung von Inhalten wie Texte, Bilder und Hyperlinks in Dokumenten. Die strukturierte Darstellung von Inhalten nennt man Web-Elemente. Derzeit befindet sich die HTML-Entwicklung in der Version 5 (HTML5-W3C Working Draft 25 August 2009) [Ian09] . Weitere Elemente wie CSS[10], DHTML[11], XML[12] und Ajax[13] sind nach und nach hinzugefügt und erweitert worden.

3.3.1 Grundlagen der Web-Elemente

Web-Elemente sind in einem Dokument jene Inhalte, die als Tabellen, Textboxen, Textareas, Formularelemente oder Bilder dargestellt werden. Die Maus- und Tastatur-Eingaben auf diesen Elementen können durch sogenannte „Record/Replay“ Tools aufgezeichnet und beim Testen wiedergegeben werden. Die Web-Elemente sind die Hauptbestandteile einer Internet-Webseite. Die Darstellung erfolgt in einem Webbrowser, die den HTML Text interpretiert. Die Interpretation einer Seite ist dem Browser-Hersteller überlassen. Das W3C[14] Konsortium entwickelte bald die Acid-Browsertests (Abbildung 19), um den Browser-Hersteller die Möglichkeit zu geben, ihre Browser auf Webstandards zu überprüfen.

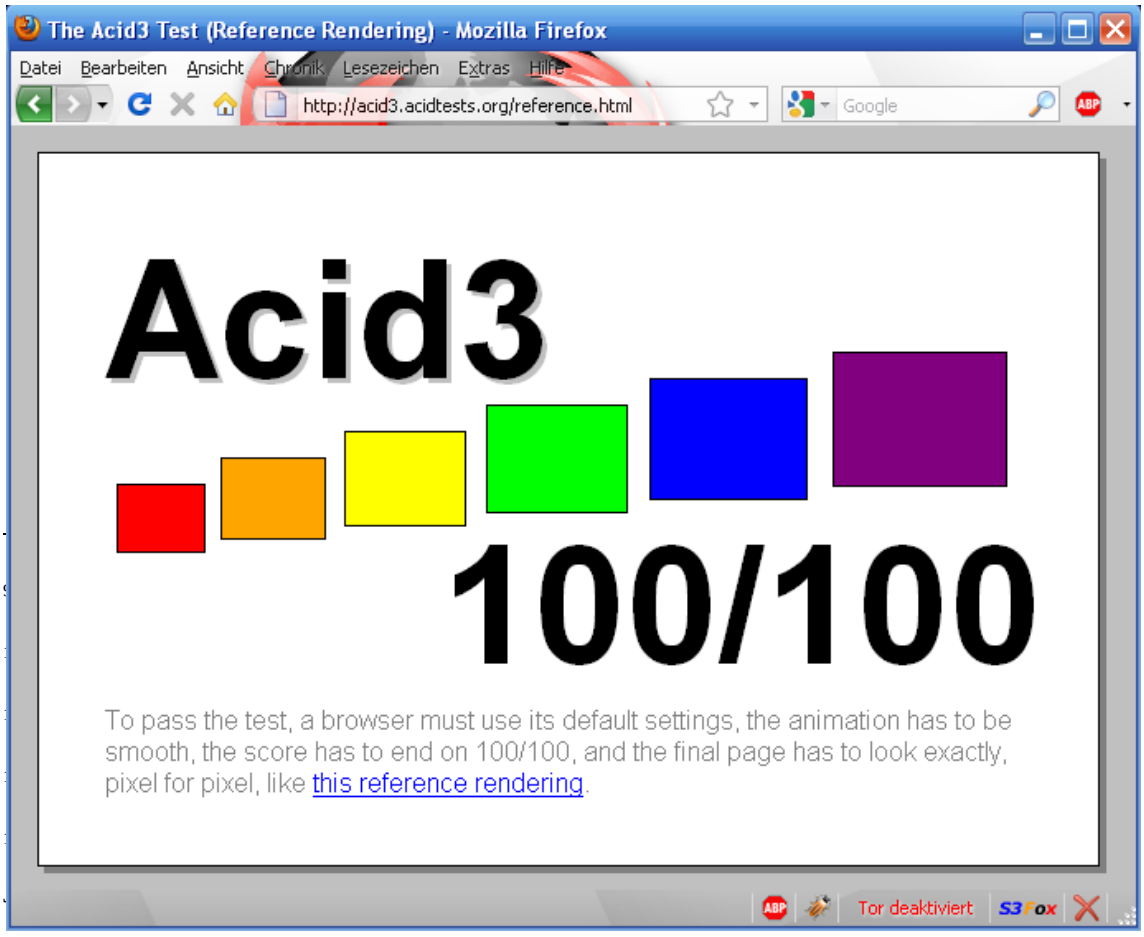

Abbildung 19: Acid 3 Referenz Test

3.3.2 Web-Elemente

Web-Elemente sind HTML-Elemente, die durch ein tag-Paar markiert werden. Ein tag-Paar besteht aus einem Starttag und einem Endtag. Ein Starttag beginnt immer mit einem Spitze-Klammer-Zeichen „<“, gefolgt vom Elementnamen und optionalen Attributlisten. Mit der schließenden Spitze-Klammer „>“ wird der Starttag geschlossen. Der Endtag besteht aus dem Zeichen „</“, dem Elementnamen und dem abschließenden „>“. Im Endtag dürfen keine Attribute stehen. Ein Starttag und Endtag bilden mit dem dazwischenliegenden Inhalt ein Webelement. Obwohl es sich bei HTML um eine beschreibende Textauszeichnung handelt, werden in den HTML Elementen keine Angaben zur Präsentation gemacht. Diese ist dem Webbrowser überlassen und hängt von der Ausgabe-Umgebung ab. Mittels CSS wird auf die Präsentation Einfluss genommen.

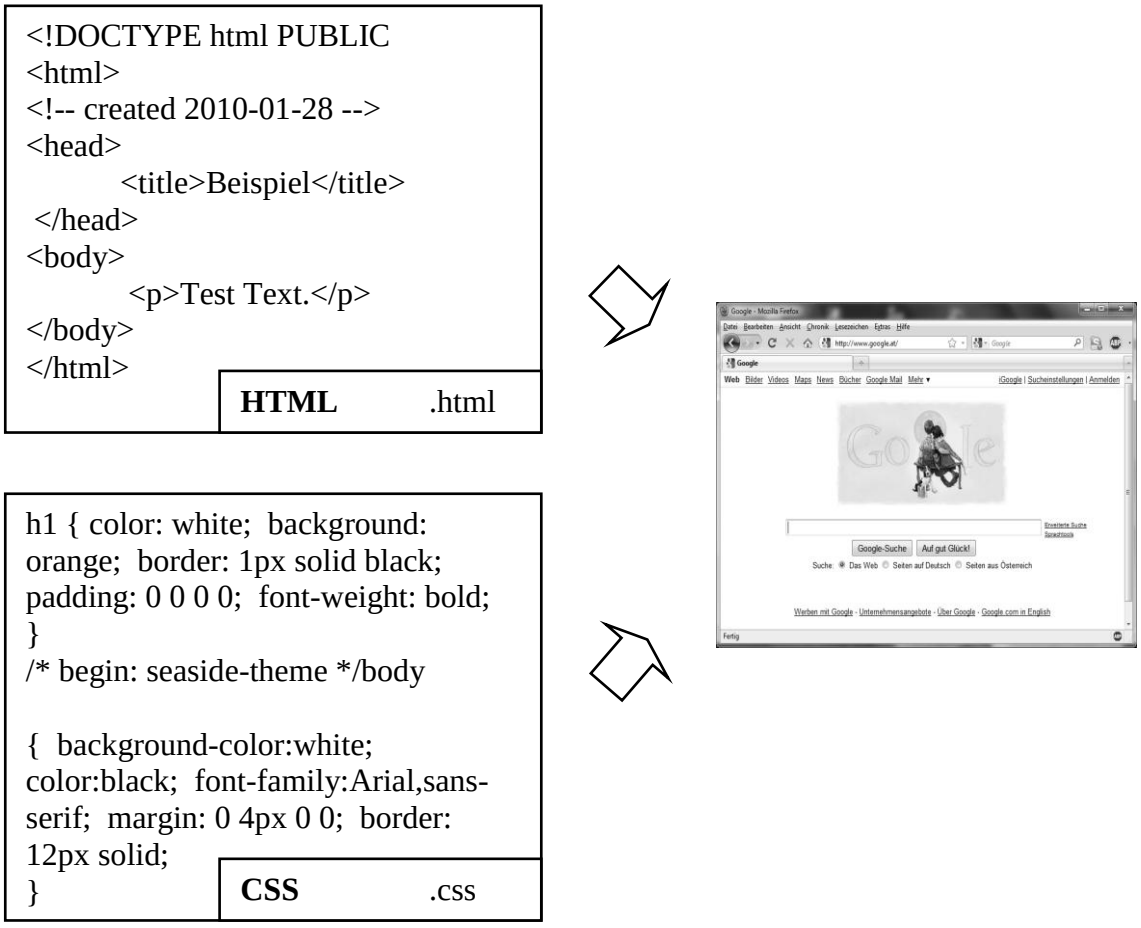

Abbildung 20: Webbrowser Interpretation von HTML und CSS Sprache

3.3.3 Einteilung der Web-Elemente

Web-Elemente können in zwei Gruppen unterteilt werden, in Formular-Elemente sowie spezielle Webelemente. Formular-Elemente sind zum Beispiel Eingabefelder, Textboxen, Buttons, Checkboxen und Radiobuttons. Diese werden mit dem <input> tag erzeugt und durch ein type-Attribut gesteuert. Andere Web-Elemente sind zum Verlinken oder Weiterleiten des Anwenders von einer Seite zur nächsten Seite, zum Einbinden eines Bildes, eines Dropdowns oder einer Listbox.

3.3.3.1 Textbox

Eine Textbox ist ein Formular-Web-Element einer grafischen Benutzeroberfläche (Webbrowser), das Benutzereingaben oder Programmausgaben aufnimmt. Textboxen werden als Eingabefelder für Suchbegriffe in Formulare verwendet. Zum Spezifizieren einer Textbox wird das type-Attribut „text“ verwendet. Die Größe der Eingabefelder kann sowohl in der Länge des Textfeldes als auch in der Länge des Eingabetextes festgelegt werden. Das Attribut „size“ legt fest, wie groß das Textfeld sein soll, und das Attribut „maxlength“ legt fest, wie viele Zeichen der darin einzugebende Text maximal haben darf.

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2  <html>
3  <head>
4  <title></title>
5  </head>
6  <form method="post" action="mailto:test@test.com">
7  <input type="text" value="Wert">Text</input>
8  <br>
9  </form>
10 </body>
11 </html>
```

Abbildung 21 : HTML Sprache einer Textbox

Eine Textbox wird im Browser als ein Rechteck dargestellt, in dem eine Eingabe getätigt werden kann (Abbildung 22).

Abbildung 22: Darstellung einer Textbox im Webbrowser

Neben dem type-Wert „text“ gibt es noch andere Arten von Daten, die in das Feld eingegeben werden können. So können zum Beispiel numerische Felder definiert werden, in die nur Zahlen eingegeben werden können, oder Felder, die nur ein Datum zulassen.

In Abbildung 23 werden die möglichen Arten der Felder dargestellt.

date	Dieser Wert erlaubt nur die Eingabe von einem Datum.
text	In diesem Fall kann man alle möglichen Zeichen eingeben.
password	Der Wert password ist für die Eingabe eines Passworts, das während der Eingabe nicht sichtbar ist. Die Eingabe der Zeichen wird durch Sternchen dargestellt.
int	Dieser Wert erlaubt nur die Eingabe von numerischen Daten.

Abbildung 23: Feldarten

3.3.3.2 Textarea

Eine spezielle Form der Textboxen stellt die Textarea[15] dar. Im Allgemeinen kann man eine Textarea als Textbox mit mehreren Zeilen definieren. In der HTML-Sprache existieren jedoch gravierende Unterschiede. Eine Textarea ist kein Formular und besitzt eigene Starttags, Endtags sowie Attribute. Das tag <textarea> ermöglicht es, mehrzeilige Felder für die Texteingabe zu definieren. Hier lassen sich dann zum Beilspiel längere Kommentare oder Mitteilungen eingeben. Die Syntax sieht hier zwingend vor, das Attribut „name“ zu verwenden, um eine eindeutige Zuweisung zu ermöglichen. Die Feldgröße für das mehrzeilige Texteingabefeld wird durch die Attribute „cols“ und „rows“ angegeben. Das Attribut „cols“ ist für die Spaltenanzahl notwendig und wird mit einem positiven numerischen Wert angegeben. Das Attribut „rows“ gibt die Zeilenanzahl an und wird ebenfalls mit einem positiven numerischen Wert angegeben.

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2  <html>
3  <head>
4  <title></title>
5  </head>
6  <form method="post" action="mailto:test@test.com">
7  <textarea name="info1" cols="10" rows="10"> Textarea</textarea>
8  <br>
9  </form>
10 </body>
11 </html>
```

Abbildung 24: HTML Sprache der Textarea

Im Webbrowser sieht der Quelltext aus Abbildung 24 wie folgt aus.

[15] Textarea: engl. für Mehrzeilige Texteingabefelder

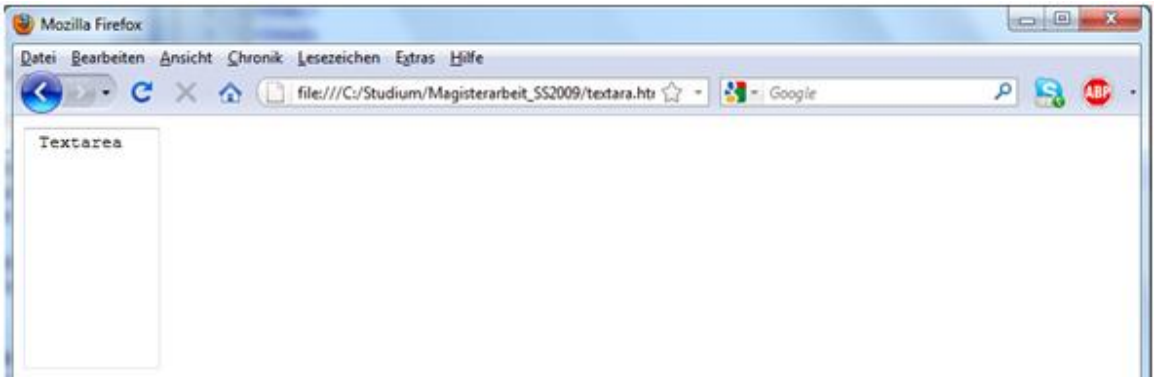

Abbildung 25: Textarea im Webbrowser

3.3.3.3 Checkbox

Ein weiteres Formular-Element in der HTML-Sprache ist die Checkbox. Eine Checkbox ist ein Standardbedienelement grafischer Benutzeroberflächen, das als kleines rechteckiges Kästchen dargestellt wird. Die Checkbox kann entweder den Zustand „markiert“ oder „nicht markiert“ annehmen. Innerhalb einer Checkbox Gruppe kann eine oder auch mehrere Checkboxen gleichzeitig markiert werden.

Als Syntax zur Realisierung der Checkboxen wird das tag <input> verwendet. Durch Hinzufügen des type-Attributes mit dem Wert „checkbox“ wird festgelegt, um welchen Typ es sich handelt. Das value-Attribut mit dem Wert „Text“ ist für die Bezeichnung zuständig und wird erst beim Empfänger sichtbar.

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2  <html>
3  <head>
4  <title></title>
5  </head>
6  <form method="post" action="mailto:test@test.com">
7  <input name="info1" type="checkbox" value="HTML"> HTML</input>
8  <br>
9  <input name="info2" type="checkbox" value="XML"> XML</input>
10 <br>
11 </form>
12 </body>
13 </html>
```

Abbildung 26: Checkbox in HTML Sprache

Dieser Quelltext bewirkt im Browser folgende Darstellung (Abbildung 27).

Abbildung 27: Checkbox im Browser

3.3.3.4 Radiobutton

Der Radiobutton ist ein kleiner Kreis, der durch Mausklick markiert wird und zu den Formular-Elementen gehört. Er wird durch das type-Attribut „radio“ definiert. Der Radiobutton kann entweder den Zustand „markiert“ oder „nicht markiert“ annehmen. Im Gegensatz zur Checkbox kann von einer Radiobutton Gruppe nur einer markiert werden.

Die Syntax vom Radiobutton ist ähnlich der Checkbox und unterscheidet sich grundsätzlich nur durch den Wert „radio“ im type-Attribut.

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2  <html>
3  <head>
4  <title></title>
5  </head>
6  <form method="post" action="mailto:test@test.com">
7  <input name="info3" type="radio" value="rot"> rot
8  <br>
9  <input name="info3" type="radio" value="grün"> grün
10 <br>
11 <input name="info3" type="radio" value="blau"> blau
12 <br>
13 </form>
14 </body>
15 </html>
```

Abbildung 28: Radiobutton HTML Sprache

Im Browser wird der Quellcode von Abbildung 28 wie in Abbildung 29 dargestellt.

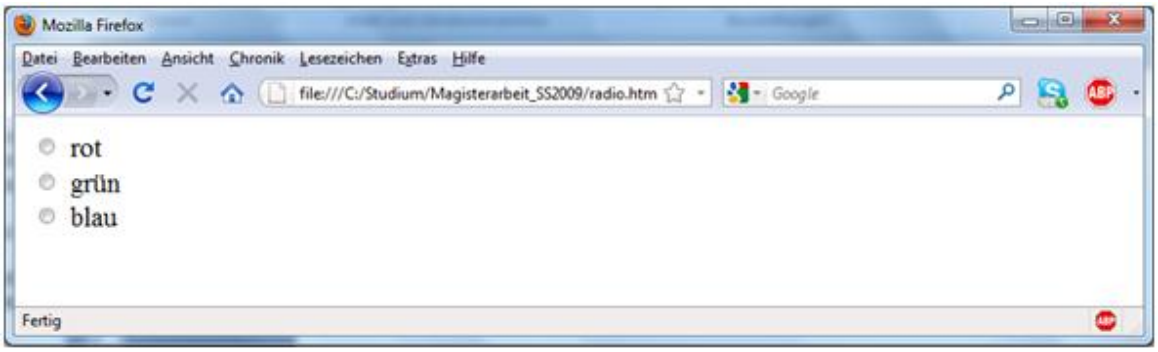

Abbildung 29: Radiobutton im Webbrowser

3.3.3.5 Dropdown Single Select Combobox

Eine weitere Feldart ist die Dropdown Combobox. Die Dropdown Combobox ermöglicht durch Anklicken des Feldes eine Auswahl verschiedener Optionen. Dabei öffnet sich mit einem Mausklick die Auswahlliste mit den möglichen Optionen, von der man eine auswählen kann. Die Syntax für ein Feld einer Auswahlliste wird durch das tag <select> erstellt. Die Definition mit dem tag <option> wird zwischen den Starttag <select> und dem Endtag </select> geschrieben. Dabei können so viele Listeneinträge wie notwendig in die Definition eingetragen werden. Das tag <option> hat ein Attribut, das häufig eingesetzt wird. Das selected-Attribut ermöglicht, einen Listeneintrag fest zu selektieren.

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2  <html>
3  <head>
4  <title></title>
5  </head>
6  <form method="post" action="mailto:test@test.com">
7  <select name="Optionen" size=1>
8  <option value="A1">Option 1
9  <option value="A2" selected>Option 2
10 <option value="A3">Option 3
11 </select>
12 </form>
13 </body>
14 </html>
```

Abbildung 30: HTML Code einer Dropdown Single Select Combobox

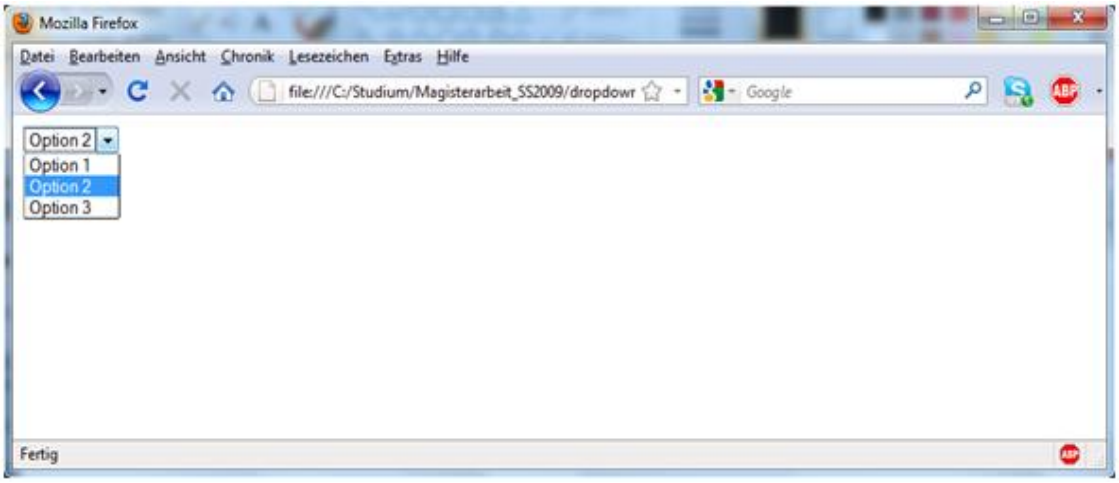

Abbildung 31: Dropdown Single Select Combobox im Webbrowser

3.3.3.6 Listbox Single Select

Das Listbox Multiple Select ist eine Art Dropdown Single Select Combobox mit einer größeren Auswahlliste. Das size-Attribut wird mit einem Wert, das der Anzahl der option-tags entspricht, angegeben und im select-tag eingetragen.

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2  <html>
3  <head>
4  <title></title>
5  </head>
6  <form method="post" action="mailto:test@test.com">
7  <select name="Optionen" size=5 >
8  <option value="A1">Option 1
9  <option value="A2" selected>Option 2
10 <option value="A3">Option 3
11 </select>
12 </form>
13 </body>
14 </html>
```

Abbildung 32: Listbox Single Select als HTML Code

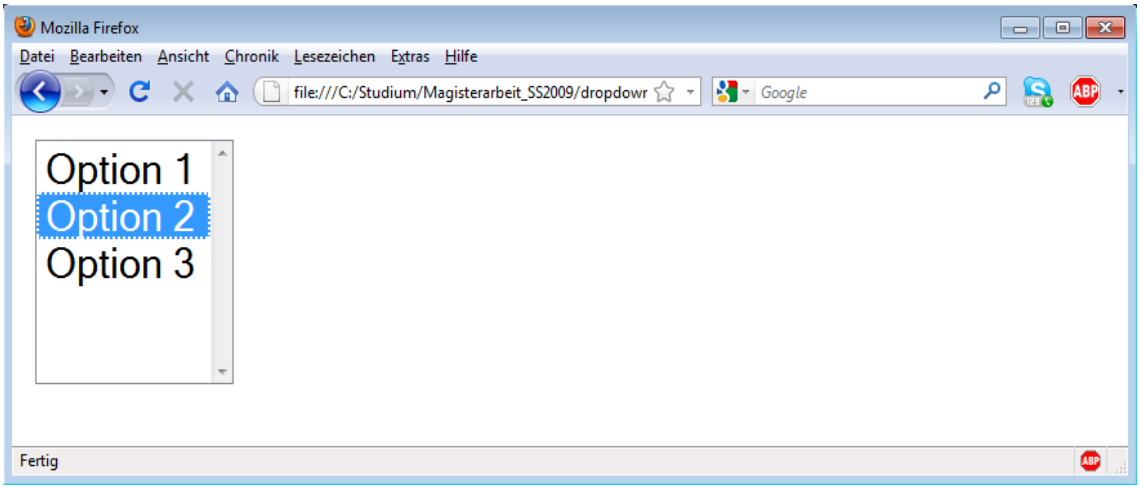

Abbildung 33: Listbox Single Select im Webbrowser

3.3.3.7 Listbox Multiple Select

Das Listbox Multiple Select unterscheidet sich vom Listbox Single Select durch das Attribut „multiple“. Dieses Attribut wird in den select-tag eingetragen und der Wert gibt die Grenze der gleichzeitig ausgewählten Optionen an.

```
1   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2   <html>
3   <head>
4   <title></title>
5   </head>
6   <form method="post" action="mailto:test@test.com">
7   <select name="Optionen" size=5 multiple="2">
8   <option value="A1">Option 1
9   <option value="A2" selected>Option 2
10  <option value="A3">Option 3
11  </select>
12  </form>
13  </body>
14  </html>
```

Abbildung 34: Listbox Multi Select als HTML-Code

Im Webbrowser sieht die Listbox von Abbildung 34 aus wie in Abbildung 35.

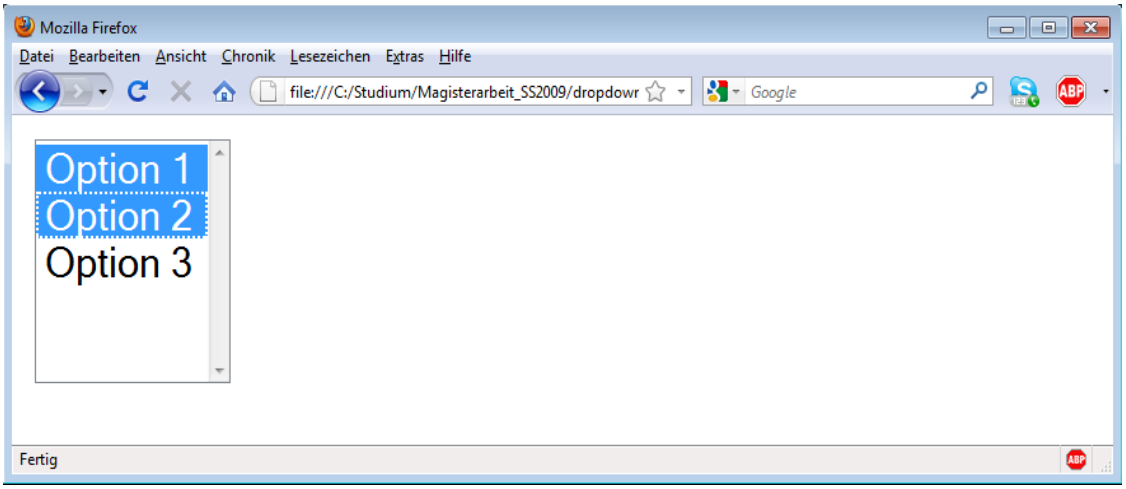

Abbildung 35: Listbox Multiple Select im Webbrowser

3.3.3.8 Button

Ein Button ist für das Versenden eines Formulars zuständig. Für einen Button wird der input-tag mit einem type-Attribut verwendet, das zwei Werte „submit“ und „reset“ annehmen kann. Neben dem type-Attribut muss auch die Beschriftung des Buttons mit dem value-Attribut erfolgen.

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2  <html>
3  <head>
4  <title></title>
5  </head>
6  <form method="post" action="mailto:test@test.com">
7  <input type="button" value="Wert">Text</input>
8  <br>
9  </form>
10 </body>
11 </html>
```

Abbildung 36: Button als HTML Code

Ein Button wird im Webbrowser als Rechteck dargestellt (Abbildung 37) und die Farbe des Buttons verändert sich, wenn sich der Cursor auf dem Button befindet. Bei jedem Mausklick wird eine Aktion gestartet.

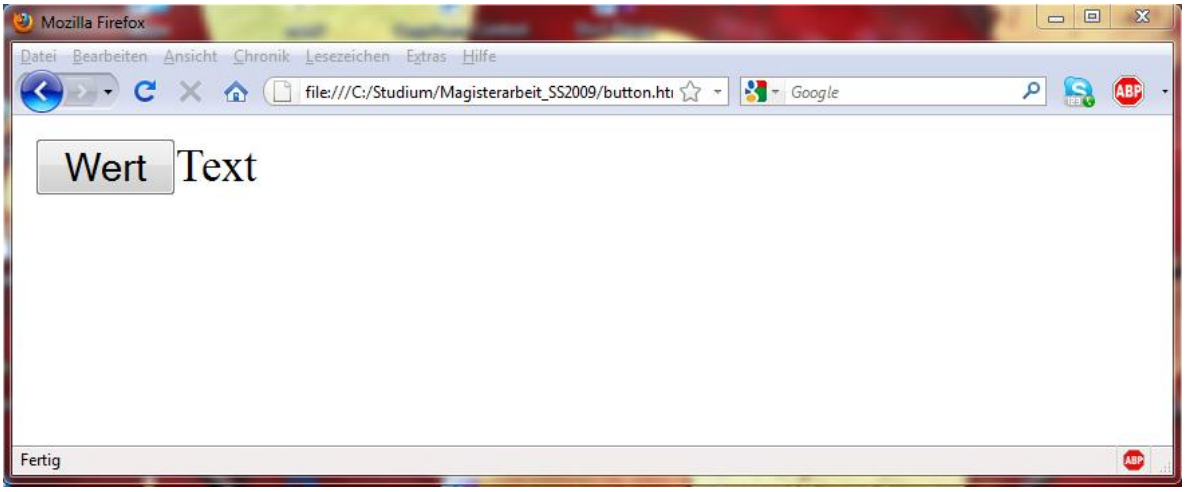

Abbildung 37: Button im Webbrowser

3.3.3.9 Image

Ein Image ist ein grafisches Element auf einer Webseite, das mittels des img-tag eingefügt wird. Seit HTML 4 gibt es auch die Möglichkeit, Grafiken und Bilder über das object-tag einzubinden. In Abbildung 38 sind drei Beispiele angeführt wie man ein Bild in eine Webseite einfügen kann. Im ersten Beilspiel muss das Bild in derselben Ebene auf dem Server liegen wie die HTML-Datei. Im zweiten Beispiel liegt das Bild in einem Unterorder „src“ und im dritten Beispiel ist das Test-Bild auf einem entfernten Server.

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2  <html>
3  <head>
4  <title></title>
5  </head>
6  <img src="Testimage.png"/>
7  <br>
8  <br>
9  <img src="src/Testimage.png"/>
10 <br>
11 <br>
12 <img src="http://upload.wikimedia.org/wikipedia/commons/d/d9/Test.png"/>
13 <!-- [Sca09] -->
14 </body>
15 </html>
```

Abbildung 38: Image im HTML Code [Sca09]

Die Darstellung der Bilder aus dem HTML Code sieht wie in Abbildung 39 aus.

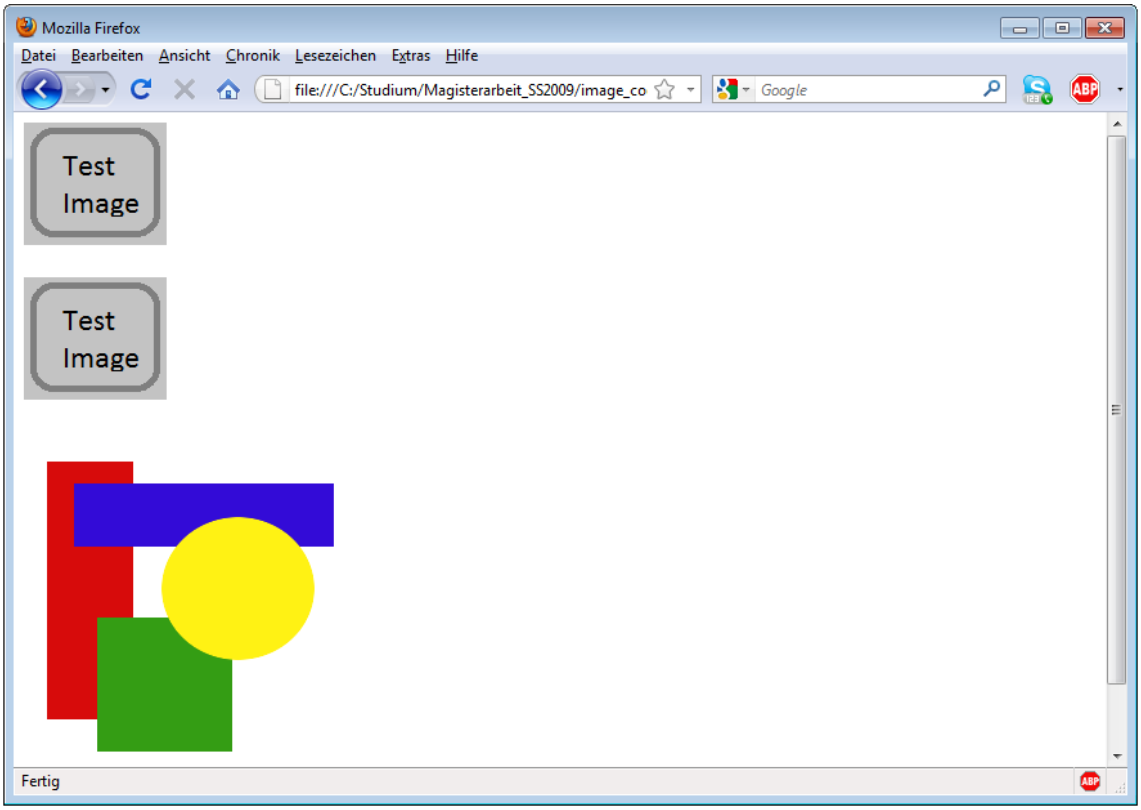

Abbildung 39: Image Darstellung im Webbrowser [Sca09]

3.3.3.10 Hyperlinks

Verbindungen zu anderen Seiten werden mittels Hyperlinks Elementen realisiert. Diese Links oder Verknüpfungen sind das zentrale Element bei der Erstellung von Webseiten und Hauptbestandteil jeder Webseite. Das sogenannte Surfen im WWW ist das Springen von einer Seite zu einer anderen Seite mittels Links. Ein Link kann auf der gleichen Seite zu anderen Textpassagen oder auf ein anderes Dokumenten auf einer fremden Seite verweisen. Ein Link wird durch einen einfachen Mausklick ausgelöst. Die Syntax eines Links wird mit dem tag <a> umgesetzt. Dieser tag setzt einen „anchor“ auf der Webseite und macht aus dem normalen Text ein Hypertextdokument. Die notwendigen Attribute für einen funktionierenden Hyperlink sind „href“ und „name“, wenn es sich um einen internen Verweis handelt bzw. „href“, wenn auf eine externe Seite verlinkt wird. Bei internen Links (Abbildung 40) benötigt man zwei Links, das eine ist der Hauptlink und das andere das Ziel. Vom Hauptlink wird man dann mit dem href-Attribut zum entsprechenden Ziel-name-Attribut verwiesen. Der href-Wert ist der interne Name mit einem vorgestellten Rautenzeichen „#“. Externe Links werden mit der richtigen URL angegeben.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title></title>
</head>
<a href="http://www.tuwien.ac.at">TU-Wien</a>
<br>
<a href="#intern">Interne Link</a>
<br>
<a href="http://www.tuwien.ac.at"><img src="Testimage.png"/></a>
<br>
<br>
<br>
<br>
<br>
<a name="intern">Ziel</a>
</body>
</html>
```

Abbildung 40: HTML Code von einem externen und einem internen Hyperlink.

Ohne definierte Formatierung sieht die Darstellung von Browser zu Browser unterschiedlich aus. Im folgenden Beispiel (Abbildung 41) sieht die Darstellung eines Hyperlinks blau und unterstrichen aus. Eine Grafik als Link ist eine spezielle Darstellung im Webbrowser, wo statt dem Linktext eine Grafik mit dem img-tag eingefügt wird. Auch dieser Link wird wie alle anderen Links grafisch blau umrandet dargestellt.

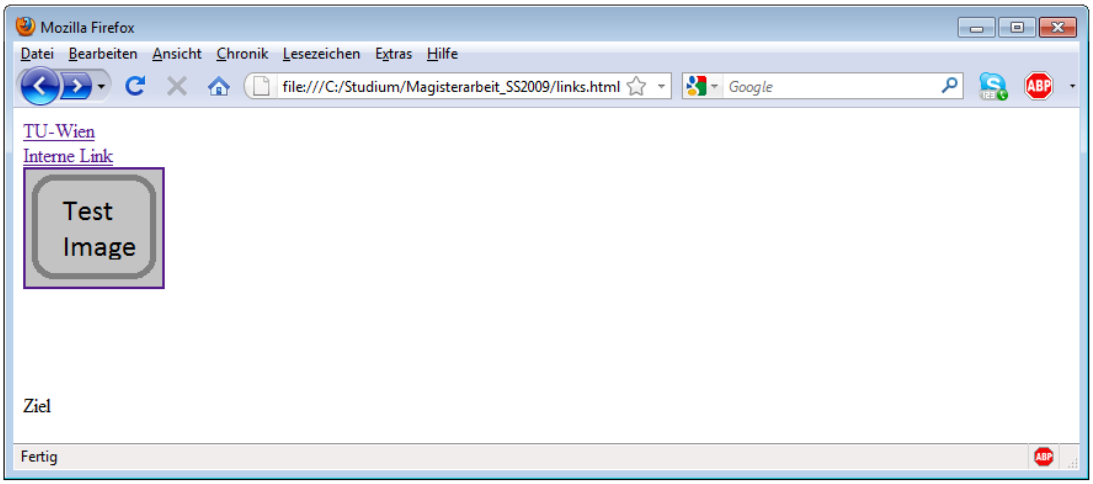

Abbildung 41: Standard Link Darstellung im Webbrowser

3.3.3.11 Table

Ein Table ist für die Gestaltung einer Webseite sehr wichtig, da dadurch die genaue Positionierung von bestimmten Webelementen wie Text und Bilder auf der Webseite in Spalten ermöglicht wird. In HTML wird eine Tabelle mit dem table-Starttag begonnen und mit dem schließenden table-Endtag beendet. Die einzelnen Zeilen-tags sind <tr></tr> und dazwischen befinden sich die einzelnen Spalten-tags <td></td>. Um einen Tabellenkopf oder sogenannte Kopfzellen darzustellen, wird in der ersten Zeile statt des <td> ein <th> tag eingetragen. Dieses tag stellt den Text der ersten Zeile fett und zentriert dar.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title></title>
</head>
<body>
<table>
<tr><th>Kopf1</th><th>Kopf2</th><th>Kopf3</th></tr>
<tr><td>Zeile1_Spalte1</td><td>Zeile1_Spalte2</td><td>Zeile1_Spalte3</td></tr>
<tr><td>Zeile2_Spalte1</td><td>Zeile2_Spalte2</td><td>Zeile2_Spalte3</td></tr>
<tr><td>Zeile3_Spalte1</td><td>Zeile3_Spalte2</td><td>Zeile3_Spalte3</td></tr>
</table>
<br>
<table border="1">
<tr><th>Kopf1</th><th>Kopf2</th><th>Kopf3</th></tr>
<tr><td>Zeile1_Spalte1</td><td>Zeile1_Spalte2</td><td>Zeile1_Spalte3</td></tr>
<tr><td>Zeile2_Spalte1</td><td>Zeile2_Spalte2</td><td>Zeile2_Spalte3</td></tr>
<tr><td>Zeile3_Spalte1</td><td>Zeile3_Spalte2</td><td>Zeile3_Spalte3</td></tr>
</table>
</body>
</html>
```

Abbildung 42: HTML Code von Tables

Wie aus dem HTML Code von Abbildung 42 entnommen werden kann, ist es möglich, die Darstellung der Tabelle im <table> tag durch weitere Attribute zu verändern und diese an die gewünschte Webseite anzupassen. In diesem Beispiel (Abbildung 42) habe ich ein border-Attribut eingefügt, um einen Rahmen darzustellen (Abbildung 43).

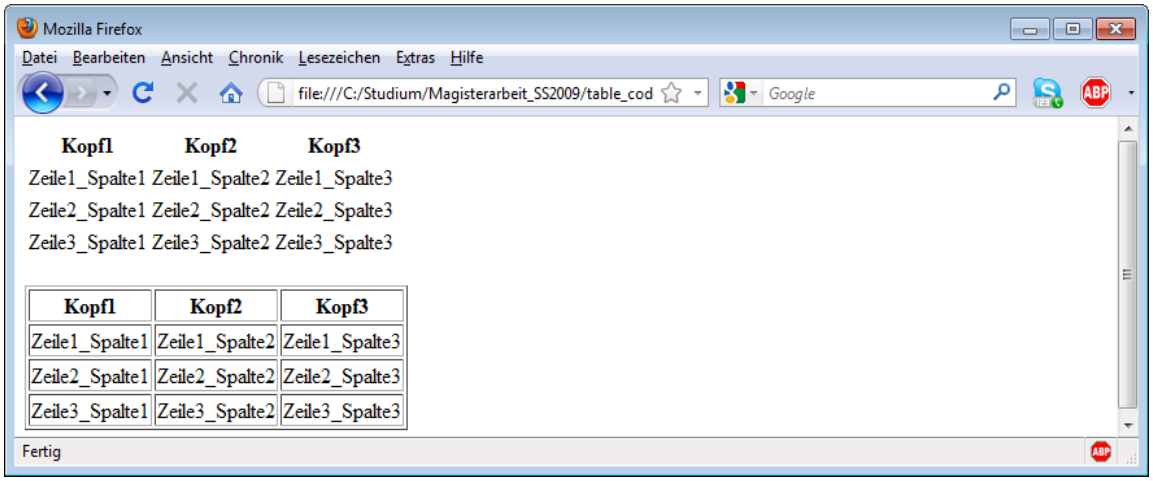

Abbildung 43: Darstellung von Tables im Webbrowser

3.3.4 Identifizierung

Für die Identifizierung der einzelnen Webelemente spielen tags sowie die Attribute der einzelnen Elemente eine große Rolle. tags werden in erster Linie dazu verwendet, um ein Element zu finden. Da diese mehrfach vorkommen können, sind die einzelnen Attribute und deren Werte der tags für das Auffinden der Webelemente sehr wichtig. Hier kommt der sogenannte DOM-Baum zum Tragen.

3.3.4.1 DOM-Baum

Der DOM-Baum[16] auch DOM-Knotenbaum genannt, ist eine Spezifikation einer Schnittstelle (API) für den Zugriff auf ein HTML und XML Dokument [Mgj08] . Diese entstand Mitte der 90er Jahre auf Drängen einiger Entwickler und wurde 1998 von der W3C standardisiert. Es existieren mehrere Versionen (Levels) mit jeweils verschiedenen Modulen. Die Verarbeitung erfolgt mittels Einlesen des HTML- oder XML-Dokuments und dem Erzeugen des DOM-Knotenbaumes. Mit den verschiedenen Methoden der API ist dann ein Zugriff auf einzelne Inhalte, Knoten und der Struktur möglich. Der DOM-Baum wird sehr häufig mit JavaScripts benutzt. Es wird ein Code in JavaScript geschrieben, der den DOM nutzt, um die Webseite und seine Elemente zu manipulieren.

Im folgenden Beispiel wird dem Web-Element mit der *Id="dummyId"* ein neues href-Attribut erstellt und dieses dann mit dem Wert *„test"* belegt. (Tabelle 19)

```
obj=document.getElementById('dummyId');
obj.setAttribute('href','#');
obj.href='test';
```

Tabelle 19: JavaScript zum manipulieren einer Webseite

Mit der JavaScript-Konsole in Mozilla Firefox ist es möglich, JavaScripts ohne permanente Änderung der HTML Struktur zu simulieren. Das Plug-In „Firebug"[17] ermöglicht die gleichzeitige Darstellung des DOM-Baums und der JavaScript-Konsole unterhalb des geränderten HTML-Dokumentes. In Abbildung 44 sieht man die Original HTML-Seite.

[16] DOM: Document Object Model, engl. für Dokument Objekt Modell

[17] Firebug: http://getfirebug.com, ist ein Plug-In für den Mozilla Firefox Webbrowser

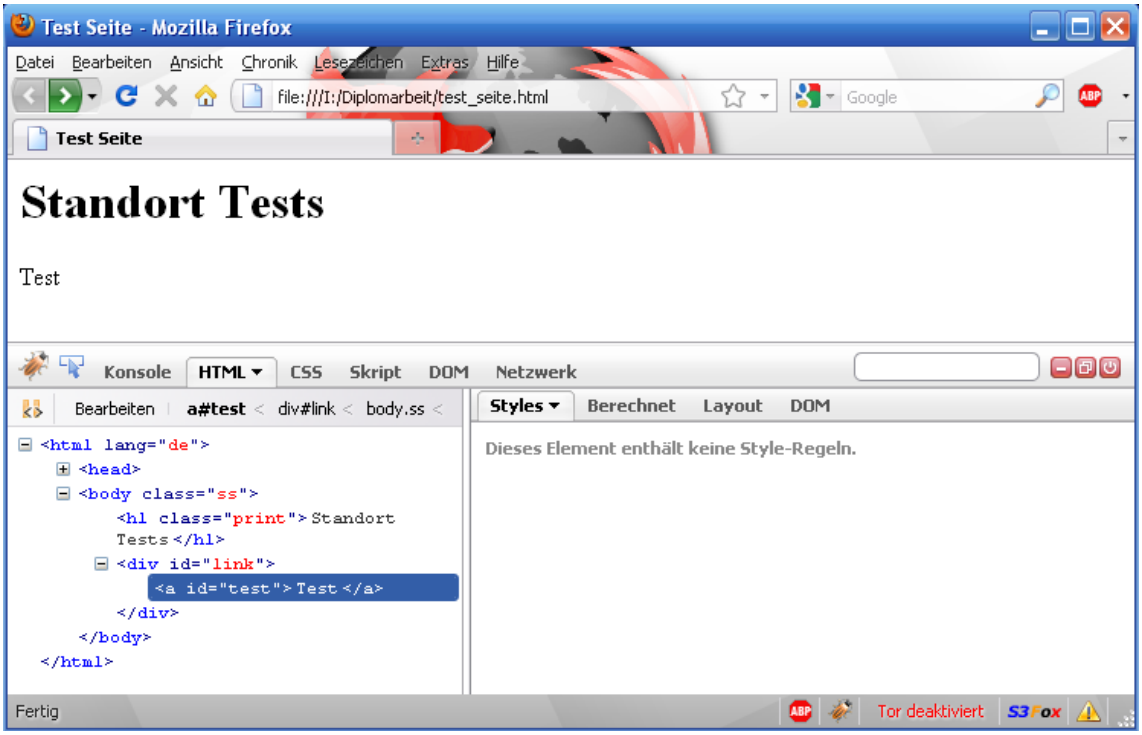

Abbildung 44: DOM Baum vor Ausführen des JavaScripts

Nach Initialisieren des JavaScript Codes (Tabelle 20) wird der DOM Baum (Abbildung 45) verändert.

```
obj=document.getElementById('test');

inn=document.getElementById('test').innerHTML = 'Neuer Name';

obj.setAttribute('href','url');
```

Tabelle 20: JavaScript Code

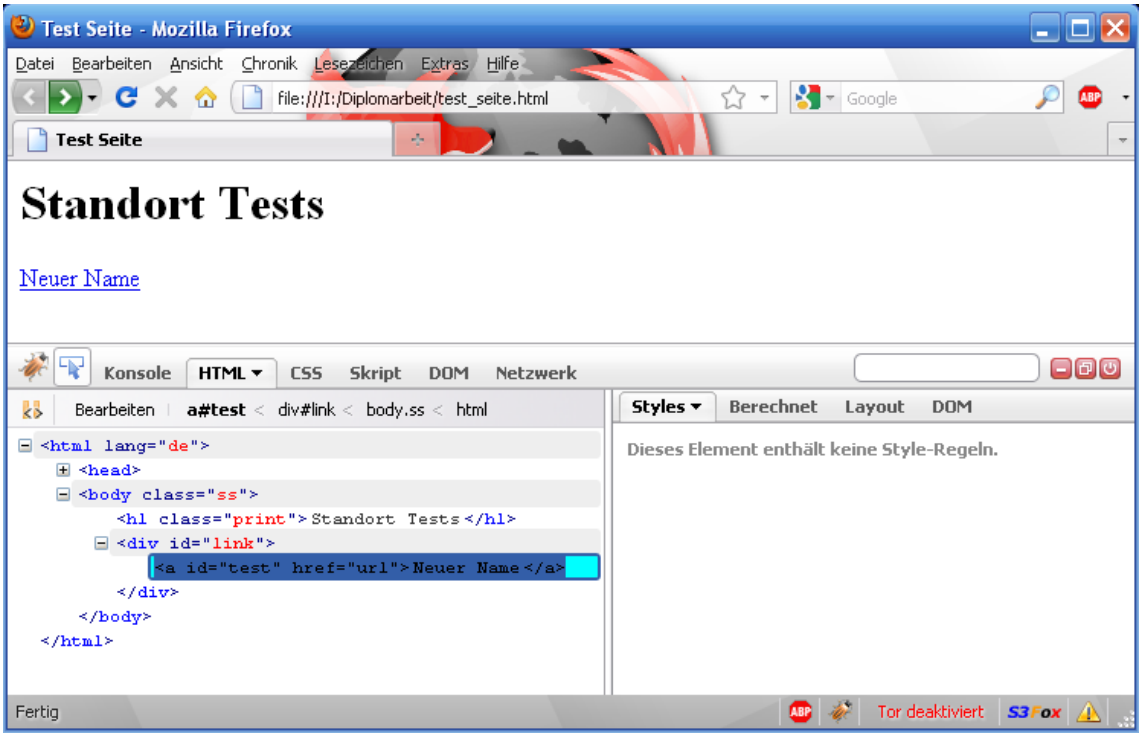

Abbildung 45: DOM Baum nach Ausführen des JavaScripts

Die wichtigsten Knotentypen im DOM sind:

- Ein Dokumentknoten stellt die gesamte Baumstruktur dar.
- Ein Dokumentfragmentknoten stellt einen Teil der Baumstruktur dar.
- Ein Elementknoten entspricht exakt einem Element in HTML oder XML.
- Ein Attributknoten entspricht exakt einem Attribut in HTML oder XML.
- Ein Textknoten stellt den textuellen Inhalt eines Elements oder Attributs dar.

Tabelle 21: DOM Knotentypen [Wik05]

In Abbildung 46 ist der HTML-Code (von Abbildung 42 auf Seite 59) als DOM-Knotenbaum dargestellt. Das Dokument kann man sich dabei als Stammbaum vorstellen und die einzelnen Elemente als Äste (Knoten). Einige Knoten sind in der Abbildung 46 zur Erklärung teilweise expandiert, um die darunterliegenden Kindknoten zu zeigen. Wie man in dieser Abbildung sehen kann, sind die Tabellen eine eigene Verzweigung und alle darin vorkommenden tags (wie zum Beispiel <th>, <tr> und <td>) einzelne Knoten.

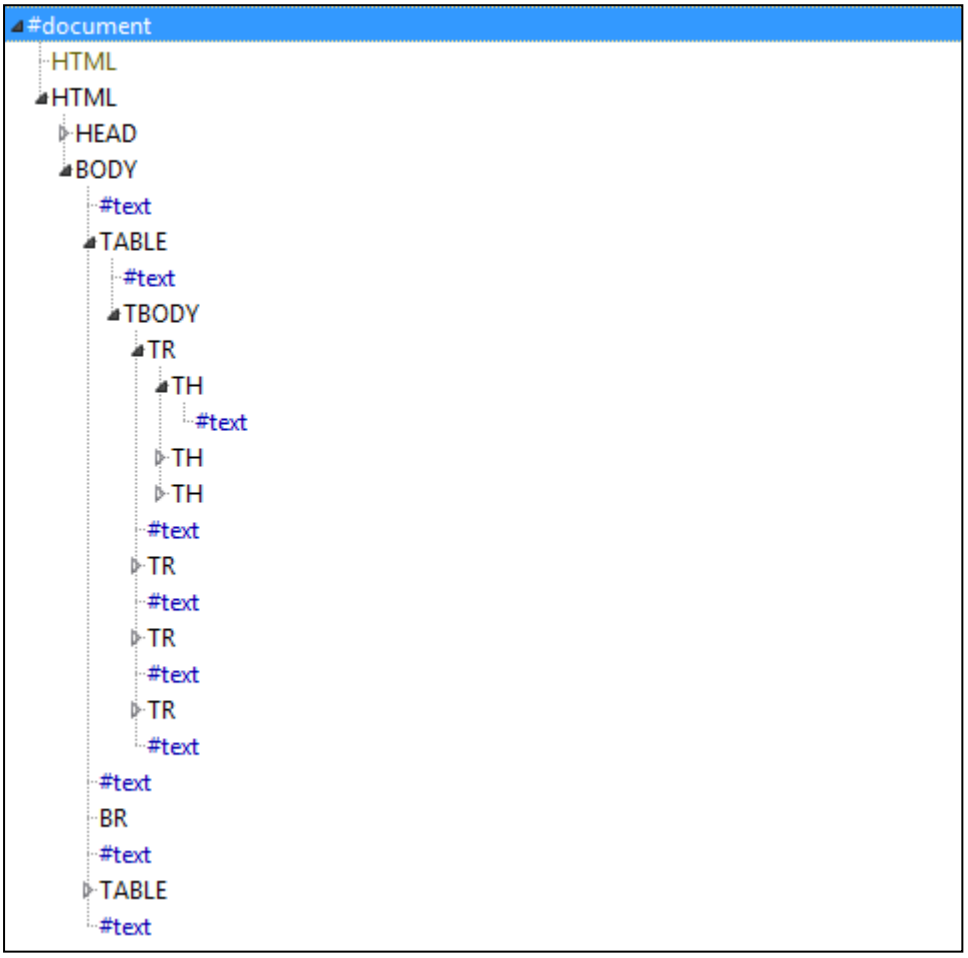

Abbildung 46: DOM-Baum einer HTML Webseite

3.3.5 Web-Testwerkzeuge

3.3.5.1 Wozu Testwerkzeuge?

Wie ich im Abschnitt „2.5.1 Was sind automatisierte Tests“ auf Seite 33 kurz erklärt habe, verwendet man Testwerkzeuge mit dem Ziel, das automatisierte Testen zu unterstützen. Testwerkzeuge können bei der Planung, Ausführung und Auswertung von Tests eine große Hilfe sein. Es gibt eine große Auswahl von Testwerkzeugen, die je nach Verwendungszweck eingesetzt werden können. Der folgende Abschnitt soll einen kurzen Einblick in die Welt der Testwerkzeuge vermitteln.

Da nach Martin Pol [Mar00] Testwerkzeuge automatisierte Hilfsmittel für Testaktivitäten sind und eine sehr umfangreiche Auswahl an Testwerkzeugen [Ric10] zur Verfügung steht, werde ich die Arten der Werkzeuge nach ihren Tätigkeiten eingrenzen.

3.3.5.2 Arten von Web Testwerkzeugen

Für die Unterstützung der Arbeit beim Testen gibt es unterschiedliche Werkzeuge, die jedoch nicht alle für jede Tätigkeit eingesetzt werden können. Nach Edward Kit [Kit95] kann man die Testwerkzeuge in folgende Tätigkeiten aufteilen. Jede diese Tätigkeit benötigt ihre eigenen Werkzeuge.

- Reviews und Inspektionen
- Planung von Tests
- Testdesign und -entwicklung
- Testausführung und -auswertung
- Supportwerkzeuge beim Testen von Software

Tabelle 22: Gliederung der Testwerkzeuge in Tätigkeiten

Eine Liste von nützlichen Web-Testwerkzeugen findet man auf [Ric10] . In meiner Arbeit gehe ich auf die Testausführung und -auswertung ein.

3.3.5.3 Testausführung und -auswertung

Die Ausführung und die Auswertung der Tests beinhaltet das Auswählen von Testfällen, das Aufsetzen der Umgebung auf einem Testsystem, das Ausführen der Tests, das Aufzeichnen der Ausgaben und Ergebnisse, das Analysieren möglicher Fehler und das Messen des Fortschritts der gesamten Tests. Die Tester zeichnen mit sogenannten „Record/Replay“ Testwerkzeugen (Abbildung 47) Testfälle auf, um diese in einem Testablauf automatisiert ablaufen zu lassen. Die „Record/Replay“ Testwerkzeuge zeichnen im Record-Modus alle Benutzeroperationen wie zum Beispiel alle Tastatur-(Key-Action) und Maus-(Maus-Action) Eingaben des Testers und die Ausgaben der Software auf. Diese aufgenommen Tests können zu jeder Zeit im Testwerkzeug abgespielt und die Ausgaben validiert werden, indem diese Ausgaben mit den zuvor vom Tester aufgezeichneten validieren Ergebnisse verglichen werden.

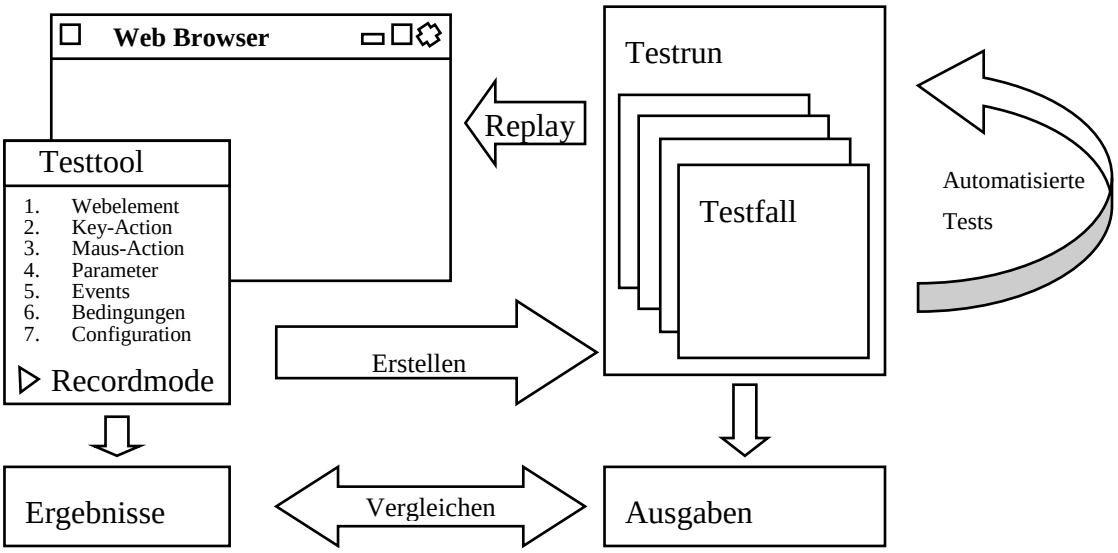

Abbildung 47: Automatisierte Webtests

Testwerkzeuge (Record/Replay) werden in zwei Gruppen aufgeteilt, die nativen (intrusive) Werkzeuge und die nicht eindringenden (non-intrusive) Werkzeuge. Ein natives Werkzeug befindet sich wie das zu testende Programm auf dem gleichen Server oder Computer und beansprucht auch einen Teil der Performance des Systems. Beim nicht eindringenden Werkzeug wird die Performance des Systems nicht beeinträchtigt, da sich das Werkzeug auf einem anderen Rechner befindet.

4. Testwerkzeuge

Im vorherigen Kapitel habe ich die Testwerkzeuge und die Notwendigkeit beim automatisierten Testen erklärt. Nun werde ich einige speziell ausgesuchte Testwerkzeuge im Detail beschreiben und erklären, wie diese eingesetzt werden können. Mit einigen kleinen Beispielen werde ich die von mir erklärten Testwerkzeuge erläutern.

4.1 Unit Tests

Die bekannteste Art von automatisierten Code Tests sind die sogenannten Unit-Tests. Ein Unit-Test ist ein Stück Programmcode, das der Entwickler nur zum Testen eines bestimmten Teiles eines anderen Programmcodes schreibt. Unit-Tests gehören zu den White-Box-Tests, da der Test die Implementierungsdetails seines Prüfers kennt und von diesem Wissen Gebrauch macht. Im Unit-Test werden kleinere Programmteile abgekapselt von anderen Programmteilen getestet. Diese Programmteile einer Software werden Module genannt. Die Granularität der unabhängig getesteten Module reicht dabei von einzelnen Methoden über ganze Klassen bis hin zu Komponenten. Diese Module werden mit „Unit-Tests“[18] auf ihre Funktionalität hin überprüft, um mögliche Fehler aufzudecken. Einzelne Units-Tests werden nach Beendigung der Implementierung während der Integration zu einem Integrationstest zusammengestellt.

„Unit-Tests sind ein vergleichsweise kostengünstiger Weg zu besserem Quelltext in kürzerer Zeit.“ [Bra04]

Obwohl Unit-Tests einfach zu erstellen und kostengünstig sind, werden diese Tests oft nicht beachtet, da Software-Entwickler der Meinung sind, dass Tests zu viel Zeit kostet.

„Tests zu schreiben, kostet zu viel Zeit.“ [Bra04]

Unit-Tests sind auf alle bekannten Programmiersprachen anwendbar. Diese Unit-Tests sind sogenannte Modultestsoftware.

4.2 Modultestsoftware

Modultestsoftware sind Unit-Tests-Software und dienen zum Nachweisen von Fehlern in den Modulen von Softwareprogrammen. Module sind (wie in Unit Tests erklärt) hauptsächlich Klassen oder Methoden. Testframes gibt es für fast alle Programmiersprachen. Der bekannteste Testframe für automatisierte Softwaretests ist xUnit. Dieses Framework erlaubt das Überprüfen verschiedener Elemente (Units) von Software, wie etwa Funktionen und Klassen. Das erste xUnit-Framework wurde von Kent Beck unter dem Namen SUnit für die Programmiersprache Smalltalk entworfen.

[18] Unit Tests: engl. für Modul Test

[Wik07] Aufbauend zu diesem Framework wurden dann für fast alle bekannten Programmiersprachen Unit-Test-Frameworks erstellt.

Der bekannteste Vertreter ist JUnit für die Sprache Java. Weitere sind PHPUnit für das PHP-Framework, JSUnit für JavaScripts, JUnitPerf, XMLUnit und andere mehr.

4.2.1 JUnit

JUnit[19] ist ein kleines mächtiges Open-Source-Software Java-Framework-Werkzeug zum Schreiben und Ausführen automatisierter Unit-Tests, mit dem Java-Klassen automatisiert getestet werden können. Es steht unter der *„Common Public License“*[20]. JUnit ist für das Testen von Java Programmen in einem Java Umfeld entworfen worden. JUnit wird für Unit-, Komponenten- und White-Box-Tests eingesetzt, bei denen die interne Struktur der zu testenden Komponente bekannt ist. Bei JUnit werden dazu Testklassen mit mehreren Testmethoden für die zu testenden Klassen erzeugt. Mehrere Testklassen können dann zu Test-Suiten zusammengefasst werden.

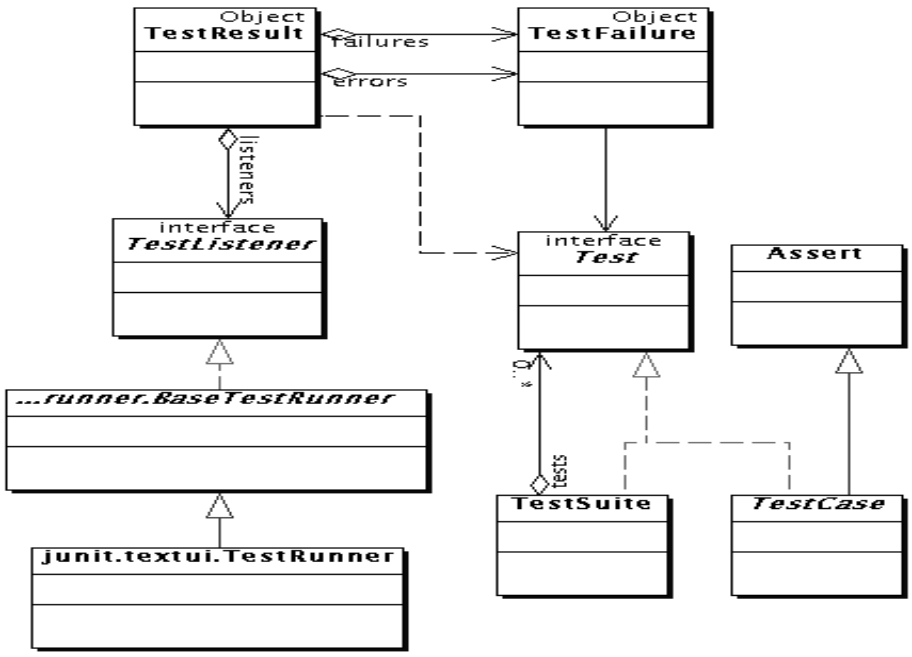

Abbildung 48: JUnit Framework [Qin09]

[19] JUnit: Java Unit

[20] Common Public License : CPL http://junit.sourceforge.net/cpl-v10.html

4.2.2 PHPUnit

PHPUnit ist ein Plattform-unabhängiges in PHP[21] geschriebenes freies Framework zum Testen von PHP-Skripten. Es ist besonders für automatisierte Tests einzelner Units geeignet. Es basiert auf dem xUnit-Konzept, welches auch für andere Programmiersprachen genutzt wird, wie zum Beispiel in JUnit für Java.

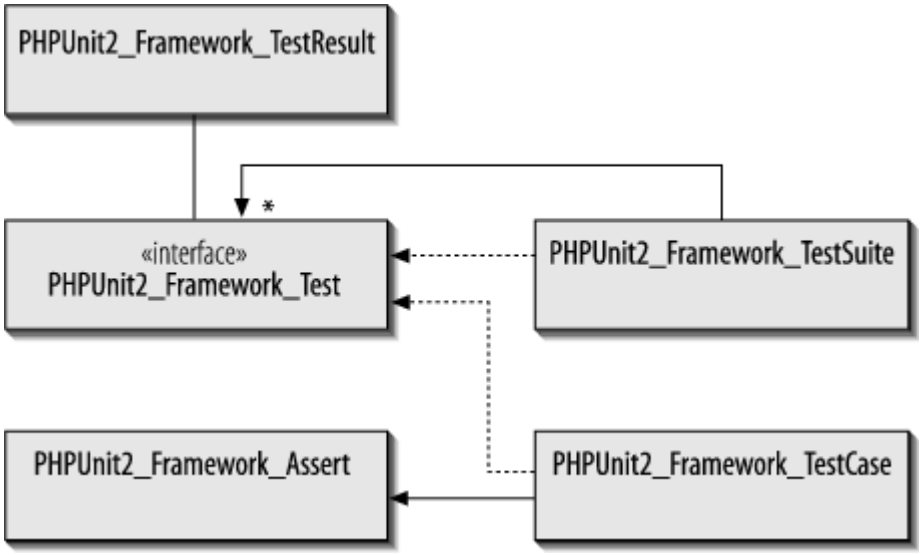

Abbildung 49: PHPUnit Klassen und Schnittstellen [Ber10]

4.2.3 Simpletest

Simpletest ist ein Open-Source Unit-Test-Framework für die Skriptsprache PHP, die von Markus Baker entwickelt wurde und JUnit sowie PHPUnit sehr ähnlich ist. Simpletest basiert (wie JUnit und PHPUnit) auf xUnit und ermöglicht es, auch Testfälle von anderen Unit-Tests zu übernehmen. Für die PHP-Testfälle benötigt man einen Webserver[22] (XAMPP[23] - Apache-Server[24]) mit einem PHP-Plug-in. Vor dem Test müssen die einzelnen Test-Module in das PHP-Skript geladen werden. In einer neuen Testklasse *„TestOfRankings“* wird die *„WebTestCase“* Klasse im Modul „web_tester.php“ erweitert. In einer neuen Funktion werden die einzelnen Testschritte

[21] PHP: Hypertext Preprocessor [TPG10]

[22] Webserver: Ist ein Computer, der Dokumente an Clients wie z. B. Webbrowser überträgt. [Wik10]

[23] XAMPP: Apache Friends ist ein non-profit Projekt zur Förderung des Apache Web-Servers und verbundener Technologien wie MySQL, PHP und Perl. [Kai10]

[24] Apache Server: The Apache Software Foundation provides support for the Apache community of open-source software projects. [TAS10]

mit *„get()“*, *„setField()“*, *„click()“*, *„assertTitle()“* und *„assertText()“* implementiert. Eine detaillierte Dokumentation über alle Module, Klassen und Funktionen findet man auf der Homepage von Simpletest[25].

```php
<?php
require_once('simpletest/autorun.php');
require_once('simpletest/web_tester.php');
require_once('simpletest/reporter.php');

class TestOfRankings extends WebTestCase {
    function testWeAreTopOfGoogle() {
        $this->get('http://www.google.com');
        $this->setField('q', 'simpletest');
        $this->click("Auf gut Glück!");
        $this->assertTitle('SimpleTest - Unit Testing for PHP');
    }
}

class TestOfAbout extends WebTestCase {
    function testOurAboutPageGivesFreeReignToOurEgo() {
        $this->get('http://www.orf.at');
        $this->click('Nordirisches Parlament bekommt Justizministerium');
        $this->assertTitle('news.ORF.at');
        $this->assertText('Nordirisches Parlament bekommt Justizministerium');
    }
}
?>
```

Abbildung 50: Simpletest Code Beispiel

Das PHP Test-Skript wird im Browser geöffnet und läuft automatisch ab. Das Ergebnis des Testlaufes wird sofort im Webbrowser dargestellt.

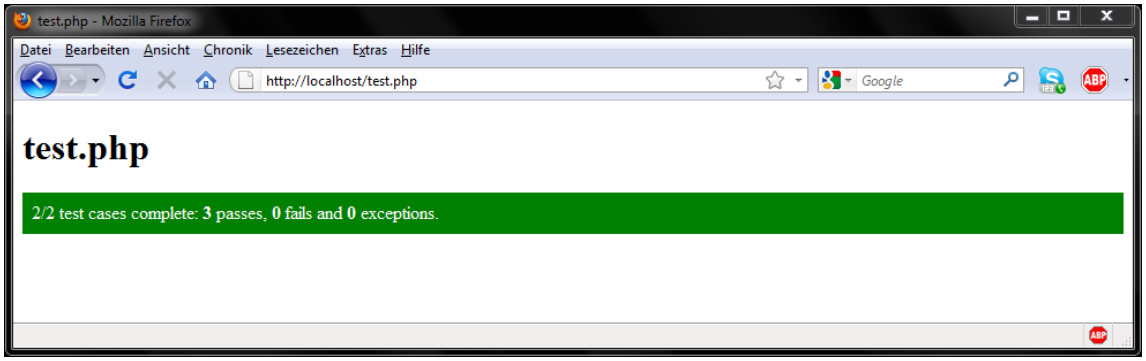

Abbildung 51: Simpletest Ergebnisse im Webbrowser

[25] Simpletest: http://www.simpletest.org

Ein Fehler im Code oder auf der Webseite wird direkt im PHP-Skript im Webbrowser durch die Art des Fehlers sowie die Fehler-Zeile dargestellt.

Tabelle 23: Fehler beim Simpletest Ablauf

Die Module im Handbuch sind in Bereiche wie „Unit tester“, „Group tests“, „Mock objects“, „Partial mocks“, „Reporting“, „Exceptions“, „Web tester“, „Testing forms“, „Authentication“ und „Scriptable browser“ unterteilt (Tabelle 24).

Unit tester	Das Kernsystem des Regressionstest Frameworks
Group tests	Für die Gruppierung der Tests
Mock objects	Zum Erstellen von Mock Objekten
Partial mocks	Zum Erstellen von partiellen Mock Objekten
Reporting	Erstellen von Reports
Expectations	Fehlermeldungen
Web tester	Testen von Web-Elementen
Testing forms	Testen von Web-Formularen
Authentication	Testen von Authentifizierungen
Scriptable browser	Testen von PHP-Skripten

Tabelle 24: Online-Handbuch Aufteilung

Einschränkungen von Simpletest findet man im Bereich von JavaScripts, dynamischen Webseiten, IFrames und XPath Unterstützung. Für die 3.0-Version ist die Entwicklung und Unterstützung von JavaScripts von Simpletest geplant. Diese lassen sich zum Beispiel mit Selenium unterstützen.

4.2.4 Selenium

Ein weiteres Testframework für Web-Applikationen ist das Selenium Projekt, das von der Firma ThoughtWorks entwickelt wurde. Das Framework wurde als freie Software

unter der Apache-2.0-Lizenz veröffentlicht. Mit dem Selenium Testframework ist es möglich, Interaktionen mit einer Web-Anwendung aufnehmen und diese automatisiert zu wiederholen. Das Selenium Projekt beinhaltet mehrere Produkte wie das IDE, Remote Control, Grid, Core, on Rails, on Ruby, CubicTest (ein Plug-In[26] für Eclipse) und Bromine.

Selenium IDE
Selenium Remote Control
Selenium Grid
Selenium Code
Selenium on Rails
Selenium on Ruby
Selenium CubicTest
Selenium Bromine

Tabelle 25: Selenium Produkte

Selenium-IDE[27] ist ein reiner Black-Box Test für einen Webbrowser. Mit einem zusätzlichen Plug-In für Mozilla Firefox[28] ist es möglich, die einzelnen Schritte einer Web-Applikation im Internet Webbrowser aufzunehmen und diese automatisiert abspielen zu lassen. Die einzelnen Schritte, genannt Kommandos[29], beinhalten Einstellungen wie *„Command“*, *„Target“*, *„Value“* und *„Description“*. Im Command werden die eigentlichen Befehle für den Webbrowser angegeben, wie zum Beispiel das Öffnen einer URL oder das Klicken eines Buttons. Die IDE erlaubt auch, Überprüfungen mit *„verify“* und *„assets“* manuell einzufügen oder diese direkt beim Aufnehmen aus dem Webbrowser mit dem Kontext-Menü zu selektieren. Die aufgenommenen Testschritte lassen sich in einzelne Tests oder ganze „*Testruns“* gruppieren. Mehrere Tests lassen sich in einem Testscript zusammenfassen. Die einzelnen Testscripts kann man als XML-Datei lokal speichern.

[26] Plug-In: von engl. to *plug in*, „einstöpseln, anschließen“, deutsch etwa „Erweiterungsmodul“, ist ein Computerprogramm [Wik08]

[27] IDE: engl. integrated development environment, integrierte Entwicklungsumgebung

[28] Mozilla Firefox: Internet Browser, http://www.mozilla-europe.org/de/firefox/

[29] Kommandos: engl. commands

Molybdenum ist (neben Selenium DIE) das bekannteste Mozilla Firefox Add-On für Web-Tests, mit einem einfachen User Interface. An einem kleinen Beispiel möchte ich die Handhabung von Molybdenum erklären.

1. **Installation unter Mozilla Firefox 3.6 Windows 7**

   Zuerst öffnet man den Browser im Betriebssystem Microsoft Windows 7 (die gleiche Prozedur funktioniert auch mit anderen Betriebssystemen, wie Linux oder Mac OS X), danach klickt man auf Extras und dann auf Add-ons. Im Add-ons Popup unter *„Add-ons herunterladen“* Tab trägt man im *„Alle Add-ons durchsuchen“* Molybdenum ein und betätigt mit der *„Enter-Taste“*. Das Molybdenum Add-on wird aufgelistet und mit *„Zu Firefox hinzufügen“* installiert. Nach einem Neustart des Webbrowsers steht dann das Molybdenum Add-on zur Verfügung.

2. **Öffnen und die erste Aufnahme**

   Im Menü-Eintrag *„Extras“* findet man den Punkt Molybdenum. Mit einem Klick oder *„Strg+Umschalt+M“* öffnet sich das Molybdenum Anwendung Pop-up.

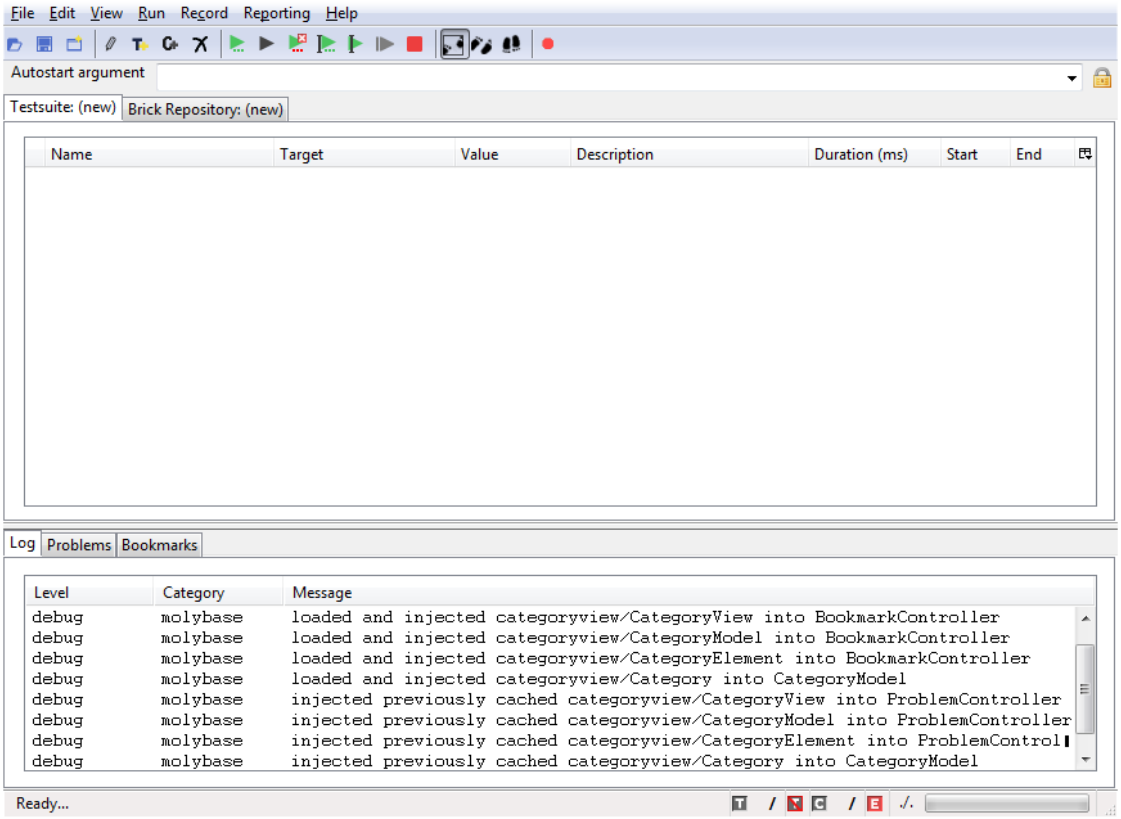

Abbildung 52: Molybdenum Web Test Anwendung

Der rote Kreis startet die Aufnahme *„record“*. Im Webbrowser hat kann man nun wie gewohnt eine Webseite laden indem man den URL in der Adressleiste eingibt z.B. *http://www.molyb.org*. Mit „Enter“ bestätigt man die Eingabe. Danach klickt man einen Link auf dieser Webseite an. Mit der Maus markiert

man einen beliebigen Text und wählt im Kontextmenü *„verify TextPresent“* aus. Diese Aufnahme funktioniert bei jedem Web-Element (z.B. Textboxen, Textflächen, Radio Buttons, Check Boxen, usw). Das Aufnehmen von Ajax-Request ist auch möglich.

3. **Wiedergabe**

   Die Wiedergabe erfolgt mit der F9-Taste oder mit dem grünen Pfeil *„Run all Tests“*. Nach Klicken dieses Buttons werden die einzelnen aufgenommen Schritte nach einander abgespielt.

4. **Automatisierte Wiedergabe**

   Automatisierte Tests kann man auf drei Arten durchführen. Die einfachste Variante lässt sich in der Kommandozeile starten. Hierzu wird der zuvor aufgenommen Selenium-Test als XML-Datei auf die lokale Festplatte gespeichert und in der Kommandozeile mit dem *„firefox.exe“* Befehl gestartet. Ein ANT[30]-Skript ist die zweite Art für ein automatisches Ausführen der Selenium Tests. Die dritte Möglichkeit erfolgt mit dem Maven-Task.

5. **Wiedergabe in der Kommandozeile**

   Der Kommandozeilen-Befehl wird direkt im Firefox Ordner abgesetzt und mit einigen Parametern versehen. Das *„`--no-remote`“* startet den Webbrowser mit einem anderen Profil.

```
C:\Program Files\Mozilla Firefox>firefox.exe --no-remote -P
mytestprofile -mo c:\seleniumtest.xml -report c:\mytest.html  -
headless -autostartarg myvarvalue
```

Tabelle 26: Kommandozeile für einen automatischen Test

[30] Apache Ant (Ant englisch für Ameise) ist ein in Java geschriebenes Werkzeug zum automatisierten Erzeugen von Programmen aus Quelltext. [Wik04]

In der folgenden Tabelle 27 sieht man die Selenium Browser Version Kompatibilität.

Browser	**Selenium-IDE**	**Selenium-RC**	**Operating Systems**
Firefox 3	1.0 Beta-1 & 1.0 Beta-2: Record and playback tests	Start browser, run tests	Windows, Linux, Mac
Firefox 2	1.0 Beta-1: Record and playback tests	Start browser, run tests	Windows, Linux, Mac
IE 8		Under development	Windows
IE 7	Test execution only via Selenium-RC*	Start browser, run tests	Windows
Safari 3	Test execution only via Selenium-RC	Start browser, run tests	Mac
Safari 2	Test execution only via Selenium-RC	Start browser, run tests	Mac
Opera 9	Test execution only via Selenium-RC	Start browser, run tests	Windows, Linux, Mac
Opera 8	Test execution only via Selenium-RC	Start browser, run tests	Windows, Linux, Mac
Google Chrome	Test execution only via Selenium-RC(Windows)	Start browser, run tests	Windows
Others	Test execution only via Selenium-RC	Partial support possible	As applicable

Tabelle 27: Unterstützte Browser und Betriebssysteme [Tho10]

4.2.5 JsUnit

JsUnit ist ein open-source Unit Testframework für Client JavaScripts im Webbrowser und basiert auf dem xUnit Framework. JsUnit wurde von JUnit an JavaScripts portiert und wird von Sourceforge[31] gehostet. Es ist zum automatisierten Testen des clientseitigen JavaScripts im Jahre 2001 entwickelt worden.

[31] Sourceforge: www.SourceForge.net is the world's largest open source software development web site.

Ein JavaScript ist eine interpretierte Programmiersprache mit objektorientierten Fähigkeiten. [Fla07] JavaScript wird in Webbrowsern verwendet und ermöglicht Benutzern das Steuern und Ändern eines Dokuments. Die eingebettete JavaScript Version führt die in HTML Seiten implementierten Skripten direkt im Webbrowser auf dem Clientrechner aus. Diese Ausführung wird als clientseitiges JavaScript bezeichnet, da es nicht vom Webserver ausgeführt wird.

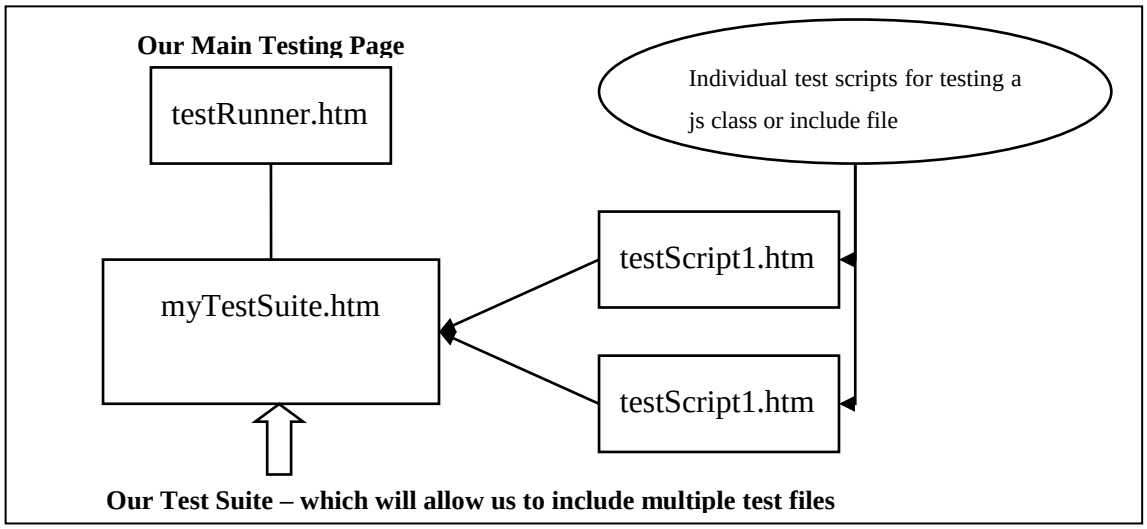

Abbildung 53: Mögliche JsUnit Tests Suite Einbindung [Plu06]

Die Funktion wird in ein HTML, die sogenannte Testseite, eingebettet und durch die Einbindung der *„jsUnitCore.js“* Bibliothek implementiert.

Abbildung 53 zeigt eine mögliche Einbindung von eigenen Testfällen in eine TestSuite. In diesem Beispiel sieht man mehrere individuelle HTML-Test-Scripts, die zu einer Test-Suite zusammengestellt werden. Diese TestSuite ermöglicht dem Tester, mehrere Test-Dateien ohne großen Aufwand zu implementieren. Abbildung 54 veranschaulicht, wie eine Bibliothek in eine HTML-Seite eingebunden werden kann.

```
<script language=”javascript”
src="http://localhost/jsunit/app/jsUnitCore.js"></script>
```

Abbildung 54: Einbindung der jsUnitCore.js Bibliothek in eine HTML Seite

Aussagen Funktionen in JsUnit:

Die Test-Funktionen in JsUnit sind denen der xUnit-Aussagen ähnlich. Wie aus Tabelle 28 ersichtlich ist sind die *„comments“* in den eckigen Klammern optional als Kommentare zu verwenden, wogegen die Werte *„booleanValue, value1, value2“* unbedingt gesetzt gehören.

- assert([comments], booleanValue)
- assertTrue([comments], booleanValue)
- assertFalse([comments], booleanValue)
- assertEquals([comments], value1, value2)
- assertNotEquals([comments], value1, value2)
- assertNull([comments], value)
- assertNotNull([comments], value)
- assertUndefined([comments], value)
- assertNotUndefined([comments], value)
- assertNaN([comments], value)
- assertNotNaN([comments], value)
- fail(comments)

Tabelle 28: Assertion Funktionen in JsUnit

4.2.6 JUnitPerf

Eine weitere xUnit Entwicklung für Performancetests ist JUnitPerf. Es wurde von Mike Clark entwickelt. Das JUnitPerf-Framework ist kostenlos erhältlich. Hauptsächlich geht es bei JUnitPerf Tests um Zeitlimit-Tests *„TimedTest“*, Last-Tests *„LoadTest“* und mehrfach ausgeführte Tests *„RepeatedTest“*.

JUnitPerf erweitert das Konzept von JUnit um die Testklassen *„TimedTest“* und *„LoadTest“*. TimedTest gibt einen Fehler aus, wenn eine Maximalzeit während des Testlaufs überschritten wird. Der LoadTest simuliert die Anzahl der Nutzer, die gleichzeitig auf das Testsystem zugreifen und somit das System belasten. Der RepeatedTest wiederholt eine Test-Sequenz.

```
import com.clarkware.junitperf.*;
import junit.framework.Test;

public class ExampleThroughputUnderLoadTest {
        public static Test suite() {
                int maxUsers = 10;
                long maxElapsedTime = 1500;
                Test testCase = new ExampleTestCase ("testOneSecondResponse");
                Test loadTest = new LoadTest(testCase, maxUsers);
                Test timedTest = new TimedTest(loadTest, maxElapsedTime);
                return timedTest;
        }
        public static void main(String[] args) {
        junit.textui.TestRunner.run(suite());
```

Tabelle 29: JUnitPerf Test Source Code [Joh05]

4.3 Shell- Skript

Ein *„Shell-Skript“* ist eine Textdatei, das entsprechende Kommandozeilen-Befehle nacheinander ausführt. Ein *„Shell-Skript“* erlaubt dem Programmierer Funktionen (Tabelle 30), Bedingungen (Tabelle 31) und Parameter (Tabelle 32) für komplexe Kommandozeilen-Programme zu erstellen. Eine Textdatei wird mit der ersten Zeile *„#!/bin/bash“* als Bash Shell-Skript begonnen. Diese Zeile interpretiert und wertet die einzelnen Kommandos im Skript aus.

```
# Funktion
build_failed() {exit 1}
echolog() {echo "$1" >>$LOGFILE 2>&1}
```

Tabelle 30: Shell Funktion

```
# Bedingung
if [ -e ${BUILDDIR}/report ]; then
  rm -r ${BUILDDIR}/report
fi
```

Tabelle 31: Shell Bedingung

```
# Parameter
WORKDIR=${HOME}/test_ordner
```

Tabelle 32: Shell Parameter

5. Evaluierung

In diesem Kapitel werde ich ein *„Web Applikation Testing Tool“* analysieren und bewerten. Es existiert eine Vielzahl automatisierter Test Werkzeuge auf dem Markt, die teils frei und teils kommerziell angeboten werden. Tabelle 33 listet eine kleine Auswahl der verfügbaren Test Werkzeuge auf. Für meine Analyse und Bewertung werde ich das freiverfügbare Selenium HQ Web Application Testing Solution Tool hernehmen.

• **SeleniumHQ - Web Application Testing Solution**
• Lixto WATS – Web Application Testing Solution
• HP Quality Center – Mercury-QuickTest

Tabelle 33: Test Programme

Selenium HQ WATS ist ein Test-Framework zum Erstellen und Ausführen automatischer *„Web-Applikation“*-Tests. Das Selenium Framework wird oft mit Selenium IDE gleichgesetzt, jedoch ist die IDE nur eine Anwendung in einem großen Paket. Dieses Paket beinhaltet Selenium Core, Selenium IDE und Selenium Remote Control, das sogenannte RC. Diese Produkte können entweder alleine oder gemeinsam eingesetzt werden. Die Kern-Einheit dieses Frameworks bildet Selenium Core.

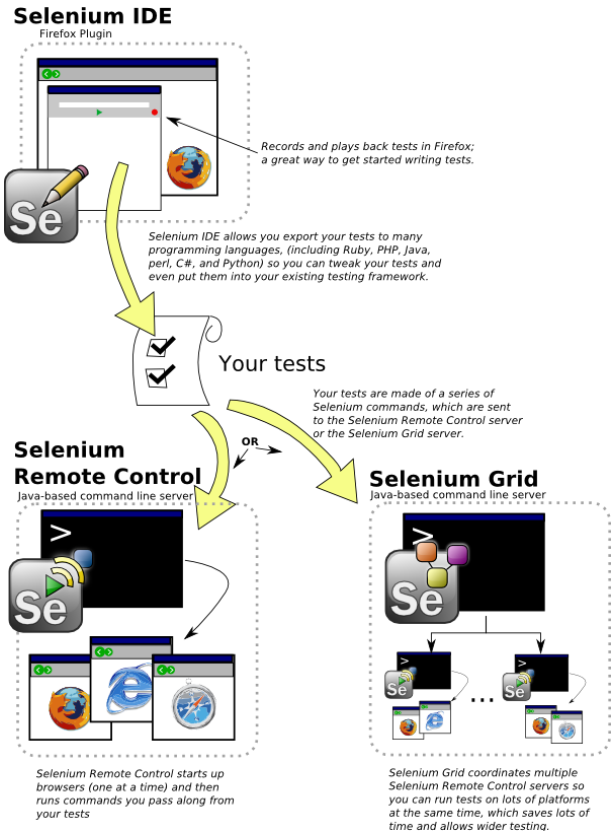

Abbildung 55: Selenium Produkte [Tho10]

5.1 Selenium Core

Selenium Core basiert auf JavaScripts und beinhaltet das Kommando API und den eigentlichen Testrunner. Der Testrunner ermöglicht es, die in HTML gespeicherten Testscripts und die Testfälle in einem Webbrowser abzuspielen. Die Testscripts werden beim Aufnehmen in einer HTML Struktur gespeichert, wobei die einzelnen Kommandos in einer Tabellenzeile und die Tabellenzellen mit den jeweiligen Kommandos, Zielen (Target) und dem Wert (Value) angelegt werden. Im Testrunner öffnet man einzelne Testskripte (Testsuite, beinhaltet mehrere Testfälle) und ladet diese mit dem Go-Button. Im Control Panel des Testrunners kann man die gesamte Testsuite oder einen Testfall starten. Das Testskript muss wie der Testrunner auf einem Testserver liegen. Der Testrunner simuliert Benutzerinteraktionen mithilfe von JavaScripts und testet die Webseite durch Ausführen der einzelnen Kommandos. Nach erfolgreichem Wiedergeben der einzelnen Kommandos kann man davon ausgehen, dass sich die Webseite richtig (gewünscht) verhält. Neben Mozilla Firefox werden auch viele andere Webbrowser und Betriebssysteme durch Selenium Core unterstützt. (siehe Tabelle 27 auf der Seite 74)

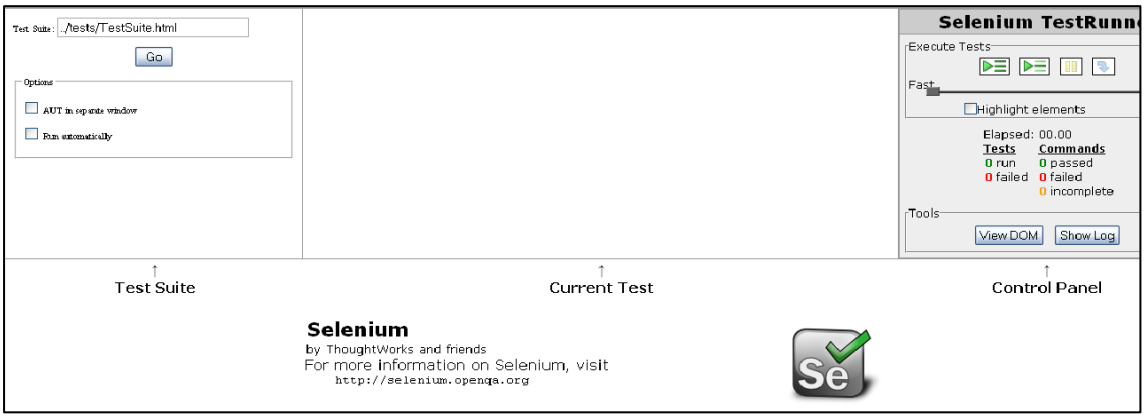

Abbildung 56: Testrunner in einem Webbrowser

In der Abbildung 56 sieht man den Testrunner in einem Webbrowser. Beim Laden des Testrunners „TestRunner.html“ im Selenium Core wird diese Seite mit drei iFrames aufgebaut. Die Testsuite ladet man in der linken Spalte. Im „Current Test“ werden die Kommandos des aktuellen Testfalls dargestellt. Während der Wiedergabe werden die einzelnen Kommandos in Serie abgearbeitet und je nach Ausgabe farblich markiert. Die Markierung erfolgt in drei Farben, gelb (während des Ausführens), rot (im Fehlerfall) und grün (im Erfolgsfall). In der rechten Spalte, dem „Control Panel“, werden die während des Tests gesammelten Informationen sowie die Bedienungselemente für das Starten oder Pausieren der Tests dargestellt. Der eigentliche Web-Applikation Test wird im unteren Bereich des Fensters durchgeführt. Um Testfälle zu erstellen, benötigt man Selenium IDE.

5.2 Selenium IDE

Obwohl Testfälle und Testskripte auch manuell erstellt werden können, ist es sehr hilfreich, diese mit Hilfe eines Werkzeuges zu tun. Selenium IDE[32] nimmt dem Anwender eine Menge Arbeit ab und unterstützt ihn beim Erstellen der Testfälle. IDE wird als Mozilla Firefox Plug-In installiert und im Menü „Extras/Selenium IDE“ gestartet. Durch die automatische Aufnahme (der rote Button rechts im Menü) werden die ausgeführten Befehle und alle Browser-Interaktionen sequenziell in einem Testfall als HTML Code gespeichert (Tabelle 34). Diese sogenannten Kommandos werden in einzelnen Tabellenzeilen und Tabellenzellen gespeichert. Die Wiedergabe der Kommandos erfolgt dann wieder sequentiell als Block mit dem eigentlichen Kommando, dem Ziel und dem Wert. Für ein vollständiges Kommando benötigt man nicht unbedingt alle drei Zeilen. Manche Kommandos stehen manchmal auch ganz alleine, ohne ein Ziel oder Wert da. Die Kommandos können auch mit Variablen versehen werden, um zum Beispiel bestimmte Werte aus der Webseite auszulesen und diese in einem andern Kommando zu verwenden. Das ermöglicht dem Anwender, komplexe und dynamische Testskripte zu erstellen. Diese Testskripte können sich auf Veränderungen auf der Webseite zum Teil anpassen und somit auch reagieren.

```
<tr>
        <td>waitForText</td>                                  -> Kommando
        <td>//*[@id=&amp;quot;PhotoViewerTime&amp;quot;]</td>   -> Ziel
        <td>3/3</td>                                          -> Wert
</tr>
```

Tabelle 34: Auszug aus einem Testfall

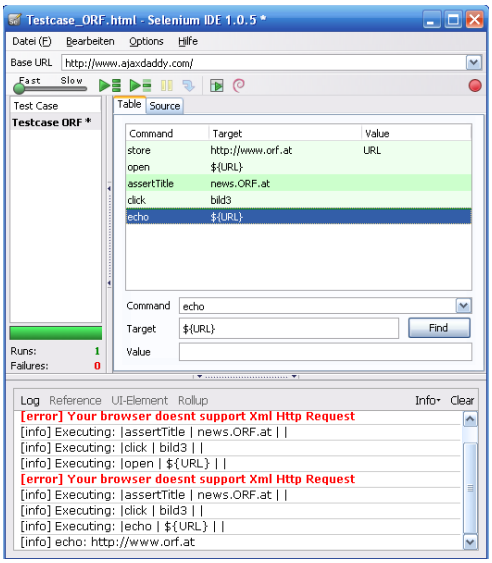

Abbildung 57: Selenium IDE mit einem erfolgreichen Testrun

[32] IDE: Integrated Development Environment

Selenium IDE ist sehr einfach und verständlich aufgebaut. Ein roter Button auf der rechten Seite ermöglicht dem Anwender das automatische Aufnehmen der einzelnen Kommandos. Zum Debuggen stehen einige Buttons zur Auswahl. Mit einem Breakpoint kann man das Skript an einem bestimmten Punkt anhalten, um es Schritt für Schritt ablaufen zu lassen. Dabei kann man Fehler und andere Probleme feststellen und das Testskript auch für eine veränderte Webseite anpassen. Durch Plug-Ins hat man die Möglichkeit, eigene Erweiterungen für IDE zu schreiben oder einzubauen. Dadurch lässt sich das Werkzeug beliebig erweitern und ausbauen, um zukünftige Probleme lösen zu können. In den Optionen kann man das HTML-Skript in fast jede beliebige Programmiersprache formatieren. Eine Export-Funktion speichert die Testskripte in andere Formate.

5.3 Selenium Remote Control

Selenium Remote Control (RC) ermöglicht das Arbeiten mit verschiedenen Web-Browsern, wo hingegen Selenium IDE auf Mozilla Firefox beschränkt ist. Selenium RC muss sich auf einem Testwebserver befinden und auch dort mittels eines Java Kommandozeilen Befehls gestartet werden. Selenium RC läuft dann als Prozess mit und nimmt die Befehle und Kommandos der Testfälle entgegen. Somit lassen sich auch andere Webbrowser starten und die Tests durchführen. Mittels dem „interactive“ Parameter kann man auch direkt in der Kommandozeile ein Kommando anlegen und dieses im Selenium RC ausführen. Der interaktive Befehl wird dann wie folgt in die Kommandozeile eingegeben.

```
cmd=open&1=http://www.yahoo.com
```

Tabelle 35: Interaktive Befehle

Das Starten des Selenium Servers in der Kommandozeile (Tabelle 36) erfolgt mit einem Java Programm und der „selenium-server.jar“ Datei. Es ist auch möglich beim Starten des Servers einige Parameter optional hinten die Jar-Datei mit einem Minus „-“ anzuhängen, um bestimmte Browser Eigenschaften zu initialisieren. Diese Prozedur kann auch erleichtern werden, indem man den Kommandozeilen-Befehl in ein Batch oder Shell Skript (Tabelle 37) einfügt und dieses bequem aus dem Desktop des Betriebssystems heraus startet.

```
java -jar selenium-server.jar -interactive -ensureCleanSession -
browserSessionReuse -singleWindow
```

Tabelle 36: Kommandozeilen Selenium Server Startbefehl

Mit diesem Befehl (Tabelle 36) wird der Remote Control Server gestartet, der zwischen der Entwickler-Plattform und dem Browser läuft. Der Remote Server startet je nach Programmierung (in der Sprache C#, Java, Ruby, perl, PHP oder .Net) ein oder mehrere Browser-Fenster und legt die „Selense“ Kommandos an, die in der Core abgearbeitet werden.

```
@echo on
c:
cd "C:\selenium-grid-1.0.4\selenium-server-1.0.3"
java -jar selenium-server.jar -interactive -ensureCleanSession -
browserSessionReuse -singleWindow
```

Tabelle 37: Batch Datei zum bequemen Starten des Servers

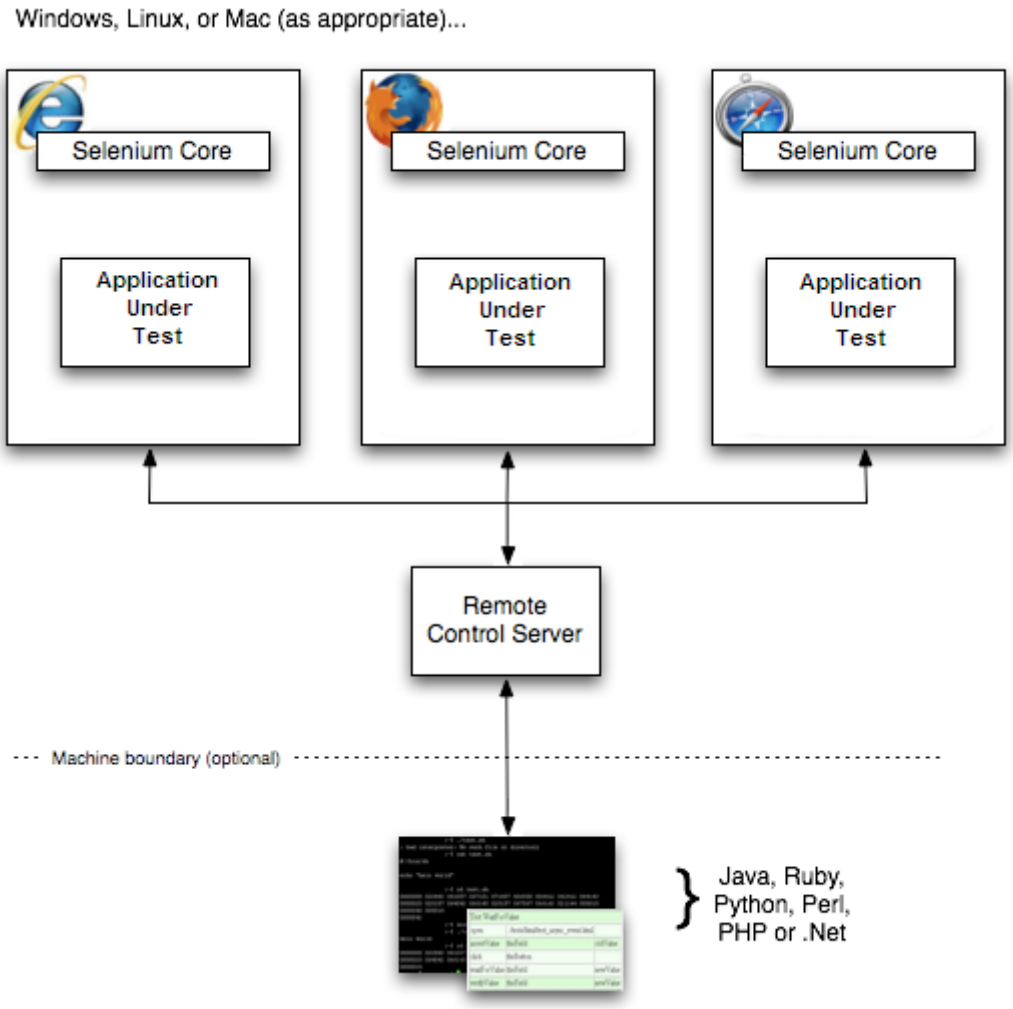

Abbildung 58: Selenium RC Architektur [Tho10]

Während Selenium IDE nur Befehle in einem Webbrowser, dem *„Mozilla Firefox“*, aufnehmen und abarbeiten kann, ist Selenium RC auch in der Lage, Testfälle anderer Entwicklerumgebungen abzuarbeiten. Man schreibt dazu den Code in der eigenen Entwicklerumgebung und schickt diese Kommandos dann direkt dem Remote Control zu. Selenium IDE ermöglicht den Export des HTML Codes in verschiedene Programmiersprachen, um diese direkt in der jeweiligen Entwicklerumgebung verwenden zu können. Dieser Export soll die Entwickler unterstützen, schnell eigene Testskripts aufzunehmen und in die eigene Testumgebung einzubauen. Erfahrene Entwickler können auch ohne diesen Export einen eigenen Test-Code in einer beliebigen Programmiersprache schreiben. Die von Selenium Framework unterstützte Programmiersprachen reichen von Java, C#, PHP bis zu Perl, Python und .Net.

5.3.1 Java

JUnit ist das bekannteste Test-Framework für die Java Umgebung. Selenium Tests in Java benötigen den sogenannten *„Selenium Java Client Driver“*, dieser muss in das *„classpath“* der Entwicklerumgebung eingebunden und die Bibliotheken mit einem *„import“* in das Java-Programm aufgerufen werden (Tabelle 38). JUnit und das Selenium Framework bilden die eigentliche Testumgebung für die GUI Web Applikation Tests.

```
package com.example.tests;

import com.thoughtworks.selenium.*;
import java.util.regex.Pattern;

public class NewTest extends SeleneseTestCase {
    public void setUp() throws Exception {
        setUp("http://www.google.com/", "*firefox");
    }
      public void testNew() throws Exception {
          selenium.open("/");
          selenium.type("q", "selenium rc");
          selenium.click("btnG");
          selenium.waitForPageToLoad("30000");
          assertTrue(selenium.isTextPresent("Results * for selenium
rc"));
    }
}
```

Tabelle 38: Java Implementierung [Tho10]

Abbildung 59 zeigt ein Beispiel mit der Eclipse Entwicklerumgebung.

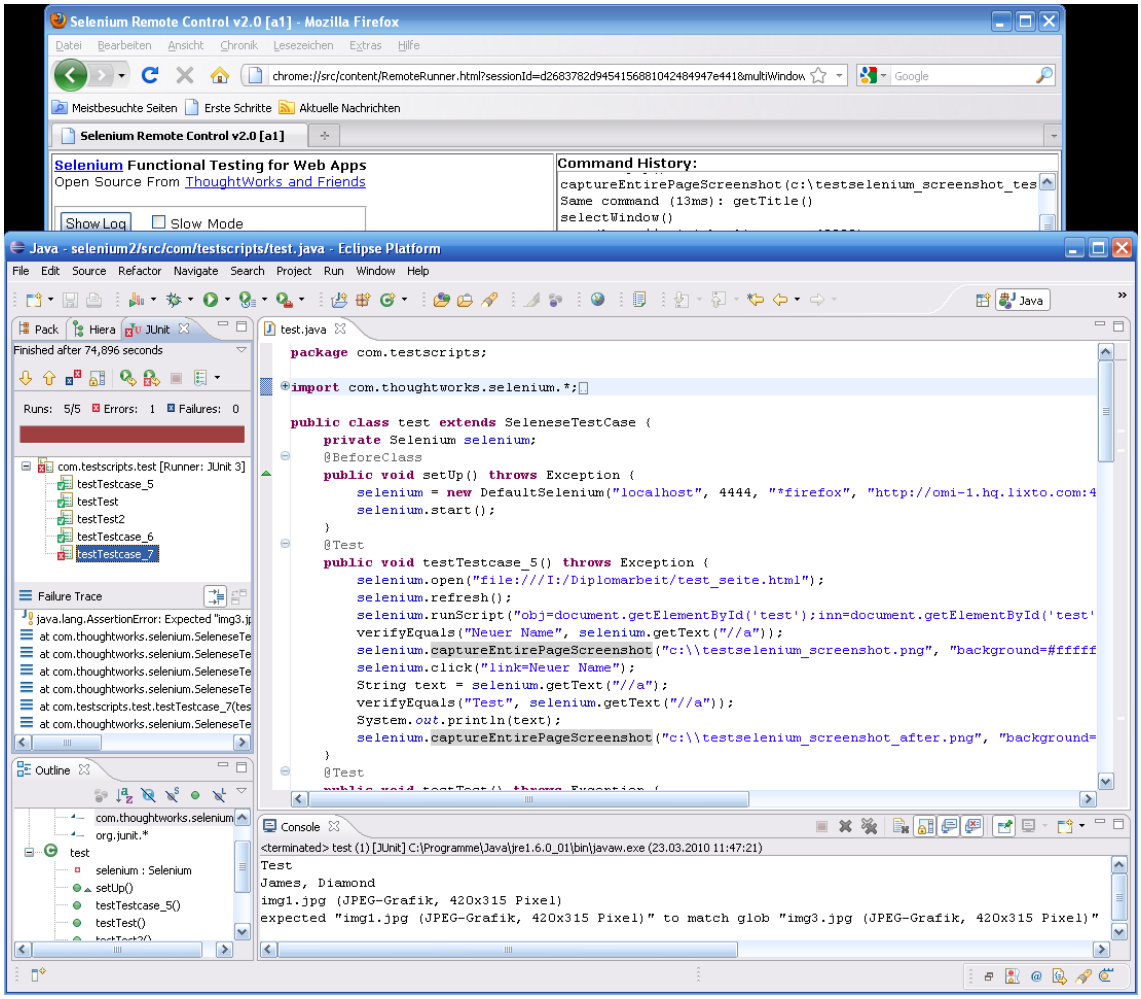

Abbildung 59: Eclipse Entwicklungsumgebung startet das Selenium RC

5.3.2 PHP

Wie für Java vorgesehen, ist es auch in PHP notwendig, die Selenium-Bibliotheken in den PHP Code einzubinden.

```
<?php
require_once 'PHPUnit/Extensions/SeleniumTestCase.php';
class Example extends PHPUnit_Extensions_SeleniumTestCase
{
  function setUp()
  { $this->setBrowser("*firefox");
    $this->setBrowserUrl("http://www.google.com/");}
  function testMyTestCase()
  { $this->open("/");
    $this->type("q", "selenium rc");
    $this->click("btnG");
    $this->waitForPageToLoad("30000");
    $this->assertTrue($this->isTextPresent("Results * for selenium
rc"));}}
?>
```

Tabelle 39: PHP Code [Tho10]

In PHP werden die Kommandos mit *„$this->“* an den Selenium Remote-Control übergeben.

5.4 Kommandos

Die Selenium-Kommandos, auch *„Selense“* genannt, sind die Hauptbestandteile des Selenium Tests. Eine sequentielle Aneinanderreihung der Kommandos bildet das Testskript. Die Kommandos kann man in drei Bereiche aufteilen: Verifizierung, Lokalisierung und Matching.

5.4.1 Verifizierung

Die Verifizierung ist notwendig, um bestimmte Eigenschaften und Gegebenheiten auf der Webseite zu ermitteln und zu bestätigen. Wozu Verifizierung notwendig ist, lässt sich anhand einiger Fragen erklären (Tabelle 40).

Ist ein bestimmtes Element irgendwo auf der Webseite vorhanden?
Existiert eine spezielle Textpassage auf der Webseite?
Ist eine spezielle Textpassage an einer speziellen Position auf der Webseite zu finden?

Tabelle 40: Fragen zur Verifizierung

Die drei gängigsten Verifizierung-Kommandos sind *„verifyTextPresent“*, *„verifyElementPresent“*, *„verifyText“*. Mit diesen Kommandos lassen sich alle Fragen von Tabelle 40 beantworten. Im folgenden Beispiel wird das *„verifyText“*-Kommando dargestellt.

Command	**Target**	**Value**
verifyText	//table/tr/td/div/p	This is my text and it occurs right after the div inside the table.

Tabelle 41: Beispiel für verifyText [Tho10]

5.4.2 Elemente Lokalisieren

Webelemente lassen sich anhand von Positionsangaben lokalisieren. Diese können zum Beispiel durch den DOM, XPath oder dem Identifier lokalisiert werden.

Die einfachste Methode zur Lokalisierung eines Elements ist der „Link Text“. Anhand des Link-Textes wird ein Hyperlink auf der Webseite gefunden.

Im folgenden Beispiel wird ein Hyperlink mit dem Ziel-Befehl (engl. target) *„link=Continue“* lokalisiert.

```
<html>
 <body>
  <p>Are you sure you want to do this?</p>
  <a href="continue.html">Continue</a>
  <a href="cancel.html">Cancel</a>
</body>
<html>


link=Continue (4)
link=Cancel (5)
```

Tabelle 42: Lokalisierung durch Link-Text [Tho10]

Die DOM Lokalisierung verwendet JavaScripts zum Evaluieren und Lokalisieren der Elemente. Die DOM Objekte werden mit *„document“* beginnend und Punkt *„.“* zwischen den einzelnen Knoten sowie einem Index angegeben (Tabelle 43).

Das JavaScript *„document“*-Objekt ermöglicht das Auslesen und Manipulieren der einzelnen Zustände von Formularelemente. Die einzelnen Formulare werden durch JavaScript in einem Array festgehalten und das erste Formular ist mit dem Index [0] ansprechbar. Mit diesem JavaScript Objekt und den dazugehörigen Methoden (z.B. *„write()“*, *„getElementId()“*) ist es möglich, alle Elemente in einem HTML-Dokument zu identifizieren, anzusprechen und zu manipulieren.

```
<html>
  <body>
   <form id="loginForm">
    <input name="username" type="text" />
    <input name="password" type="password" />
    <input name="continue" type="submit" value="Login" />
    <input name="continue" type="button" value="Clear" />
   </form>
 </body>
 <html>


dom=document.getElementById('loginForm') (3)
dom=document.forms['loginForm'] (3)
dom=document.forms[0] (3)
document.forms[0].username (4)
document.forms[0].elements['username'] (4)
document.forms[0].elements[0] (4)
document.forms[0].elements[3] (7)
```

Tabelle 43: DOM Beispiel [Tho10]

Die XPath-Methode ist eine Sprache zur Lokalisierung eines Knotens in einem XML Dokument. Im Selenium benutzt man diese Sprache, um Elemente auf der Web Applikation anzuwählen. Absolute XPath’s beinhalten dazu alle Elemente beginnend mit der HTML-Wurzel. Neben dem absoluten Pfad kann man die Identifizierung mit dem relativen Pfad oder über die *„Id“* und Elementnamen durchführen. Ein XPath beginnt immer mit *„//“*.

Die folgenden Beispiele (Tabelle 44) dienen zur Veranschaulichung der möglichen Nutzung. Das *„xpath="* im Beispiel ist in Selenium nicht notwendig.

- `xpath=/html/body/form[1]` (3) - *Absolute path (would break if the HTML was changed only slightly)*
- `//form[1]` (3) - *First form element in the HTML*
- `xpath=//form[@id='loginForm']` (3) - *The form element with attribute named 'id' and the value 'loginForm'*
- `xpath=//form[input/\@name='username']` (4) - *First form element with an input child element with attribute named 'name' and the value 'username'*
- `//input[@name='username']` (4) - *First input element with attribute named 'name' and the value 'username'*
- `//form[@id='loginForm']/input[1]` (4) - *First input child element of the form element with attribute named 'id' and the value 'loginForm'*
- `//input[@name='continue'][@type='button']` (7) - *Input with attribute named 'name' and the value 'continue' and attribute named 'type' and the value 'button'*
- `//form[@id='loginForm']/input[4]` (7) - *Fourth input child element of the form element with attribute named 'id' and value 'loginForm'*

Tabelle 44: Beispiele für XPaths [Tho10]

Die Identifizierung erfolgt in erster Linie durch die sogenannte *„Identifier"*-Methode. Mehrfach-Benennungen werden in dieser Methode ignoriert und das erste Element mit dem passenden Namen benutzt.

```
<html>
 <body>
  <form id="loginForm">
   <input name="username" type="text" />
   <input name="password" type="password" />
   <input name="continue" type="submit" value="Login" />
  </form>
 </body>
<html>
identifier=loginForm (3)
identifier=username (4)
identifier=continue (5)
continue (5)
```

Tabelle 45: Identifier Beispiel [Tho10]

5.5 Fallbeispiel-Lösungsansatz

Mein Lösungsansatz auf die Fragestellung *„Wie kann man Selenium-Tests automatisieren und wie generiert man tägliche Selenium-Test-Reports?“* sieht wie folgt aus. Ich habe eine automatisierte Testumgebung mit Hilfe einer *„Shell-Skript“* erstellt, das täglich Reports über die einzelnen Testfälle und Testskripte ausgibt.

„Welche Werkzeuge und Infrastruktur ist dazu notwendig?“

Als Testwerkzeug verwende ich Selenium und das freie *„Linux“* Betriebssystem, das mir mit *„Cron“, „Ant“* und *„Shell-Skript“* eine gute Voraussetzung für die Automatisierung bietet.

5.5.1 Testvorbedingungen

Ich möchte eine virtuelle, in Entwicklung stehende Webseite automatisiert testen. Die dazu verwendeten Technologien (Linux, Selenium) stehen Lizenz-frei zur Verfügung.

5.5.2 Testumgebung

Als Testumgebung habe ich zwei Server ausgesucht. *„Server 1“* dient hier als Run-Server und *„Server 2“* als Selenium Remote-Control-Server. Das ganze lässt sich auch virtualisieren und auf einem Computer direkt testen.

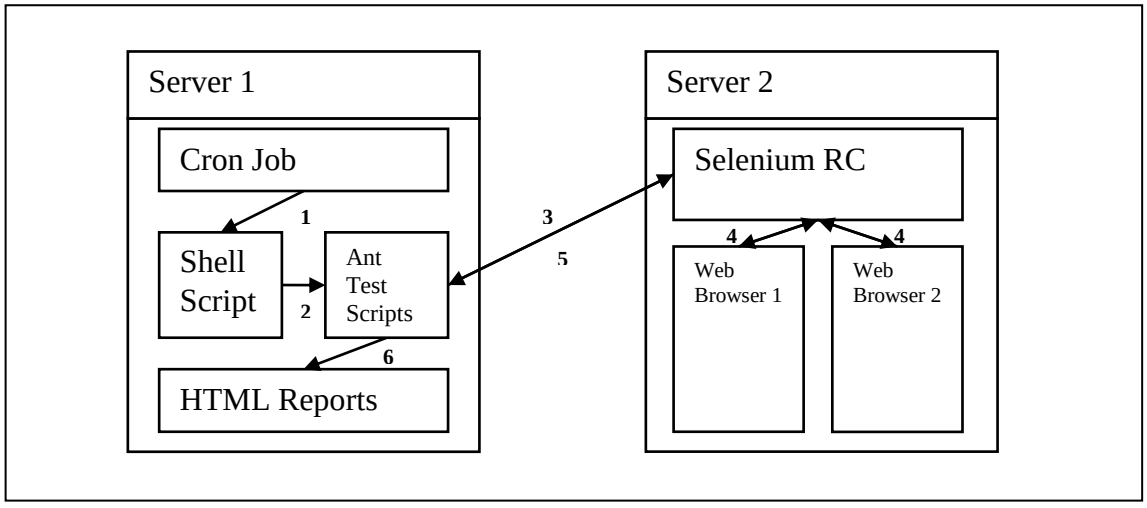

Abbildung 60: Server Testumgebung

Im ersten Schritt wird täglich ein „*cron job*“[33] gestartet. Die Datei habe ich dafür mit dem Befehl „*crontab -e* “ in der Kommandozeile mit den Zeilen (Tabelle 46) ergänzt. Diese ermöglicht mir, das Shell Skript „*build.sh*“ jeden Tag um 8 Uhr in der Früh automatisch zu starten.

[33] Cron ist ein Unix Deamon zum Verwalten immer wiederkehrender Aufgaben.

```
# run Tests
00 08   * * *   /home/serkan /build.sh
```

Tabelle 46: Cron Job einstellen

Der Projekt Ordner beinhaltet den Quell-Code *„src“*, die Bibliotheken *„lib“*, das Shell Skript *„build.sh“* sowie die ANT-Skripte *„build.xml“* und *„build-testsuite.xml“*.

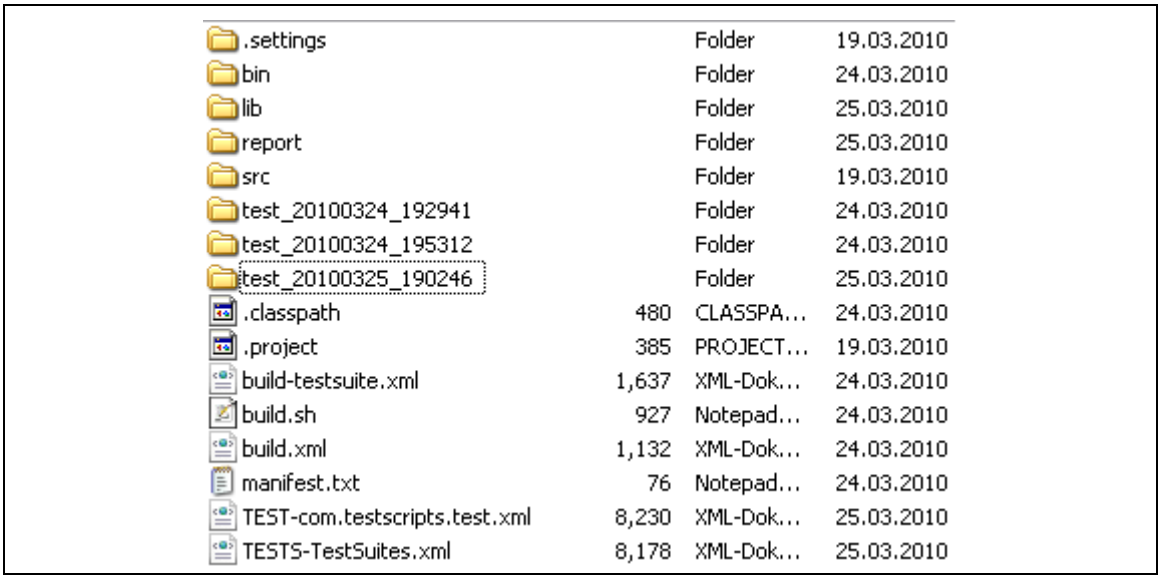

Abbildung 61: Projekt Ordner auf Server 1

Das Shell Skript ruft ein ANT-Testskript auf und startet die Selenium-Tests. Diese Tests werden sequentiell in die Selenium Remote Control übermittelt und ausgeführt. Nach Beendigung der Tests werden die Resultate *„report“* im Server 1 als HTML-Dateien gespeichert. Das Shell Skript kopiert diese dann in den richtigen Apache-Server Ordner, um diese im Browser zugänglich zu machen. In meinem Beispiel werden die Dateien direkt in einen dummy Ordner *„test_%datum%_%zeit%“* kopiert, wo auch die Log-Datei gespeichert wird. Mit dieser Struktur bekomme ich täglich Log-Dateien zum Testdurchlauf und Resultate als HTML gespeichert.

Das Shell Skript (Tabelle 47) wird in einem Durchlauf ausgeführt und startet dabei mehrere ANT-Befehle. Dazu verwendet der ANT-Befehl Standard-mäßig die *„build.xml“* Datei. Es ist jedoch auch möglich, mit *„-f build1.xml“* andere Skripte aufzurufen. Der erste Befehl *„ant clean“* löscht dabei die alten Klassen Dateien und die Projekt.jar Datei. Das Projekt wird dazu im nächsten Schritt *„ant compile“* neu kompiliert und mit *„ant compress“* eine neue Projekt.jar Datei erzeugt. Diese Dateien werden im nächsten Schritt für den Testdurchlauf benötigt. Ich habe diesen Schritt extra eingefügt, um bei verändertem Quell-Code immer eine saubere Struktur zu haben.

Im letzten Schritt wird das *„ant test“* Target im build.xml Skript aufgerufen. Dieses Target ermöglicht uns, einen weiteren ANT Befehl *„<ant antfile="build-testsuite.xml"/>“* auszuführen.

```
#!/usr/bin/env bash
# Test Shell Script

# set parameters
WORKDIR=${HOME}/test_ordner
BUILDDIR=/home/serkan/test_ordner
DEPLOYDIR="$BUILDDIR/test_$(date "+%Y%m%d_%H%M%S")"

#export JAVA_HOME=/usr/local/jdk1.6.0_10
#export ANT_HOME=/home/serkan/apache-ant-1.7.0

if [ -e ${BUILDDIR}/report ]; then
  rm -r ${BUILDDIR}/report
fi

### functions ###
build_failed() {exit 1}
echolog() {echo "$1" >>$LOGFILE 2>&1}

mkdir $DEPLOYDIR
LOGFILE=$DEPLOYDIR/build.log
NOW=`date`
echolog "Test build from  $NOW"
echolog ""

echolog "Clean"
cd ${BUILDDIR}
#${ANT_HOME}/bin/
ant clean >>$LOGFILE 2>&1 || build_failed
echolog "Start Compile"
ant compile >>$LOGFILE 2>&1 || build_failed
echolog "Start Compress"
ant compress >>$LOGFILE 2>&1 || build_failed
echolog "Start Tests"
ant test >>$LOGFILE 2>&1 || build_failed
echolog "Copy reports to correct folder"
cp -r ${BUILDDIR}/report $DEPLOYDIR
```

Tabelle 47: Shell Skript

```
<?xml version="1.0" ?>
<project default="main">

    <target name="main" depends="clean, compile, compress, test"
        description="Main target">
        <echo>
             Clean, Compile, Building the .jar file and start tests.
        </echo>
    </target>

    <path id="classpath.test">
        <pathelement location="lib/selenium-server.jar" />
        <pathelement location="lib/junit-4.6.jar" />
        <pathelement location="lib/selenium-java-client-driver.jar" />
        <pathelement location="lib/Project.jar" />
    </path>

    <target name="clean" description="- cleans files">
             <delete dir="bin/com/testscripts/" includes="*.class"/>
             <delete dir="lib" includes="Project.jar"/>
    </target>

    <target name="compile" description="Compilation target">
        <javac srcdir="src/com/testscripts" destdir="bin">
               <classpath refid="classpath.test"/>
        </javac>
    </target>

    <target name="compress" description="Compression target">
        <jar jarfile="lib/Project.jar" basedir="bin" />
    </target>

    <target name="test">
        <ant antfile="build-testsuite.xml"/>
    </target>

</project>
```

Tabelle 48: Build.xml Datei

Die Testsuite beinhaltet zwei Testfälle, die jedoch beliebig erweitert werden können. In Tabelle 49 sieht man die beiden Testfälle in den Targets *„Testcase1“* und *„Testcase2“*.

```
<?xml version="1.0"?>

<project name="Tests" default="testAllDelete">

    <description>
            Tests
    </description>

    <path id="classpath.test" description="- classpath settings">
        <pathelement location="lib/selenium-server.jar" />
        <pathelement location="lib/junit-4.6.jar" />
        <pathelement location="lib/selenium-java-client-driver.jar" />
        <pathelement location="lib/Project.jar" />
    </path>

   <target name="testAllDelete" description="- starts all tests">
        <echo>=== Clean:</echo>
        <antcall target="clean"/>
        <echo>=== Testall:</echo>
        <antcall target="testAll"/>
    </target>

    <target name="clean" description="- clean target">
    </target>

    <target name="testAll" description="- starts all tests">
        <echo>=== Tests:</echo>
        <antcall target="Testcase1"/>
        <antcall target="Testcase2"/>
        <echo>=== Reports:</echo>
        <junitreport todir=".">
            <fileset dir=".">
                <include name="TEST-*.xml"/>
            </fileset>
            <report format="frames" todir="./report"/>
        </junitreport>
    </target>

    <target name="Testcase1" description="- testcase1">
        <junit printsummary="yes" fork="yes" showoutput="no">
          <classpath refid="classpath.test" />
          <formatter type="xml" usefile="true"/>
          <test name="com.testscripts.test_suite"/>
        </junit>
    </target>

    <target name="Testcase2" description="- testcase2">
        <junit printsummary="yes" fork="yes" showoutput="no">
          <classpath refid="classpath.test" />
          <formatter type="xml" usefile="true"/>
          <test name="com.testscripts.testcase2"/>
        </junit>
    </target>

</project>
```

Tabelle 49: Testsuite Datei *"Build-testsuite.xml"*

5.5.3 Testfälle

Die beiden Testfälle habe ich mit Selenium IDE aufgenommen und als JUnit Code exportiert. Den exportierten JUnit Code verwende ich als Java-Datei in meiner Projektstruktur. Mit dieser Strategie kann man mit Selenium IDE beliebig viele Testfälle und sogar neue Testsuiten direkt erstellen und exportieren. Die in diesem Fallbespiel erstellten Testfälle sind im Anhang zu finden.

Testfall 1

Der erste Testfall beinhaltet drei einzelne Testblöcke.

Im ersten Testblock wird eine HTML-Seite auf der lokalen Festplatte geladen und diese mit einem JavaScript manipuliert. Das JavaScript fügt dem Link ein href-Attribut hinzu, so dass der nächste Schritt im Test diesen Link lokalisiert und benützt.

```
selenium.runScript("obj=document.getElementById('test');inn=document.g
etElementById('test').innerHTML = 'Neuer
Name';obj.setAttribute('href','test_seite.html');");
```

Tabelle 50: JavaScript im Testfall

Im zweiten Testblock wird die *„http://www.ajaxdaddy.com/demo-bsn-autocomplete.html“* geladen und ein Ajax-Test durchgeführt. Dabei schreibt der Testfall den Text *„james“* in eine Textbox hinein und wartet auf den darauffolgenden JavaScript. Dieser JavaScript lässt eine Box erscheinen, die im nächsten Schritt angeklickt und der Text in die Textbox übernommen wird. Das wichtigste bei diesem Test ist das Timing, da die Box nur wenige Sekunden auf dem Bildschirm sichtbar bleibt und innerhalb dieser Zeit der nächste Schritt im Test durchgeführt werden muss.

Im dritten Testblock wird wieder ein Ajax Test durchgeführt, in dem auf virtuelle Buttons geklickt wird. Das Testende wird durch ein Vergleichen eines Bildes

Abbildung 62: Vergleich des Testbildes [Sar07]

durchgeführt.

Testfall 2

Im diesem Testfall wird wieder ein Ajax-Test durchgeführt, in dem auf virtuelle Buttons geklickt wird. Auch hier wird zum Abschluss ein Bild verglichen.

5.5.4 Resultate

Die Ergebnisse werden während des Testens als XML-Dateien angelegt und am Ende des Testes als HTML zusammengestellt. Das Resultat öffnet man mit der *„index.html“* Datei im *„result“* Ordner. Die Log-Datei gibt Auskunft über den gesamten Testverlauf.

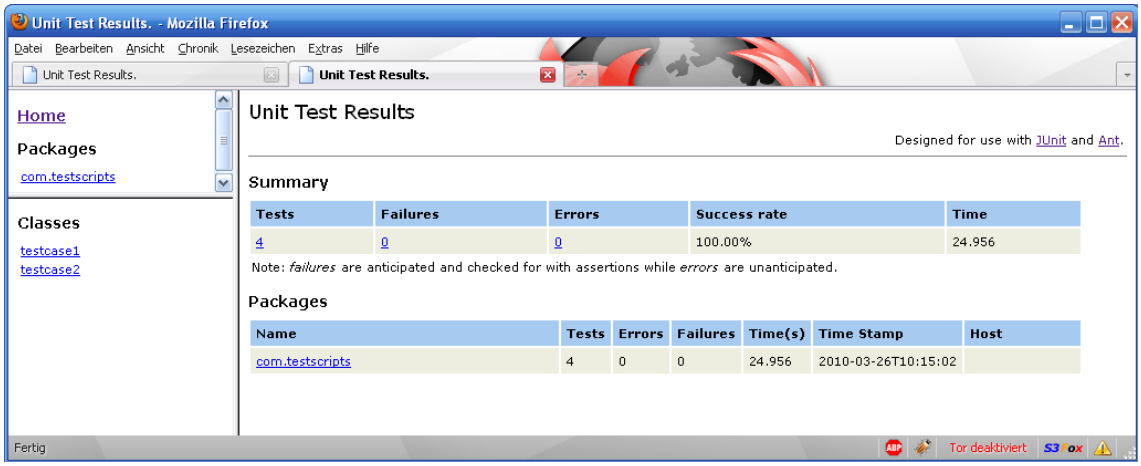

Abbildung 64: HTML Resultate im Web-Browser

Die Links im Report führen dann zu den Detailansichten der einzelnen Testklassen und Tests.

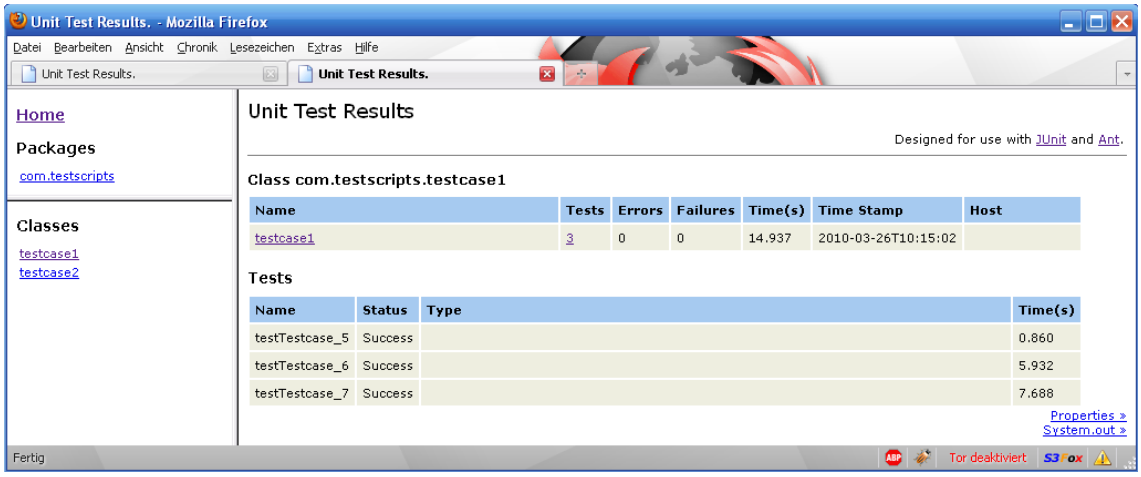

Abbildung 63: Detailansicht im Report

5.6 Alternative Lösung

Die im vorherigen Kapitel gezeigte Lösung kann man alternativ mit einer Windows Testumgebung realisieren. Dazu verwendet man wie vorher auch das Selenium Remote Control und ein Windows Batch-Skript. Diese kann man mit einem Cronw-Service täglich automatisch starten und abarbeiten. Die Ergebnisse werden auch hier als HTML-Dateien ausgegeben, die über den Web-Browser dargestellt werden können. Das ermöglicht Unternehmen, die ihre Systeme auf eine Microsoft-Windows-Umgebung aufbauen, eine alternative Lösung.

5.7 Ergebnisse

In dieser Arbeit habe ich die Grundlagen, Methoden und die Notwendigkeit des Testens für Web-Applikationen identifiziert. Dabei bin ich auf eine große Anzahl von Testwerkzeuge gestoßen, die alle das Thema „Testen von Web-Applikationen“ sehr ernst und nüchtern zu lösen versuchen. Beim Betrachten dieser Lösungen fand ich das Selenium HQ Framework. Das Selenium HQ Web Application Testing Solution Tool geht dabei einen eigenen Weg und versucht, die jetzigen Testansätze (Unit Tests) mit dem des Webtestens zu kombinieren. In diesem Ansatz werden so viele Test- und Programmierumgebungen, Betriebssysteme und unterschiedliche Webbrowser wie nur möglich eingebunden. Das bietet den Entwicklern einen großen Spielraum für den Einsatz der Selenium Testumgebung. In meiner Arbeit fand ich heraus, dass die automatisierten Selenium-Tests mit einigen Erweiterungen auch in der Praxis eingesetzt werden können. Die von mir vorgenommenen Erweiterungen schaffen eine voll automatisierte Testumgebung, die täglich Test-Szenarien durcharbeitet und dazu Reports erstellt. Kleine bis mittlere Unternehmen und freie Web-Entwickler finden damit eine leistungsfähige Testumgebung vor.

6. Abbildungsverzeichnis

7. Tabellenverzeichnis

8. Literaturverzeichnis

[Alp94] **Alper, Marcel. 1994.** *Professionelle Softwaretests.* Braunschweig [u.a.] : Vieweg, 1994.

[Aut72] **Autorenkollektiv. 1972.** *Lexikon Technik und exakte Naturwissenschaften Bd.1.* Frankfurt/Main : Fischer, 1972. ISBN-13: 978-3411011551.

[BFJ07] **Becker Michael, Fraikin Falk, Jungmayr Stefan, Schnizler Moritz, Schoolmann Andreas, Winter Mario. 2007.** IT Transfer Office. [Online] 24. 05. 2007 [Zitat vom: 27. 01. 2010] http://www.ito.tu-darmstadt.de/publs/pdf/FraikinTestVonKomponenten.pdf.

[Ber10] **Bergmann, Sebastian. 2010.** http://www.phpunit.de. [Online] 14. 02. 2010 [Zitat vom: 02. 03. 2010] http://www.phpunit.de.

[Bra04] **Braig, Übersetzt von Steffen Gemkow und Andreas. 2004.** *Unit-Tests mit JUnit.* Münschen Wien : Carl Hanser Verlag, 2004. ISBN 3-446-22824-1.

[Bro08] **Brockhaus. 2008.** *Der Brockhaus in einem Band.* München: Brockhaus in der Wissensmedia, 2008. ISBN-10: 3765316830.

[Den92] **Denert, Ernst. 1991, 1992.** *Software-Engineering.* Berlin: Springer, 1991, 1992. ISBN 3-540-53404-0.

[Dor01] **Dormann, Markus. 2001.** Seminar in SE und SQM WS2 2001/02. [Online] 13. 11. 2001 [Zitat vom: 08. 02. 2010] http://www.ifi.uzh.ch/req/courses/seminar_ws01/.

[Duh10] **Duhr, Stefan.** http://www.ib.hu-berlin.de. [Online] [Zitat vom: 09. 01. 2010] http://www.ib.hu-berlin.de/~rfunk/HSt/qm_im/qualitaet.pdf.

[DRP01] **Dustin Elfriede, Rashka Jeff, Paul John. 2001.** *Software automatisch testen.* Berlin Heidelberg : Springer-Verlag, 2001. ISBN 3-540-67639-2.

[Ele10] Elektroniknet: Home. [Online] Weka Fachmedien GmbH. [Zitat vom: 09. 02. 2010] http://www.elektroniknet.de/?id=16354.

[Fla07] **Flanagan, David. 2007.** *JavaScript: Das umfassende Referenzwerk.* Köln: O'Reilly Verlag GmbH & Co. KG, 2007. ISBN: 978-3-89721-491-0.

[Fun10] **Funk, Robert.** Prof. Dr. rer. pol. Robert Funk. [Online] [Zitat vom: 12. 01. 2010] http://www.ib.hu-berlin.de/~rfunk/HSt/qm_im/qualitaet.pdf.

[GiL09] **Gietl Gerhard, Lobinger Werner. 2009.** *Leitfaden für Qualitätsauditoren und Durchführung von Audits nach ISO 9001:2008, 3.Auflage, Planung.* München: Carl Hanser Verlag, 2009. ISBN 978-3-446-41835-6.

[GuS98] **Gumm Heinz-Peter, Sommer Manfred. 1998.** *Einführung in die Informatik.* München/Wien: Oldenbourg, 1998. ISBN 3-486-24422-1.

[Hag05] **Hagen, Johannes. 2005.** Java-Performancemessung. [Online] 02. 06. 2005 [Zitat vom: 16. 03. 2010] http://ebus.informatik.uni-leipzig.de/www/media/lehre/seminar-javatools05/semtools05-hagen-slides.pdf.

[Har03] **Hartmut, Ernst. 2003.** *Grundkurs Informatik 3. Auflage.* Braunschweig/Wiesbaden: Vieweg, 2003. ISBN 3-528-25717-2.

[HCL10] **HCL Technologies.** Test Automation. [Online] HCL Technologies: Offshore Software Development, Outsourcing Software Development Services, Custom Application:. [Zitat vom: 27. 01. 2010] http://www.hcltech.com/enterprise-transformation-services/independent-verification-and-validation/success-stories/23/.

[HUS06] **Heinle Nick, Ulrich Bill, Speidel Pena. 2006.** *Webdesign mit JavaScript & Ajax.* Köln : O`Reilly Verlag, 2006. ISBN-10 3-89721-471-7.

[HöM05] **Höfer Andreas, Dr. Müller Matthias. 2005.** Institute for Program Structures and Data Organization. [Online] 19. 10. 2005 [Zitat vom: 11. 03. 2010] http://www.ipd.uka.de/.

[Hof08] **Hoffmann, Dirk W. 2008.** *Software-Qualität.* Berlin-Heidelberg : Springer, 2008. ISBN 978-3-540-76323-9.

[Hor09] **Horn, Torsten. 2009.** JUnit. [Online] 06. 06. 2009 [Zitat vom: 01. 03. 2010] http://www.torsten-horn.de/techdocs/java-junit.htm.

[How10] **Hower, Rick. 2010.** Web Test Tools. [Online] Software QA and Testing Resource Center, 12. 01. 2010 [Zitat vom: 08. 02. 2010] http://www.softwareqatest.com/qatweb1.html.

[HTW10] **HTWK Leipzig.** Swt Wiki. [Online] [Zitat vom: 27. 01. 2010] http://www.imn.htwk-leipzig.de/~weicker/pmwiki/pmwiki.php/Main/Blackbox-Test.

[Ian09] **Ian Hickson, David Hyatt. 2009.** HTML 5. [Online] W3C, 25. 09. 2009 [Zitat vom: 10. 02. 2010] http://www.w3.org/TR/html5/.

[IEE90] **IEEE. 1990.** *Fachwörterbuch der Terminologie der Softwaretechnik.* New York, NY : IEEE, 1990.

[INS10] **INSO - Industrial Software. 2010.** INSO - Software Testen. [Online] INSO - Industrial Software, 31. 03. 2010 [Zitat vom: 31. 03. 2010] http://www.inso.tuwien.ac.at/lectures/software_testen/.

[IOS10] **International Organization for Standardization.** ISO Online. [Online] International Organization for Standardization. [Zitat vom: 09. 01. 2010] http://www.iso.org/iso/home.htm.

[Kir08] **Kirschner, Dr. Denise. 2007, 2008.** Matlab functions for PRCC and eFAST. [Online] University of Michigan, 2007, 2008. [Zitat vom: 26. 01. 2010] http://malthus.micro.med.umich.edu/lab/usadata/.

[Kit95] **Kit, Edward. 1995.** *Software Testing in the Real World - Improving the Process.* Great Britain : Addision-Wesley, 1995. ISBN 0-201-87756-2.

[Kla07] **Klaus, Franz. 2007.** *Handbuch zum Testen von Web-Applikationen.* Berlin Heidelberg New York : Springer, 2007. ISBN 978-3-540-24539-1.

[Kle09] **Kleuker, Stephan. 2009.** *Formale Modelle der Softwareentwicklung Model-Checking, Verifikation, Analyse und Simulation.* Wiesbaden : Vieweg+Teubner, GWV Fachverlage GmbH, 2009. ISBN 978-3-8348-0669-7.

[Kob03] **Kobert, Thomad. 2003.** *HTML 4.* Bonn : Verlag moderne Industrie Buch AG & Co. KG, Landsberg Königswinterer Str. 418, 2003. ISBN 3-8266-7197-X.

[Lig02] **Liggesmeyer, Peter. 2002.** *Software-Qualität: Testen, Analysieren und Verifizieren von Software.* Heidelberg, Berlin : Spektrum, Akademischer Verlag, 2002. ISBN 3-8274-1118-1.

[Mgj08] **Mgjbot. 2008.** About the Document Object Model - MDC. [Online] Mozilla Developer Center, 10. 06. 2008 [Zitat vom: 16. 03. 2010] https://developer.mozilla.org/en/About_the_Document_Object_Model.

[Obj10] **Object Mentor.** Welcome to JUnit.org! | JUnit.org. [Online] Object Mentor. [Zitat vom: 01. 03. 2010] http://www.junit.org.

[PJR09] **Page Alan, Johnston Ken, Rollison Bj. 2009.** *HOW WE TEST SOFTWARE AT MICROSOFT.* Redmond, Washington: Microsoft Press, 2009. Body Part No. X14-71546.

[PeS07] **Pehnke Oliver, Schmid Benjamin. 2007.** eXXcellent solutions Homepage. [Online] 2007. [Zitat vom: 16. 01. 2010] http://www.exxcellent.de.

[PeY09] **Pezze Mauro, Young Michal. 2009.** *Software testen und analysieren, Prozesse, Prinzipien und Techniken.* München: Oldenbourg Verlag, 2009. ISBN 978-3-486-58521-6.

[Plu06] **Plush, Jim. 2006.** AJAX and Unit Testing - it's time to mingle. *Jim Plush's Programming Paradise.* [Online] 13. 02. 2006 [Zitat vom: 11. 03. 2010] http://www.litfuel.net/plush/?postid=117.

[PKS00] **Pol Martin, Koomen Tim, Spillner Andreas. 2000.** *Management und Optimierung des Testprozesses: ein praktischer Leitfaden für Testen von Software, mit TPI und TMap.* Heidelberg : dpunkt.verlag, 2000. ISBN 3-932-58865-7.

[Pro04] **Protogerakis, Michael. 2004.** Informatik im Maschinenbau. [Online] 16. 01. 2004 [Zitat vom: 16. 03. 2010] http://www.zlw-ima.rwth-aachen.de/lehre/vorlesungen_uebungen/informatik2/download/referat_teststrategien.pdf.

[Qin09] **Qing, Yu Hong. 2009.** Hong Qing Yu's Home Page. [Online] 01. 07. 2009 [Zitat vom: 02. 03. 2010] http://www.cs.le.ac.uk/people/hqy1/.

[Rau05] **Rausch, Andreas. 2005.** AG Softwarearchitektur. [Online] 16. 02 2005. [Zitat vom: 09. 02. 2010] http://agrausch.informatik.uni-kl.de/publikationen/repository/journal/jour004/Goal.pdf.

[ReP99] **Rechenberg Peter, Popberger Gustav. 1999.** *Informatik-Handbuch 2., aktualisierte und erweiterte Auflage.* München Wien : Carl Hanser Verlag, 1999. ISBN 3-446-19601-3.

[Ric06] **Richter, Jörg. 2006.** DeepaMehta -- Software Engineering. [Online] 06. 10. 2006 [Zitat vom: 16. 01. 2010] http://www.deepamehta.de/docs/softwareengineering.html.

[Ruf05] **Ruff, Stefan. 2005.** WebTest_InANutshell Infometis. [Online] 13. 01. 2005 [Zitat vom: 13. 01. 2010] https://svn.canoo.com/trunk/webtest-book/german/WebTest_InANutshel-Infometis.pdf.

[Sar07] **Sardan. 2007.** AjaxDaddy - Ajax Examples in Action. [Online] 2007. [Zitat vom: 25. 03. 2010] http://www.ajaxdaddy.com.

[Sca09] **Scarce. 2009.** Datei:Test.png - Wikipedia. [Online] 25. 07. 2009 [Zitat vom: 05. 02. 2010] http://de.wikipedia.org/wiki/Datei:Test.png.

[Sei10] **Seidler, Kai Oswald. 2010.** apache friends - apache, mysql, php und perl installation leicht gemacht. [Online] Apache Friend, 09. 03. 2010 [Zitat vom: 09. 03. 2010] http://www.apachefriends.org/de/index.html.

[SBS06] **Sneed Harry, Baumgartner Manfred, Seid Richard. 2006.** *Der Systemtest - Anforderungsbasiertes Testen von Software-Systemen.* München/Wien : Carl Hanser, 2006. ISBN 3-446-40793-6.

[Spi05] **Spillner Andreas, Tilo Linz. 2005.** *Basiswissen Softwaretest.* Heidelberg : dpunkt-Verl., 2005.

[Spi07] **—. 2007.** *Basiswissen Softwaretest.* Heidelberg: dpunkt Verl., 2007.

[Ste05] **Stefan Münz, Wolfgang Nefzger. 2005.** *HTML Handbuch.* Poing : Franzis-Verlag, 2005. ISBN 3-7723-6654-6.

[Tha96] **Thaller, Georg Erwin. 1996.** *ISO 9001, Software-Entwicklung in der Praxis.* Hannover: Verlag Heinz Heise GmbH & Co KG, 1996. ISBN 3-88229-080-3.

[Tha02] **—. 2001.** *ISO 9001:2000 Software-Entwicklung in der Praxis.* Hannover: Verlag Heinz Heise GmbH & Co KG, 2001. ISBN 3-88229-189-3.

[Tha00] **—. 2000.** *Software-Qualität, Der Weg zu Spitzenleistungen in der Software-Entwicklung.* Berlin: VDE Verlag, 2000. ISBN 3-8007-2494-4.

[Tha03] **—. 2002.** *Software-Test. Verifikation und Validation.* Hannover: Heise, 2002. ISBN 3-88229-198-2.

[TAS10] **The Apache Software Foundation. 2010.** Welcome! - The Apache Software Foundation. [Online] The Apache Software Foundation, 02. 03. 2010 [Zitat vom: 09. 03. 2010] http://www.apache.org/.

[TPG10] **The PHP Group.** PHP: Hypertext Preprocessor. [Online] [Zitat vom: 16. 03. 2010] http://www.php.net/.

[Tho10] **ThoughtWorks.** Selenium web application testing system. [Online] [Zitat vom: 01. 03. 2010] http://seleniumhq.org/.

[W3C10] **W3C.** World Wide Web Consortium (W3C). [Online] [Zitat vom: 05. 02. 2010] http://www.w3.org/.

[Wal01] **Wallmüller, Ernst. 2001.** *Software-Qualitätsmanagement in der Praxis.* München Wien: Carl Hanser Verlag, 2001. ISBN 3-446-21367-8.

[Wes06] **Westphal, Frank. 2006.** *Testgetriebene Entwicklung mit JUnit & FIT.* Heidelberg: dpunkt Verlag, 2006. ISBN 3-89864-220-8.

[Wik04] **Wikipedia. 2010.** Apache Ant – Wikipedia. [Online] Wikipedia, 18. 02. 2010 [Zitat vom: 10. 03. 2010] http://de.wikipedia.org/wiki/Apache_Ant.

[Wik05] **—.** Document Object Model - Wikipedia. [Online] Wikipedia. [Zitat vom: 05. 02. 2010] http://de.wikipedia.org/wiki/Document_Object_Model.

[Wik06] **—. 2009.** Dynamisches Software-Testverfahren - Wikipedia. [Online] Wikipedia, 19. 11. 2009 [Zitat vom: 09. 02. 2010] http://de.wikipedia.org/wiki/Dynamisches_Software-Testverfahren.

[Wik01] **—.** http://de.wikipedia.org/wiki/Softwaretest. *Softwaretest - Wikipedia.* [Online] Wikipedia. [Zitat vom: 12. 01. 2010] http://de.wikipedia.org/wiki/Softwaretest.

[Wik07] **—.** Liste von Modultest-Software. [Online] Wikipedia. [Zitat vom: 01. 03. 2010] http://de.wikipedia.org/wiki/Liste_von_Modultest-Software.

[Wik08] **—. 2010.** Plug-in - WIkipedia. [Online] Wikipedia, 26. 02. 2010 [Zitat vom: 04. 03. 2010] http://de.wikipedia.org/wiki/Plug-in.

[Wik03] **—.** Programmierschnittstelle - Wikipedia. [Online] Wikipedia. [Zitat vom: 25. 01. 2010] http://de.wikipedia.org/wiki/Programmierschnittstelle.

[Wik02] **—.** Webanwendung - Wikipedia. [Online] [Zitat vom: 12. 01. 2010] http://de.wikipedia.org/wiki/Web-Anwendung.

[Wik10] **—. 2010.** Webserver - Wikipedia. [Online] Wikipedia, 08. 03. 2010 [Zitat vom: 09. 03. 2010] http://de.wikipedia.org/wiki/Webserver.

[Win04] **Winkler, Dietmar. 2004.** Quality Software Engineering (QSE) Research. [Online] 04. 02. 2004 [Zitat vom: 08. 02. 2010] http://qse.ifs.tuwien.ac.at/courses/skriptum/download/11P_Auto_wid_20040204.pdf.

[Wir04] **Wirsing, Martin. 2004.** Programmierung und Softwaretechnik (PST). [Online] 21. 12. 2004 [Zitat vom: 26. 01. 2010] http://www.pst.informatik.uni-muenchen.de/lehre/WS0405/mse/folien/D2-BlackWhite6p.pdf.

[Wir10] **Wirth & Horn - Informationssysteme GmbH - www.wirth-horn.de.** Der Wörterbuch-Verlag Nr.1: Langenscheidt:. [Online] [Zitat vom: 09. 01. 2010] http://www.langenscheidt.de.

9. Anhang

9.1 Testfall Java Code

```
package com.testscripts;
package com.testscripts;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import com.thoughtworks.selenium.DefaultSelenium;
import com.thoughtworks.selenium.SeleneseTestCase;
import com.thoughtworks.selenium.Selenium;
public class testcase1 extends SeleneseTestCase {
private Selenium selenium;
@BeforeClass
public void setUp() throws Exception {
selenium = new DefaultSelenium("serkan.avci.com", 4444, "*firefox",
"http://www.orf.at");
selenium.start();}
@Test
public void testTestcase_5() throws Exception {
selenium.open("file:///I:/Diplomarbeit/test_seite.html");
selenium.refresh();
selenium.runScript("obj=document.getElementById('test');inn=document.g
etElementById('test').innerHTML = 'Neuer
Name';obj.setAttribute('href','test_seite.html');");
verifyEquals("Neuer Name", selenium.getText("//a"));
selenium.captureEntirePageScreenshot("c:\\test.png",
"background=#ffffff");
selenium.click("link=Neuer Name");
String text = selenium.getText("//a");
verifyEquals("Test", selenium.getText("//a"));
System.out.println(text);
selenium.captureEntirePageScreenshot("c:\\test1.png",
"background=#ffffff");}
@Test
public void testTestcase_6() throws Exception {
selenium.open("http://www.ajaxdaddy.com/demo-bsn-autocomplete.html");
selenium.selectFrame("DaddyDemo");
selenium.typeKeys("testinput", "james");
for (int second = 0;; second++) {
if (second >= 60) fail("timeout");
try { if
(selenium.isElementPresent("//*[@id=\"as_ul\"]/li/a/span[3]"))
break; } catch (Exception e) {}
Thread.sleep(1000);}
selenium.click("//*[@id=\"as_ul\"]/li/a/span[3]");
String name = selenium.getText("//*[@id=\"as_ul\"]/li/a/span[3]");
System.out.println(name);
selenium.type("testinput", name);}
@Test
public void testTestcase_7() throws Exception {
selenium.open("http://www.ajaxdaddy.com/demo-bsn-autocomplete.html");
assertEquals("BrandSNew Autocomplete Examples", selenium.getTitle());
selenium.click("link=Next");
selenium.waitForPageToLoad("30000");
```

```
assertEquals("SlideShow Viewer Examples", selenium.getTitle());
selenium.selectFrame("DaddyDemo");
selenium.click("//*[@src=\"images/img1_t.jpg\"]");
selenium.click("//map[@id='PhotoViewerToolbarMap']/area[4]");
for (int second = 0;; second++) {
if (second >= 60) fail("timeout");
try { if
("2/3".equals(selenium.getText("//*[@id=\"PhotoViewerTime\"]")))
break; } catch (Exception e) {}
Thread.sleep(1000);}
selenium.captureEntirePageScreenshot("c:\\pic1.png",
"background=#ffffff");
selenium.click("//map[@id='PhotoViewerToolbarMap']/area[4]");
for (int second = 0;; second++) {
if (second >= 60) fail("timeout");
try { if
("3/3".equals(selenium.getText("//*[@id=\"PhotoViewerTime\"]")))
break; } catch (Exception e) {}
Thread.sleep(1000);}
selenium.captureEntirePageScreenshot("c:\\pic2.png",
"background=#ffffff");
selenium.click("//map[@id='PhotoViewerToolbarMap']/area[2]");
for (int second = 0;; second++) {
if (second >= 60) fail("timeout");
try { if
("2/3".equals(selenium.getText("//*[@id=\"PhotoViewerTime\"]")))
break; } catch (Exception e) {}
Thread.sleep(1000);}
selenium.captureEntirePageScreenshot("c:\\pic3.png",
"background=#ffffff");
selenium.click("//map[@id='PhotoViewerToolbarMap']/area[2]");
for (int second = 0;; second++) {
if (second >= 60) fail("timeout");
try { if
("1/3".equals(selenium.getText("//*[@id=\"PhotoViewerTime\"]")))
break; } catch (Exception e) {}
Thread.sleep(1000);}
selenium.captureEntirePageScreenshot("c:\\pic4.png",
"background=#ffffff");
selenium.click("//map[@id='PhotoViewerToolbarMap']/area[5]");
for (int second = 0;; second++) {
if (second >= 60) fail("timeout");
try { if
("3/3".equals(selenium.getText("//*[@id=\"PhotoViewerTime\"]")))
break; } catch (Exception e) {}
Thread.sleep(1000);}
selenium.captureEntirePageScreenshot("c:\\pic5.png",
"background=#ffffff");
selenium.click("//map[@id='PhotoViewerToolbarMap']/area[7]");
selenium.waitForPopUp("", "30000");
selenium.selectPopUp("");
selenium.captureEntirePageScreenshot("c:\\pic6.png",
"background=#ffffff");
String title = selenium.getTitle();
System.out.println(title);
assertEquals("img3.jpg (JPEG-Grafik, 420x315 Pixel)",
selenium.getTitle());
selenium.close();}
@AfterClass public void tearDown(){selenium.stop();}
}
```

Abbildung 65: Testfall Skript *„testcase1.java“*

```
package com.testscripts;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import com.thoughtworks.selenium.DefaultSelenium;
import com.thoughtworks.selenium.SeleneseTestCase;
import com.thoughtworks.selenium.Selenium;
public class testcase2 extends SeleneseTestCase {
private Selenium selenium;
@BeforeClass
public void setUp() throws Exception {
selenium = new DefaultSelenium("serkan.avci.com",
4444, "*firefox", "http://www.orf.at");
selenium.start();}
@Test
public void testTestcase_7() throws Exception {
selenium.open("http://www.ajaxdaddy.com/demo-bsn-autocomplete.html");
assertEquals("BrandSNew Autocomplete Examples", selenium.getTitle());
selenium.click("link=Next");
selenium.waitForPageToLoad("30000");
assertEquals("SlideShow Viewer Examples", selenium.getTitle());
selenium.selectFrame("DaddyDemo");
selenium.click("//*[@src=\"images/img1_t.jpg\"]");
selenium.click("//map[@id='PhotoViewerToolbarMap']/area[4]");
for (int second = 0;; second++) {
if (second >= 60) fail("timeout");
try { if
("2/3".equals(selenium.getText("//*[@id=\"PhotoViewerTime\"]")))
break; } catch (Exception e) {}
Thread.sleep(1000);}
selenium.click("//map[@id='PhotoViewerToolbarMap']/area[4]");
for (int second = 0;; second++) { if (second >= 60) fail("timeout");
try { if
("3/3".equals(selenium.getText("//*[@id=\"PhotoViewerTime\"]")))
break; } catch (Exception e) {}
Thread.sleep(1000);}
selenium.click("//map[@id='PhotoViewerToolbarMap']/area[2]");
for (int second = 0;; second++) { if (second >= 60) fail("timeout");
try { if
("2/3".equals(selenium.getText("//*[@id=\"PhotoViewerTime\"]")))
break; } catch (Exception e) {}
Thread.sleep(1000);}
selenium.click("//map[@id='PhotoViewerToolbarMap']/area[2]");
for (int second = 0;; second++) { if (second >= 60) fail("timeout");
try { if
("1/3".equals(selenium.getText("//*[@id=\"PhotoViewerTime\"]")))
break; } catch (Exception e) {}
Thread.sleep(1000);}
selenium.click("//map[@id='PhotoViewerToolbarMap']/area[5]");
for (int second = 0;; second++) { if (second >= 60) fail("timeout");
try { if
("3/3".equals(selenium.getText("//*[@id=\"PhotoViewerTime\"]")))
break; } catch (Exception e) {}
Thread.sleep(1000);}
selenium.captureEntirePageScreenshot("c:\\pic5.png",
"background=#ffffff");
selenium.click("//map[@id='PhotoViewerToolbarMap']/area[7]");
selenium.waitForPopUp("", "30000");
selenium.selectPopUp("");
String title = selenium.getTitle();
System.out.println(title);
```

```
assertEquals("img3.jpg (JPEG-Grafik, 420x315 Pixel)",
selenium.getTitle());
selenium.close();}
@AfterClass
public void tearDown(){selenium.stop();}
}
```

Abbildung 66: Testfall Skript *„testcase2.java“*

9.2 Log Datei

```
Test build from  Fri Mar 26 11:14:57 CET 2010

Clean
Buildfile: build.xml

clean:

BUILD SUCCESSFUL
Total time: 0 seconds
Start Compile
Buildfile: build.xml

compile:
    [javac] Compiling 14 source files to
            /home/serkan/test_ordner/bin

BUILD SUCCESSFUL
Total time: 1 second
Buildfile: build.xml

compress:
      [jar] Building jar: /home/serkan/test_ordner/lib/Project.jar

BUILD SUCCESSFUL
Total time: 0 seconds
Start Tests
Buildfile: build.xml

test:

testAllDelete:
     [echo] === Clean:

clean:
     [echo] === Testall:

testAll:
     [echo] === Tests:
Testcase1:
    [junit] Running com.testscripts.testcase1
    [junit] Tests run: 3, Failures: 0, Errors: 0, Time elapsed: 14.937
sec
Testcase2:
    [junit] Running com.testscripts.testcase2
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 10.019
sec
     [echo] === Reports:
[junitreport] Processing /home/serkan/test_ordner/TESTS-TestSuites.xml
to /tmp/null1985086376
[junitreport]  Loading  stylesheet  jar:file:/usr/share/java/ant/ant-
```

```
junit-
1.7.0.jar!/org/apache/tools/ant/taskdefs/optional/junit/xsl/junit-
frames.xsl
[junitreport] Transform time: 755ms
[junitreport] Deleting: /tmp/null1985086376

BUILD SUCCESSFUL
Total time: 26 seconds
```

Tabelle 51: Log Datei