Orientation in Objects

Grails mit Selenium testen

) Schulung)

) Beratung)

) Entwicklung)

) Artikel)

AUTOR

Sven Lange
Orientation in Objects GmbH

Veröffentlicht am: 26.5.2011

SELENIUM RC FÜR GRAILS ANWENDUNGEN

Bei der Umsetzung größerer Softwareprojekte ist es wichtig, dass man sich Gedanken darüber macht, wie und womit getestet wird. Damit verringert man das Risiko, dass Modifikationen oder Erweiterungen am Projekt bestehende Funktionalität negativ beeinträchtigt und dadurch schwere Folgenkosten in Kauf genommen werden müssen.

Hat man sich dazu entschieden automatisierte funktionale Tests durchführen zu wollen, geht es an die Evaluierung und Auswahl passender Frameworks. In diesem Artikel wird gezeigt welche Frameworks sich derzeit für das automatisierte funktionale Testen von Grails Anwendungen anbieten. Die Test Frameworks WebTest und Selenium werden näher vorgestellt, wobei Selenium besonderes Augenmerk geschenkt wird, indem das Plugin näher untersucht und der Testablauf beschrieben wird.

GRUNDLAGEN

FUNKTIONALES TESTEN

Unter funktionalen Tests werden hier Black-Box Tests verstanden. Die genauen internen Abläufe zwischen verschiedenen Systemkomponenten sind nicht interessant. Es wird versucht die geforderten fachlichen Funktionalitäten des Gesamtsystems zu verifizieren. Bei einer Web-Anwendung wird die Verifizierung über die grafische Oberfläche, also in einem Web-Browser, durchgeführt.

Wichtig ist, dass die Tests oder Teile davon immer wieder automatisiert wiederholt werden können, damit Fehler, die durch Modifikationen hervorgerufen wurden, schnell gefunden und behoben werden können.

GRAILS EVENT SYSTEM

Das Grails Framework basiert auf dem Gant Build System [Gant]. Hierbei handelt es sich um eine Ant-Abstraktion auf Basis von Groovy. Mit Gant hat man also die Mächtigkeit des Ant-Build Systems und zusätzlich die Mächtigkeit der Groovy Skript Sprache vereint. Die auf der Konsole aufgerufenen Grails Commands (z.B. grails run-app) werden von dem Gant Build System ausgeführt. Diese Grails Commands sind also Groovy Skripte. Eine besondere Form dieser Groovy Skripte ist das _Events.groovy Skript. Über dieses Skript kann man Event Handler beim Grails Event System registrieren. Grails schaut an den folgenden Orten im Dateisystem nach den _Events Skripten:

- USER_HOME/.grails/scripts - Benutzerspezifische Event Handler
- PROJECT_HOME/scripts - Anwendungsspezifische Event Handler
- PLUGINS_HOME/*/scripts - Pluginspezifische Event Handler
- GLOBAL_PLUGINS_HOME/*/scripts - Event Handler von globalen Plugins

AUSLÖSEN EINES EVENTS

Bevor hier beschrieben wird, wie auf ein auftretendes Event reagiert werden kann, soll vorweg noch gezeigt werden, wie Events erzeugt werden.

Um in einem Skript ein Event auslösen zu können, muss das Grails interne Skript _GrailsEvents inkludiert werden. Wurde es inkludiert, wird bei jedem Target-Aufruf am Anfang und Ende automatisch ein Event generiert. Der Eventname setzt sich aus dem großgeschriebenen Namen des Targets und der Endung Start oder End zusammen. Das folgende Beispiel Skript würde also die Events MacheEtwasStart und MacheEtwasEnd auslösen.

```
includeTargets << grailsScript("_GrailsEvents")
target(macheEtwas: "Beschreibung des Targets.") {
 // ...
}
```

Beispiel 1: Grails Gant Skript löst Event automatisch aus

Zudem können auch Events explizit ausgegeben werden. Dies kann mit dem Aufruf der Event-Closure gemacht werden, wie im folgenden Beispiel gezeigt wird. Dort wird ein Event mit dem Namen IchBinEinEvent ausgegeben. Dieses Event besitzt eine Liste von Zeichenketten als Argument.

```
includeTargets << grailsScript("_GrailsEvents")
event("IchBinEinEvent", ["Hier", "werden", "Argumente",
"mitgegeben"])
```

Beispiel 2: Explizit ausgelöstes Event in Grails Gant Skript

REAGIEREN AUF EVENTS

Wie vorweg schon beschrieben, müssen Event Handler in einem _Event.groovy Groovy Skript definiert werden. Man definiert eine Closure nach einem festgelegten Namensmuster. Der Name startet mit dem kleingeschriebenen Wort event und endet mit dem Eventnamen. Das folgende Code-Beispiel zeigt einen Event Handler für ein Event mit dem Namen IchBinEinEvent und gibt dessen Argumente auf der Konsole aus.

```
includeTargets << grailsScript("_GrailsEvents")
eventIchBinEinEvent = {arguments ->
 arguments.each {
  println it
 }
}
```

Beispiel 3: Event Handler für Event mit dem Namen IchBinEinEvent

GRAILS TEST SYSTEM

Im Folgenden sollen kurz einige Begriffe aus dem Grails Test System vorgestellt werden.

Es gibt insgesamt vier Testphasen. Eine Phase gibt an in welchem Zustand sich die Grails Anwendung gerade befindet. Abhängig von der Phase wird die zu testende Grails Anwendung in einen bestimmten Zustand versetzt, bevor die Tests gestartet werden. Ebenfalls abhängig von der Phase werden nach dem Testablauf abschließende Aufräumarbeiten durchgeführt.

Testphase	Beschreibung
unit	Keine Vorbereitungen oder abschließenden Aufräumarbeiten nötig.
integration	Vor dem Test werden ein Persistenz- und ein Mock-Anwendungskontext bereitgestellt, die abschließend wieder eliminiert werden.
functional	Vor dem Test wird ein Persistenz- und ein vollwertiger Anwendungskontext bereitgestellt, die abschließend wieder eliminiert werden.
other	Vom Grails Framework wird nichts vorbereitet oder abschließend durchgeführt.

Tabelle 1: Grails Testphasen

Weiterhin gibt es sogenannte Testtypen (dt. für types). Testtypen geben den Mechanismus vor, wie oder womit getestet werden soll. Grails bringt von Haus aus den JUnit4GrailsTestType für JUnit 4 Tests mit. Plugins können Grails um weitere Testtypen erweitern, wie es das Spock [SpoP] oder das Selenium Plugin [SelP] tun. Weitere Details hierzu gibt es im Kapitel Testablauf oder der Grails Referenz Dokumentation [GraRD].

TEST FRAMEWORKS FÜR GRAILS ANWENDUNGEN

Es gibt bereits einige Grails Plugins, die von einem Entwicklungsteam aufgegriffen wurden und ins Projekt schnell integriert werden können. Die Plugins enthalten normalerweise einige Grails Skripte, die leere Testdateien generieren können. Die generierten Dateien ermöglichen es schnell, einfach und basierend auf Best Practices neue Tests zu schreiben. Insgesamt bieten einem diese Plugins also die Einsparung von Zeit für die Integration eines Test-Frameworks und der Testerstellung.

WEBTEST

Das erste und vielleicht auch bekannteste Grails Plugin für das Erstellen und Durchführen von funktionalen Tests ist das WebTest Plugin [WebP]. Es ist momentan in der Version 3.0.0 erhältlich und basiert auf dem freien Open-Source Test Framework WebTest in der Version 3.0.

WebTest beinhaltet HTMLUnit, welches einen HTML Browser simuliert, mit dem auf schnelle Weise Anfragen an einen Server gesendet werden, wie in der folgenden Abbildung dargestellt.

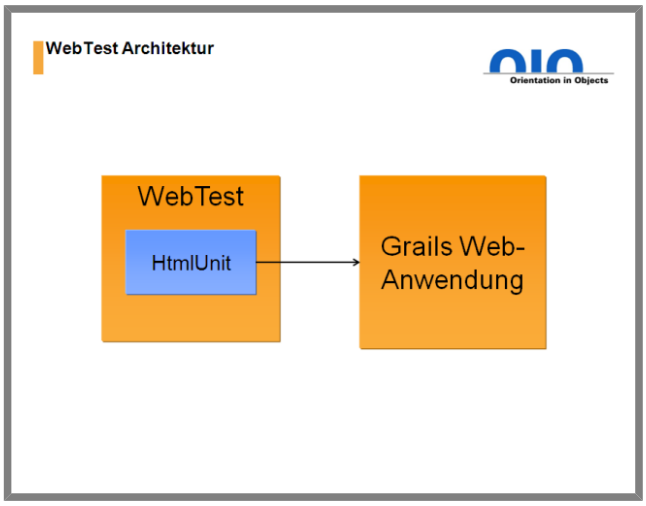

Abbildung 1: Canoo WebTest Architektur

Die Tests können in XML oder Groovy verfasst werden. Es ist jedoch vorteilhafter die Tests in Groovy zu verfassen, da die Tests dann leichter lesbar sind und man zusätzlich alle Groovy Sprachfeatures zur Verfügung hat.

Wie bereits angedeutet ist die Test-Ausführung mit WebTest sehr schnell. Zudem bekommt man einen sehr detailreichen Testbericht, der gerade bei Fehlerfällen ausgesprochen hilfreich sein kann.

Nicht von Vorteil ist die Tatsache, dass WebTest nur einen Browser simuliert und somit nicht das Verhalten eines realen Browsers nachstellt, schon gar nicht das Verhalten verschiedener Browser. Ebenfalls negativ ist die wenig umfangreiche JavaScript Unterstützung des Frameworks. Stark JavaScript-lastige Seiten lassen sich nur sehr mühsam mit WebTest testen. [WebKC] [Hüt10] Hier trumpfen andere Test Frameworks auf.

SELENIUM

Ein weiteres sehr populäres Grails Plugin für funktionales Testen ist das Selenium RC Plugin [SelP]. Es basiert auf dem freien Open-Source Test Framework Selenium und ist derzeit in der Version 1.0.2 verfügbar.

Kurz gesagt handelt es sich bei Selenium um eine Fernsteuerung für Browser. Die Browser werden per JavaScript gesteuert, welches in ein HTML Inline-Frame eingebettet wird. Dieser Trick ermöglicht das realitätsnahe Testen mit den meisten am Markt gängigen Browsern.

Wenn man von Selenium Core spricht meint man den Teil, der als JavaScript im Browser sitzt und dessen Steuerung übernimmt. Damit ist das Interagieren mit der eigentlichen Web-Anwendung gemeint. Angetrieben und in den Browser injiziert wird der Core von Selenium RC, wobei Selenium RC nur einen Mittelsmann für die Tests darstellt. Die eigentlichen Tests, die mit Groovy verfasst werden können, werden von Selenium RC in Form von Selenium Commands empfangen und an den Core weitergeleitet. Die folgende Abbildung versucht den beschriebenen Aufbau zu verdeutlichen.

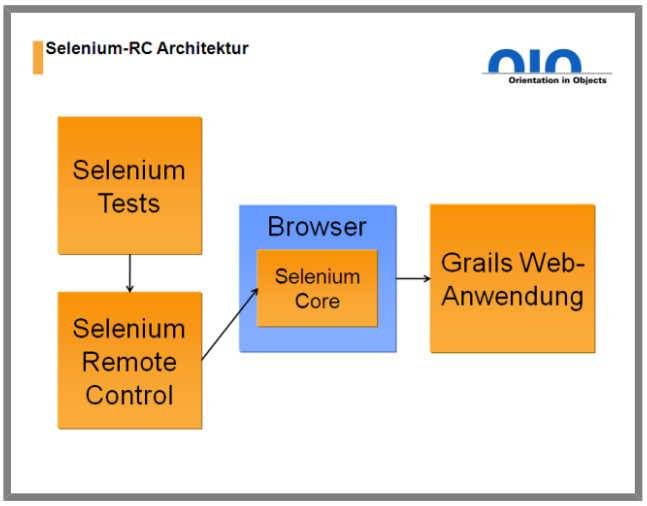

Abbildung 2: Selenium Architektur

Ein großer Vorteil in diesem Szenario ist, dass man für die Testskriptsprache eine große Auswahl hat. Die Tests können bspw. in Groovy, Java, C# oder Ruby verfasst werden. Damit stehen Schleifen, If-Satements (Konditionale Operatoren) und weitere Sprachfeatures zur Verfügung, die es erlauben aufwändigere Tests leichter zu realisieren.

SELENIUM COMMANDS

Selenium Commands, auch Selenese genannt, stellen die API des Test-Frameworks dar. Die API dient als Sprache zum definieren von Tests. Die Commands sagen Selenium was getan werden soll und werden in die drei Kategorien Action, Accessor und Assertation unterteilt.

Actions führen Befehle wie beispielsweise das Klicken eines Buttons aus und verändern somit den Status einer Anwendung. Vielen Actions kann man das Suffix andWait anhängen, wodurch Selenium auf das Laden der nächsten Seite wartet.

Accessors versetzen jemanden in die Lage den aktuellen Status einer Seite zu speichern, beispielsweise den Titel einer Seite mit dem Command storeTitle.

Assertations prüfen den aktuellen Status einer Seite der Anwendung auf Korrektheit. Es gibt die drei Assertation-Modi assert, verify und waitFor, mit denen man z.B. auf einen Text mit assertText, verifyText oder waitForText prüfen kann. Dabei sorgt eine negative Rückgabe von assert zum Fehlschlagen und Abbruch eines Tests. Bei verify wird im Negativfall nur ein Hinweis geloggt und der Test weiter ausgeführt. waitFor eignet sich speziell für Anwendungen, die viel JavaScript einsetzen.

GRID

Im Laufe eines Projekts nimmt die Anzahl der Tests üblicherweise stetig zu, so dass auch die Laufzeit eines kompletten Testdurchlaufs stetig immer länger dauert. Grade bei Selenium Tests, die über Selenium RC laufen, können unerträglich lange Laufzeiten entstehen.

Abhilfe schaffen kann hier das Projekt Selenium Grid [SelG]. Die Selenium Tests werden nicht mehr sequenziell über eine Selenium RC Instanz ausgeführt, sondern parallel über mehrere Selenium RC Instanzen wie in der folgenden Abbildung dargestellt.

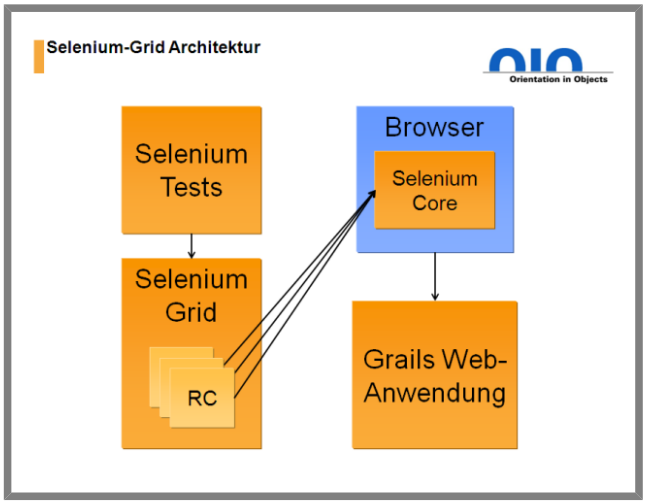

Abbildung 3: Selenium Grid Architektur

Dieses Feature ist jedoch noch nicht Bestandteil des Grails Plugins. Immerhin ist es auf einer Todo-Liste vermerkt.

WEITERE

Selbstverständlich gibt es noch weitere Test Frameworks wie z.B. Webdriver [WebdP], Geb [GebP] oder G-Func [GFuP]. Gerade Geb in Verbindung mit Spock ist derzeit in vieler Munde. Ob es sich schon bald breiter Verwendung erfreuen kann, bleibt abzuwarten. Sehr ruhig hingegen ist es in letzter Zeit um G-Func geworden. Keines dieser Frameworks wird im Rahmen dieses Artikels näher betrachtet.

AUSWAHL EINES FRAMEWORKS

Aus der vorrangegangenen Vorstellung der beiden Test-Frameworks WebTest und Selenium kann Folgendes abgeleitet werden:

Der Einsatz von WebTest kommt dann in Frage, wenn es sich bei der Anwendung nicht um eine stark JavaScript lastige Anwendung handelt. Auch darf cross-browser Funktionalität keine große Rolle spielen.

Zu Selenium sollte man greifen, wenn es darum geht Anwendungen zu testen, die viel JavaScript in ihrer Benutzeroberfläche verwenden. Zudem verzeiht Selenium eher nicht valides HTML und ist cross-browser fähig.

Da heutzutage die meisten Web-Anwendungen über umfangreiche JavaScript Elemente und Frameworks verfügen, soll im folgenden Kapitel gezeigt werden, wie man mit dem dafür geeigneten Testframework Selenium Tests schreiben kann.

FUNKTIONALE TESTS MIT SELENIUM RC

Nachfolgend wird gezeigt, wie man mit dem Grails Selenium RC Plugin in der Version 1.0.2 funktionale Tests schreiben kann. Bei der verwendeten Plugin Version handelt es sich um das sechste Release, welches Anfang Juni 2010 erschienen ist [SelR]. Im Anschluss daran wird der Testablauf detailiert beschrieben, um einen technischen Einblick zu bekommen, was alles im Hintergrund bei der Testdurchführung abläuft.

ERSTELLUNG VON TESTFÄLLEN

Das Plugin enthält zwei Grails Skripte, die auf der Konsole ausgeführt werden können und dabei leere Musterdateien für das Testen generieren.

Das erste Skript lautet grails create-selenium-test. Mit ihm kann man eine leere Testdatei anlegen, in die dann anschließend Testfälle eingebettet werden können. Das folgende Beispiel zeigt eine generierte Testdatei.

```
package example
import grails.plugins.selenium.*
import org.junit.*
import static org.junit.Assert.*
import static org.hamcrest.Matchers.*
@Mixin(SeleniumAware)
class BeispielTests {
   @Before void setUp() {
   }
   @After void tearDown() {
   }
   @Test void eineTestMethode() {
      selenium.open 'http://localhost:8080/appName'
      selenium.clickAndWait 'MeinButton'
   }
}
```

Beispiel 4: Mit create-selenium-test generierte Testdatei

Durch die Groovy Mixin-Annotation der Testklasse kann auf die aktuelle Selenium Instanz zugegriffen werden. Bei der bereitgestellten Instanz handelt es sich allerdings nicht um ein Standard Selenium Objekt [StdSO]. Die hier zur Verfügung gestellte Instanz wurde um einige nützliche Methoden erweitert. Detaillierte Informationen hierzu können der Plugin Dokumentation [SelPD] entnommen werden.

Das zweite mitgelieferte Skript heißt create-selenium-page. Mit diesem Skript ist es möglich eine leere Ausgangsklasse für ein Page Objekt zu generieren. Page Objekte beruhen auf der Idee des Page Object Patterns [POP] und sind dann sehr hilfreich, wenn man viele Testklassen in einem Projekt hat. Ein Page Objekt repräsentiert genau eine Seite in einer Web-Anwendung. Testklassen sprechen nicht mehr direkt mit den Web-Seiten einer Anwendung, sondern nur noch über entsprechende Page Objekte, sofern diese vorhanden sind. Dies hat den Vorteil, dass man nicht mehr zahlreiche Testklassen anpassen muss, sobald sich eine Seite der Anwendung ändert, sondern nur noch eine Stelle und zwar im Page Object.

DER TESTABLAUF

Der Testablauf einer Grails-Anwendung gestaltet sich wie folgt: Nachdem das Grails Command test-app auf der Konsole abgesetzt wurde, wird das Einstiegs-Grails-Skript (grails/grails.bat) im bin-Verzeichnis von Grails ausgeführt. Dieses stößt ein anderes Skript mit dem Namen startGrails im bin-Verzeichnis an. Dies hat zur Folge, dass die Java Klasse GrailsStarter ausgeführt wird. Sie ist verantwortlich für das Festlegen etlicher Systemvariablen und dem Aufbereiten eines ClassLoaders. Im Anschluss daran wird die Java Klasse GrailsScriptRunner ausgeführt. Diese Klasse hat zur Aufgabe die Benutzereingaben auf der Kommandozeile zu verwalten. Das heißt, es wird identifiziert, welches Skript auszuführen ist und welche Parameter an die entsprechenden Skripte weitergeleitet werden müssen.

Um alle Selenium Tests auszuführen eignet sich nachfolgender Befehl auf der Kommandozeile. Dabei gibt functional die auszuführende Testphase und selenium den auszuführenden Testtypen an:

```
grails test-app functional:selenium
```

Beispiel 5: Kommandozeilenbefehl um alle Selenium Tests auszuführen

EVENTS WÄHREND DES TESTABLAUFS

Führt man das Grails-interne Test Skript aus, werden von diesem Events ausgelöst. Im Folgenden werden die Events aufgezeigt, auf die das Selenium-RC Plugin bei einem Testdurchlauf reagiert. Das Plugin registriert noch weitere Event Handler, aber diese werden hier nicht betrachtet.

ALLTESTSSTART

Dieses Event signalisiert lediglich, dass Grails begonnen hat einen Testdurchlauf durchzuführen. Das Selenium Plugin reagiert darauf mit der Anmeldung des Selenium Test-Typen für die Testphase mit dem Namen functional.

TESTSUITESTART

Dieses Event tritt auf, wenn eine Gruppe von Tests durchgeführt wird. Das Event gibt über das mitgegebene Argument type bekannt, welchen Testtypen die auszuführenden Tests haben. Hat dieser Parameter den Wert selenium, werden Selenium und ein Browser gestartet. Desweiteren werden zwei weitere Event Handler registriert. Es wird der TestContextNotifier registriert, der den Remote Runner (Frame im Browser) darüber informiert, welcher Testfall und welcher Test momentan abgespielt werden. Bei dem zweiten Event Handler mit dem Namen ScreenshotGrabber handelt es sich um eine Instanz, die dafür sorgt, dass bei einem Fehlerfall ein Screenshot gemacht wird. Dieser kann bei der Aufdeckung des Fehlers hilfreich sein. Nachdem Selenium gestartet ist und die beiden neuen Event Handler registriert sind, ist der TestSuiteStart Event Handler mit seinen Aufgaben fertig und die Tests werden durchgeführt.

TESTSUITEEND

Sind alle Tests einer Testphase für einen Testtypen durchgelaufen, wird das Event TestSuiteEnd generiert. Bei diesem Event reagiert das Plugin mit dem Beenden von Selenium.

ABSCHLUSS DES TESTS

Nachdem Selenium beendet ist wird zum Abschluss noch ein JUnit Testbericht generiert. Dieser ist bei weitem nicht so detailreich wie Berichte von WebTest, gibt aber einen Überblick über das was passiert ist.

ZUSAMMENFASSUNG

Grails Kommandos, die über die Kommandozeile aufgerufen werden können, werden von Gant ausgeführt. Die Kommandos sind eigentlich Groovy Skripte und werden von der Klasse GrailsScriptRunner einschließlich aller Kommandozeilenparameter aufgerufen. Auf der Skript Ebene steht ein Event System bereit. Auf Events können Plugins mit eigenen Aktionen reagieren und so Abläufe in Grails modifizieren oder erweitern.

Man muss die Vor- und Nachteile der zur Verfügung stehenden Test Frameworks gut abwägen, bevor man sich für eines entscheidet. Neben WebTest und Selenium gibt es wie erwähnt noch weitere Test Frameworks, die bei einer Evaluierung berücksichtigt werden könnten.

Das Selenium-RC Plugin eignet sich zum funktionalen Testen von stark JavaScript-lastigen Web-Anwendungen. Es macht Gebrauch vom Grails Event System, indem es den Testtypen selenium der funktionalen Testphase hinzufügt und dafür sorgt, dass alle Selenium Tests von Selenium ausgeführt werden. Die Testdurchführung selbst ist recht langsam und die Testberichte nicht sehr detailreich.

AUSBLICK

Das nächste Major-Release Selenium 2 [Sel2] wird von vielen Testern sehnsüchtig erwartet. Selenium 2 wird Browser nicht mehr per JavaScript steuern, sondern über native Browserschnittstellen. Diese Ansteuerung wird durch die WebDriver API gekapselt. Dadurch werden die Testmöglichkeiten erweitert und Ausführungszeiten minimiert. Auch wird es möglich sein Selenium Tests entweder in einem Browser, oder wie bei WebTest mit HtmlUnit laufen zu lassen, was die Ausführungszeit weiter minimiert.

Sehr interessant wäre die Integration von Selenium Grid in das vorhandene Selenium RC Plugin oder in ein eigenständiges Plugin, damit Testausführungszeiten effektiv verringert werden können. Dies wurde aufgrund von Zeitmangel vom Autor des Selenium-RC Plugins bisher nicht umgesetzt, jedoch ist das Vorhaben geplant. Bis es soweit ist, sollte man Tests so erstellen, dass diese in keinster Weise voneinander abhängig sind. Man muss also aktuell nicht nur darauf zu achten, dass die Tests einzeln ablaufen, sonder auch darauf, dass sie nicht parallel ablaufen können.

REFERENZEN

- [WebKC] WebTest Key Characteristics
  (http://webtest.canoo.com/webtest/manual/keyCharacteristics.html)
- [CvsS] Canoo WebTest vs. Selenium
  Raible, Matt
  2007-05-31
  (http://raibledesigns.com/rd/entry/canoo_webtest_vs_selenium)
- [SRC] Selenium Remote Control
  Sanchez, Ivan
  2008-08-29
  (http://isanchez.net/2008/08/29/slides-from-my-talk-at-skillsmatter)
- [WebP] WebTest Plugin
  (http://grails.org/plugin/webtest)
- [Hüt10] Funktionales Testen von Rich Internet Applications
  Hüttermann, Michael
  JavaSPEKTRUM; 2008-05-01
- [Gant] Gant
  (http://gant.codehaus.org/)
- [SelP] Selenium RC Plugin
  (http://grails.org/plugin/selenium-rc)
- [WebdP] Webdriver Functional Testing Plugin
  (http://grails.org/plugin/webdriver)
- [GebP] Geb
  (http://geb.codehaus.org/)
- [GFuP] Grails Functional Testing Plugin
  (http://grails.org/plugin/functional-test)
- [GFuP] Grails Selenium RC Plugin Release
  (http://svn.grails-plugins.codehaus.org/changelog/grails-plugins/grails-selenium-rc/trunk)
- [StdSO] Interface Selenium
  (http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/com/thoughtworks/selenium/Selenium.html)
- [SelPD] selenium-rc - Reference Documentation
  (http://robfletcher.github.com/grails-selenium-rc/docs/manual/index.html)
- [POP] Page Object pattern
  Stewart, Simon M.
  2009-10-12
  (http://code.google.com/p/selenium/wiki/PageObjects)
- [Sel2] Selenium 2.0 and WebDriver
  (http://seleniumhq.org/docs/09_webdriver.html)
- [SelG] Selenium Grid
  (http://selenium-grid.seleniumhq.org/)
- [GraRD] Grails Referenz Dokumentation
  (http://grails.org/doc/1.3.5/)
- [SpoP] Grails Spock Plugin
  (http://www.grails.org/plugin/spock)