



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

TEST-DRIVEN DEVELOPMENT MIT RUBY ON RAILS

AM BEISPIEL DER O/ZB WEBAPPLIKATION

PROJEKTARBEIT

vorgelegt von

Briewig, Martin, B.Sc. (Matr.-Nr.: 43509)

Leibel, Michael, B.Sc. (Matr.-Nr.: 43674)

Betreuer: Prof. Dr. Frank Schaefer

Fachbereich: Informatik (Master)

Inhaltsverzeichnis

1	Einleitung	1
1.1	Aufgabenbeschreibung	1
2	Hintergrundwissen	3
2.1	Das o/ZB Projekt	3
2.2	Das Testsystem	3
2.3	Test-Driven Development	3
2.4	o/ZB Darlehensvertrag	3
3	Allgemeine Dokumentationen	4
3.1	Das korrigierte ER-Diagramm	4
3.2	Das korrigierte Relationenmodell	4
3.3	Deployment und Versionskontrolle	4
3.4	Datenbank Migration	9
4	Korrekturen, Bugfixes	10
4.1	WebImport	10
4.2	Punkteberechnung	10
5	Testdriven Development	11
5.1	Analyse	11
5.2	Implementierung	11
6	Neue Features	12
6.1	Dokumenten-Export	12
7	Ergebnis und Ausblick	13
7.1	Ergebnis	13
7.2	Ausblick	13
8	Anhang	14

Abbildungsverzeichnis	15
Tabellenverzeichnis	16
Listings	17

Abkürzungsverzeichnis

AP Arbeitspaket

o/ZB Ohne Zins Bewegung

RoR Ruby on Rails

SSH Secure Shell

1 Einleitung

Die ohne Zins Bewegung Stuttgart arbeitet nun seit einigen Jahren mit der Hochschule Karlsruhe zusammen an einer Webanwendung. Diese Webanwendung soll die Geschäftsprozesse der Bewegung unterstützen und verwalten. Im Laufe der letzten Jahre ist aus der ursprünglichen Idee eine komplexe Webanwendung entstanden, die nun gründlich auf Fehler überprüft und gegebenenfalls ausgebessert werden soll. Diese Aufgabe wird nun uns, Herr Briewig und Herr Leibel, anvertraut. Im Folgenden wird eine detaillierte Aufgabenstellung formuliert, an dieser sich diese Projektarbeit messen lassen wird.

1.1 Aufgabenbeschreibung

Die Aufgaben dieser Projektarbeit sind recht vielfältig. Sie lassen sich in 3 Phasen einteilen. Darüber hinaus gibt es für jede Phase ein Zeitlimit von etwa 3 Wochen.

1.1.1 Phase 1: Korrekturen, Dokumentieren

(Duedate: 02.05.2013)

Die erste Phase wird von den Korrekturen am ER-Diagramm und dem dazugehörigen Relationsmodell dominiert. Das ER-Diagramm weist noch ein paar Unstimmigkeiten auf, die im Laufe der Zeit entstanden sind. Diese gilt es zu bereinigen und die vorgenommenen Korrekturen in der Implementierung umzusetzen. Ziel ist ein ER-Diagramm, welches den nicht-historisierten Zustand darstellt. Dieses ER-Diagramm soll 1:1 in der Implementierung vorzufinden zu sein. Es soll vereinfacht dargestellt werden, lediglich die Primär- und Fremdschlüssel sollen den Entitäten als Zusatzinformationen dienen.

Neben den Korrekturen am ER-Diagramm sind Dokumentationen der Vorgehens- und Funktionsweise des Datenbank Migrationstools notwendig. Hierfür sind zwei Batch-Skripte angefertigt worden, deren Nutzungs- und Funktionsweisen zu dokumentieren sind.

Im Laufe der Zeit sind die verschiedensten Techniken zur Bereitstellung und Versionierung der o/zb Webanwendung verwendet worden. Es gilt nun, einen einheitlichen Vorgang zu definieren der für die nachfolgenden Projektgruppen verwendet werden soll. Basierend auf der anzufertigten Dokumentation, kann die Bereitstellung und Versionierung stetig verbessert werden.

Zusätzlich soll in dieser Phase ein kritischer Fehler bereinigt werden. Der WebImport, welcher für den Import der Kontobewegungen verantwortlich ist, arbeitet nicht richtig. Zur Zeit ist der Datei-Upload auf dem Testsystem nicht möglich. Darüber hinaus kommt es bei einer lokalen Testumgebung zu Laufzeitfehlern. Auch die im Webfrontend angezeigte Anzahl der importierten Datensätze ist nicht korrekt.

1.1.2 Phase 2: Korrektur der Punkteberechnung, Word Export

(Duedate: 23.05.2013)

1.1.3 Phase 3: Einrichtung Testdriven Development

(Duedate: 13.06.2013)

2 Hintergrundwissen

2.1 Das o/ZB Projekt

2.2 Das Testsystem

2.3 Test-Driven Development

2.3.1 Definition

2.3.2 Unit-Tests

2.3.3 Functional-Tests

2.3.4 Integration-Tests

2.4 o/ZB Darlehensvertrag

3 Allgemeine Dokumentationen

3.1 Das korrigierte ER-Diagramm

3.2 Das korrigierte Relationenmodell

3.3 Deployment und Versionskontrolle

In diesem Abschnitt werden die Begriffe *Deployment* und *Versionskontrolle* erläutert. In jedem Teilabschnitt wird die aktuelle Umsetzung beschrieben. Der letzte Teil dieses Abschnitts beschäftigt sich mit dem *Commit & Deployment Workflow*.

3.3.1 Deployment

In der Softwareentwicklung wird unter dem Begriff *Deployment* der Prozess zur *Bereitstellung* und *Verteilung* einer Software verstanden. Der Prozess ist von Software zu Software unterschiedlich, da z.B. die Konfiguration einer Software immer an die Umgebung angepasst werden muss, in der diese zum Einsatz kommt.

Im Falle der o/zb muss der Deployment Prozess die Bereitstellung der o/zb Webanwendung auf einem Server bewerkstelligen. Dies wirft die folgenden Fragen auf: Welcher Webserver wird eingesetzt? Wie gelangt die RoR Webanwendung auf den Server? Und wie arbeiten Anwendungsserver und Webserver auf dem Server zusammen? Diese Fragen werden die nun folgenden Teilabschnitte (kurz) klären.

Webserver - Apache

Ein Webserver überträgt Daten an einen Client. Dabei kann dieser die Daten nur lokal oder auch weltweit zur Verfügung stellen. Er dient im Falle der o/zb Webanwendung dazu, die von RoR erzeugten Webseiten an den Clienten auszuliefern. Dies geschieht sobald er die Anfrage eines Clienten aufgenommen und an die RoR Anwendung weitergeleitet hat. Daraufhin verar-

beitet die RoR Anwendung diese Anfrage und liefert dem Webserver das Ergebnis zurück. Der Webserver sendet nun diese Daten an den Clienten.

Auf dem Testsystem der o/zb (s. 2.2 auf Seite 3) kommt ein Apache Webserver in der Version 2.2.16 zum Einsatz. Auf dem Testsystem laufen bis zu vier Apache Instanzen gleichzeitig, um die Anfragen der Benutzer bedienen zu können. Weitere Informationen zum Apache Webserver sind unter der Adresse <http://www.apache.org> erreichbar.

Anwendungsserver - Phusion Passenger

Ein *Anwendungsserver* bietet einer Anwendung die benötigte Laufzeitumgebung, damit diese auch ausgeführt werden kann. Dazu stellt der Anwendungsserver der Anwendung spezielle Dienste zur Verfügung, die die Anwendung zur Ausführung benötigt.

Im Fall der o/zb Webanwendung wird der weit verbreitete und leistungsstarke *Phusion Passenger* Anwendungsserver in der Version 3.0.19 genutzt. Phusion Passenger ist ein Modul für den Apache Webserver und ist als ein sogenanntes *RubyGem* (entspricht etwa einem Softwarepaket speziell für Ruby) verfügbar. Zudem ist es auch unter dem Namen *mod_rails* oder *mod_rack* bekannt. Weitere Informationen können auf der Webseite von Phusion <https://www.phusionpassenger.com/> entnommen werden.¹

Capistrano

Wie Eingangs dieses Abschnittes erwähnt worden ist, sind bei einem Deployment Prozess die Schritte notwendig, die die Bereitstellung der Software bewerkstelligen. Diese Schritte müssen nicht immer manuell ausgeführt werden, sondern können automatisiert werden. Dafür eignet sich im Falle der o/zb Webanwendung die Software *Capistrano*. Diese Software ist ein Open Source Werkzeug, das (Batch-) Skripte auf Servern, z.B. mit der Hilfe einer *Secure Shell (SSH)*, ausführt. Dem zur Folge ist ihr Haupteinsatzzweck in der Softwareverteilung wiederzufinden. Capistrano ist genauso wie die o/zb Webanwendung auch in Ruby geschrieben und als Ruby-Gem verfügbar.²

Bei Capistrano bestimmen sogenannte Deployment Rezepte („Deployment Recipies“) wie der Deployment Prozess verläuft. Auch für die o/zb Webanwendung wurde ein Rezept geschrieben, welches im Abschnitt 3.3.3 auf Seite 7 beschrieben wird.

¹vgl. http://de.wikipedia.org/wiki/Phusion_Passenger

²vgl. [http://de.wikipedia.org/wiki/Capistrano_\(Software\)](http://de.wikipedia.org/wiki/Capistrano_(Software))

3.3.2 Versionsverwaltung

Für einen stabilen und guten Softwareentwicklungsprozess ist eine Versionsverwaltung in der heutigen Zeit unabdingbar. Die Hauptaufgaben bestehen aus der Protokollierung der vorgenommenen Änderungen an Quelltexten, Skripten und anderen Dokumenten. Der Wiederherstellung von alten Zuständen, sodass versehentliche Änderungen oder Änderungen, die z.B. zu Laufzeitfehlern führten, zurückgenommen werden können. Die Archivierung jedes neuen Projektzustands. Die Koordinierung des gemeinsamen Datei-Zugriffs der am Projekt beteiligten Entwickler. Und zu guter Letzt ermöglicht eine Versionsverwaltung die gleichzeitige Erzeugung mehrerer Entwicklungszweige (sogenannter „Branches“) eines Projektes.³

Git

Für das Projekt der o/zb Stuttgart wird die weit verbreitete, freie Software *Git* verwendet. Es wurde ursprünglich für die Quelltext-Verwaltung des Linux Kernels entwickelt.

Git ist im Gegensatz zu den Traditionellen Versionsverwaltungen wie z.B. *SVN* oder *Mercurial* ein verteiltes Versionsverwaltungssystem. Es gibt keinen zentralen Server auf dem das Projekt gespeichert wird, sodass jeder Entwickler eine lokale Kopie des gesamten Repositorys vorliegen hat - *clone*. Dem zur Folge hat der Entwickler die Möglichkeit auch ohne Netzwerkzugriff die einzelnen Zustände seiner Arbeit festzuhalten - *commit*. Besteht wieder ein Netzwerkzugriff kann er seine Änderungen auf das von den Entwicklern gemeinsam genutztes Projekt-Depot (*Repository*) hochladen - *push*. Zuvor muss er sich jedoch mit dem gemeinsamen Repository synchronisieren - *pull*.⁴

Das aktuelle, gemeinsame Repository des o/zb Projektes wird von dem bekannten Git-Hoster *GitHub* bereitgestellt. Die Adresse zum Repository lautet: <https://github.com/Avenel/FirstApp>.

3.3.3 Workflow, Umsetzung

In diesem Teilabschnitt werden die vorher erläuterten Konzepte *Deployment* und *Versionsverwaltung* in Zusammenhang gebracht. Es wird ein Arbeitsablauf („Workflow“) definiert, der den Deployment Prozess beschreibt. Dieser Arbeitsablauf wird in Abbildung 3.1 auf der nächsten Seite dargestellt.

³vgl. <http://de.wikipedia.org/wiki/Versionsverwaltung>

⁴vgl. <http://de.wikipedia.org/wiki/Git>

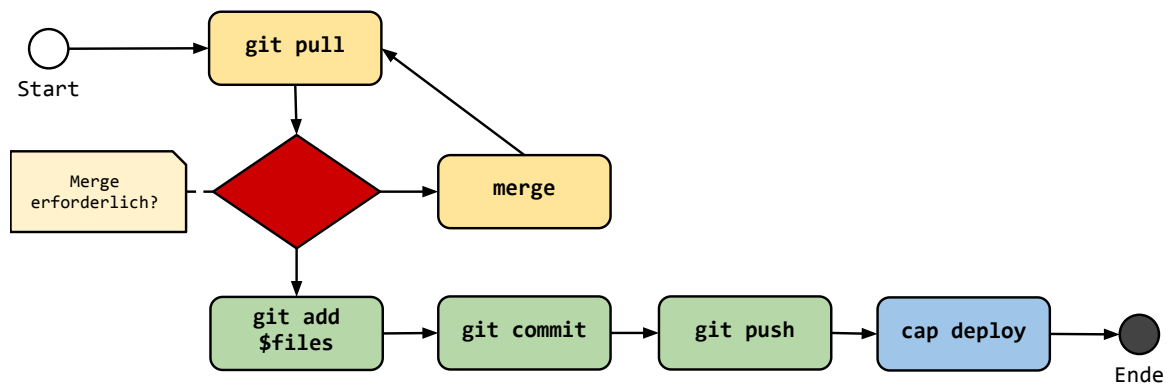


Abbildung 3.1: Der Deployment Workflow

Möchte der Entwickler seine Arbeit auf dem Server bereitstellen, ist er angehalten sich den aktuellen Projektstatus aus dem gemeinsamen Repository zu holen. Er ist dafür verantwortlich sein Projekt vor jedem Commit auf den neuesten Stand zu bringen. Dies geschieht mit der Anweisung *git pull*. Kommt es zu (Datei-) Konflikten die Git nicht automatisch selber lösen kann, muss der Entwickler selber eingreifen (*merge*). Er wiederholt diesen Vorgang solange, bis sein Projekt konfliktfrei und auf dem neuesten Stand ist. Erst dann kann er ausgewählte Änderungen am Projekt für einen neuen *Commit* hinzufügen (*git add*). Im Anschluss schließt er den Commit-Prozess mit dem Befehl *git commit -am "Kommentar"* ab. Damit auch das gemeinsame Repository auf den aktuellsten Stand gebracht wird, erfolgt der Befehl *git push*. Dieser lädt die neuesten Änderungen hoch.

Wurde das gemeinsame Repository nun auf den neuesten Stand gebracht, kann der letzte Schritt im Deployment Workflow durchgeführt werden. Mit *cap deploy* wird das, im nächsten Abschnitt beschriebene, Deploymentskript ausgeführt.

Das Capistrano Deploymentskript (Rezept)

Das Capistrano Deploymentskript bzw. Rezept ist vorerst nicht im öffentlich zugänglichen Git Repository zu finden, da es durchaus sensible Informationen enthält. Ist es vorhanden befindet es sich hier: *config/deploy.rb*.

In dem Deploymentskript werden zuerst sämtliche Variablen festgelegt, der Name der Anwendung und die für einen sicheren SSH Zugriff notwendigen Daten (Serveradresse, sowie auch der Deployment-Benutzer: „*ozbapp*“). Darüber hinaus werden Informationen zum Repository angegeben, in dem die Projektdateien liegen. Im Anschluss wird der Deployment Ort auf dem Server, sowie eigene Aktionen während des Deployment Vorgangs festgelegt.

```

1 # Name application
2 set :application, "ozbapp"
3
4 # Setup deployment user and server ip
5 server "188.64.45.50", :web, :app, :db, :primary => true
6 set :user, "ozbapp"
7 set :use_sudo, false
8 ssh_options[:forward_agent] = true
9
10 # Setup git repository information
11 set :scm, "git"
12 set :repository, "https://github.com/Avenel/FirstApp.git"
13 set :branch, "master"
14
15 # Setup where to deploy the app on the server
16 set :deploy_to, "/home/#{user}/apps/#{application}"
17 set :deploy_via, :remote_cache
18
19 namespace :deploy do
20
21   desc "Tell Passenger to restart the app."
22   task :restart do
23     run touch "#{current_path}/ozbapp/tmp/restart.txt"
24   end
25
26   desc "Renew SymLink"
27   task :renew_symlink do
28     run "rm /home/ozbapp/ozbapp"
29     run "ln -s /home/ozbapp/apps/ozbapp/current/ozbapp /home/ozbapp/ozbapp"
30   end
31
32 end
33
34 # Execute renew_symlink after update_code
35 after 'deploy:update_code', 'deploy:renew_symlink'

```

Listing 3.1: Capistranot Deployment Rezept

3.4 Datenbank Migration

4 Korrekturen, Bugfixes

4.1 WebImport

4.2 Punkteberechnung

5 Testdriven Development

5.1 Analyse

5.2 Implementierung

5.2.1 RSpec

6 Neue Features

6.1 Dokumenten-Export

6.1.1 Analyse

6.1.2 Implementierung

7 Ergebnis und Ausblick

7.1 Ergebnis

7.2 Ausblick

8 Anhang

Abbildungsverzeichnis

3.1	Der Deployment Workflow	7
-----	-----------------------------------	---

Tabellenverzeichnis

Listings

3.1	Capistranot Deployment Rezept	8
-----	---	---