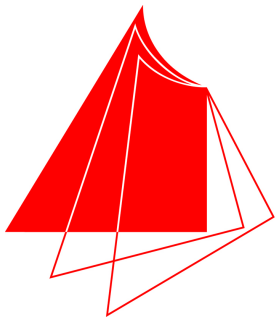

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

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

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 gegebenfalls 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 weißt 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 RubyGem 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.

---

[1] vgl. `http://de.wikipedia.org/wiki/Phusion_Passenger`
[2] vgl. `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“) defininert, der den Deployment Prozess beschreibt. Dieser Arbeitsablauf wird in Abbildung 3.1 auf der nächsten Seite dargestellt.

[3] vgl. `http://de.wikipedia.org/wiki/Versionsverwaltung`
[4] vgl. `http://de.wikipedia.org/wiki/Git`

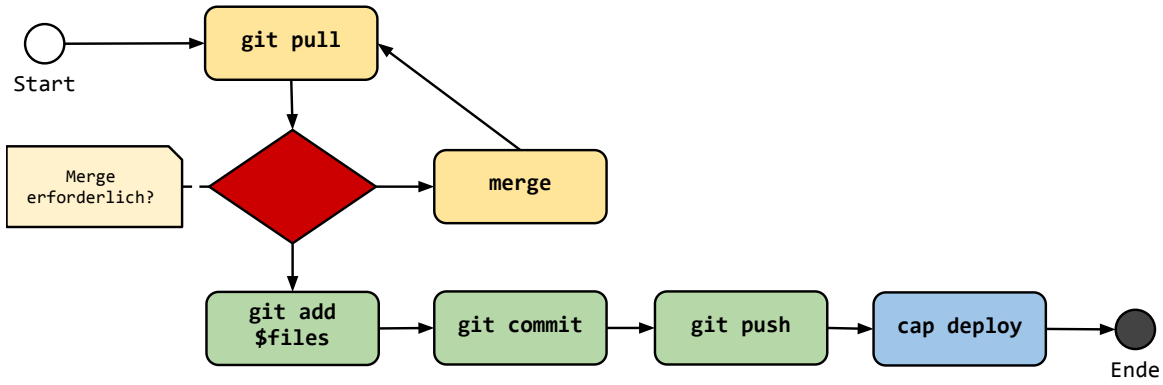

Abbildung 3.1: Der Deployment Workflow

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.

```
# Name application
set :application, "ozbapp"

# Setup deployment user and server ip
server "188.64.45.50", :web, :app, :db, :primary => true
set :user, "ozbapp"
set :use_sudo, false
ssh_options[:forward_agent] = true

# Setup git repository information
set :scm, "git"
set :repository, "https://github.com/Avenel/FirstApp.git"
set :branch, "master"

# Setup where to deploy the app on the server
set :deploy_to, "/home/#{user}/apps/#{application}"
set :deploy_via, :remote_cache

namespace :deploy do

    desc "Tell Passenger to restart the app."
    task :restart do
        run touch "#{current_path}/ozbapp/tmp/restart.txt"
    end

    desc "Renew SymLink"
    task :renew_symlink do
        run "rm /home/ozbapp/ozbapp"
        run "ln -s /home/ozbapp/apps/ozbapp/current/ozbapp /home/ozbapp/ozbapp"
    end

end

# Execute renew_symlink after update_code
after 'deploy:update_code', 'deploy:renew_symlink'
```

Listing 3.1: Capistrano Deployment Rezept

3.4 Datenbank Migration

In der vergangenen Zeit wurde ein Java Datenbank Migrationstool geschrieben, welches die Daten von dem aktuellen Produktivsystem in das neue System übertragen soll. Um den Umgang mit diesem Tool zu erleichtern sind zwei Batch Skripte angefertigt worden. Diese Batch Skript sind im Ordner *tools* des Git Repositorys und auch auf dem Server im *Home Verzeichnis ozbapp* zu finden. Um die Batch Skripte ausführen zu können ist eine SSH Verbindung erforderlich. Diese kann unter Windows mit dem Programm *Putty* (`http://www.putty.org/`) oder unter Linux mit dem Befehl *ssh username@host* (`http://wiki.ubuntuusers.de/SSH`) geöffnet werden. Der Username bzw. der Hostname ist in beiden Fällen *ozbapp* bzw. *ozbapp.mooo.com*. Die benötigte Konfiguration für Putty ist der Abbildung 3.2 zu entnehmen. Ist man verbunden befindet man sich automatisch schon im *Home Verzeichnis* in dem die Batch Skripte vorliegen.

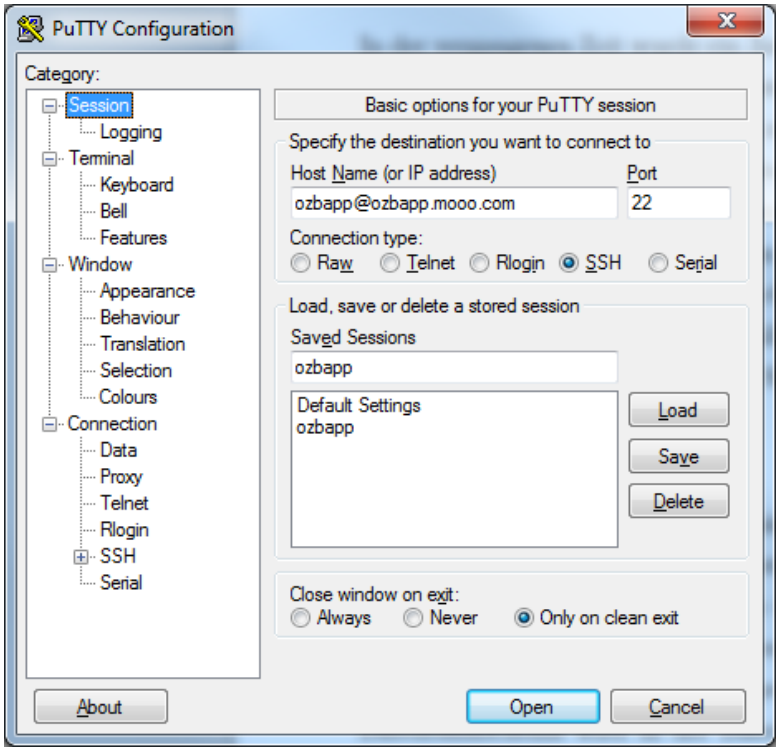

Abbildung 3.2: Putty Konfiguration

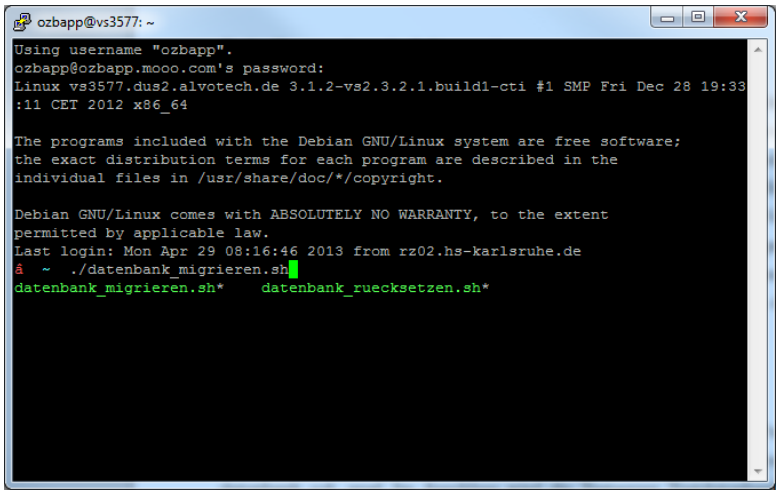

Abbildung 3.3: Offene SSH Session in Putty

Im Folgenden werden die Funktionsweise und die Benutzung dieser beiden Batch Skripte beschrieben.

3.4.1 Datenbank migrieren

Dieses Batch Skript importiert zunächst einen aktuellen Stand der Produktivdatenbank, der als MySql Dump zur Verfügung gestellt wird, in die auf dem Testserver liegende Produktivdatenbank *ozb_prod*. Im Anschluss wird die Testserver Testdatenbank *ozb_test* geleert und neu angelegt. Ist dies geschehen, werden mit Hilfe des Datenbank Migrationstools die Daten aus der Produktivdatenbank *ozb_prod* in die Testdatenbank *ozb_test* übertragen. Das neue Datenbankschema wird in der Datei *create_tables.txt* beschrieben. Sind Veränderungen am Datenbankschema vorgenommen worden, müssen diese in dieser Datei übernommen werden. Ausgeführt wird dieses Skript mit dem Befehl *./datenbank_migrieren.sh*.

```
1 #!/bin/bash
2 PASS=""
3 echo "drop database ozb_prod" | mysql -u root -p$PASS
4 echo "create database ozb_prod" | mysql -u root -p$PASS
5 mysql -u root -p$PASS ozb_prod < ozb_prod.sql
6 echo "drop database ozb_test" | mysql -u root -p$PASS
7 echo "create database ozb_test" | mysql -u root -p$PASS
8 java -jar OZBMigration.jar -i ./create_tables.txt -u root -p $PASS
9 mysqldump -u root -p$PASS ozb_test > dump.sql
```

Listing 3.2: datenbank_migrieren.sh

3.4.2 Datenbank zurücksetzen

In diesem Batch Skript wird die Testdatenbank auf den ursprünglichen Zustand zurückgesetzt. Der ursprüngliche Zustand liegt in dem MySql Dump *dump.sql*. Zur Ausführung genügt auch hier der folgende Befehl *./datenbank_ruecksetzen.sh*.

```
1 #1/bin/bash
2 PASS=""
3 echo "drop database ozb_test" | mysql -u root -p$PASS
4 echo "create database ozb_test" | mysql -u root -p$PASS
5 mysql -u root -p$PASS ozb_test < dump.sql
```

Listing 3.3: datenbank_migrieren.sh

4 Korrekturen, Bugfixes

4.1 Webimport

In der o/zb Webanwendung sorgt der sogenannte *Webimport* für die Übertragung der in einer CSV Datei hinterlegten Kontobewegungen. Weitere Informationen zu dem Webimport kann der Projekt Dokuementation WS 12/13 (ab S.53) entnommen werden.

Zum Zeitpunkt der Aufnahme der Arbeiten an diesem Projekt funktionierte der Webimport nicht korrekt. Die folgenden Punkte sind aufgefallen:

- Datei Upload auf dem Testsystem funktioniert nicht.
- Laufzeitfehler müssen abgefangen werden.
- Die Anzahl der importierten Datensätze stimmt nicht.

4.1.1 Bugfix: Datei Upload

Bei diesem Fehler akzeptierte der Webimport scheinbar keine Datei. Der Grund hierfür liegt in den abgebildeten Zeilen in Listing 4.1.3 auf Seite 14.

```
1     if (collect_konten.size == 0 )
2       @error += "Keine der zu importierenden Konten in der Datenbank eingetragen"
3     else
4       collect_konten.uniq.each do |ktoNr|
5         b = Buchung.find(
```

Listing 4.1: webimport_controller.rb

In Zeile 4 (bzw. Zeile 426) wurde ursprünglich der Befehl *uniq!* verwendet. Methoden mit einem Ausrufezeichen (!) benutzen meistens eine unsichere Implementierungsweise der Methode. Dieser Befehl gibt laut der offiziellen RubyDoc [1] entweder ein *Array* oder *nil* zurück. Der

[1] vgl. http://www.ruby-doc.org/core-1.9.3/Array.html#method-i-uniq-21

Rückgabewert *nil* tritt ein, sobald das Array keine Elemente vorweisen kann. Da dieser Fall nicht abgefangen worden ist, kommt es in der Zeile 4 (bzw. 426) zu einem Fehler. Abhilfe schafft hier die Methode *uniq*. Diese gibt in jedem Fall ein (ggf. leeres) *Array* zurück.

4.1.2 Laufzeitfehler

Konnte ein Datensatz z.B. aufgrund eines MySql Fehlers nicht importiert werden, wird eine Exception geworfen. Aufgrund dieser Exception wurde der Importvorgang nicht korrekt zu Ende geführt, sondern direkt abgebrochen. Der Benutzer sieht in diesem Moment eine Fehlerseite und kann nicht mehr agieren, da er keine Informationen darüber erhält, was schiefgegangen ist. Der häufigste Fehler tritt auf, wenn versucht worden ist eine Buchung noch ein weiteres Mal zu importieren (MySql Fehler: *duplicate key error*). Dies gilt es mit Hilfe des Exception Handlings zu korrigieren.

Die Exceptions treten beim Speichervorgang auf, dem zur Folge wurde um jeden Speichervorgang ein *begin-rescue* Block gesetzt. Dieser fängt die Exception auf und hängt die Fehlermeldung der *@error* Variable an. Diese stellt dem Benutzer ggf. die Informationen darüber, was schiefgelaufen ist, dar.

```
1                begin
2                  b.save
3                  collect_konten << kontonummer
4                  imported_records += 1
5                rescue Exception => e
6                  @error += "Etwas ist schiefgelaufen.<br /><br />"
7                  @error += e.message + "<br /><br />"
8                end
```

Listing 4.2: webimport_controller.rb

4.1.3 Anzahl der importierten Datensätze

```
1                begin
2                  b.save
3                  collect_konten << kontonummer
4                  # Nur 1x zaehlen!
5                  #imported_records += 1
6                rescue Exception => e
7                  @error += "Etwas ist schiefgelaufen.<br /><br />"
```

```
8                     @error += e.message + "<br /><br />"
9                   end
```

Listing 4.3: webimport_controller.rb

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

6.2 Deployment E-Mail Benachrichtigung

7 Ergebnis und Ausblick

7.1 Ergebnis

7.2 Ausblick

8 Anhang

Abbildungsverzeichnis

Tabellenverzeichnis

Listings