Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Entwurf und Entwicklung einer Ranking Website von Open Source Projekten für Crowdsourcing Zwecke

Bachelorarbeit

im Studiengang Informatik

von

Yauheni Stoliar

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Jun.-Prof. Dr. Robert Jäschke
Betreuer: Dipl.-Math. Raphael Pham

Hannover, 07.03.2013

Abstract

Open Source community has a low participation in software testing. The heterogeneous crowdsourced code cannot be merged with the master branch directly without testing. This leads to additional outlay in the collaborative development. An explicit platform for software testing with its own administration, personalized content and access to the test case related knowledgebase simplifies creation of tests. The adoption of this platform by the open source community, will lead to increased interest in software testing.

Zusammenfassung

Open-Source-Community weist eine geringe Beteiligung an Software Tests auf. Der aus unterschiedlichen Quellen stammende Code kann ohne Tests nicht direkt ins Hauptprojekt übernommen werden. Das führt zu einem zusätzlichen Aufwand in der kollaborativen Entwicklung. Eine explizite Plattform für Software Tests mit eigener Verwaltung, personifizierten Inhalten und einem Zugriff auf bereits implementierte Testfälle vereinfacht das Schreiben von Tests. Wird diese Plattform von der Open-Source-Community genutzt, so wird sich die Beteiligung an Software Tests erhöhen.

Inhaltsverzeichnis

1. Einleitung **1**
1.1. Motivation . . . 1
1.2. Ziel . . . 2
1.3. Gliederung . . . 3

2. Grundlagen **4**
2.1. GitHub . . . 4
2.1.1. Sprechen Sie GitHub? . . . 5
2.2. Hintergründe der Motivation . . . 6
2.2.1. Individuelle Motivation . . . 7
2.2.2. Zwischenmenschliche Motivation . . . 8
2.2.3. Zusammenfassung . . . 9
2.3. Entwicklungswerkzeug . . . 9
2.3.1. Grails . . . 9
2.3.2. Twitter Bootstrap . . . 11

3. Analyse **12**
3.1. Benutzerprofile . . . 12
3.1.1. Erfahrene Benutzer . . . 12
3.1.2. Interessenten . . . 12
3.1.3. Passanten . . . 13
3.2. Ablaufszenarien . . . 14
3.3. Basissystemanalyse . . . 14
3.4. Erweiterungsanalyse . . . 15
3.5. Analyse der sozialen Aspekte . . . 16
3.6. Richtig motivieren . . . 16
3.6.1. Bequeme Umgebung . . . 16
3.6.2. Motivierende Umgebung . . . 16
3.6.3. Profitable Umgebung . . . 17
3.7. Zusammenfassung . . . 17
3.7.1. Projekteigentümer . . . 17
3.7.2. Projektteilnehmer . . . 18

4. Ranking **21**
4.1. Gemeinsame Bewertungskriterien . . . 21
4.1.1. Basispunkte . . . 21
4.1.2. Bonuspunkte . . . 21
4.2. Personalisierte Darstellung . . . 22
4.2.1. Soziale Punkte . . . 22

4.3. Zusammenfassung 24

5. Implementierung **25**
5.1. Wer macht was 25
5.1.1. Benutzermanagement 25
5.1.2. Nachrichten 27
5.1.3. Import von GitHub 27
5.1.4. Testfälle 28
5.2. Ranking 31
5.3. Aktualisierungen 32
5.4. Design 33
5.5. Zusammenfassung 33

6. Ausblick **35**
6.1. Modularität 35
6.2. Komfort und Notwendigkeit 36
6.3. Anpassung des Rankings 36
6.4. Zusammenfassung 36

A. Anhang **38**
A.1. Inbetriebnahme 38
A.1.1. Datenbank 38
A.1.2. Server 38
A.1.3. Konfiguration 39
A.1.4. Installation 40

Literaturverzeichnis **41**

1. Einleitung

1.1. Motivation

Wenn man sich umsieht, so wird man kaum etwas finden, was ohne Hilfe des Computers entstanden ist. Bücher, Möbel, Unterhaltungselektronik – diese Liste kann unendlich fortgeführt werden – um all dies zu entwerfen und herzustellen, wurde auf die Möglichkeiten des Computers zurückgegriffen. Ohne Software wäre das alles nicht möglich. Die Einsatzgebiete der Computer sind unzählig und damit auch die von Software. Fast jeder Mensch ist heute auf die eine oder andere Weise mit der Computernutzung konfrontiert.

Permanent steigende Anforderungen an Software stimulieren das Entstehen neuer Arten der Softwareentwicklung. Immer kürzer werdende Nutzungszyklen implizieren Mitwirkungsbereitschaft seitens der Entwickler. Es muss schneller, besser und sicherer produziert werden. Die Anzahl der Softwareprodukte steigt dabei und befindet sich mittlerweile im 7-stelligen Bereich [3, S. ES-1]. So entsteht Bedarf nach schnellem Informations- und Erfahrungsaustausch. Es wird in Gruppen entwickelt, und sogar dafür existieren Konzepte zur Produktivitätssteigerung, die es erlauben, während der parallelen Zusammenarbeit auftretende Konflikte zu vermeiden.

Als Beispiel dafür kann die Crowdsourcing-Plattform „GitHub“ dienen, die jedem eine Möglichkeit bietet, eigene Projekte in Zusammenarbeit mit anderen zu entwickeln oder an fremden Projekten teilzunehmen. Die geografische Lage spielt dabei keine Rolle. Alles wird online erledigt.

Solche neuen Methoden bieten zwar steigende Produktivität, bringen dennoch auch neue Herausforderungen mit sich. Mit steigender Entwickleranzahl mehrt sich auch die Anzahl der Programmierstile. Das Gesamtkonzept muss kommuniziert und verstanden werden. Ist das nicht der Fall, kommt es zu Fehlern.

Wenn wir vor dem Hintergrund riesiger Softwareeinsatzgebiete, heterogener Entwicklungsumgebungen und schneller Lernstrategien über die Auswirkungen eines logischen oder technischen Fehlers nachdenken, wird klar, wie gravierend solche Fehler wären. Es wäre logisch, sich an der Stelle zu fragen, was man tun kann, um solche Situationen, wo ein Fehler eine große Anzahl von Nutzern betrifft, zu vermeiden. Die Antwort existiert seit Jahrzehnten – Softwaretests. Je höher das Areal der Softwarenutzung ist, umso höher ist der Bedarf nach Softwaretests und deren Wert. Sicherlich ist diese Methode nicht die einzige, die zur Qualität der Software beiträgt. Sie ist jedoch eine erprobte und funktionierende.

1.2. Ziel

Diese Bachelorarbeit untersucht die technischen und psychologischen Aspekte der sozialen Software-Entwicklung. Der wirtschaftliche Nutzen der Software Tests wird mit den konkreten Zahlen in einem Abschlussbericht für das National Institute of Standards and Technology (NIST), USA beschrieben [3].

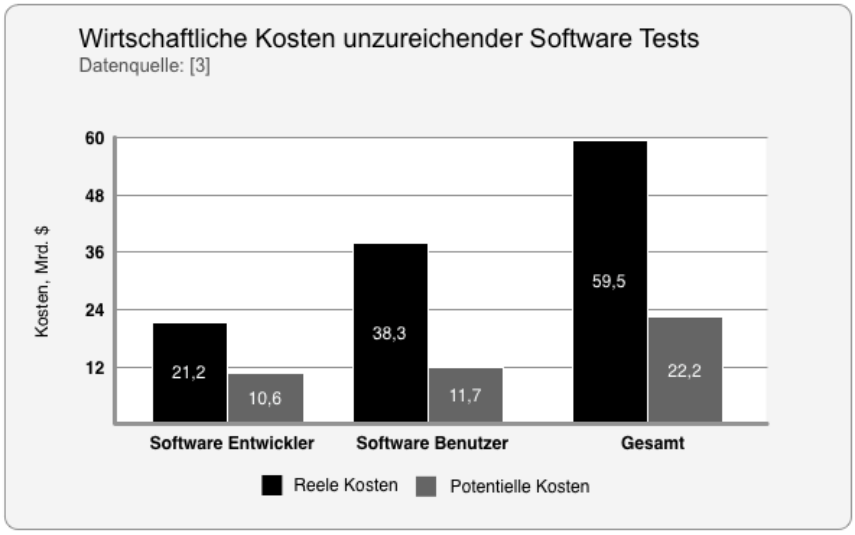

Abbildung 1.1.: Wirtschaftliche Kosten unzureichender Softwaretests

In Abbildung 1.1. ist deutlich erkennbar, wie viel man durch bessere Test-Infrastruktur sparen könnte. Aber wenn Softwaretests so nützlich sind, aus welchem Grund müssen die Entwickler immer noch dazu motiviert werden, diese zu schreiben? Einerseits ist es der materielle Faktor: Auch wenn die späteren Kosten durch Tests minimiert werden können, entstehen dennoch erhebliche initiale Kosten, die sowohl seitens der Entwickler, als auch wegen der Kurzsichtigkeit des Projektmanagements oder des knappen Budgets gespart werden wollen. Einen anderen Einblick darin bietet uns der psychologische Aspekt der Softwaretests. Eine plausible Erklärung wurde bereits im 1979 von Glenford J. Myers [2, S. 6] gegeben. Er postulierte, dass Testen ein psychologisch destruktiver Prozess ist, und folglich fällt es den meisten Entwicklern schwer, sich in die Rolle des Softwaretesters zu versetzen, weil sie auf Erschaffung gestimmt sind. Diese Rolle erfordert eine Denkweise, die der des Entwicklers komplett entgegengesetzt ist.

Literaturrecherche soll helfen, unterschiedliche theoretische Konzepte zusammenzufassen und auf dieser Basis eine neue Methode auszuarbeiten, die den Softwareentwicklern einen leichteren Einstieg in die Tests ermöglicht.

In dieser Arbeit möchten wir zeigen, wie man eine Umgebung entwickeln kann, welche die Entwickler aus der Open Source Community anspricht. Unsere primären Ziele sind: eine Schnittstelle mit den vorhandenen Entwicklungsplattformen, leicht zu bewältigende Anforderungen an Teilnehmer, klare Rückmeldungen und Wettbewerb.

Wir wollen eine Plattform entwickeln, die außer den bereits aktiven Programmierer auch solche anspricht, die bis heute zwar präsent, aber untätig geblieben sind. Feine Strukturierung der Aufgaben soll den Komplexitätsgrad so reduzieren, dass jeder motiviert wird mitzumachen.

Außerdem wollen wir die sozialen Aspekte der Entwicklungsnetzwerke am Beispiel von GitHub studieren. Durch eine Auswertung des sozialen Verhaltens eine soll dem Benutzer vereinfachte Suche nach attraktiveren Anwendungsfällen geboten werden.

Als praktisches Ergebnis soll eine Webseite entstehen. Sie wird Accounts beinhalten, sowie die Möglichkeit, diese zu verwalten. Benutzer der Website erhalten eine Import-Funktion für ihre GitHub-Projekte. Nach dem Import werden die Anwender Testfälle für die eigenen Projekte erstellen, die von anderen Benutzern gelöst werden können. Die Wahl der Testfälle bleibt jedem Benutzer überlassen, allerdings wird die Darstellung durch ein Ranking beeinflusst. Die Lösungen werden von Eigentümern bewertet und die Bewertung fließt in das Ranking mit ein. Die Projekte mit einem höheren Rang werden dem Benutzer zuerst angezeigt. Das Ermitteln der Punkte für das Ranking wird vom System anhand eines Algorithmus durchgeführt.

1.3. Gliederung

Diese Arbeit ist in sechs Teile gegliedert. Die Motivation dieser Arbeit, die angestrebten Ziele und die Gliederung werden im ersten Kapitel geschildert. Im Kapitel 2 werden die Grundlagen wie GitHub, Taxonomie der Motivation und Entwicklungswerkzeuge beschrieben und die in dieser Arbeit verwendeten Begriffe erklärt. Die Anforderungsanalyse und die Erwartungen an den Prototypen sind im Kapitel 3 zu finden. Das Ranking ist ein Schlüsselthema dieser Arbeit und wird separat im Kapitel 4 analysiert und beschrieben. Der Prozess der Implementierung mit Code-Beispielen ist im Kapitel 5 zu finden. Nicht alle Ideen wurden in dem während dieser Arbeit entstandenen Prototypen implementiert. Unsere Visionen der weiteren Entwicklung des Systems befinden sich im Kapitel 6. Im Anhang sind die für die Inbetriebnahme notwendigen Schritte erklärt.

2. Grundlagen

2.1. GitHub

GitHub ist einer der größten [4] Anbieter für Projekt-Hosting und ein beliebtes soziales Netzwerk für Entwickler. Die Wachstumsraten der Teilnehmer über die Jahre sind bemerkenswert: Über 3 Mio. Nutzer sind an ca. 5 Mio. Projekten tätig.

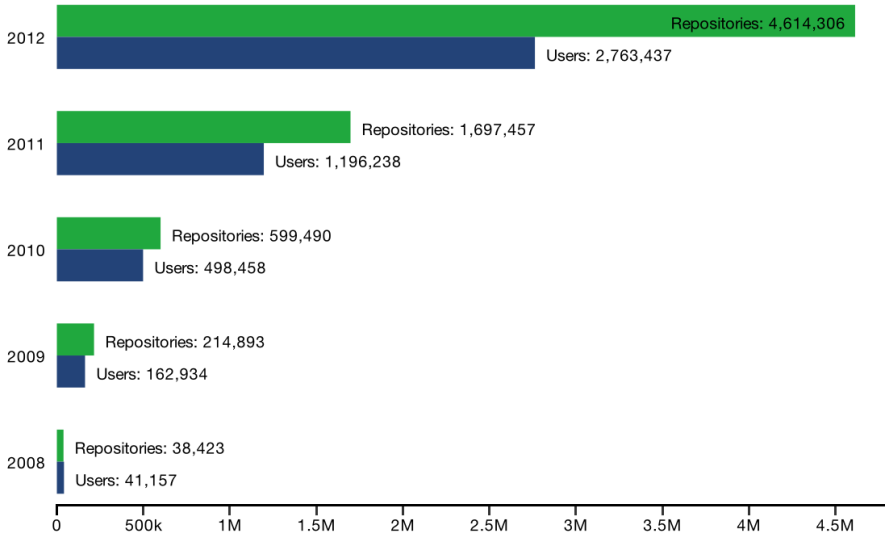

Abbildung 2.1.: GitHub Nutzer- und Projektstatistiken. Quelle: [5]

GitHub ist aus Git, einem Open-Source-Projekt von Linus Torvalds, entstanden. Neben Git existieren auch andere Versionskontrollsysteme. Die Idee von Git liegt darin, dass die Änderungen nicht wie bei den anderen Systemen direkt am Hauptstamm des Projektes durchgeführt werden. Zuerst werden die Projektdateien auf den eigenen Rechner geladen, und anschließend wird an dieser Kopie des Projektes gearbeitet. GitHub stellt dem Benutzer ein attraktives grafisches Webinterface mit allen Vorteilen von Git bereit. Außerdem erhält man mit GitHub ein Instrument für die Steuerung der kollaborativen Seite des Entwicklungsprozesses: Rechtemanagement, Wiki und Issuemanagement.

Die aktive Mitarbeit an verschiedenen Projekten spiegelt sich positiv im Profil des Benutzers wieder. Als Projekteigentümer kann man sehen, wie erfahren der Benutzer, von dem ein Pull Request kommt, ist und somit entsprechend mehr oder weniger aufmerksam vorgehen.

2.1.1. Sprechen Sie GitHub?

Mit GitHub ist eine eigene Terminologie hinzugekommen, die die Prozesse des oben beschriebenen Ablaufs kurz und bündig erklärt. Hier werden Begriffe, die später in dieser Arbeit genutzt werden, vorgestellt.

Repository: ein Ordner für die Projektdateien. Ein Repository beinhaltet nicht nur Code. Es können unter anderem auch Konfigurationsdateien, sowie Dokumentation vorkommen. Jede Datei wird aus einem Repository ausgecheckt und wieder in ein Repository mit einem akzeptierten Pull Request hochgeladen. Es existieren zwei Arten von Repositories: private und öffentliche, wobei nur die Letzteren für uns von Interesse sind – private Repositories veröffentlichen ihren Inhalt nicht.

Das Besondere an GitHub ist die Automatisierung der Änderungsübernahme. Wenn man vor der Einführung von GitHub eine Änderung zu einem fremden Projekt beitragen wollte, so musste man sich den Quellcode herunterladen, Änderungen implementieren und dem Verantwortlichen zukommen lassen. Danach mussten die Änderungen manuell eingearbeitet werden. Mit GitHub erhalten die Entwickler die Möglichkeiten, diesen Prozess mittels vordefinierter Aktionen wie **Fork**, **Pull Request** und von der Seite des Eigentümers (**Merge**), erheblich zu beschleunigen.

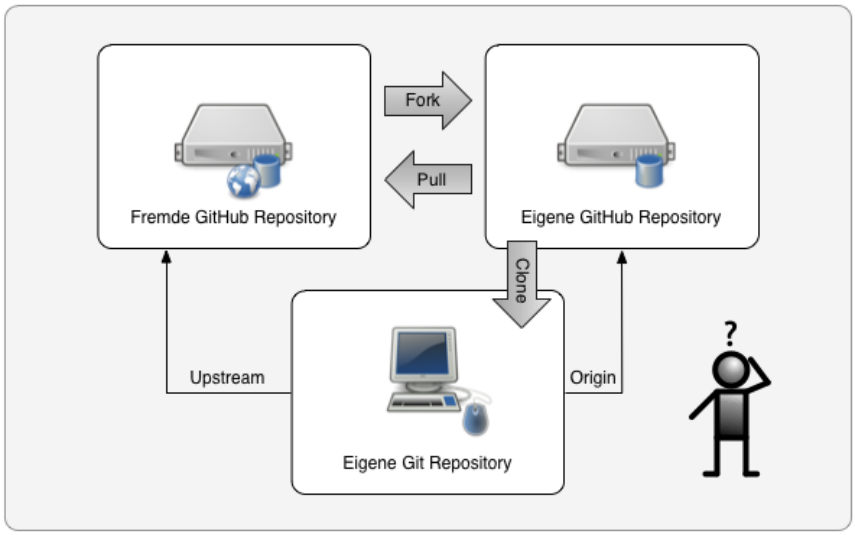

Abbildung 2.2.: GitHub Aktivitätsstruktur

Fork: eine Projektkopie. Möchte man bei einem Projekt mitwirken oder sich einen Einblick in die Projektstruktur verschaffen, so wird eine Kopie des Originalprojektes im eigenen GitHub-Account erstellt. Zu beachten ist, dass dabei keine lokale Kopie erstellt wird.

Clone: eine lokale Kopie eines **Forks**. Aus Sicht dieser Kopie existieren ein **Origin** und ein **Upstream**. Mit solcher Differenzierung ist es möglich, auch die letzten Änderungen im Originalprojekt zu verfolgen.

Branch: eine Unterteilung im Repository. Mit Branches ist es möglich, parallel z. B. zu der sicheren eine experimentelle Entwicklung zu führen, zwischen ihnen zu wechseln und diese bei Bedarf zusammenzuführen.

Push: Befehl zum Laden der lokalen Daten in einen Fork.

Pull Request: wird erstellt, wenn man die Ergebnisse der eigenen Arbeit dem Projekteigentümer bereitstellen möchte. Für den Request wählt man einen Namen und erstellt eine Beschreibung der Änderungen. Es besteht die Möglichkeit, zu dem Pull Request Kommentare zu hinterlassen. Sollte eine nachträgliche Bearbeitung des Pull Requests notwendig sein, so geschieht das durch die Bearbeitung eines lokalen Branchs und durch einen anschließenden Push mit allen Änderungen in den Online-Branch.

Merge: wird durchgeführt, wenn die in einem Pull Request vorgeschlagenen Änderungen übernommen werden. Man kann zwischen einem automatischen und einem manuellen Merge wählen, wobei man häufiger zum Letzteren tendiert.

Follow: kann erfolgen, wenn ein Interesse an Aktivitäten eines Entwicklers bei GitHub besteht: Ist der Code gut, oder sind es andere Faktoren, kann man mit Follow seinen eigenen Personenkreis zusammenstellen und über dessen Aktionen informiert bleiben.

Watch: funktioniert analog zu der Follow-Aktion, allerdings geht es dabei um Projekte. Man wird über die Änderungen am Projekt benachrichtigt.

Issue: dient wie ein Vermerk über die anstehenden Änderungen. In einem Issue kann man beschreiben, welche Fehler noch zu beheben oder welche Ideen noch zu implementieren sind. Ein Issue bleibt offen, solange das darin beschriebene Problem besteht, und wird anschließend geschlossen.

2.2. Hintergründe der Motivation

Man beschäftigt sich schon lange mit den Ursachen der Motivation. Noch in 1966 schrieb Brunner [6, S. 127]

> The will to learn is an intrinsic motive, one that finds both its source and its rewards in its own exercise.

Weiterhin erläutert er, dass der Erfolg des Lernprozesses vom Ursprung der Motivation abhängig ist. Ist der Lernpfad vorgegeben, so kann es dazu kommen, dass er nicht den Interessen des Lernenden entspricht und somit nicht von diesen unterstützt wird. Im Jahr 1987 haben Thomas Malone und Mark Lepper einen Versuch unternommen, das Wissen über die Motivation zu strukturieren [7]. Im Rahmen eines Versuchs, wo einer Schülergruppe Lerninhalte mittels eines Spiels vermittelt wurden, untersuchten die Autoren die Änderungen der Motivation in den Versuchsgruppen anhand der Änderungen im Spielprozess. Basierend auf den Ergebnissen entwickelten sie eine Übersicht der Motivationsfaktoren und strukturierten diese zugleich.

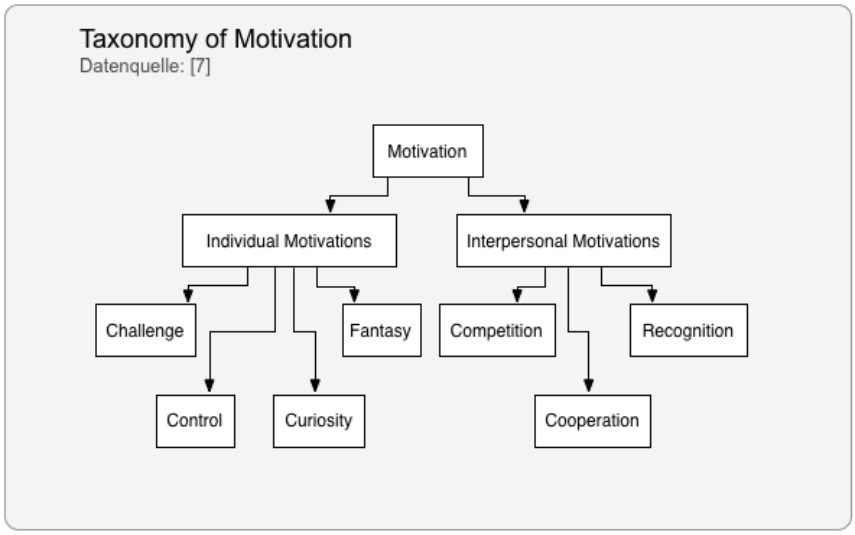

Abbildung 2.3.: Struktur der Motivation

Wir gehen dieser Studie nach und werden einige der Motivationsfaktoren in unserem System verwenden. Dafür müssen wir uns vorerst mit der Terminologie bekannt machen.

Intrinsische Motivation ist der aus eigener Interesse entstandene Wunsch, sich mit einer Tätigkeit zu beschäftigen. Dieser kann durch Faktoren wie Spaß, Neugierde, Faszination, Genuss [7] o. Ä. hervorgerufen sein.

Extrinsische Motivation entsteht beim Versuch, einem Ziel nachzugehen oder alternativ eine möglicherweise ungünstige Situation zu vermeiden.

Stellen wir uns eine Situation vor, in der eine Person ein nützliches Computerprogramm entwickeln möchte. Der Entwicklungsprozess ist dabei ein Ergebnis der intrinsischen Motivation. Der Weg bis zum Punkt, wo man ein wirklich nützliches Programm selbst schreiben kann, impliziert das Lernen der zahlreichen Rechnergrundlagen und womöglich auch der Programmierumgebungsfeinheiten. Diese zusätzlichen Informationen sind das Ergebnis der extrinsischen Motivation. Nun gehen wir noch etwas detaillierter auf die Bausteine der Motivation ein.

2.2.1. Individuelle Motivation

Schauen wir uns zuerst die von Malone und Lepper [7] vorgeschlagene Aufteilung der persönlichen Motivation an.

Challenge oder **Herausforderung**. Dabei wird auf folgende Faktoren geachtet:

- *Klares Ziel*. Unter vielen anderen werden z. B. die Computerspiele bevorzugt, die ein klares Ziel verfolgen. Diese Eigenschaft wurde als das wichtig-

ste Kriterium bei der Auswahl eingestuft. Außerdem muss darauf geachtet werden, dass das Ziel klar und potenziell erreichbar definiert wird.

- *Ungewissheit* über das Endergebnis ist ein weiteres Motivationsfaktor, dessen Spanne sorgfältig gewählt werden muss. Wird das Ergebnis von Anfang an als zu leicht erreichbar oder gar unerreichbar angesehen, so mindert das die Attraktivität.
- *Rückmeldung* über die eigene Leistung wird benötigt, um das stetige Interessenniveau des Lernenden zu erhalten. Es ist wichtig, dass die Rückmeldung häufig, klar, konstruktiv und motivierend kommuniziert wird.

Curiosity oder **Neugier** hat einen ausgeprägten Einfluss auf die intrinsische Motivation. Dieser Faktor wird vorgestellt als eine Kombination aus sensorischen und kognitiven Komponenten.

- *Sensorische Neugier* entsteht aus den Details, die unsere Aufmerksamkeit durch sensorische Wahrnehmung anziehen. Das kann in unserem Fall Design, Layout, Farbpaletten o. Ä. sein.

Control oder **Kontrolle** ist ein sehr starkes Motivationsfaktor, weil es den Menschen einen Gefühl vermittelt, selbst die Quelle der Änderungen in der Umgebung zu sein und diese nicht als Ergebnis der externen Faktoren anzusehen.

- Die *Wahl*, falls vorhanden, ist bekannt als ein starkes Mittel zur Erhöhung der intrinsischen Motivation. Am Beispiel der Lehrumgebung kann dem Teilnehmer die Wahl gegeben sein, die Aufgaben selbständig aussuchen zu können, den Schwierigkeitsgrad zu bestimmen und sogar zu entscheiden, ob die Aufgabe genommen wird oder nicht. Zu beachten ist, dass die Anzahl der Wahlmöglichkeiten nicht allzu großzügig gewählt werden darf und, wie in [7] zitiert, mit ca. fünf bis sieben Positionen optimal sei.

2.2.2. Zwischenmenschliche Motivation

Häufig hängen die Motive von anderen Personen in der Umgebung ab. Ein Wettbewerb wirkt meistens motivierend und lässt erkennen, wo eigene Schwächen liegen. Hier geht es um Motivation, die ohne andere Menschen nicht vorhanden wäre.

Competition oder **Wettbewerb**. Sieht man, dass eigene Lösung besser oder schlechter als die eines Anderen bewertet wurde, so dient es gleichzeitig wie eine Rückmeldung und ist ein Belohnungs- oder Motivationsfaktor. Durch das Vorhandensein der alternativen Lösung ist es möglich zu erkennen, wo eigene Stärken oder Schwächen liegen.

Recognition oder **Anerkennung** ist eine wichtige Voraussetzung für den Motivationsaufbau. Die Ergebnisse der Umfrage über Beliebtheit von Computerspielen haben gezeigt, dass Punktestandsführung als einer der wichtigsten Faktoren genannt wird.

2.2.3. Zusammenfassung

Anhand der oben genannten Motivationsfaktoren werden wir versuchen, die heterogenen Interessen der Benutzer anzusprechen. Gehen wir davon aus, dass die Teilnehmer aus einer Vielzahl von Gründen mitmachen, so wird es klar, dass wir, um die initiale Motivation zu erhalten und eventuell weiter darauf aufzubauen, auf individuelle Herangehensweisen angewiesen sind.

2.3. Entwicklungswerkzeug

Bei der Entwicklung des Prototypen greifen wir auf Werkzeuge der Webentwicklung zurück, die zwar neu sind, dennoch auf den bewährten Konzepten der Software Entwicklung basieren. Als Grundgerüst dient uns das Grails Framework. Als Designgrundlage nutzen wir die Elemente des grafischen Twitter Bootstrap Frameworks.

2.3.1. Grails

Grails ist ein Webentwicklung-Framework, das auf der Programmiersprache Groovy basiert. Das Framework baut auf dem MVC-Muster auf. Die Idee der schnellen Prototyp-Entwicklung wird durch die Hintergrundmechanismen von Grails unterstützt. Die Struktur des Prototyps beruht auf folgenden Konzepten:

Groovy ist eine dynamische Programmiersprache. Sie ist Java sehr ähnlich und ermöglicht einen leichten Einstieg für Java-Entwickler. Außerdem wurde die Syntax von Groovy im Vergleich zu Java sehr stark überarbeitet mit der Intention, Bündigkeit und Lesbarkeit zu erhöhen.

Convention over Configuration ist das Hauptkonzept von Grails. Das wird besonders sichtbar, wenn man die frühere Java-Web-Entwicklung kennt. Die Anzahl der Konfigurationsdateien ist deutlich reduziert worden. Dies wurde durch Einhalten bestimmter Regeln möglich. Als Beispiel kann der Aufbau der URL im Browser dienen: *http://<server>/<app>/<controller>/<action>*. Hätten wir eine Applikation „Automat“, welche Autos verwaltet, mit einem Controller AutoController.groovy, welcher eine *list()* Aktion beinhaltet, und würden diese aufrufen, so würde die Browser-Adresszeile Folgendes beinhalten: *http://<server>/automat/auto/list*. Man muss allerdings anmerken, dass das ein Standardverhalten ist. Bei Bedarf kann man durch die selbst angelegte Konfigurationsdateien dieses Verhalten entsprechend den eigenen Wünschen ändern.

DRY ist ein weiteres Schlüsselkonzept von Grails Framework. Die Abkürzung steht für „Don’t repeat yourself“. Dem Benutzer werden Mittel gegeben, seinen Code nur an einer einzigen Stelle zu schreiben und dennoch in verschiedenen Programmstellen immer wieder zu verwenden.

i18n ist eine Abkürzung des Wortes „Internationalisierung“.

CRUD ist eine weitere Abkürzung und bedeutet „Create Read Update Delete“. Das sind Methoden, die ein Benutzer am häufigsten bei der Bedienung einer Grails Applikation aufruft.

Templating ist ein Prozess der Codegenerierung. Dieses Konzept beschleunigt die Entwicklung immens. In Grails ist es möglich, die Controller und Views zu generieren. Generierte Controller-Klassen beinhalten CRUD Methoden. Die Views enthalten alle HTML-Elemente, um die Variablen der Modell-Klassen mit Werten zu initialisieren oder darzustellen.

Im Folgenden werden einige Begriffe erklärt, die bei der Implementierung verwendet werden:

Model ist eine Groovy Klasse, mit deren Hilfe ein Akteur innerhalb der Web-Applikation beschrieben wird. Anhand der darin beschriebenen Felder werden vom Framework die Hilfsmethoden generiert. Die Modelle in Grails werden in der Regel mit Hilfe der mitgelieferten Tools in der Datenbank gespeichert.

View wird mit Hilfe der HTML und Grails-spezifischen Elementen aufgebaut und bekommt die Daten vom Controller. Diese Dateien sind ein Mittel, um HTML Code nicht mit Groovy Code zu mischen. Die Idee der Views kommt von den Java Server Pages (JSP). Analog dazu werden diese Dateien Groovy Server Pages (GSP) genannt.

Controller sind Groovy Klassen, welche zwischen Model und View interagieren. Sie sind für Verarbeitung der Benutzereingaben aus dem Browser zuständig. Anhand solcher Angaben können z. B. Controller Operationen am Modell vornehmen. Mit deren Hilfe werden auch Daten für die Ausgabe (View) vorbereitet. Auch die interne Abläufe, die bestimmen, wohin die Benutzer nach jeder Aktion weitergeleitet werden, gehören in die Controller.

Services sind den Controllern sehr ähnlich, mit der Ausnahme, dass diese überall in der Applikation verfügbar sind, sobald sie durch eine **Injection** zugeschaltet worden sind.

Injection ist eine Methode, um zur Laufzeit Komponenten miteinander zu verlinken. Mit Hilfe der Injection ist es möglich, die Service-Methoden an mehreren Programmstellen zu verwenden, ohne jedes mal den Code und dessen Abhängigkeiten neu schreiben zu müssen.

Plugin ist ein bereits implementiertes Element einer Web-Applikation, der aus einem externen Repository geladen werden kann. Mit Hilfe der Plugins kann man schnell Funktionen wie Email-Versand oder Sicherheits- und Zugriffsstruktur zum eigenen Projekt hinzufügen.

Grails unterstützt zahlreiche Datenbanken. Eine H2 Datenbank wird in jedes Projekt standardmäßig eingebettet. Wir greifen zu Wartungs- und Debugging-Zwecke allerdings auf eine weitere sehr verbreitete Open-Source-Datenbank zurück – PostgreSQL.

Sobald die Implementierung abgeschlossen ist, wird ein Grails Projekt in der IDE gebaut und so verpackt, dass eine nahtlose Bereitstellung in einem Tomcat Server mög-

lich ist.

2.3.2. Twitter Bootstrap

Ein übersichtliches und benutzerfreundliches Design ist häufig entscheidend für eine Web-Applikation. Darum entscheiden wir uns für Twitter Bootstrap – eine vorgefertigte Sammlung von GUI-Elementen für den Aufbau von Webseiten. Sehr viele namhafte Webseiten wie Twitter, GitHub usw. verwenden dieses Framework. Außerdem hält das Projekt (06.01.2013) Platz 1 in den Kategorien „Popular Starred Repositories“ und „Popular Forked Repositories“ [9][10].

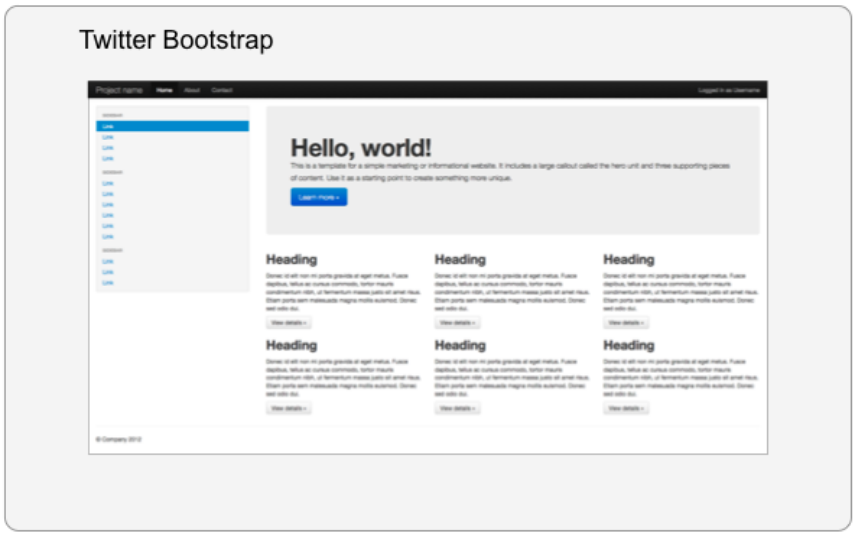

Abbildung 2.4.: Layout mit Twitter Bootstrap. Quelle: [11]

Abgesehen von einem sauberen Design und flexiblem Layout ist auch die Nutzung von Twitter Bootstrap sogar für einen unerfahrenen Webentwickler keine große Herausforderung. Die Entwicklung erfolgt mit reinem HTML. Der Unterschied besteht darin, dass man auf existierende CSS-Klassen zurückgreift.

Da es auch bei GitHub verwendet wird, bekommen wir durch die Nutzung von Twitter Bootstrap ein bekanntes Design, was zu einem schnelleren Zurechtfinden auf der Seite beitragen soll.

3. Analyse

In diesem Kapitel beginnen wir mit der Analyse der vorhandenen Situation und fahren anschließend mit dem Einbringen von Ideen fort, die zur Lösung der im ersten Kapitel genannten Probleme beitragen. Werden wir mit einem Problem konfrontiert, dessen Lösung nicht auf Anhieb ersichtlich ist, versuchen wir, den Ingenieursprinzipien nachgehend, es zu zerlegen, und sich die einzelnen Teilprobleme aufs Neue anzusehen.

3.1. Benutzerprofile

Wir beginnen unsere Analyse mit der Betrachtung der Benutzergruppen. Kennen wir unsere Zielgruppen, so können wir deren Bedürfnisse besser berücksichtigen. Bei dem Versuch, die TestHub-Benutzer zu beschreiben, greifen wir auf einen technischen Bericht zum Thema „Creating a Shared Understanding of Testing Culture on a Social Coding Site“ [1] zurück. Die Ergebnisse einer umfassenden Umfrage, bei der über 600 GitHub Nutzer befragt wurden, davon 33 im einen detaillierten Interview, liefern uns einige Beschreibungen, mit Hilfe derer wir Benutzergruppen definieren können.

3.1.1. Erfahrene Benutzer

Hierzu gehören Entwickler, die bereits erste Erfahrungen mit Open Source besitzen. Sie sind eine wertvolle Gruppe, da wir davon ausgehen können, dass Programmiererfahrung bereits vorhanden ist und aktiv eingesetzt wird. Sollten in dieser Gruppe Benutzer sein, die Code ohne Tests liefern, so wäre es die Aufgabe der Projekteigentümer den Bedarf nach Tests zu äußern und gleichzeitig auf eine Plattform wie TestHub hinzuweisen. Wie in [1] zu finden, muss nicht jedes Codesegment getestet werden. Damit wäre die Steuerung der Testabdeckung dem Projekteigentümer überlassen. Wird ein Code geliefert, der eine bereits mit Tests abgedeckte Funktionalität implementiert, so müssten die Entwickler nicht zusätzlichen Aufwand aufbringen. Uns interessiert aber viel mehr das Gegenteil. Wenn es sich dabei um testbedürftige Commits handelt, braucht der Projekteigentümer ein effektives Mittel, um den Code, der getestet werden muss, hervorzuheben und dessen Testbedarf klar zu machen.

3.1.2. Interessenten

Zu dieser Gruppe zählen die Entwickler, die sich momentan mit „Social Coding“ vertraut machen und womöglich noch keine starren Muster gebildet haben. Es wäre daher

wichtig, von Anfang an den Bedarf nach Software Tests zu äußern. Diese Aussage wird in [1] bestätigt. Einige Projekteigentümer sind der Meinung, allein das Vorhandensein von Tests in dem Projekt sollte ein Hinweis genug sein, um den Entwickler mitzuteilen, dass nicht nur Code, sondern ein getesteter Code erwartet wird. Die Umfrage zeigt aber, dass die Meinungen dazu nicht immer eindeutig sind. Wenn wir uns als Ziel setzen, möglichst Viele damit vertraut zu machen, dass Tests wichtig und willkommen sind, so sollte der Vorgang simpel und strukturiert sein, damit die neuen Benutzer sich nicht überfordert fühlen. Dafür braucht man einen expliziten Hinweis, z. B. in Form einer „Readme-Datei“ im GitHub Repository mit einer kurzen Erklärung und womöglich einem Vorschlag, sich bei TestHub zu beteiligen und dadurch zur Wertsteigerung des Projektes beizutragen.

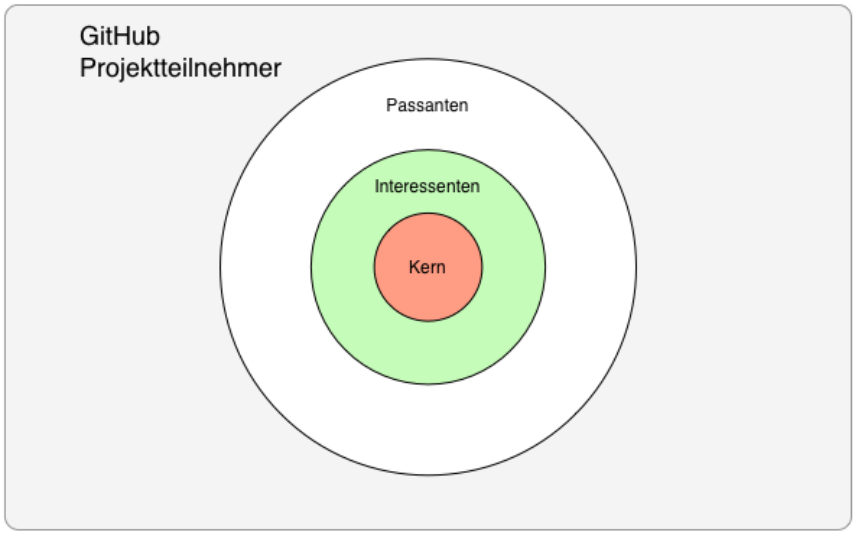

Abbildung 3.1.: Benutzerstruktur bei GitHub-Projekten. Quelle: [12]

3.1.3. Passanten

Eine weitere mögliche TestHub-Benutzergruppe wird in [12] beschrieben. Dort untersucht man ein durch die Plattformen wie GitHub ermöglichtes Phänomen des „Drive-By“ Commits. Die Leichtigkeit, mit der man an der Software-Entwicklung teilnehmen kann, bringt immer mehr Menschen dazu, sich an der Open-Source-Entwicklung zu beteiligen. Interessant dabei ist folgender Fakt: Meistens sind es wenige Entwickler, die sich mit der Hauptfunktionalität beschäftigen. Dabei ist ein hoher Grad an Know-How und eine große Branchenerfahrung häufig eine Voraussetzung. Allerdings erfordert nicht jede Aufgabe lange Einarbeitung und viel Wissen. Wird man sich dessen bewusst, so kann man als Projekteigentümer sich diese Tatsache zunutze machen und durch Strukturierung und Zerlegung der größeren und direkter Verwendung der kleineren Aufgaben eine ausreichende Abdeckung in unkritischen Bereichen durch die bisher nicht beanspruchten „Drive-By“ Commits sichern. Währenddessen können

die erfahrenen Benutzer weiterhin mit der Kernentwicklung fortfahren.

3.2. Ablaufszenarien

Jetzt, wenn die Zielgruppen bekannt sind, überlegen wir, wie die möglichen Abläufe zwischen dem Benutzer und dem System aussehen könnten. Dazu versuchen wir, die Funktionen der TestHub aus verschiedenen Sichten schrittweise zu betrachten.

- Ein Programmierer bemerkt einen kleinen Fehler in der Software, die er nutzt. Er findet das Programm auf GitHub und bietet die Korrektur an. Dazu ist er bereit einen Testfall zu schreiben, damit der Fehler nicht wieder vorkommt.
- Ein Programmierer möchte sich mit Software Tests vertraut machen und sucht nach einer klaren und leichten Möglichkeit, damit anzufangen.
- Ein Projekteigentümer möchte sein Projekt besser strukturieren und wertvoller darstellen, um mehr Interesse seitens Entwickler aufzubauen.

Versuchen wir diese Fälle zu analysieren und ein Werkzeug derart zu entwickeln, dass die oben beschriebenen Szenarien aller Beteiligten effizient gelöst werden können.

3.3. Basissystemanalyse

Greifen wir zu einer „Bottom Up“-Methode, so müssen wir mit dem anfangen, was den Kern unseres Systems bilden soll – Software Tests.

Ein Benutzer soll in der Lage sein, Testfälle aufzustellen.

Es wäre nicht sehr effizient, eigene Testfälle lösen zu wollen, woraus eine weitere Anforderung resultiert. Das Problem „Makers vs. Breakers“ [13, S. 63] liefert uns die Begründung dafür. Die Programmierer arbeiten mit einer kreativen Einstellung. Das Testen erfordert im Gegensatz dazu eine destruktive Denkweise. Außerdem sind die Tester bei fremdem Code objektiv, da es nicht um eigene Fehler geht.

Ein weiterer Benutzer soll die aufgestellten Testfälle implementieren können.

Da wir das Open Source Plattform GitHub als Ausgangspunkt betrachten, werden wir ein System entwickeln, welches uns erlaubt, einen Bezug zu den Projekten auf GitHub herzustellen.

Ein Benutzer soll seine GitHub-Projekte importieren können.

Betrachten wir diese Anforderungen, so kann man an dieser Stelle bereits Basis für ein System erkennen, welches eines Benutzer- und Rechtemanagements bedarf. Wenn wir uns allerdings lediglich damit abfinden, wäre kaum jemand daran interessiert, es zu benutzen. Wir müssen uns also überlegen, welche Mittel uns zur Verfügung stehen,

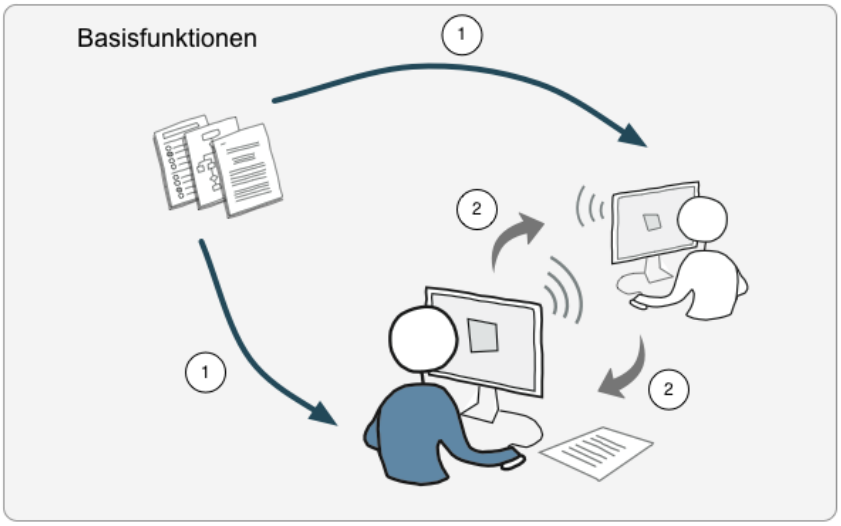

Abbildung 3.2.: Basisfunktionen für TestHub-Benutzer. 1 - Projektimport aus GitHub. 2 - Aufstellen und Implementieren von Testfällen

die neben der Hauptfunktionalität auch die Interesse der Zielgruppe wecken und diese auch konstant erhalten.

3.4. Erweiterungsanalyse

Bisher sind wir davon ausgegangen, dass die TestHub-Nutzer von GitHub auf die Seite geleitet werden und sich daraufhin mit Tests beschäftigen. Dies ermöglicht es, TestHub als eine Art Erweiterung für Tests zu bestimmten Projekten darzustellen. Damit würden wir als Plattformentwickler unseren Benutzer keine Möglichkeit geben, auf TestHub selbst zu navigieren und direkt nach den Projekten zu suchen, die offene Testfälle anbieten. Sollten wir auch diese Benutzer berücksichtigen, so entstehen weitere Anforderungen, die interne TestHub-Funktionen ansprechen.

Ein Benutzer soll in der Lage sein, nach Projekten zu suchen.

Ist ein Entwickler hauptsächlich mit Java-Projekten beschäftigt und hat bereits Kenntnis darüber, wie TestHub funktioniert, oder er entdeckt die Plattform nicht durch einen Hinweis auf der GitHub Projektseite, sondern z. B. durch Suchmaschinen, so soll durch die vorhandene Suchfunktion eine Möglichkeit entstehen, auch selbständig alle Projekte zu finden, die sich auf Programmiersprache Java beziehen.

3.5. Analyse der sozialen Aspekte

Eine weitere Verbesserung gegenüber der Basisfunktionalität kann dadurch entstehen, dass wir soziale Verhaltensmuster des Benutzers berücksichtigen. Auf GitHub ist es möglich, dass ein Benutzer andere beobachtet. Das geschieht meistens, wenn generelles Interesse an Aktivitäten oder Projekten anderer Benutzer besteht. Ist das der Fall, so gehen wir davon aus, dass dieses Interesse sich auch auf die offenen Testfälle erstreckt, und formulieren daraus eine weitere Anforderung.

Einem Benutzer soll eine anhand seiner Interessen bereits vorsortierte Projektliste angeboten werden.

Durch diese Eigenschaft wollen wir die Aufmerksamkeit des Benutzers ohne zusätzlichen Aufwand seinerseits auf ähnliche Projekte lenken. Hierbei implizieren wir durch die Beobachtung auf GitHub vorhandenes Interesse für die Technologie und nutzen es, um die Motivation für die Bearbeitung weiterer Testfälle zu steigern.

3.6. Richtig motivieren

Um den Fall zu verhindern, dass die Nutzung von TestHub zu einer einmaligen Angelegenheit wird, werden wir bei der Planung und Implementierung die Erkenntnisse der Studie über die Motivation in der Lernumgebung [7] berücksichtigen. Dabei formulieren wir zusätzliche Anforderungen, unter anderem zur Implementierung der obigen Erkenntnisse.

3.6.1. Bequeme Umgebung

Die Nutzung des Systems soll mit möglichst wenig Aufwand verbunden sein.

Testen ist häufig ein mühseliger Prozess [2, S. 6] und die Teilnahme an der Open-Source-Entwicklung ist freiwillig. Daher können wir nicht erwarten, dass die Benutzer ein unhandliches System mehr als ein Mal verwenden. Es soll ergonomisch aufgebaut sein. Wichtige Funktionen sollen mittels weniger Mausklicks erreichbar sein, benutzerfreundliche Darstellungen sollen lange Suche vermeiden.

3.6.2. Motivierende Umgebung

Open-Source-Entwicklung erfordert ein hohes Maß an Eigenmotivation. Auch wenn dabei die intrinsischen Motive eine primäre Rolle spielen und sehr vielfältig sind, sind die Mittel zu deren Erhaltung für alle gleich [7]. Damit ergänzen wir unsere Anforderungsliste.

Die Ziele und Vorteile des TestHubs sollen klar sein.

Diese Anforderung kommt aus der Motivationsstudie [7]. Sind wir in der Lage, unsere Benutzer zu überzeugen, dass ihre Tests wichtig sind, werden wir mit einer hohen Beteiligung belohnt. Viele erfolgreich gelöste Testfälle steigern die Selbsteinschätzung und sorgen für hohe Zufriedenheit.

Der Prozess der Testimplementierung soll Neugier und Interesse der Benutzer ansprechen.

Lassen wir die Benutzer auf das Ergebnis deren Testimplementierung in Form einer Bewertung und Rückmeldung seitens der Projekteigentümer warten, lässt dies eine Verbesserung eigener Fertigkeiten zu und steigert die Besucherraten.

Dem Benutzer die Kontrolle überlassen.

Eine hohe Anzahl an Testfällen wird für eine Vielfalt in der Technologie sowie in den Schwierigkeitsgraden sorgen und somit wird sich jeder TestHub Benutzer eine passende Aufgabe aussuchen können.

3.6.3. Profitable Umgebung

Implementierung einer profitablen Umgebung innerhalb des TestHubs.

Ein wichtiger Faktor für TestHub ist die Implementierung eines Bonussystems. Damit ermöglichen wir es dem Benutzer, seine Leistungen zu akkumulieren. Die Teilnehmer mit einer hohen Beteiligung profitieren davon in Form einer hohen Bewertung eigener Projekte und somit durch besseres Ranking der Projekte in der Gesamtliste. Außerdem dient dies als ein starker Motivationsfaktor, indem es Wettbewerb zwischen allen TestHub-Teilnehmern ermöglicht.

3.7. Zusammenfassung

Mit Hilfe der Analyse in diesem Kapitel wissen wir, für wen wir TestHub entwickeln, und welchen Zweck unser System erfüllen muss. Damit sind wir in der Lage, die Szenarien noch einmal durchzugehen – diesmal mit allen Schritten, wie wir diese implementieren werden.

3.7.1. Projekteigentümer

Initialisierung von TestHub beginnt, indem die Projekteigentümer eigene Repositories in TestHub importieren. Für diesen Schritt ist die Eingabe des GitHub-Benutzernamen notwendig. Zu jedem testbedürftigen Projekt werden anschließend Testfälle erstellt. Dazu wird dem Projekteigentümer eine Maske bereitgestellt, in welcher er den Namen des Testfalls, die Zugehörigkeit zu einem Repository, den zu testenden Code und eine entsprechende Beschreibung eingibt. Zur Implementierung der Testfälle wird in einer anderen wissenschaftlichen Arbeit ein Editor entwickelt. Aus diesem Grund wird

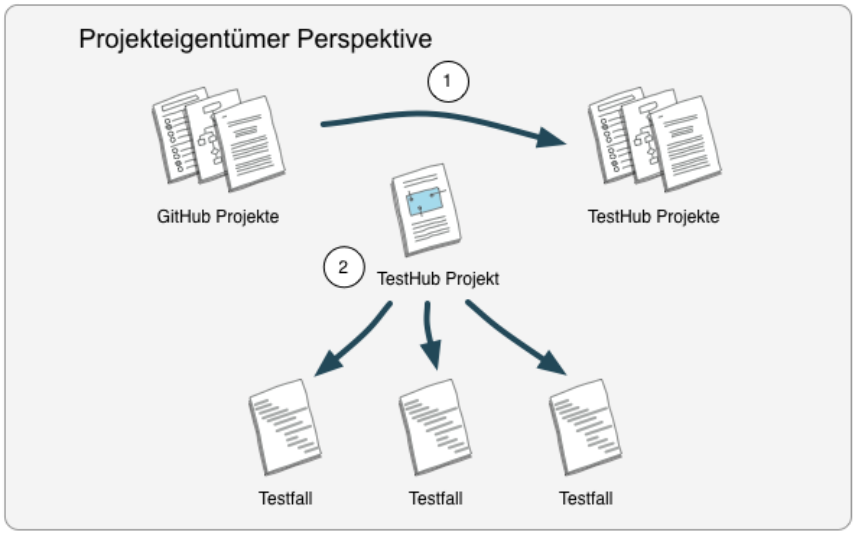

Abbildung 3.3.: Vorgehensweise der Projekteigentümer. 1 - Projektimport aus GitHub. 2 - Erstellen von Testfällen

an dieser Stelle darauf nicht weiter eingegangen. Das System wird nur die Projekte zur Implementierung auflisten, zu denen Testfälle erstellt wurden. Projekte ohne Testfälle werden nur dem Projekteigentümer angezeigt. Damit können zwar alle Projekte importiert werden, eine Veröffentlichung der Testfälle bleibt jedoch dem Eigentümer überlassen.

Da die Testfälle von verschiedenen Personen gelöst werden, bleibt die Qualität nicht konstant. Der Projekteigentümer wird die Testfälle bewerten und einen Kommentar mit der Bewertung hinterlassen. Dadurch entsteht eine Möglichkeit, dem Autor einer Lösung eine Rückmeldung zu geben.

Um den Zugriff auf das System übersichtlicher zu gestalten, werden wir dem Projekteigentümer getrennte Sichten anbieten. Darin wird man noch zu bewertende und bereits bewertete Testfälle finden. Diese Sichten sollen die Navigation erleichtern und zum Komfort beitragen.

3.7.2. Projektteilnehmer

Projektteilnehmer bekommen die Repositories aufgelistet, zu denen Testfälle existieren. Anhand dieser Liste können Projekte entsprechend eigener Vorlieben ausgewählt werden. Nach der Wahl des Projektes werden dem Benutzer alle zu diesem Projekt ausgeschriebenen Testfälle angezeigt. Nach der Wahl eines Testfalls werden alle relevanten Informationen eingeblendet. Entscheidet sich der Benutzer, den Testfall zu implementieren, so wird er zu der entsprechenden Editor-Maske geleitet.

Da der Testfall-Vorgang vom Moment der Erstellung bis zur Endbewertung eine Kommunikation zwischen dem Eigentümer und dem Teilnehmer impliziert, wird an der Stel-

le ein weiteres Detail klar: wir brauchen eine Kontaktmöglichkeit zu jedem TestHub Teilnehmer. Auf die Einzelheiten gehen wir während der Implementierung ein.

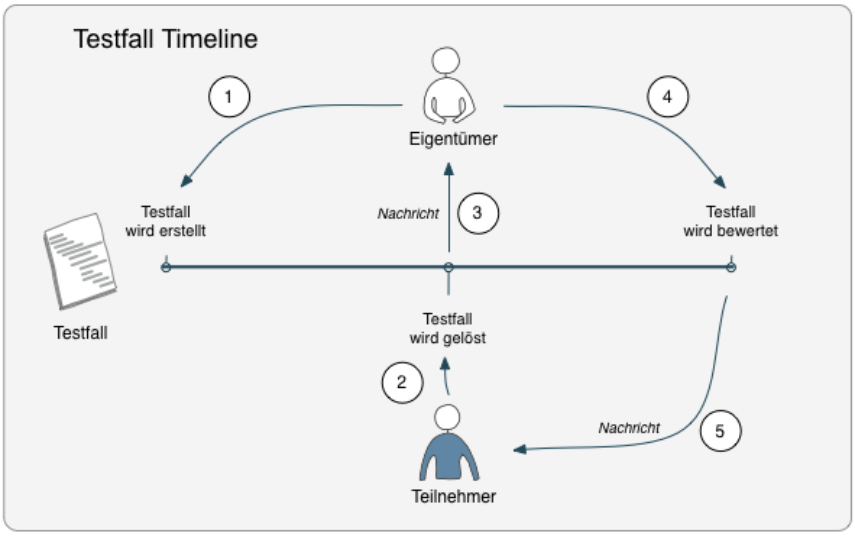

Abbildung 3.4.: Lebensdauer eines Testfalls

Betrachten wir die Kontaktmöglichkeit als gegeben, so entsteht folgender Ablauf:

1. Ein Projekteigentümer erstellt einen Testfall. Der Testfall wird zur Implementierung freigegeben.
2. Ein Benutzer löst einen Testfall.
3. Der Eigentümer des Testfalls wird informiert.
4. Daraufhin bewertet der Eigentümer die Lösung und gibt seinen Kommentar dazu ab.
5. Der Benutzer, der diesen Testfall gelöst hat, wird informiert.

Senden wir dabei in jeder Mitteilung die Links auf die entsprechenden Vorgänge, so erhalten die Benutzer eine Übersicht eigener Aktivitäten und eine gute Möglichkeit, auf dem Laufenden zu bleiben.

Zu beachten ist auch der Fakt, dass sich die Rollen der Projekteigentümer und Projektteilnehmer überschneiden können. Das TestHub-System soll dem Benutzer eine Möglichkeit bieten, eigene Projekte zu verwalten und zusätzlich auch in den anderen Projekten als Teilnehmer zu agieren.

Wächst die Anzahl der Benutzer beträchtlich, so könnte man schnell mehrere Seiten durchsuchen müssen, bis man relevante Systemeinträge findet. Daher brauchen wir eine Suchfunktion, mit deren Hilfe die relevanten Einträge schnell gefunden werden. Da wir bei den Benutzern, die eigene Projekte bereits importiert haben, über zusätzliche Informationen verfügen, können wir diese ausnutzen, um die Suche präziser zu formulieren.

Für die aktiven Benutzer, oder diejenigen, die länger dabei sind, wird zusätzlich eine Übersicht der Testfälle, die sie gelöst haben, benötigt. Dafür ist eine weitere Sicht, in welcher sich bereits bewerteten Testfälle befinden, erforderlich.

4. Ranking

Einige Anforderungen aus dem Kapitel 3 haben bereits das Thema „Ranking“ angesprochen. In diesem Kapitel gehen wir auf die einzelnen Schritte bei der Auswertung von Benutzeraktivitäten und deren Folgen für das Ranking der einzelnen Projekten ein.

4.1. Gemeinsame Bewertungskriterien

Eine der Hauptgründe, wofür wir Ranking in TestHub brauchen, ist die Möglichkeit, jedem Benutzer die Projekte bereits vorsortiert zu präsentieren. Bei der Entscheidung, wie die Projekte zu bewerten sind, spielen mehrere Faktoren eine Rolle.

4.1.1. Basispunkte

Jedes Projekt erhält eine Basisbewertung. Das Ermitteln dieser Bewertung erfolgt im Rahmen einer anderen wissenschaftlichen Arbeit. Es ist geplant, die Punkte zu jedem Projekt über eine API-Schnittstelle abzufragen. Unter Rücksprache mit dem Autor der anderen Arbeit wurde auf die Möglichkeit eingegangen, Basisbewertung als einen Wert in Prozent darzustellen. Anhand dessen wird jedem Projekt momentan die Vergabe von 100% vorgegeben („Default-Wert“).

4.1.2. Bonuspunkte

Wir betrachten die Motivation als einen der wichtigsten Antriebsfaktoren für das Lösen von Testfällen auf TestHub. Dabei setzt sich die Motivation aus mehreren Elementen zusammen. Die wichtigsten sind dabei Neugier, Herausforderung, Rückmeldung auf die eigene Aktionen und Wettbewerb. Wir wollen alle diese Elemente ansprechen, indem wir dem Projekteigentümer die Möglichkeit geben, Testfälle quantitativ zu bewerten.

Betrachtet der Eigentümer die Lösung zu einem Testfall, so kann er mit Hilfe der quantitativen Bewertung dem Autor mitteilen, wie gut er die Lösung findet. Falls erwünscht, wird er die Bewertung in einem begleitenden Kommentar begründen können.

Wir wollen darauf aufbauen und dem Benutzer durch das Lösen von mehreren Testfällen ermöglichen, die Bewertungspunkte als sog. „Bonuspunkte“ auf seinem Konto zu akkumulieren. Damit erreichen wir Folgendes:

- Die Punkte auf dem Konto stellen eine Art Punktestand dar und geben somit dem Benutzer eine Möglichkeit, sich mit anderen zu vergleichen.
- Der eigene Aufwand für bereits gelöste Testfälle geht nicht verloren, sondern wird festgehalten.
- Die Punkte auf dem Konto werden bei der Gesamtbewertung berücksichtigt.

Der letzte Punkt beschreibt ein Schlüsselmoment im Ranking. Erlauben wir dem Benutzer die Punkte zu sammeln und lassen danach diese Punkte in das Ranking einfließen, so geben wir jedem Benutzer die Möglichkeit, durch das Lösen von Testfällen die Bewertung von eigenen Projekten zu beeinflussen. Legen wir diesen Fakt dem Benutzer offen dar, so stärken wir die Motivation jedes TestHub-Projekteigentümers, sich auch als Teilnehmer bei den anderen Projekten zu beteiligen, um Bonuspunkte zu sammeln und damit eigene Projekte im Gesamtranking steigen zu lassen. Dadurch erhalten wir einen Ausgleich zwischen dem Erstellen von Testfällen für die bessere Qualität des Projektes und dem Lösen von Testfällen in den anderen Projekten, um Bonuspunkte zu erhalten.

Um die Beteiligung beim Lösen von Testfällen noch mehr zu erhöhen, lassen wir die Bonuspunkte gleich für alle eigenen Projekte ins Ranking einfließen. Der Fakt, dass *alle* eigenen Projekte von Bonuspunkten profitieren, kann die Projekteigentümer wiederum dazu bringen, die Testfälle in allen Projekten zu erstellen, um den Rang auszunutzen. Das bringt Vielfalt und gibt allen anderen Benutzern bessere Wahlmöglichkeiten.

4.2. Personalisierte Darstellung

Hier beschreiben wir, wie uns die soziale Seite des GitHubs hilft, unseren Benutzern eine Projektliste nach deren Vorlieben sortiert anzubieten.

4.2.1. Soziale Punkte

Das Motto von GitHub lautet: „Build software better, together.“ Die sozialen Motive in der Open-Source-Entwicklung sind das, was GitHub ausmacht. Das spiegelt sich auch in dessen API Schnittstelle. Mit der Initiierung des Import-Vorgangs auf TestHub erhalten wir auch darauf Zugriff, welche Personen der Benutzer beobachtet. Anhand dieser Daten führen wir ein zusätzliches Bewertungskriterium ein. Die sozialen Punkte fließen zwar auch in die Gesamtbewertung mit ein, unterscheiden sich dennoch von den Basis- und Bonuspunkten insofern, dass diese für jeden TestHub Benutzer unterschiedlich ausfallen.

Betrachten wir die Gesamtliste aller Projekte geordnet nach deren Basis- und Bonuspunkten, so wird diese Liste für alle TestHub-Benutzer gleich aussehen. Fügen wir die sozialen Punkte hinzu, so erhalten wir unterschiedliche Projektlisten für jeden Benutzer. Bei der Bestimmung, wie hoch der Anteil dieser Punkte sein darf, berücksichtigen

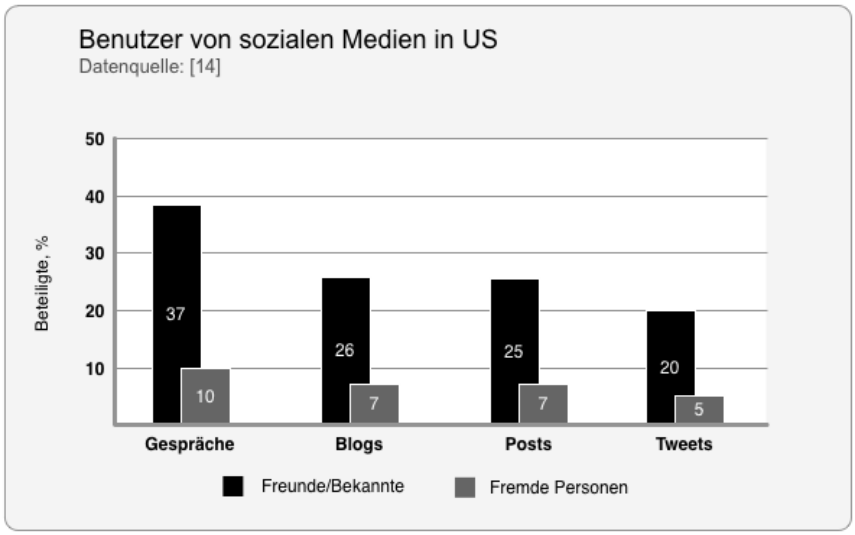

Abbildung 4.1.: Benutzer von sozialen Medien in US, welche Meinungen von Freunden/Bekannten und unbekannten Personen bezüglich Firmen/Produkten vertrauen. Quelle: [15]

wir die Ergebnisse einer Umfrage auf dem US-Markt [14]. Die Ergebnisse zeigen, dass die Empfehlungen aus den sozialen Netzwerken die Entscheidungen bei der Wahl eines Produktes beeinflussen. Dabei spielt die Quelle eine große Rolle. Eine direkte Empfehlung, die von einem Freund ausgesprochen wird, ist weitaus wirksamer als eine Twitter-Mitteilung von einem Unbekannten.

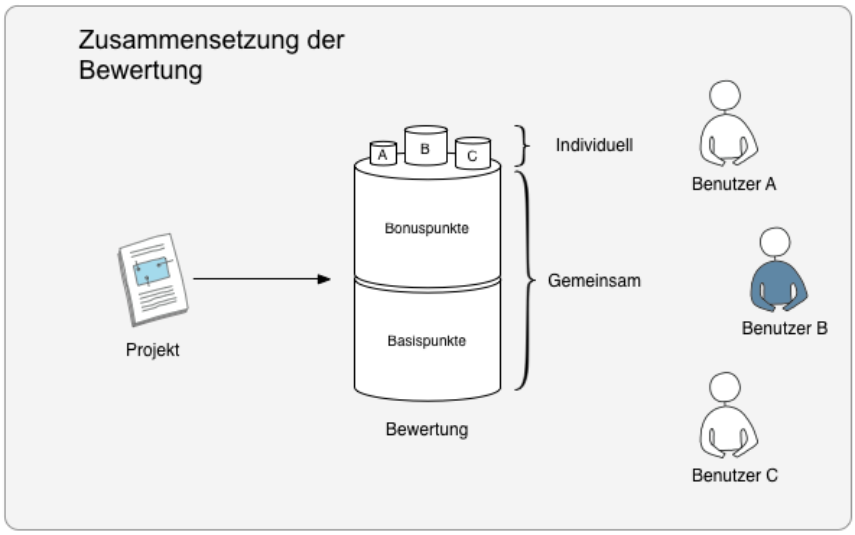

Abbildung 4.2.: Struktur der Projektbewertung.

Da wir bei TestHub die sozialen Informationen berücksichtigen wollen und dabei den

Bekanntheitsgrade der Personen, die unsere Benutzer bei GitHub beobachten, nicht auswerten können, entscheiden wir uns dazu, den kleinsten in der Umfrage ermittelten Einfluss von 5 % zu benutzen.

4.3. Zusammenfassung

Das oben beschriebe Modell der Projektbewertung hängt von mehreren Faktoren ab. Dadurch, dass wir mehrere Quellen berücksichtigen, wollen wir eine stabile und objektive Bewertung ermöglichen. Dabei bauen wir auf dem Grundbedürfnis der Projekte nach Tests, den Aktivitäten der Eigentümer und den sozialen Aspekten zwischen den Benutzer auf.

Durch Ranking wollen wir in erster Linie eine bessere Beteiligung der Nutzer erreichen. Dabei sollen Motivation und Streben nach Verbesserung der eigenen Projekte die primären Antriebsfaktoren sein. Außerdem soll das System durch die personalisierte Ansicht für den Benutzer attraktiver werden.

5. Implementierung

5.1. Wer macht was

In diesem Kapitel beschreiben wir den Aufbau der TestHub Applikation mit Hilfe des modernen Web-Frameworks „Grails“. Zuerst betrachten wir die einzelnen Komponenten und fügen diese am Ende zusammen.

5.1.1. Benutzermanagement

Bei TestHub handelt es sich um eine Web-Applikation. Die grobe Idee ist, dass die Benutzer sich registrieren, um den Zugriff auf die Funktionen des Systems zu erhalten. Da es sich dabei um ein Mehrbenutzersystem handelt, brauchen wir mindestens einen Benutzer, der die administrativen Aufgaben übernimmt. Somit sind wir auf eine Implementierung angewiesen, die mehrere Benutzer sowie deren Rollen berücksichtigt. Außerdem brauchen wir ein Benutzerinterface mit Login- und Anmeldemöglichkeiten.

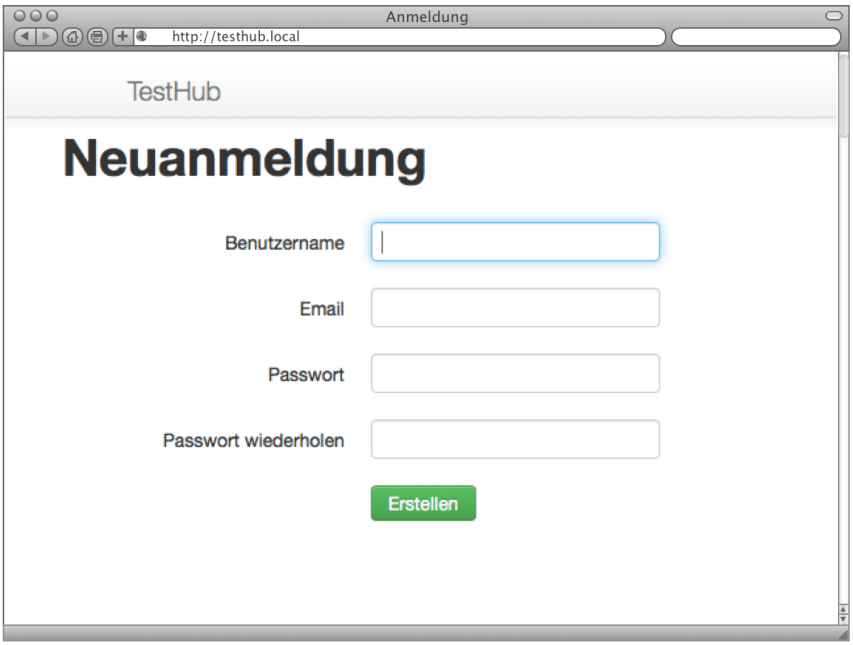

Abbildung 5.1.: Anmeldeformular

Um alle oben genannten Anforderungen zu implementieren, greifen wir zu einem im

Grails Repository bereits existierenden Plugin – Spring Security Core. Nach der Installation können wir die Einstellungen für die Benutzer, Rollen und dem darauf basierenden Berechtigungssystem vornehmen. Dazu rufen wir mit Hilfe der Grails-Konsole das Postinstallationsscript auf und benennen die Klassen für Benutzer, Rolle und URL-Berechtigungen. Dabei entstehen entsprechend die Klassen *User, Role, Requestmap* und die Referenzklasse *UserRole*, die lediglich die IDs von User- und Role-Instanzen beinhaltet.

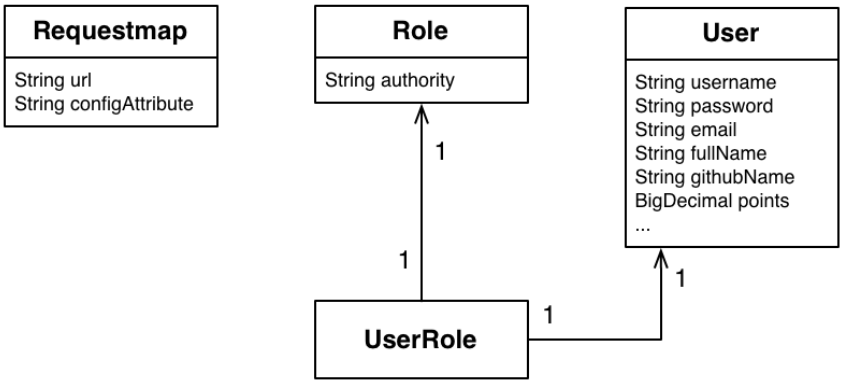

Abbildung 5.2.: Benutzermanagement Klassen

Bevor wir die Implementierung des Benutzermanagements abschließen, müssen wir noch die User Klasse anpassen, sodass diese zusätzliche Variablen enthält.

fullName ein String zum Speichern von Benutzerdefinierten Namen. Dient zur Trennung des Namens von dem Login.

githubName ein String zum Speichern des GitHub-Logins. Wenn gesetzt, wird anhand dieser Variable der Import aus GitHub durchgeführt.

githubName eine Dezimalzahl zum Speichern von Punkten. Darin fließen alle Punkte aus der Benutzerbewertungen ein.

Die restlichen Variablen wie username, password, email usw. sind bereits vom Plugin automatisch implementiert, sowie die zugehörigen Klassenmethoden, Kontroller und zum Teil auch Views mit Hilfe des ergänzenden Plugins Spring Security UI.

Die Klasse *Rolle* erhält den String *authority* zum Speichern der Benutzerrollen.

Die Requestmap-Klasse beinhaltet zwei String-Variablen: *url* und *configAttribute*. Sie werden eingesetzt, um zu ermitteln, welche URL welchen Benutzerrollen zur Verfügung steht. Wollen wir die Rollenverwaltung ausschließlich dem Administrator erlauben, so müssen wir lediglich */role/** * als *url* und *hasRole('ROLE_ADMIN')* als *configAttribute* anlegen. Um den Rest kümmert sich die Logik im Plugin.

Aus Sicherheitsgründen wollen wir restriktiv vorgehen und nehmen daher folgende Einstellung vor: der Zugriff für alle Benutzeraktionen wird zuerst verboten. Wollen wir z. B., dass die Benutzer ihre Profile ansehen oder bearbeiten können, muss danach

ein entsprechender Eintrag stattfinden. Diese und andere Einstellungen werden in der Hauptkonfigurationsdatei gemacht. Sie befindet sich unter /conf/Config.groovy und wird weiter als *Config* referenziert.

Im letzten Schritt tragen wir in die *Config* den Passwort-Algorithmus ein: SHA-512. Damit werden die Passwörter vor dem Eintragen in die Datenbank kodiert.

5.1.2. Nachrichten

Die Nachrichten im TestHub-System werden an mehreren Stellen verwendet. Anmeldung und Kommunikation zwischen den Teilnehmern lässt häufige Aufrufe im Programmcode vermuten. Um den Code sauber zu halten, muss der Aufruf der Email-Funktion am besten so simpel wie möglich gehalten werden. Im Normalfall sollten wir mit der Adresse des Empfängers und der Nachricht selbst auskommen.

Auch an dieser Stelle steht uns ein bereits implementiertes Plugin aus dem Grails Repository zur Verfügung – Mail. Die Installation ist schnell und unkompliziert. Danach müssen wir lediglich ein Email-Gateway in der *Config* hinterlegen.

Die Dokumentation des Plugin bietet umfangreiche Hilfe in Form von Konfigurationsbeispielen an. Nachdem wir die Daten eines speziell für diesen Prototypen eingerichteten Gmail Accounts hinterlegt haben, sind wir fertig und können Emails versenden.

```
1 sendMail {
2    to "user@testhub.local"
3    subject "Hello User"
4    body "Software Tests help to save the world."
5 }
```

Listing 5.1: Aufruf der Mail Funktion

5.1.3. Import von GitHub

Bei der Implementierung dieser Funktion sind wir auf die API-Schnittstelle von GitHub angewiesen. Die Sicherheitspolitik von GitHub schränkt die Anzahl der Zugriffe auf die API Schnittstelle für nicht registrierte Benutzer stark ein. Um eine Sperre zu vermeiden, muss man bei jeder Anfrage einen Schlüssel mitschicken. Den erhält man, nachdem im GitHub-Profil eine Applikation angelegt wurde. Durch das Mitsenden des Schlüssels ist die Limitierung nicht aufgehoben, aber dennoch für die Zwecke der Prototyp-Entwicklung ausreichend erweitert.

Als Antwort auf die API-Anfragen bekommen wir Informationen im JSON-Format. Somit müssen wir bei der Implementierung zuerst eine Möglichkeit suchen, JSON zu parsen. Dafür bietet uns Groovy eine JsonSlurper-Klasse an.

Mit Hilfe dieser Klasse können wir JSON-String für die weitere Verarbeitung zu einer Map zusammenfassen.

```
def url = new URL("https://api.github.com/...") // app id mitsenden
def connection = url.openConnection()
def jsonRepoData = new JsonSlurper().parseText( connection.content.text )
```

Listing 5.2: API Anfrage

Nachdem wir anhand des GitHub-Logins einen Anhaltspunkt für die Anfrage haben, führen wir diese aus und erhalten alle Repositories des Benutzers. Die erhaltenen Daten enthalten detaillierte Informationen, von denen einige für unsere Zwecke nicht relevant sind, es aber später werden könnten. Daher haben wir beschlossen unsere Klassen so zu entwickeln, dass die Repositories und deren Eigentümer die von GitHub gelieferte Informationsstrukturen nachbilden. Da wir mit Hilfe der JsonSlurper-Klasse unsere Anfrage als eine Map vorliegen haben, fangen wir an, durch diese Map zu iterieren und dabei aus jedem Eintrag eine Repository Instanz zu erstellen. Die Eigentümer-Instanz ist dabei in jedem Repository-Eintrag als eine eingebettete Map enthalten und muss separat geparst werden.

```
jsonRepoData.each { repo -> // Jeder Eintrag wird mit "repo" angesprochen
    parseOwner(repo.owner)
    parseRepository(repo)
}
```

Listing 5.3: Iteration durch JsonSlurper Map

Das Erstellen von Objekten wird von Groovy mit der Methode bindData(obj, map) unterstützt. Dabei erfolgt eine automatische Zuweisung von Map-*Values* an die Objekt-Variablen, falls die Variablennamen den Map-*Keys* entsprechen.

```
def parseOwner(oMap) { // vereinfachte Darstellung
    def repoUser = User.findByGithubName(oMap.login)
    def owner = new Owner(user: repoUser)
    bindData(owner, oMap)
}
```

Listing 5.4: Erstellen eines Owner Objektes

Das Klassenmodell unserer Applikation besteht dabei aus dem Repository und Owner, welche die Projekte und deren Eigentümer entsprechend abbilden.

5.1.4. Testfälle

Nachdem ein Benutzer seinen GitHub-Login eingetragen hat, stehen ihm seine importierten Repositories in Form von TestHub-Projekten zur Verfügung. Möchte er nun anfangen die Testfälle für die anderen Benutzer auszuschreiben, so wird er auf der Programmebene die Objekte der Klasse *Issue* erstellen, die Testfälle darstellt. Wie sie genau aussieht, bleibt außerhalb des Fokus dieser Arbeit. Deswegen haben wir bei

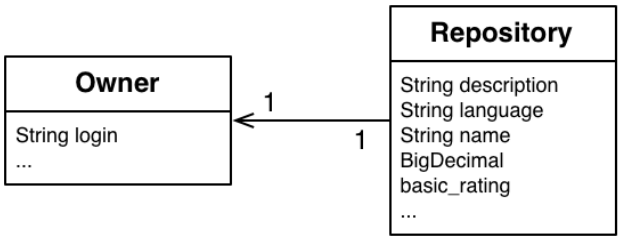

Abbildung 5.3.: Klassen für GitHub Import

den Testfällen nur das Nötigste implementiert. Name, Beschreibung und Code zum Testen werden vom Eigentümer beim Anlegen des Testfalls angegeben und in den Variablen *name, description* und *question* gespeichert. Bei der Angabe der Lösung wird die Referenz auf den Autor in *solutionProvider* gespeichert. Die Bewertung und der Kommentar des Eigentümers kommen entsprechend in *value* und *comment*. Der Status des Testfalls wird mit Hilfe des Enum-Konstruktes festgehalten. Initial ist ein Testfall „offen“. Nachdem die Lösung angegeben wurde, ändert sich der Status zu „gelöst“. Die bereits bewerteten Testfälle erhalten den Status „geschlossen“.

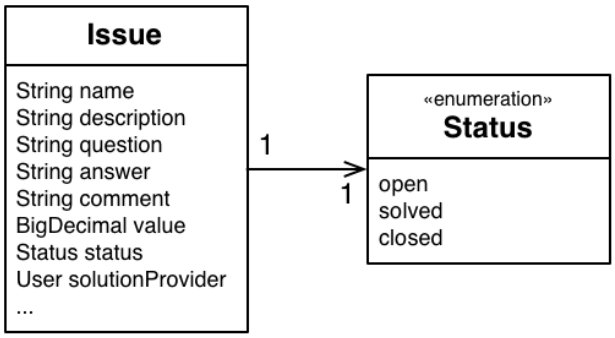

Abbildung 5.4.: Testfall Klasse mit Status Enum

Außer den Standardfunktionen wie CRUD braucht ein Testfall spezielle Methoden z. B. bei der Angabe einer Lösung.

Nachdem ein Testfall vom Eigentümer bewertet wurde, wollen wir die Bewertungspunkte auf das Konto des Autors der Lösung ablegen. Die Methode ist der aus dem Listing 5.5 identisch, abgesehen von der Nachricht und der Punktezuweisung.

issueInstance.solutionProvider.points += issueInstance.value

Jeder Benutzer erhält einen persönlichen Bereich. Darin findet er alle eigenen Projekte, Testfälle, die er noch bewerten muss, und eine Liste mit abgeschlossenen Testfällen. Diese Ansicht entspricht der allgemeinen mit der Ausnahme, dass nach dem Eigentümer gefiltert wird.

```
def updateSolved(Long id, Long version) { // vereinfachte Version
      def issueInstance = Issue.get(id) // Testfall finden
      issueInstance.properties = params // Variablen aktualisieren
      issueInstance.solutionProvider = springSecurityService.currentUser //
          Author der Loesung setzen
      issueInstance.status = Status.SOLVED // Status aktualisieren
      sendMail { // Email Benachrichtigung senden
         to issueInstance.user.email
         subject message(code: 'issue.solved.label')
         html g.render(template:"solutionSubmitted")
      }
   }
```

Listing 5.5: Aktion nach der Lösung eines Testfalls

Um die Nutzung für die TestHub-Teilnehmer noch komfortabler zu machen, implementieren wir eine Suchfunktion. Mit Hilfe dieser Funktion werden die Benutzer alle Projekte und Testfälle nach Schlüsselworten durchsuchen können. Diese Suche kann außerdem noch verfeinert werden. Dazu werten wir das Benutzerprofil aus und prüfen, welche Programmiersprachen er in seinen Projekten verwendet. Danach kann der Benutzer mittels eines einzelnen Klicks die Suchergebnisse auf Projekte mit diesen Sprachen eingrenzen.

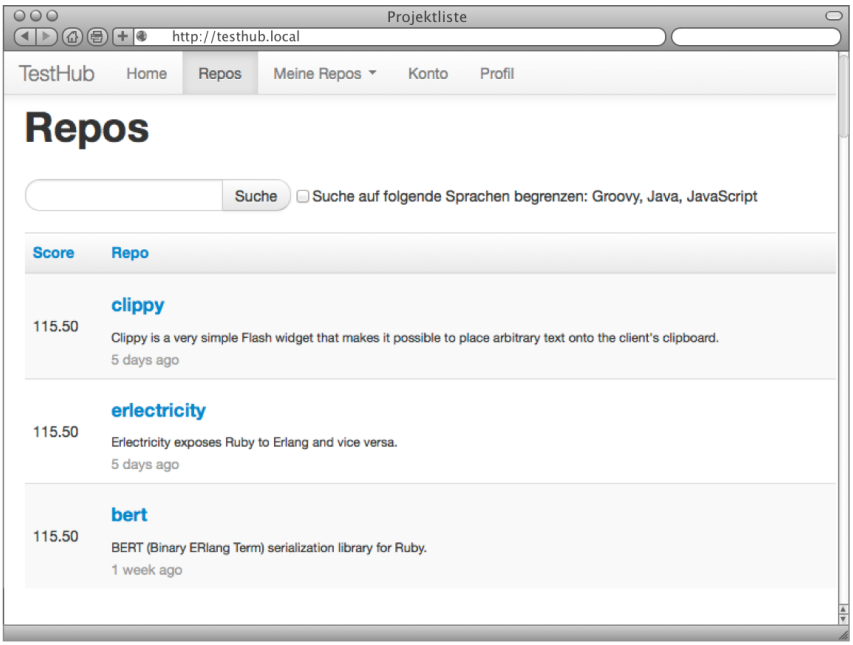

Abbildung 5.5.: Suche mit Filteroption

5.2. Ranking

Die Implementierung des Rankings beginnt in der Repository-Klasse. Die Basispunkte werden für jede Instanz der Klasse auf 100 festgelegt. Als nächstes wollen wir die Bonuspunkte hinzuzählen. Dafür müssen wird durch alle Projekte iterieren und die Kontopunkte der Projekteigentümer zu den Basispunkten des Projekt addieren. Danach werden die Projekte nach dem neuen Wert sortiert und in der View dem aufrufenden Benutzer angezeigt. Nach wie vor müssen wir berücksichtigen, dass die eigenen Projekte nicht mit in die Liste aufzunehmen sind.

```
1  def list() { // vereinfachte Version
2     def me = springSecurityService.currentUser // aktueller Benutzer
3     def sql = new Sql(dataSource) // Datenbank Instanz
4     def queryData = """
5        SELECT r.id as id, (r.basic_rating+u.points) as score
6        FROM repository r
7        JOIN "user" u on (r.user_id = u.id)
8        WHERE r.user_id <> ${me.id} // hier lassen wir eigenen Projekte aus
9        """
10    def rowsData = sql.rows(queryData)
11 }
```

Listing 5.6: Aufbau der Projektliste anhand von Basis- und Bonuspunkten

Außer der allgemeinen Projektbewertung wollen wir noch die Liste so aufbauen, dass sie für jeden Benutzer unterschiedlich ist. Dazu wollen wir die GitHub-Daten ausnutzen.

Die API-Anfrage an GitHub lässt uns die Liste der anderen Benutzer anfordern, die der aktuelle Benutzer auf GitHub beobachtet. Die Ausführung dieser Anfrage für jedes Projekt wäre nicht nur langsam, sondern auch rechenintensiv. Diese Logik wollen wir in das eigene Klassensystem verlagern und die API-Anfragen lediglich dazu nutzen, die Beobachter Listen aktuell zu halten.

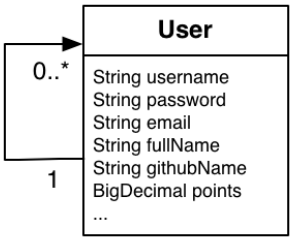

Abbildung 5.6.: User Klasse mit Selbstreferenz

Zuerst müssen wir alle Nutzer unseres System finden, die einen GitHub-Login hinterlegt haben. Danach iterieren wir über diese Liste und senden, je Benutzer, eine Anfrage an GitHub, um die Liste der anderen Benutzer zu erhalten, die er beobachtet.

```
def updateAllUsers() { // vereinfachte Version
      def c = User.createCriteria()
      def githubUsers = c.list () { isNotNull("githubName") } // nur
          Benutzer mit Bezug auf GitHub
      githubUsers.each { user ->
         updateMyFollowingList(user.githubName) // die Liste der
             beobachteten Benutzer aktualisieren
      }
   }
```

Listing 5.7: Aktualisierung der Beobachtungslisten für Benutzer mit GitHub Login

Mit Hilfe dieses Programmdesigns können wir nun beim Aufbau der Projektliste die Abfrage so ergänzen, dass eine zusätzliche Überprüfung stattfindet, ob der aktuelle Benutzer den Projekteigentümer auf GitHub beobachtet.

```
...
(r.basic_rating // Basiswert
   +u.points // Bonuspunkte
   +(CASE
         WHEN (u.id IN (SELECT user_id // Falls User in Liste
                        FROM user_user
                        WHERE user_following_id = ${me.id}))
         THEN round(((r.basic_rating+u.points)/100*5),2) // Punkte um 5 %
             erhoehen
         ELSE 0 // Sonst um 0 erhoehen
      END)) as score
...
```

Listing 5.8: Geänderte Anfragezeile, um Beobachtungsverhalten zu berücksichtigen

5.3. Aktualisierungen

Da sich die Projekte und Beobachtungslisten auf GitHub ändern, müssen wir unsere Daten aktuell halten. Es wäre sicherlich möglich, das mit Hilfe eines Cron-Jobs o. Ä. zu realisieren, aber damit wären wir vom Betriebssystem abhängig. Um diese Abhängigkeit zu vermeiden, können wir dies mit internen Mitteln des Grails-Frameworks realisieren: „Quartz plugin for Grails“.

Mit Hilfe dieses Plugins können wir die Aufgaben samt deren Wiederholzyklen direkt in der Applikation in Form von Groovy-Klassen anlegen. In TestHub brauchen wir zwei solche Klassen, um Projekte und Beobachtungslisten aktuell zu halten. Dabei müssen wir nur Zeitspanne und Methode angeben.

Der Code für Aktualisierung von Beobachtungslisten ist identisch, abgesehen von der aufgerufenen Methode. Die Logik dahinter iteriert durch die entsprechenden Collections, führt für jede Instanz einen API-Aufruf durch und aktualisiert die Daten.

```
class RepoUpdateJob {
   def repositoryService // Service injection
   static triggers = {
      simple repeatInterval: 300000l // Wiederholzyklus: 5 minuten
   }

   def execute() {
      repositoryService.updateAllRepos() // Update Methode
   }
}
```

Listing 5.9: Beobachtung der Änderungen auf GitHub

5.4. Design

Beim Design haben wir uns an GitHub orientiert, um den Benutzern ein Gefühl der vertrauten Umgebung zu bieten. Da GitHub mit Hilfe von Twitter Bootstrap implementiert ist, haben wir dieses Design-Framework auch für das TestHub-Layout benutzt. Normalerweise impliziert die Nutzung von Twitter Bootstrap eine Einbindung von JavaScript und CSS-Dateien in jede Seite. Bei Grails können wir den Aufwand dank des Mesh Layout Systems reduzieren. Man kann sich das so vorstellen, dass für die Darstellung ein einziges Dokument verwendet wird. Lediglich einzelne Bereiche werden darin bei der Darstellung unterschiedlicher Inhalte ausgetauscht. Somit können wir bestimmen, ob wir Code-Änderung in allen View-Dokumenten oder nur in einigen bestimmten vornehmen.

Ein weiteres Plugin aus dem Grails Repository bringt die Twitter Bootstrap Ressourcen mit sich und erlaubt uns, mit nur einer Zeile die Ressourcen einzubinden. Danach stehen unseren Designvorhaben alle Bootstrap-CSS-Klassen zur Verfügung.

Die Entwicklung des Frontends wird durch die Einbindung von Twitter Bootstrap stark beschleunigt. Wir müssen uns nur darum kümmern, wie wir die Informationen darstellen wollen. Ist eine Tabelle oder ein Definitionsset besser, so suchen wir eine bessere Struktur aus und geben die CSS-Klassen an.

Durch die Verwendung von Twitter Bootstrap wird unser Design adaptiv: Es passt sich der jeweiligen Bildschirmauflösung an. Damit sind wir angesichts der steigenden Tendenz permanenter Verfügbarkeit, an einem Rechner oder unterwegs mit Hilfe eines Tabletts auf die Zukunft vorbereitet.

5.5. Zusammenfassung

Während der Implementierung haben wir intensiv die Vorteile eines breit entwickelten Frameworks genutzt. Plugins, Generierung von Code und ein bereits eingebettetes MVC-Muster ermöglichten uns die Entwicklungszeit gegenüber den konventionellen Entwicklungsmethoden zu reduzieren. Die größte Herausforderung bei der Implemen-

tierung lag daran, den Bewertungsmechanismus zu implementieren. Die API-Zugriffe mussten über einen GitHub Account ID authentifiziert werden, um eine Sperre zu verhindern. Nach der Einrichtung aller Plugins und Implementierung der Anforderungen ist ein Klassensystem, wie in Abbildung 5.7 zu sehen, entstanden.

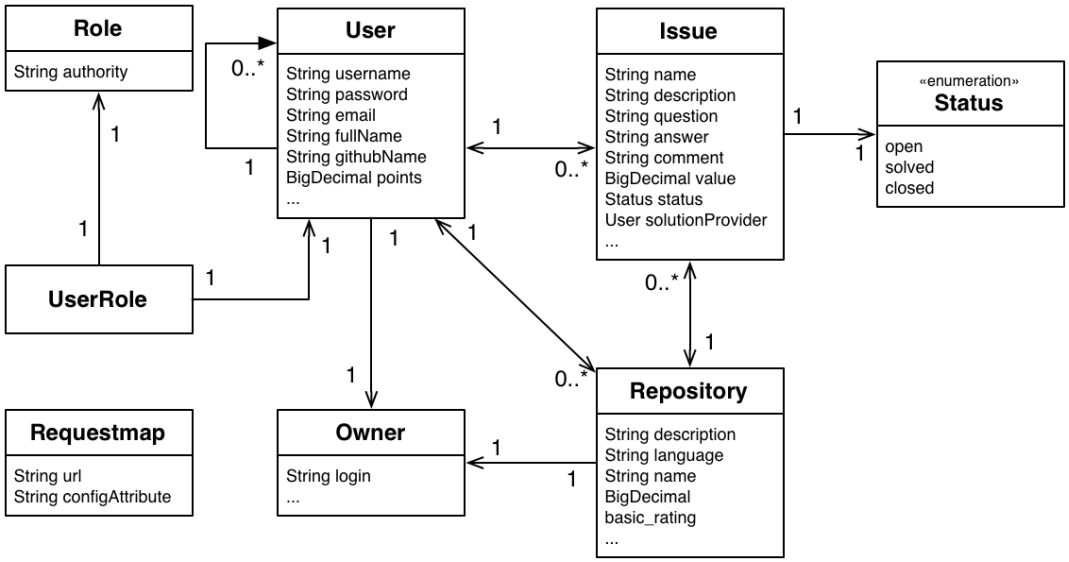

Abbildung 5.7.: Vereinfachtes Klassendiagramm der TestHub

Die Controller-Klassen binden diese Struktur mit Ausnahme des Repository-Controllers vollständig ab. Bei letzterem wurde eine Parallelisierung für einen komfortablen Zugriff auf eigene Instanzen erforderlich.

Insgesamt hat die Implementierung gezeigt, dass der Schwerpunkt bei der Entwicklung von TestHub viel mehr in der theoretischen Ausarbeitung liegt. Sobald ein theoretisches Modell existiert, konnten wir ein passendes Werkzeug – in unserem Fall das Grails Framework – auswählen und anhand des Modells eine schnelle Prototyp-Implementierung vornehmen.

6. Ausblick

Während der Entwicklung von TestHub kamen immer wieder Ideen auf, die für die Implementierung im Rahmen eines Prototypen zu aufwendig wären, sich aber für die Weiterentwicklung als nützlich erweisen könnten. Solche Ideen wollen wir in diesem Kapitel vorstellen.

6.1. Modularität

Diese Applikation wurde so entwickelt, dass die Kopplung zwischen den Modulen schwach ist. Dadurch erreichen wir nicht nur eine bessere Übersichtlichkeit, sondern erlauben auch eine leichtere Weiterentwicklung auf der Basis von TestHub.

In weiteren wissenschaftlichen Arbeiten werden derzeit Module entwickelt, die sich gut ins TestHub integrieren lassen könnten. Darunter ist eine Methode zur Ermittlung des Testbedarfs eines Projekts, die den TestHub-Benutzern eine präzise Auswertung liefert, sowie ein ausgefeilter Editor, der durch seine Features womöglich den Bedarf nach einer IDE aufhebt.

Außerdem könnte man die Integration durch Importmöglichkeiten für andere Plattformen (z. B. SourceForge) ausbauen. Dadurch wären mehr Programmierer angesprochen.

Man könnte sich eine Entwicklung des TestHubs in Richtung einer Lernumgebung vorstellen. Dazu muss eine Differenzierung der Testarten nach Typ und Bewertung stattfinden. Danach könnte man den Benutzern mit schlechten Bewertungen Testfälle vom selben Typ und Programmiersprache als Referenz anbieten. Damit wäre nicht nur der Kommentar bei der Bewertung, sondern auch die vielen positiven Beispiele anderer Benutzer bei der Verbesserung eigener Tests hilfreich.

Statistiken wären eine große Hilfe für die Entwickler des TestHubs, sowie für externe Analytiker. Damit wäre man in der Lage, das Verhalten der Teilnehmer als Gruppe zu beobachten, Trends zu erkennen und bei Problemen womöglich früh zu reagieren. Auch für die Teilnehmer selbst wären die Nutzungsstatistiken über z. B. Anteile der Programmiersprachen und Testarten eine gute Orientierung. Weiterhin wären auch Statistiken auf der persönlichen Ebene denkbar. Diese könnte man für die Darstellung der eigenen Erfahrungsstufe benutzen. Daran wären auch sicherlich Unternehmen interessiert, die nach erfahrenen Mitarbeitern suchen.

6.2. Komfort und Notwendigkeit

Bei der Implementierung von TestHub haben wir der Entwicklung einer komfortablen Umgebung für die Benutzer viel Aufmerksamkeit geschenkt. Eine wirkliche Beurteilung des Komforts erhält man jedoch erst durch direktes User-Feedback. Ein eingebauter Mechanismus zur Meinungserhebung wäre für die Administration von TestHub eine gute Möglichkeit, die Reaktion der Nutzer zu erfahren.

In der heutigen Welt sind Englischkenntnisse ein de facto Standard für die Programmierer. Dennoch könnte es Entwickler geben, die lieber in ihrer eigenen Sprache arbeiten. Dafür bietet Grails einen eingebauten i18n Mechanismus. Sollten die Benutzer es wollen, so könnte die Applikation mehrsprachig werden.

So komfortabel ein System auch ist, es nützt nichts, wenn es unbekannt bleibt. Möchte man dafür werben, so wäre sicherlich ein markantes Design, das zwar an vertraute Entwicklungsplattformen erinnert, aber gleichzeitig auch eigene Erkennungsmarke darstellt, von großer Wichtigkeit.

Eine eigene API würde Plattformen wie GitHub o. Ä. eine Möglichkeit liefern, TestHub in ein eigenes Design zu integrieren und die TestHub Funktionen direkt verfügbar zu machen. Damit wäre es z. B. möglich, auf GitHub die Projekte, die auch auf TestHub präsent sind, mit einem Abzeichen und Anzahl der offenen Testfällen zu kennzeichnen. Somit müsste nicht mehr der Eigentümer selbst die Commiter bitten, Tests zu schreiben, sondern die Notwendigkeit wäre direkt sichtbar.

6.3. Anpassung des Rankings

Erst in einer laufenden Applikation kann man sehen, ob die Ideen, die im Ranking-Algorithmus implementiert sind, plausible Ergebnisse liefern oder nicht. Gegebenenfalls muss man auf die Rankingsschwankungen reagieren und den einen oder anderen Parameter einstellen. Dazu könnte man auch weitere Rankingkriterien einführen. Z. B. man kann auf GitHub nicht nur Personen, sondern auch Projekte beobachten. Damit könnte man analog der bereits implementierten Benutzerbeobachtungslisten die soziale Seite des Rankings ausbauen.

Eine Leaderboard mit den erfolgreichsten Benutzern anhand ihrer Punkte wäre eine starke Motivation für die anderen Teilnehmer. Nützlich wäre dabei die Möglichkeit, die Aktivitäten der Top 10 ansehen zu können, um ggf. deren Lösungen zu studieren.

6.4. Zusammenfassung

Eine Plattform mit Basisfunktionen wie TestHub wird in der Lage sein, akute Bedürfnisse von Open-Source-Projekten nach Tests zu decken. Möchte man damit aus der Position eines Hilfswerkzeugs hinaustreten, so braucht man eine hohe Akzeptanz und

Integration von Open-Source-Netzwerken wie GitHub o. Ä., sowie deren Benutzer. Dafür muss die TestHub Website alles bieten können, was die Nutzer benötigen. Wenn man die bereits durchgeführte Untersuchungen wie [1] betrachtet, so können daraus die wichtigsten Anforderungen abgeleitet werden. Die Erweiterungen können aus den Anregungen der Benutzer oder Ideen der Administration entstehen und sogar von der Open-Source-Community implementiert werden.

A. Anhang

A.1. Inbetriebnahme

A.1.1. Datenbank

Für die Entwicklung des Prototypen wurde eine PostgreSQL Datenbank benutzt. Es ist ratsam, dies als eine Anforderung zu betrachten, da Teile des Ranking-Algorithmus auf der PostgreSQL-Syntax aufbauen. Die Einstellungen zur Datenbank sind in der Datei DataSource.groovy im Verzeichnis conf vorzunehmen. Zuerst werden die Benutzerdaten eingetragen.

```
dataSource {
   pooled = true
   driverClassName = "org.postgresql.Driver"
   username = "testhub"
   password = "th1747"
}
```

Listing A.1: Benutzerdaten und DB Treiber

Danach erfolgt die Eingabe des Datenbanknamens und der URL.

```
environments {
   development {
      dataSource {
         dbCreate = "update"
         url = "jdbc:postgresql://192.168.178.35:5432/thdb"
         dialect = org.hibernate.dialect.PostgreSQLDialect
      }
   }
...
}
```

Listing A.2: DB Name und URL

Während der Entwicklung wurde PostgreSQL in der Version 8.4.10 benutzt.

A.1.2. Server

Für die Bereitstellung der Applikation ist ein Tomcat Server notwendig. Es sind keine besonderen Einstellungen am Server nötig. Während der Entwicklung wurde Tomcat 7 und Grails in der Version 2.1.3 benutzt.

A.1.3. Konfiguration

Vor der Inbetriebnahme müssen die Daten des Administrator-Accounts hinterlegt werden. Dies geschieht in der Datei BootStrap.groovy im Ordner conf. Zuerst legen wir die Rolle an, danach den Benutzer und schließlich weisen wir ihm die Rolle zu.

```
1 def createUsers = {
2    def adminRole = new Role(authority: 'ROLE_ADMIN').save(flush: true) //
        Rolle anlegen
3    def adminUser = new User(username: 'admin', email: "admin@testhub.local",
         enabled: true, password: 'testhub2013') // Benutzer anlegen
4    adminUser.save(flush: true)
5    UserRole.create adminUser, adminRole, true // Benutzer zuweisen
6    }
7 }
```

Listing A.3: TestHub Administrator-Account

Falls erwünscht können in der Config.groovy im selben Ordner die Sicherheitseinstellungen angepasst werden.

```
1 grails.plugins.springsecurity.ui.password.minLength=6
2 grails.plugins.springsecurity.ui.password.maxLength=32
3 grails.plugins.springsecurity.ui.register.defaultRoleNames = ['ROLE_USER']
```

Listing A.4: Sicherheitseinstellungen

Anschließend müssen in der selben Datei die Mail-Gateway-Angaben gemacht werden.

```
1 grails {
2    mail {
3       host = "smtp.gmail.com"
4       port = 465
5       username = "account_name@gmail.com"
6       password = "account_passwort"
7       props = ["mail.smtp.auth":"true",
8          "mail.smtp.socketFactory.port":"465",
9          "mail.smtp.socketFactory.class":"javax.net.ssl.SSLSocketFactory",
10         "mail.smtp.socketFactory.fallback":"false"]
11   }
12 }
```

Listing A.5: Mail-Gateway Einstellungen für Gmail

Damit die Applikation richtig mit globalen URLs umgeht, muss die Server-URL eingestellt werden. In der selben Datei muss die Zeile

grails.serverURL = "http://meine-server-adresse.de:8080/testhub"

angepasst werden.

A.1.4. Installation

Für die Installation muss die Applikation in Form einer War-Datei vorliegen. Eine Grails Applikation kann mit Hilfe der Grails-Konsole gepackt werden. Dies geschieht mit dem Befehl *war app_name*. Die War-Datei wird danach im Projektordner abgelegt.

Die Installation erfolgt durch das Kopieren der Applikation in den webapps-Ordner von Tomcat 7.

/usr/share/tomact7/webapps für Debian Linux.

Anschließend muss der Tomcat Server gestartet werden.

Literaturverzeichnis

[1] Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, Kurt Schneider:
Creating a Shared Understanding of Testing Culture on a Social Coding Site,
http://se.uni-hannover.de/pub/File/pdfpapers/Pham2012.pdf
Software Engineering Group, Leibniz Universität Hannover, Universidade Federal do Rio Grande do Norte, 2012

[2] Glenford J. Myers; Corey Sandler; Tom Badgett:
The Art of Software Testing, 3rd Edition,
John Wiley & Sons Verlag, 2012

[3] Gregory Tassey, Ph.D.:
The Economic Impacts of Inadequate Infrastructure for Software Testing,
http://www.nist.gov/director/planning/upload/report02-3.pdf
National Institute of Standards and Technology, 2002

[4] Stephen O'Grady:
Survival of the Forges,
http://de.slideshare.net/sogrady/survival-of-the-forges
RedMonk, 2011

[5] Brian Doll:
The Octoverse in 2012,
https://github.com/blog/1359-the-octoverse-in-2012
GitHub, 2012

[6] Jerome Buner:
Toward a Theory of Instruction,
Cambridge, MA: Harvard University Press., 1966

[7] Thomas W. Malone, Mark R. Lepper:
Making Learning Fun: A Taxonomy of Intrinsic Motivations for Learning, 1983
in Richard E. Snow, Marshall J. Farr (Eds.):
Aptitude, Learning, and Instruction: Cognitive and Affective Process Analyses (Vol. 3, S. 223-253),
Hillsdale, NJ: Lawrence Erlbaum Associates, 1987

[8] *Grails Webseite*, 2013
http://grails.org

[9] *Popular Starred Repositories*, 2013
https://github.com/popular/starred

[10] *Popular Forked Repositories*, 2013
https://github.com/popular/forked

[11] *Twitter Bootstrap Projekt auf GitHub*, 2013
http://twitter.github.com/bootstrap/examples/fluid.html

[12] Raphael Pham, Leif Singer, Kurt Schneider:
Building Test Suites in Social Coding Sites by Drive-By Commits,
http://se.uni-hannover.de/***
Software Engineering Group, Leibniz Universität Hannover, 2012

[13] Scott Loveland, Michael Shannon, Geoffrey Miller, Richard Prewitt:
Software testing techniques : finding the defects that matter,
Hingham, Mass.: Charles River Media, 2005

[14] *The Faces of Social Media*,
Knowledge Networks, MediaPost Communications , 2011

[15] *Mobile Social Media Usage Affects Shopping Habits*, 2013
http://www.emarketer.com/Article/Mobile-Social-Media-Usage-Affects-Shopping-Habits/1008487

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 07.03.2013

_____

Yauheni Stoliar