Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 77

Untersuchung von bekannten Gamificationsansätzen auf ihre Anwendbarkeit auf das Werkzeug der statischen Codeanalyse FindBugs

Tobias Kuhn

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Stefan Wagner
Betreuer/in:	Dipl.-Ing. Jan-Peter Ostberg
Beginn am:	22. Januar 2016
Beendet am:	22. Juli 2016
CR-Nummer:	D.2.4

Kurzfassung

Werkzeuge zur statischen Codeanalyse sind in der Lage viele verschiedene Fehler und Probleme mit vorhandener Software hervorzuheben. Das Beheben dieser Fehler kann die Software-Qualität merklich verbessern. Entwickler sind jedoch oft nicht motiviert, sich intensiv mit den Analyseergebnissen zu beschäftigen. Neben dem Aufwand die Fehler zu beheben, müssen Meldungen bewertet und Falschmeldungen aussortiert werden.

Die Methode der *Gamification* stellt Ansätze zur Verfügung um die Motivation zu steigern. Zunächst werden verbreitete Ansätze, ihre Hintergründe und Wirksamkeit erläutert. Anschließend werden vielversprechende Ansätze ausgewählt und auf das konkrete Beispiel FindBugs für statische Code-Analyse-Werkzeuge übertragen. Der resultierende Entwurf wird abschließend prototypisch mit praxisrelevanten Technologien implementiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Dieses Kapitel bespricht die Grundlagen dieser Arbeit. Die Hintergründe der Arbeit werden in Abschnitt 1.1 erklärt. Anschließend wird das Ziel in Abschnitt 1.2 erläutert und zum Schluss die Gliederung der Arbeit in Abschnitt 1.3 dargelegt.

1.1 Motivation

Softwareingenieuren steht heutzutage eine große Menge an Werkzeugen zur Verfügung, die alle Phasen der Entwicklung unterstützen können. Dazu zählen auch die Werkzeuge zur statischen Codeanalyse. Obwohl diese teils schwerwiegende Fehler entdecken, ist ihre beständige Verwendung mit Aufwand verbunden:

Die Entwickler müssen geeignete Werkzeuge auswählen und anschließend nicht nur die Konfiguration an das laufende Projekt anpassen, sondern auch die Ausgabe fortwährend überwachen. Dies trifft insbesondere zu, wenn die kritisierten Code-Stellen auf ein komplexeres Problem hindeuten oder womöglich falsch-positive Ergebnisse sind.

Eine Möglichkeit zur Steigerung der Motivation findet sich in der sogenannten *Gamification*. Diese beschreibt wie Spielelemente und -prinzipien auf andere Systeme übertragen werden können. Diese Methode erhöht die Motivation der Nutzer, auch wenn selbige komplexe oder monotone Aufgaben lösen müssen. [Kap12] *Gamification* verwendet dazu unterschiedliche Ansätze, die in dieser Arbeit auf ein konkretes Werkzeug der statischen Codeanalyse übertragen werden.

Die zugrundeliegende Absicht besteht darin, dass stärker motivierte Entwickler Werkzeuge zur statischen Codeanalyse besser einsetzen, sodass mehr Fehler behoben werden und die Qualität der Software und des Prozesses verbessert wird.

1.2 Ziel

Diese Arbeit soll zunächst untersuchen, welche Ansätze der *Gamification* einsetzbar sind. Hierfür sollen unter anderem Ansätze auf Basis von Aufgaben, sogenannten *Quests*, und Punktesystemen näher betrachtet werden. Beispielhaft für alle statischen Codeanalyse-Werkzeuge wird in dieser Arbeit *FindBugs* [Fin] untersucht.

Daher sollen im Anschluss taugliche Ansätze ausgewählt werden, die auf FindBugs angewendet werden können. Hierzu soll ein eigener Entwurf entwickelt werden, wobei insbesondere auf Langzeitmotivation und das mögliche Austricksen des Systems geachtet werden soll. Dies ist notwendig, da *Gamification* die Nutzer dazu verleiten kann, nur das Spiel an und für sich zu spielen ohne dabei die eigentlichen Aufgabenstellung zu lösen. [ZC11]

Letztendlich soll der entstandene Entwurf prototypisch implementiert und die Implementierung evaluiert werden.

1.3 Aufbau der Arbeit

Die Arbeit ist wie folgt strukturiert: Zunächst werden in Kapitel 2 die einzelnen Ansätze der *Gamification* erläutert. Darauf aufbauend beschreibt Kapitel 3 die geplante Anwendung dieser Ansätze sowohl abstrakt als auch durch einen Entwurf der Benutzerschnittstelle. Kapitel 4 erklärt die tatsächlich stattgefundene Implementierung. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick in Kapitel 5.

2 Ansätze der Gamification

Dieses Kapitel beschäftigt sich mit den Grundlagen der *Gamification* und der Herausarbeitung der einzelnen Ansätze, die in den darauffolgenden Kapiteln angewendet werden.

2.1 Gamification

Gamification stellt eine Methode dar, mit der die Prinzipien und Ästhetik von Spielen sowie spielerisches Denken auf bestehende Systeme angewendet werden, um das Engagement, die Motivation und die Leistung der Benutzer zu verbessern. [Kap12] Allerdings lässt sich *Gamification* auch über das gewünschte Resultat definieren: *Gamification* ist die Erweiterung eines Systems oder Service durch das Anbieten von Gebrauchseigenschaften, auch Affordanzen genannt, für spielerische Erfahrungen, um den Wertschöpfungsprozess der Benutzer insgesamt zu unterstützen. [HH12]

Eleganter ist diese Definition, weil einfaches Hinzufügen von Spielprinzipien allein keine spielerischen Erlebnisse schaffen [Kap12], wie sie die *Gamification* zur Steigerung von Prozessqualitäten wie z. B. der Motivation erreichen will. Beispielsweise sammeln Studenten auch ECTS-Punkte um ihr Studium abzuschließen, allerdings werden die wenigsten Studenten mit den Prüfungen ihres Studiums eine spielerische Erfahrung verbinden.

Hamari et al. berichten in einer Literaturanalyse davon, dass eine große Mehrheit der untersuchten Studien eine zumindest teilweise positive Wirkung zwischen *Gamification* und dem erwünschten Ergebnis zeigen. [HKS14] Zu ähnlichen Ergebnissen kommen auch andere Meta-Analysen [Kap12].

2.2 Spielstile

Nach Kapp [Kap12] lässt sich das Spielen als Tätigkeit wie auch einzelne Spiele in drei Arten unterscheiden:

Zunächst gibt es den **kompetitiven** Spielstil. Hierbei treten wenigstens zwei Parteien gegeneinander an. Dabei gibt es einen objektiven Maßstab, der eine willkürlich definierte Leistung jedes einzelnen Spielers im Spiel misst. Basierend auf diesem Ergebnis wird eine Rangfolge festgelegt, die zu einem eindeutigen Sieger oder einem Unentschieden führen kann. Beispiele

hierfür reichen von Spielen, in denen jede Aktion minutiös erfassbar ist, wie z. B. Schach oder Go bis hin zu Spielen, in denen nur Punkte für sehr abstrakte Handlungen vergeben werden, wie z. B. die Einnahme eines Kontrollpunktes in einem *First-Person Shooter* oder das Lösen eines Rätsels in einer bestimmten Zeit in einem Denkspiel. In Videospielen finden sich oftmals künstliche Intelligenzen, die versuchen einen menschlichen Gegner in einstellbarer Qualität zu simulieren.

Daneben gibt es mit dem **kooperativen** Spielstil die Umkehrung dieses Prinzips. Statt gegeneinander zu spielen treffen sich mehrere Parteien, um gemeinsam zu versuchen, ein bestimmtes Ziel zu erreichen. Dabei hängt das Spielerlebnis weniger von der exakten Definition ab, wann das Ziel erreicht ist, da vielmehr die Kooperation selbst das Ziel darstellt. Beispiele hierfür finden sich in modernen *Massively Multiplayer Role-Playing Games*, in denen bestimmte Teile des Spiels nur in einem Team zugänglich sind. Ebenso zählen hierzu Spiele wie *FarmVille*, in dem die soziale Interaktion und der gemeinschaftliche Fortschritt im Vordergrund steht.

Zusätzlich gibt es noch den weniger greifbaren **kreativen** Spielstil. So wie das Spielen eines Instrumentes nicht unter die ersten beiden Stile fällt, gibt es auch in Videospielen das zusätzliche Element der Selbstentfaltung. Hierzu zählen alle Spiele, die dem Spieler größere Freiheit lassen, wie er sein Ziel erreichen kann, oder die womöglich gar kein Ziel vorgeben. Eine Reinform ist beispielsweise der *Creative Mode* in *Minecraft*: Hier steht eine unbegrenze Anzahl an Würfeln zur Verfügung, um beliebige Strukturen zu erbauen. Ebenso gibt es Spiele wie *Spore* oder *Kerbal Space Program*, in denen der Spieler seine Kreatur bzw. sein Raumschiff selbst aus einer Vielzahl vorgefertigten Einheiten erstellen kann.

In der Praxis lässt sich kaum ein Spiel in einem Stil erfassen, sondern es existiert eine Mannigfaltigkeit an Mischformen. Mannschaftssportarten wie Fuß- oder Handball enthalten kooperative Elemente, da mit den anderen Spielern der eigenen Mannschaft gespielt wird. Allerdings will jeder einzelne Spieler die gegnerische Mannschaft in kompetitiver Form schlagen. Dazu sind kreative Freiheiten verfügbar, beispielsweise mit besonderen Spielzügen oder Tricks einzelner Spieler.

All diese Spielstile finden sich mehr oder weniger auch im Prozess der Softwareentwicklung wieder. Das Hauptelement ist dabei sicherlich die kooperative Fertigstellung oder Verbesserung der Software. Während kompetitive Absichten unter Entwicklern schaden können (aber längst nicht müssen), so sind diese für die Beziehung zwischen Tester und Entwickler eher erwünscht. Während der Tester möglichst viele Fehler finden sollte, möchte der Entwickler möglichst fehlerfreien Quellcode schreiben. Ebenso kann zu großer kreativer Freiraum zu Problemen der Lesbarkeit und Wartbarkeit führen, wenn keine einheitlichen Stilrichtlinien für den Quellcode verwendet werden. Allerdings ist er in vielen Projekten vorhanden, deren Sicherheitsaspekte unbedenklich sind. Als konkrete Tätigkeiten zählt dazu z. B. das Herausarbeiten einer Software-Architektur oder eines -Feinentwurfes ebenso wie die Gestaltung einer Benutzerschnittstelle.

2.3 Spielertypen

Bartle entwickelte 1996 eine Einteilung in vier Spielertypen [Bar96]. Dabei lag der Fokus darauf, die Ziele und Interaktionen von Spielern für sogenannten *Multi User Dungeons* zu erfassen. Andreasen und Downey erarbeiteten einen Bartle-Persönlichkeitstest [Bar04], der angibt, inwieweit die befragte Person mit den Spielertypen übereinstimmt. Obwohl Bartle diese Einteilung nur für eine ganz bestimmte Art von Spielen schuf, scheint sie auf andere Spiele übertragbar zu sein. Yee konnte zeigen, dass sich Motivationstypen mittels Hauptkomponentenanalyse von Umfrageantworten sehr ähnlich kategorisieren lassen. Er stellte dabei auch fest, dass sich die Typen nicht gegenseitig ausschließen. [Yee06] Durch ihre intuitive Allgemeingültigkeit ist sie heutzutage so weit verbreitet, dass sie auch in der *Gamification* benutzt wird, um die Motivationen von Spielern zu erklären. [Kap12; ZC11]

Hier sollen kurz die unterschiedlichen Spielertypen nach Bartle dargelegt werden, die von den folgenden Ansätzen abgedeckt werden.

2.3.1 Achiever

Achiever, übersetzt etwa erfolgreiche Personen, agieren in der Spielumgebung. Sie suchen gezielt nach Erfolgserlebnissen im Spiel. Sie lassen sich auf die Herausforderungen ein, die ihnen das Spiel stellt, und versuchen sie bestmöglich abzuschließen. Dabei möchten sie sich mit anderen Spielern messen und zeigen, dass ihre Leistung besser ist. Während ihr Antrieb aus der Herausforderung entsteht, sind die dabei erreichten Erfolge für sie auch ein Statussymbol.

2.3.2 Socializer

Socializer, übersetzt etwa gesellige Personen, interagieren mit anderen Spielern. Sie erfreuen sich daran, andere Menschen kennenzulernen und mit ihnen in Verbindung zu bleiben. Ein zentraler Bestandteil ist für sie die Möglichkeit beispielsweise über Chat oder Sprachübertragung mit anderen zusammenzuarbeiten. Für sie ist das Spiel teilweise nur Mittel zum Zweck. Sie lassen nicht das Spiel ihr Erlebnis gestalten, sondern gestalten ihr eigenes Erlebnis mit anderen Spielern zusammen.

2.3.3 Explorer

Explorer, übersetzt Entdecker, interagieren mit der Spielumgebung. Sie versuchen so viel wie möglich über die Welt und ihre Hintergründe in Erfahrung zu bringen. Ein großes Element ihres Spielerlebnisses besteht in der Perspektive neue Orte, Spielmechaniken oder andere Möglichkeiten zu entdecken. Sie versuchen Details des Spiels über Experimente in Erfahrung

zu bringen. Sie interessieren sich für alle Inhalte der Spielwelt, sogar für die Elemente, die eigentlich durch Programmfehler entstanden sind.

Yee erweitert diesen Typ zu „Immersion“, was ein Eintauchen in die Spielwelt bedeutet, sodass der Spieler so interagiert, als wäre er selbst im Spiel anwesend. Er fasst daher auch das aktive Rollenspiel, das Individualisieren der Spielwelt, das Entspannen und die Flucht vor der Realität zu den Zielen dieses erweiterten Typs. [Yee06]

2.3.4 Killer

Killer, übersetzt Mörder, in manchen Kontexten auch *Griefer*, übersetzt etwa Miesmacher, agieren gegenüber anderen Spielern. Für sie zählt nicht der eigentliche Spielinhalt, sondern Unruhe zu stiften und das Erlebnis der Mitspieler zu beeinträchtigen. Ihre Beweggründe sind das Zurschaustellen ihrer Macht über andere, indem sie ihnen Frustration zufügen. Im Spiel konzentrieren sie sich auf destruktive Aktionen wie Erobern, Zerstören und andere Möglichkeiten Chaos anzurichten.

2.3.5 Fähigkeitsgrade

Dabei gilt es natürlich zu berücksichtigen, dass jeder Spieler einen unterschiedliches Niveau an Fähigkeit im Umgang mit dem System aufweist. Ein Modell hierfür ist die Einteilung von Dreyfus [DD80] in fünf sinngemäß übersetzte Stufen: Einsteiger (*Novice*), Fachkundiger (*Competent*), Erfahrener (*Proficient*), Experte (*Expert*) und Meister (*Master*). Dieses wird auch heutzutage noch verwendet. [ZC11]

Dreyfus macht die Stufenfortschritte dabei an der Weiterentwicklung zu jeweils situationsbedingter Erinnerung, holistischer Einordnung, intuitiver Entscheidungsfindung und absorbierter Wahrnehmung fest. Ein Fachkundiger hat dabei reale Situationen erlebt und eingeordnet; ein Erfahrener hat solche Erfahrung gesammelt, dass er ein Bild vom ‚großen Ganzen‘ gemacht hat. Ein Experte muss nicht mehr aufwändig analysieren, sondern kann seine Entscheidungen intuitiv treffen und die Wahrnehmung des Meister hat sich mit dem System verschmolzen.

2.4 Spielprinzipien

Die folgenden Abschnitte sollen einen kurzen Überblick über verschiedene Ansätze und Methoden der *Gamification* geben. Sie basieren dazu auf [Kap12; Sch14; ZC11].

2.4.1 Fortschrittserfassung

Um den Fortschritt von Benutzern zu erfassen, gibt es Punktesysteme. Punkte erlauben eine vereinfachende Abbildung von einer beliebigen Tätigkeit auf eine numerische Skala. Dies hat gleich mehrere Vorteile. So können Benutzer sich zum Beispiel mit anderen Benutzern anhand von gesammelten Punkteständen vergleichen. Ebenso ermöglicht es dem Systembetreiber einzuschätzen, welcher Benutzer wie aktiv im System ist. Außerdem kann er gegenüber den Benutzern kommunizieren, welche Aktivitäten lohnenswerter, aber auch aufwändiger sind. Die Abgrenzung zu Belohnungssystemen ist allerdings fließend: Da Benutzer Punkte sammeln, könnte man die aufgelisteten Systeme auch als Belohnungssysteme wie unter Abschnitt 2.4.2 auffassen.

Erfahrungspunkte

Ein sehr verbreitetes Punktekonzept sind die sogenannten Erfahrungspunkte, auch *Experience Points* oder in Kurzform *XP* genannt. Jede Aktion im System, die eine Leistung erbringt, wird dem Benutzer mit einer bestimmten Menge an Erfahrungspunkten vergütet. Dabei bestimmt der benötigte Aufwand üblicherweise auch die Menge der als Belohnung erhaltenen Erfahrungspunkte. Dadurch lassen sich die Fortschritte aller Benutzer miteinander vergleichen.

Erfahrungspunkte nehmen üblicherweise strikt monoton zu, d. h. sie verfallen nicht und können auch nicht eingelöst werden. Es ist möglich, ein Maximum an Erfahrungspunkten zu definieren, ab dem alle Benutzer als gleich erfahren gelten, allerdings ist dies nicht nötig.

Üblich ist hingegen die weiter verkürzende Abbildung von Erfahrungspunkten auf Stufen oder *Levels*. Dabei handelt es sich um eine monoton steigende Funktion, die die denkbaren Erfahrungspunkte auf eine üblicherweise zwei- oder dreistellige Zahl reduziert. Dies hat den Vorteil, dass es einerseits den kognitiven Aufwand reduziert, um zwei Spieler zu vergleichen, und eine beliebige Unschärfe einbringt, um den Entwicklungsstand der Spieler besser vergleichen zu können. Zum Beispiel kann ein Benutzer mit 10 Erfahrungspunkten auf Stufe 1 eingeordnet werden und ein Benutzer mit 110 Punkten hat bereits Stufe 4 erreicht, während sowohl 1010 und 1110 Erfahrungspunkte beide Stufe 10 bedeuten.

Währungspunkte

Erlaubt man in einem Punktesystem, die gesammelten Punkte für Belohnungen im System wieder einzutauschen, so handelt es sich um ein Währungspunktesystem. Dessen Bedeutung kann von einfachen Handelssystemen bis hin zu voll entwickelten Wirtschaftssystemen reichen, vergleichbar mit denen „echter“ Währungen. Wegen dieser schnell erreichten Komplexität müssen Währungspunkte gut konfiguriert und überwacht werden, da sie sonst für übermäßige Inflation oder Deflation anfällig sein können, die die eigentlichen Ziele untergraben.

Währungspunktesysteme sind außerhalb der *Gamification* in der Wirtschaft als Bonusprogramme und Kundenkarten verbreitet, die bestimmte Aktionen mit geldwerten Vorteilen vergüten. Da in der *Gamification* der Übergang zu echten Währungssystemen fließend ist, können Währungssysteme auch zu rechtlichen Problemen führen, etwa wenn gewisse Gesetzgebungen sie als Glücksspiel oder echte Währung einstufen, welche nach bestimmten Normen reguliert wird.

Bestenliste

Bestenlisten, auch *Leaderboards* genannt, ermöglichen es den Benutzern sich mit allen anderen Nutzern zu vergleichen, die ihnen womöglich gar nicht bekannt sind. Sie listen alle Benutzer mit einer Platzierung auf, die auf einem gewissen Punktestand beruht. Dies ermöglicht den Benutzern einzuschätzen, ob sie zu den Besten gehören oder wie viele Punkte ihnen im Vergleich mit z. B. einem Freund fehlen.

2.4.2 Belohnungssysteme

Genauso wie Punktesysteme auch Belohnungen aussprechen, so erfassen die Belohnungssysteme auch den Fortschritt. Während die hier betrachteten Ansätze allerdings eher der Nominalskala zugehörig sind, tendieren die unter Abschnitt 2.4.1 genannten Punktesysteme eher zu einer Rationalskala. Während diese ein definiertes Intervall („10 Punkte mehr“) und ein Verhältnis („doppelt soviele Punkte“) angeben können, sind diese Ansätze hier nicht direkt vergleichbar. Zum Beispiel ist nicht intuitiv klar, welches Leistungsverhältnis zwischen zwei grundverschiedenen Errungenschaften besteht.

Errungenschaften

Errungenschaften, auch *Achievements* genannt, werden für das erstmalige Erreichen einer bestimmten Anforderung verliehen. Hier reicht die Bandbreite von einfachen Anforderungen wie „Benutze eine bestimmte Funktion“ über „Vollende eine bestimmte Anzahl an Aufgaben“ bis hin zu „Löse ein Problem auf eine bestimmte trickreiche Art und Weise“. Hat ein Benutzer eine Errungenschaft erreicht, so wird ihm dies explizit mitgeteilt. Zudem gibt es die Möglichkeit dem Benutzer gewisse Errungenschaften zu zeigen, die er noch erreichen kann oder die andere Benutzer bereits freigeschaltet haben. Aus den gesammelten Errungenschaften lassen sich auch Nutzungsprofile herauslesen, wenn Benutzer den Fokus auf bestimmte Anforderungen legen.

Abzeichen

Abzeichen, auch *Badges* genannt, werden dem Nutzer ähnlich wie eine Errungenschaft verliehen. Allerdings ist das Abzeichen selbst eine grafische Darstellung, die die erreichte Errungenschaft repräsentiert. Diese kann beispielsweise an unterschiedlichen Stellen im System zusätzlich zum Benutzernamen für alle sichtbar angezeigt werden.

2.4.3 Direkte Rückmeldung

Für die Benutzer eines Systems spielen alle Aktionen rund um die *Gamification* eine große Rolle: Punkte können für verschiedene Leistungen erworben oder aufgewendet werden, die gegebenenfalls verglichen werden. Belohnungen sollen sofort ausgesprochen werden, damit der Benutzer das Erfolgserlebnis mit der tatsächlichen Tätigkeit verbindet. Dazu muss der Benutzer direkte Rückmeldung von Aktionen bekommen und teilweise auch schon vorausschauend informiert werden, welche Konsequenzen bestimmte Aktionen haben.

2.4.4 Aufgaben

Aufgaben, auch *Quests* genannt, fassen einen oder mehrere zu tätigende Schritte zusammen. Dabei enthält eine Aufgabe auch Angaben über die Punkte, die ein Benutzer durch das Lösen erhalten kann. Aufgaben können durch textuelle Beschreibungen oder individuelles Stilisieren die Nutzer gezielt ansprechen und ihnen einen tieferen Sinn vermitteln. So ist es zum Beispiel im Kontext der Softwareentwicklung möglich, einen Programmierer zu informieren, dass durch das Lösen der Aufgaben ein Standard erfüllt wird, wodurch das Softwareprodukt wertvoller wird. Genauso gut ist es denkbar, dass Aufgaben einem bestimmten Motiv folgen. Beispielsweise können die Benutzer zu Geheimagenten werden, die gefährliche Missionen abschließen.

Aufgaben lassen sich üblicherweise in drei Kategorien einteilen: Zunächst sind sie offen, d. h. sie werden an einer bekannten Stelle ausgeschrieben, sodass sich interessierte Benutzer informieren können. Im zweiten Schritt nimmt der Benutzer die Aufgabe an und versucht sie zu lösen. Dieser Schritt kann, muss aber nicht unbedingt dem System mitgeteilt werden. Zuletzt wird die Aufgabe vom Benutzer abgeschlossen. Dazu muss er dem System nachweisen, dass er die geforderte Leistung tatsächlich erbracht hat. Akzeptiert das System diesen Nachweis, so wird die versprochene Belohnung freigeschaltet.

2.4.5 Ansprechende Lern- und Interessenkurve

Jeder Benutzer hat eine bestimmten Satz an Fähigkeiten und Interessen. Demgegenüber stehen die Anforderungen, die das System an den Benutzer stellen kann, einfach gesagt der

Schwierigkeitsgrad. Das Ziel des Systems muss es sein, seine Anforderungen so an den Benutzer anzupassen, dass dieser weder über- noch unterfordert ist, sondern in den sogenannten *Flow*-Zustand [CC92] höchster Produktivität gelangt.

Darüber hinaus ist auch die Interessenkurve von Bedeutung. Benutzer und Spieler können nicht dauerhaft maximal interessiert und involviert sein. Stattdessen muss das System oder Spiel seine Präsentation und Anforderungen ähnlich wie eine gut geschrieben Geschichte entwickeln. So kann z. B. zunächst das Interesse der Benutzer durch etwas Außergewöhnliches geweckt werden, um dann einen Spannungsbogen aufzubauen, der in einem großen Finale endet. Dabei muss die Interessenkurve auf verschiedenen Ebenen betrachtet werden. Das Gesamterlebnis mit dem System, jede einzelne Stufe, jeder einzelne Tag und jede einzelne Aufgabe sollten so gestaltet sein, dass der Benutzer oder Spieler weiter motiviert wird und nicht das Interesse verliert.

2.4.6 Anpassbarkeit

Ein System, dessen Gestaltung vom Benutzer beeinflusst werden kann, bietet mehrere Vorteile. Findet der Benutzer das System in einer für ihn ansprechenden Gestaltung vor, so ist es wahrscheinlicher, dass er zum System zurückkehren und es regelmäßig benutzen wird. Er hat dadurch auch eine große Freiheit sich selbst gegenüber anderen Benutzern darzustellen, wenn z. B. ein Benutzerbild ausgetauscht werden kann. Zudem kann das Anpassen mit Kosten von Währungspunkten versehen werden, um das Währungssystem zu stabilisieren. Gleichzeitig ergibt die individuelle Anpassung damit auch eine Art Statussymbol, denn der Benutzer muss über ausreichend Währungspunkte verfügen, um sie durchführen zu können.

2.4.7 Einfache Wiederholbarkeit

Spiele sind oftmals mit Fehlschlägen verbunden, z. B. wenn die vom Spieler angewendete Taktik nicht wie geplant funktioniert. Spiele ermöglichen es jedoch einfach zu einem Ausgangszustand zurückzukehren und einen neuen Versuch zu unternehmen. Videospiele ermöglichen beispielsweise das Neuladen eines Abschnitts oder ein weiteres „Leben“ zu verwenden. Brettspiele können frisch aufgebaut werden, jedes Fußballspiel beginnt mit dem Anstoß beim Spielstand von 0:0.

Das Eliminieren oder Verkürzen von negativen Konsequenzen ermuntert die Neugierde und den Drang der Benutzer, das System auszuprobieren und zu erkunden. Die Wiederholbarkeit steht auch deswegen im Vordergrund, da sie erst das Lernen und Verbessern ermöglicht. Ein Anfänger, der das erste Mal in seinem Leben beispielsweise Schach spielt, wird dem Experten in der Regel unterlegen sein, da der Erfahrene bereits unzählige Situationen erlebt und abgewogen hat. Obwohl der Anfänger mit hoher Sicherheit verlieren wird, kann er sehr viel aus dem Spiel lernen und in seinen nächsten Spielen selber anwenden.

3 Anwendung auf FindBugs

Dieses Kapitel erläutert den Entwurf, der aus der konkrete Anwendung der besprochenen Ansätze auf FindBugs entstanden ist. Dabei wurde die Software zwecks der allgemeinen Verständlichkeit auf Englisch entworfen und entwickelt. Daher werden in den nachfolgenden Texten und Abbildungen vermehrt die englischen Fachbegriffe verwendet.

3.1 Verwandte Arbeiten

Es gibt bereits verschiedene Vorgehen *Gamification* bei der Software-Entwicklung einzusetzen. Dubois et al. [DT13] haben eine Methode dargelegt, mit der *Gamification* systematisch dazu eingesetzt werden kann. Sie haben diesen Prozess auf eine studentische Projektarbeit angewendet. Die Projekte wurden einer Metrikenanalyse mittels SonarQube [Son] unterzogen. Damit konnten sie zeigen, dass die entwickelten Artefakte unter dem Einfluss von *Gamification* eine signifikant höhere Qualität aufwiesen als die Artefakte, die herkömmlich entwickelt wurden.

Berkling et al. [BT13] machten allerdings auch gemischte Erfahrungen bei den Anwendungen von *Gamification*. Sie verwendeten *Gamification*, um eine unterstützende Lernplattform zu einer Lehrveranstaltung über Software Engineering zu entwickeln. Die Rückmeldung der Studenten war zu gleichen Teilen positiv wie negativ. Ein entscheidender Faktor könnte dabei die Ästhetik der entwickelten Plattform darstellen. Diese war mit sehr vielen fotorealistischen Grafiken versehen und wirkte auf die Teilnehmer zu sehr wie ein Videospiel.

Eine Übersicht über viele Ansatzpunkte, *Gamification* in der Softwareentwicklung einzusetzen, findet sich zum Beispiel bei Fecher [Fec12], der diverse Einsatzmöglichkeiten während des gesamten Entwicklungsprozesses aufzeigt. Für diese Arbeit relevant ist insbesondere die Code-Überarbeitung während der Implementierung. Hierfür wird ein aufgabenbasierter Ansatz vorgeschlagen, der durch die Projektleitung kontrolliert und mit Belohnungen versehen wird.

Bell et al. entwickelten eine Methode zur *Gamification*, die sie HALO [SBK11] (*Highly Addictive, sociaLly Optimized*) nennen. HALO verpackt Entwickleraufgaben als *Quests*. Dabei wird auf direkte Rückmeldungen an den Benutzer und das Erreichen des *Flow-Zustandes* Wert gelegt. Hierbei soll eine starke Immersion helfen, damit der Benutzer seine volle Aufmerksamkeit auf das System richtet und nicht von äußeren Einflüssen abgelenkt wird. Die Umsetzung des Systems wurde als Plugin für eine Entwicklungsumgebung geplant, wobei ein externer Server zur Datensammlung und -verwaltung zum Einsatz kommen soll.

Bell et al. wandten HALO anschließend in Informatikvorlesungen an, in denen die Studenten eigene Projekte abgeben mussten. [BSK11] Dabei benutzten sie *Gamification* um die Studenten zum Testen zu motivieren. Um die starke Immersion zu gewährleisten, gestalteten sie die *Quests* im Stile eines Comics. Mit Hilfe einer solchen Stilisierung konnten bereits Kiniry et al. [KZ08] Lerninhalte für Studenten in ansprechender Weise aufbereiten. Sie maskierten die Lehre über formale Methoden mit den Motiven der Ausbildung eines Ninjas. Die Rückmeldungen der Studenten zeigten, dass diese Stilisierung gut angenommen und die Inhalte erfolgreich vermittelt werden.

3.2 Anwendung der Ansätze auf FindBugs

Da viele der beschriebenen Ansätze nicht nur angebracht erschienen sondern auch bereits erprobt waren, wurden sie für die Umsetzung der *Gamification* in FindBugs in dieser Arbeit übernommen. Das Kernprinzip bildet dabei die Zusammenfassung von einzelnen Arbeitspaketen zu Quests.

Das Erledigen dieser Quests soll einerseits mit einem Erfahrungspunktesystem („XP“ genannt) und andererseits mit einem Währungspunktesystem („Coins“ genannt) belohnt werden. Darüberhinaus sollen bestimmte Ereignisse auch mit Achievements und Badges prämiert werden. Die Währungspunkte sollen dabei verwendet werden können, um besondere Aktionen im System zu benutzen. Denkbar ist auch eine Verwendung für Belohnungen außerhalb des Systems, wenn ein geeigneter Kontoführer ausfindig gemacht werden kann.

Ebenfalls lohnenswert erscheint die Auftrennung in eine Serverkomponente und ein Plugin für die Entwicklungsumgebung, in diesem Falle Eclipse. Die Serverkomponente soll dabei ein eigene, zeitgemäße Web-Oberfläche besitzen, auf der alle mit dem System möglichen Interaktionen verfügbar sind.

Eine Abwägung bestand darin, ob die *Gamification* auf Entwicklerebene oder auf Projektebene stattfinden soll. Das bedeutet, ob die Nutzer einzelne Entwickler innerhalb eines Projekts oder einzelne Projekte innerhalb eines größeren Portals sind.

Zwar wäre auf Projektebene bestimmt große Motivation vorhanden, sich öffentlich im Vergleich zu anderen Projekten als besonders hochwertig hervorzutun. Andererseits limitiert dieses Vorgehen aber auch die Anzahl der teilnehmenden Projekte, denn somit wäre es nötig, dass Projektdaten öffentlich zugänglich gemacht werden.

Dies ist in vielen Fällen unrealistisch, da es berechtigte Bedenken bezüglich Sicherheit oder Technologiediebstahl geben wird. Ebenso kann ein derartiges Vorgehen durch Verschwiegenheitsverpflichtungen unmöglich gemacht werden. Darüber hinaus wäre auf Projektebene auch die Motivation zur Manipulation größer. Hier findet ein Vergleich „nur“ gegenüber anonymen

Projekten, nicht aber gegenüber individuellen Personen statt. Die Suche und Verhinderung solcher Manipulationen müsste durch externe Prüfungen erfolgen, was wiederum einen größeren Aufwand verursachen würde.

Daher erschien die Entwicklerebene lohnenswerter, denn hier gibt es gleich mehrere Vorteile für die Umsetzung. Einerseits sind die Entwickler auf dieser Ebene bereits mit Werkzeugen für z. B. Versionskontrolle, kontinuierliche Integration und eben statischer Code-Analyse vertraut. Daher wird die Hemmschwelle geringer ausfallen, ein weiteres Werkzeug einzusetzen, dessen Nutzen zunächst vermutlich kontrovers betrachtet wird. Weiterhin kann das System dadurch direkten Zugang zum Quellcode der entwickelten Software erhalten, ohne dass unerwünschte Daten nach außen kommuniziert werden.

Zudem ist es möglich, hier vorhandene Prozessstrukturen zur Überwachung zu übernehmen. Es gibt dort verschiedene Arten von Reviews [LL13], insbesondere Quellcode-Reviews sind weit verbreitet. Für die Entwickler ist es also auch ein bereits bekanntes Konzept, die Arbeit eines anderen als akzeptabel oder nicht einzustufen. Somit erschien es sinnvoll, dass es im System Reviews für die Quests gibt, mit denen sich die Entwickler gegeneinander kontrollieren. Hier gibt es bereits sehr direkte Kommunikationskanäle. Außerdem verbinden die Entwickler mit anderen Entwicklern nicht nur eine anonyme Entität sondern eine konkrete Person. Durch diese Nähe zu Mitentwicklern und Projekt dürfte es ihnen auch bedeutend leichter fallen, die Qualität der Quests oder Lösungen zu bewerten.

Darüber hinaus wird in agilen Prozessen das Konzept der Story Points [CB12] verwendet, was einem Punktesystem der *Gamification* bereits nahe kommt. Hier wird der Aufwand eines Arbeitspaketes in einer abstrakten Punktezahl ausgedrückt. Somit können mehrere Arbeitspakete dann zu größeren Einheiten gebündelt werden. Dabei ist die vergebene Punktzahl zum Aufwand des Arbeitspaketes proportional. Diese Eigenschaften sind ideal für die Auswahl der Belohnung von neu erstellten Quests. Daher erhalten neue Quests ihre konkrete Punktebelohnungen durch ein Mehrheitsvotum.

Entgegen dem Vorschlag aus [Fec12] erschien es jedoch nicht sinnvoll, dass die Projektleitung die neuen Quests erstellt. Oftmals zählt für die Projektleitung nur die Qualität als Ganzes und es besteht kein derart tiefer Einblick in den Quellcode, um genau abzuschätzen, welche Fehler wirklich relevant sind. Stattdessen sind die Entwickler, die tatsächlich am Code arbeiten, am ehesten in der Lage, die Konsequenzen der (Nicht-)Behebung eines Fehlers abzuschätzen. Dies wird dadurch verstärkt, dass sie auch diejenigen sind, die die Behebung durchführen und bewerten sollen. Somit kann jeder Nutzer des Systems auch neue Quests anlegen. Dies schließt auch nicht aus, dass eine besonders aktive Projektleitung sich hier aktiv beteiligt.

Belohnungen sollten dabei insbesondere auch für das Erstellen von Quests und für das Abstimmen in Reviews vergeben werden, denn der bester Questlöser ist nicht unbedingt der beste Questersteller. Stattdessen sollten die individuellen Präferenzen und Fähigkeiten im System berücksichtigt werden. Andernfalls könnte dies einzelne Nutzer demotivieren und die Verwendung von besonderen Funktionen behindern, da diese von der Zielgruppe nicht bezahlt werden können.

Das Ziel soll sein möglichst alle Subtypen von Spielmotivation anzusprechen. Dabei ist zu bedenken, dass diese nicht einzeln, sondern immer in Mischform auftreten. *Achiever* sollen durch die direkten Herausforderungen in Quests und möglichen Statussymbolen wie Abzeichen motiviert werden. *Socializer* sollen motiviert werden, da sie durch Interaktionen im System mit anderen Entwicklern in Kontakt treten können, z. B. wenn sie ein negative Bewertung abgegeben oder bekommen haben. *Explorer* können immer wieder neue Quests erfüllen, die vom Ersteller mit Details ausgeschmückt werden können. Es mag widersprüchlich sein, *Killer* anzusprechen, da deren Ziel eigentlich das Untergraben des Systems ist. Andererseits kann dies auch gewinnbringend zum Aufspüren von Sicherheitslücken oder Fehlern eingesetzt werden. Sie sollen durch ihre Entscheidungsgewalt in den Reviews motiviert werden. Der willkürliche Missbrauch soll durch eine Mehrheitsentscheidung verhindert werden, die nur solche Stimmen belohnt, die mit der Entscheidung übereinstimmen. Gravierende Betrugsversuche bei manipulierten Questlösungen durch *Killer* sollten in den Reviews von anderen Nutzern erkannt werden.

Eine denkbare Erweiterung wäre eine automatische Generierung von Quests anhand bestimmter Motive, wie z. B. mittelalterliche *Fantasy* oder *Cyberpunk Science Fiction*. Ebenso wäre es möglich, ganze Questreihen für Fehler bestimmter Kategorien oder in bestimmten Modulen automatisch anzulegen. Diese Funktionalitäten dürften aufgrund der Komplexität jedoch den Horizont der Implementierung überschreiten.

3.2.1 Quest-Ablauf-Diagramm

Auf Basis der dargelegten Ansätze wurde ein Ablaufdiagramm für Quests entwickelt. Dies verhält sich sehr ähnlich zu einer Zustandsmaschine, denn eine Quest befindet sich im System immer in genau einem Zustand. Der Zustand kann sich nur ändern, wenn eine Aktion im System getätigt wird. Dies wird durch Pfeile dargestellt. Die einzelnen Zustände und Übergänge werden im Folgenden erläutert.

Die Farben haben dabei folgende Bedeutungen: Rote Übergänge können nur von Administrator-Konten ausgeführt werden. Blaue Übergänge erfordern unter Umständen, dass der auslösende Benutzer eine bestimmte Menge an Stufen besitzt. Orangene Übergänge erfordern unter Umständen, dass der auslösende Benutzer eine bestimmte Menge an *Coins* investiert. Grüne Übergänge geben dem ausführenden Benutzer eine bestimmte Menge Erfahrungspunkte.

Die Questzustände wurden dem Ablauf folgend eingefärbt: Noch nicht offene Questzustände sind blau hinterlegt. Offene Questzustände sind grün hinterlegt. Zugewiesene Questzustände sind dunkelblau hinterlegt. Gelöste, aber unbewertete Questzustände sind rot hinterlegt. Geschlossene Questzustände sind lila hinterlegt.

Der erste Teil des Ablaufs vom Neuerstellen eines Quests bis zum offenen Zustand ist in Abbildung 3.1 zu sehen. Zu Beginn der Zustandsabfolge löst der Benutzer aus, dass eine neue Quest angelegt wird. Die Anzahl der gleichzeitig erstellbaren Quests ist dabei abhängig von der Stufe des Benutzers. Dadurch soll verhindert werden, dass neue Benutzer viele Quests

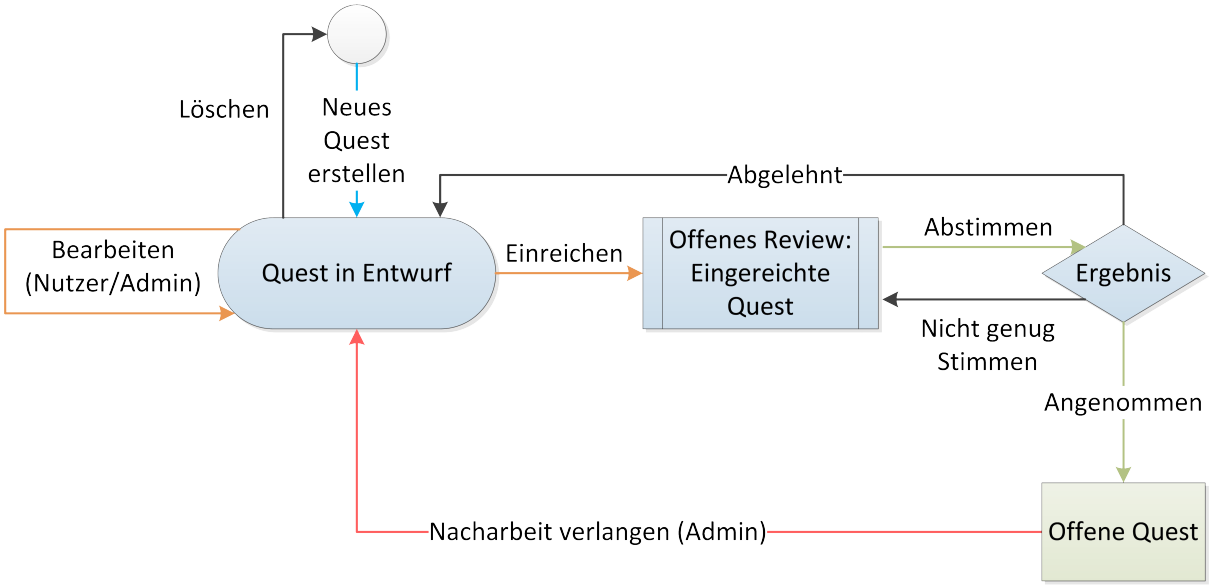

Abbildung 3.1: Erster Teil des Quest-Ablauf-Diagramms vom Neuerstellen zur „offenen Quest“

gleichzeitig anlegen und sich nicht ausreichend auf eines konzentrieren. Die Quest befindet sich dann im Entwurfszustand. In diesem kann es beliebig bearbeitet und wieder gelöscht werden. In späteren Zuständen ist die Quest inhaltlich nicht mehr veränderbar. Dadurch können sich die Benutzer in den späteren Zuständen auf die Angaben zum Umfang der zu erledigenden Arbeit und der Belohnung verlassen.

Es gibt mehrere Fälle, in denen der entwerfende Benutzer *Coins* verwenden muss. Einerseits kann er einen sogenannten *Quest-Timer* festlegen. Dieser bestimmt, wie lange ein Benutzer an der Quest arbeiten darf, bevor er die Arbeit abgeschlossen haben muss. Andererseits kann er auch besonders umfangreiche Quests erstellen, in denen er die Lösung vieler Fehler verlangt, oder er kann die Quests besonders aufwändig gestalten.

Die Bezahlung erfolgt dabei im Schritt des Einreichens. Dadurch wird eine Abstimmung über diesen Questentwurf gestartet. Dabei steigen die Kosten für das Einreichen auch, wenn bereits viele offene Quests im System vorhanden sind. Andererseits soll natürlich das Anlegen von Quests im leeren System angetrieben werden, weswegen dieser Fall mit einer Auszahlung von *Coins* belohnt wird, wenn die Quest akzeptiert wird.

Die Abstimmung über den Entwurf läuft dann so lange, bis eine ausreichende Anzahl Nutzer, z. B. die einfache Mehrheit der registrierten Benutzer, ihre Stimme abgegeben hat. Eine Stimme enthält dabei entweder ein „Ja, akzeptiert“ oder ein „Nein, abgelehnt“. Hinzu kommt, dass jede Stimme einen Vorschlag für die Belohnung der Quest abgibt. Aus der Mehrheit dieser Stimmen wird dann eine Entscheidung abgeleitet, die den Entwurf entweder ablehnt oder akzeptiert. In jedem Fall bekommen alle Nutzer, die eine korrekte Stimme abgegeben haben, Erfahrungspunkte. Eine Stimme gilt als korrekt, wenn ihre Entscheidung mit der Mehrheit

übereinstimmt. Dabei bekommen auch die früher abgegeben Stimmen mehr Erfahrungspunkte als diejenigen, die später stattgefunden haben. Dies soll einerseits verhindern, dass die Benutzer einfach eine Zufallsauswahl treffen ohne die Quest näher zu betrachten, um immer eine Belohnung zu bekommen. Außerdem sollen damit die Stimmen eingeschränkt werden, die sich einfach der bereits etablierten Mehrheit anschließen.

Es handelt sich hierbei um ein Abstimmung offener Natur, in dem alle Benutzer angesprochen werden. Dies ermöglicht beispielsweise, dass ein Entwicklungsteam das Erstellen oder Bewerten von Quests in einem eigenen Entwickler-Meeting durchführen kann.

Wird ein Entwurf abgelehnt, so kann ihn der Ersteller wieder wie vorher bearbeiten. Möchte er ihn jedoch nochmals einreichen, muss er den nötigen Betrag an *Coins* noch einmal entrichten. Falls der Entwurf angenommen wird, so wird die Quest zu einer offenen Quest und kann bearbeitet werden. Neben den Stimmen wird nun auch der Questersteller durch Erfahrungspunkte und ggf. *Coins* belohnt.

Außerdem wird die Belohnung der offenen Quest festgelegt. Dabei wird die Einstellung des Questerstellers in zweierlei Hinsicht berücksichtigt. Falls Stimmen außerhalb eines Unsicherheitsintervalles (z. B. $\pm 50\%$) um die Einstellung des Questerstellers liegen, so wird der Vorschlag trotz akzeptierender Mehrheit abgelehnt. In diesem Fall könnte ein Betrugsversuch mit unrealistisch hohen oder niedrigen Belohnungen vorliegen. Denkbar ist auch, dass die Quest selber uneindeutig formuliert ist und deswegen von den abstimmenden Nutzern falsch verstanden wird. Oder aber es gibt einfach stark abweichende Meinungen über den nötigen Aufwand, um die Quest zu lösen. Auch in diesem Fall ist weitere Kommunikation zwischen den Entwicklern nötig. Wenn jedoch die Stimmen plausibel erscheinen, so wird die Belohnung als Median aus allen Stimmen und der Einstellung des Questerstellers ermittelt.

Einem Administrator ist es bei der offenen Quest möglich, diese zurück in den Entwurf zu überführen, falls sie ihm mangelhaft erscheint oder schwerwiegende Gründe, z. B. rechtlicher Natur, gegen die Anzeige der Quest im System sprechen.

Der zweite Teil des Ablaufs von der offenen Quest bis zum abgeschlossenen Zustand findet sich in Abbildung 3.2. Offene Quests befinden sich nun in einer Art Marktplatz, denn die Benutzer können sich nun umsehen und sich einer bestimmten Quest zuweisen, ähnlich wie in einem Ticket-System wie JIRA [JIR]. Offene Quests können von jedem Nutzer gegen eine bestimmte Anzahl *Coins* hervorgehoben werden, wodurch ihre Sichtbarkeit erhöht wird.

Das Zuweisen oder Annehmen einer Quest durch einen Benutzer bedeutet, dass er die Fehler in dieser Quest beheben will, wenn er die daran festgemachte Belohnung bekommt. Dabei ist für diesen Vorgang aber eine Mindeststufe erforderlich, je nachdem wieviele Quests der Benutzer bereits bearbeitet. Dies soll verhindern, dass vor allem neue Nutzer sich zu viel Arbeit auflasten, die sie am Ende nicht erbringen können. Allerdings soll auch jeder Nutzer nicht zuviele Quests gleichzeitig bearbeiten, weswegen er *Coins* aufbringen muss, wenn er sich mehrere Quests gleichzeitig zuweisen will. Es steht einem Administrator auch frei, Quests explizit an Entwickler zu verteilen, falls er dies möchte.

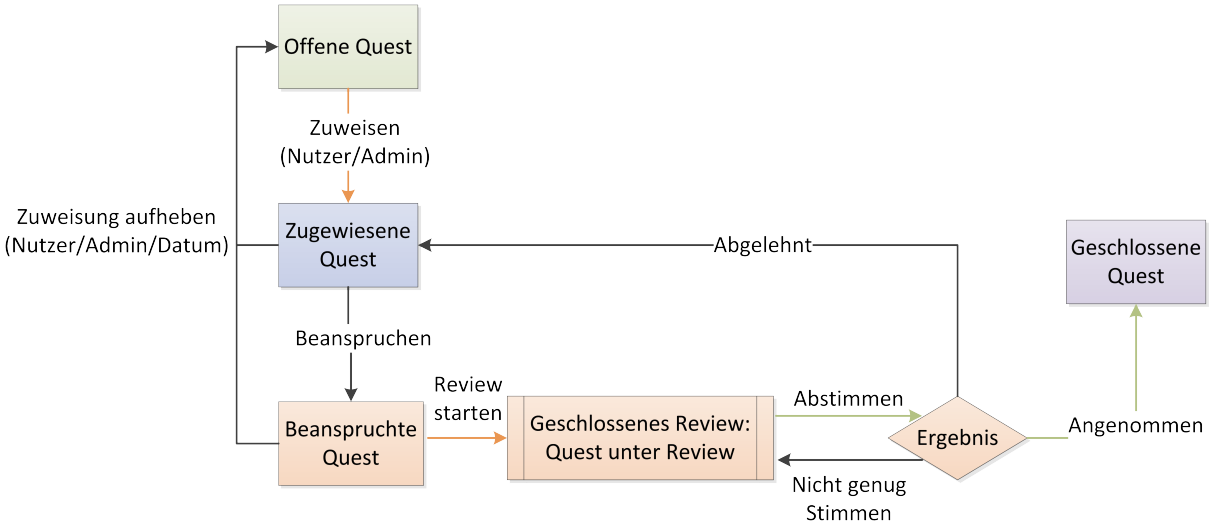

Abbildung 3.2: Zweiter Teil des Quest-Ablauf-Diagramms von der „offenen“ zur „geschlossenen Quest“

Neben dem einfachen Zuweisen an einen Benutzer gibt es auch die Möglichkeit, eine Quest durch zwei Personen kooperativ zu lösen. Dies soll vor allem Praktiken wie Pair Programming oder ein Sherpa-Vorgehen [LL13] unterstützen. Im Sherpa-Vorgehen gibt ein erfahrener Entwickler sein Wissen an einen unerfahrenen weiter. Hierzu kann ein zweiter Benutzer als Zweitbearbeiter eingetragen werden, der dieser Eintragung separat zustimmen muss. Dabei wird auch eine grobe Prozentaufteilung der Belohnung festgelegt, die allerdings aufgrund der Kooperation mit einem Bonus versehen wird. Dieser Bonus fällt größer aus, je mehr Stufen zwischen den beiden Benutzer liegen. Der Bonus für den Benutzer mit der niedrigeren Stufe ist dabei erheblich größer als der Bonus für denjenigen mit der höheren Stufe. Dies soll den Wissenstransfer fördern und es ermöglichen, dass auch neue Benutzer schnell in das System einsteigen können. Um Betrug für einzelne Benutzer unattraktiv zu machen, ist die für jeden Benutzer einzeln maximal kooperativ erreichbare Belohnung allerdings auf die festgelegte Belohnung der Quest begrenzt.

Der Benutzer hat jederzeit die Möglichkeit, sich aus einer zugewiesenen Quest wieder auszutragen. Ebenso steht diese Option dem Administrator zur Verfügung, um Situationen zu lösen, in denen ein Mitarbeiter, z. B. krankheitsbedingt, nicht mehr die Arbeit übernehmen kann. Es erlaubt ihm als Teilschritt außerdem eine zugewiesene Quest zurück in den Entwurfsstatus zu verschieben.

Allerdings kann dies auch passieren, wenn der sogenannte *Quest-Timer* abgelaufen ist. Dabei handelt es sich um ein Abgabedatum, zu dem der zugewiesene Entwickler spätestens die Quest abgeschlossen und beansprucht haben muss. Ist dies nicht der Fall, so wird er als Bearbeiter aus der Quest entfernt und die Quest steht wieder für alle zur Verfügung. Die Dauer des *Quest-Timers* kann entweder vom Questersteller festgelegt werden oder ein Standardwert, etwa eine Arbeitswoche, wird vom System festgelegt. In diesem Fall ist es dem Bearbeiter möglich, den

Quest-Timer gegen eine Zahlung von *Coins* hinauszuzögern. Dabei wird die Zahlung umso teurer, je länger der beantragte Zusatzzeitraum ist.

Um zu verhindern, dass Benutzer den *Quest-Timer* umgehen, indem sie sich einfach austragen und direkt wieder zuweisen, gibt es für die Benutzer eine Warteperiode von wenigen Tagen, wenn sie sich die gleiche Quest noch einmal zuweisen wollen.

Hat ein Benutzer die Quest gelöst und alle nötigen Arbeitsschritte abgeschlossen, so kann er die Quest und dessen Belohnung beanspruchen. Dies führt dazu, dass ein geschlossenes Review gestartet wird. Hierfür werden bis zu drei zufällige Entwickler bestimmt, die die Qualität der Lösung der Quest bewerten sollen. Es steht dem beanspruchenden Nutzer aber auch frei, bestimmte Entwickler gegen *Coins* zu benennen. Dabei wird der Preis für die Nominierung eines Benutzers zum Reviewer durch ein *Sliding Window* bestimmt: Je öfter der Benutzer in den vergangenen Wochen nominiert wurde, desto teurer wird jedes weitere Mal.

Hat eine Mehrheit der Reviewer für oder gegen das Akzeptieren der Lösung gestimmt, so erhalten wie bei der Abstimmung über eingereichte Quests alle abstimmenden Benutzer Erfahrungspunkte in Abhängigkeit, ob ihre Stimme mit der Mehrheit übereinstimmt. Ebenso erhalten frühere Stimmen wieder mehr Erfahrungspunkte. Wurde die Lösung der Quest abgelehnt, so wird es wieder den ursprünglichen Bearbeitern zugewiesen. Der *Quest-Timer* wird während der gesamten Abstimmung angehalten.

Ist die Lösung akzeptiert worden, so wird die Quest geschlossen und die Belohnungen an den Bearbeiter und den eventuell vorhandenen Zweitbearbeiter ausgezahlt.

3.2.2 Wirtschaftsübersicht

Neben den Quests sollen den Benutzern weitere Möglichkeiten gegeben werden, *Coins* für Gegenleistungen zu investieren. Eine Übersicht über die Möglichkeiten Coins zu verdienen und auszugeben findet sich in Abbildung 3.3. Die Raute in der Mitte bedeutet dabei die Menge an *Coins* in einem Benutzerkonto. Die orangenen, abgerundeten Rechtecke darüber sind die Zuflüsse. Darunter befinden sich die blauen Rechtecke, in denen es eher um Gestaltung des Systems geht, die grünen Rechtecke, in denen es um den Entwurf von Quests geht, und die grauen Rechtecke, in denen es um die anderen Funktionen während des Quest-Ablaufes geht.

Benutzer können primär *Coins* verdienen, indem sie Quests lösen. Darüber hinaus können sie in neuen Projekten eine gewisse Anfangsmenge bekommen und werden beim erfolgreichen Einreichen von neuen Quests in ein leeres System belohnt.

Dem gegenüber stehen einerseits die Möglichkeiten, das System gegen *Coins* nach ihren Vorstellungen zu gestalten. Dazu gehört das Festlegen eines persönlichen Titels und eines Avatars, einem Benutzerbild. Außerdem soll es ihnen möglich sein, die farbliche Gestaltung der Oberfläche anzupassen und offene Quests hervorzuheben.

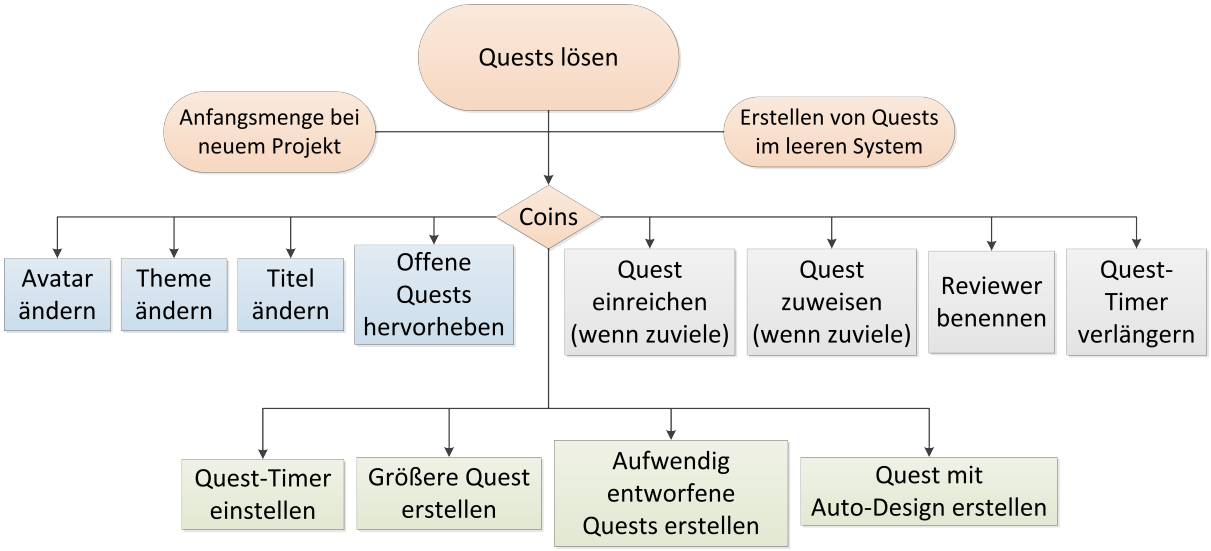

Abbildung 3.3: Übersicht über die Zu- und Abgänge der *Coins*-Währung im System

Beim Entwurf einer Quest können *Coins* investiert werden, um diese besonders umfangreich mit vielen Teilaufgaben anzulegen. Es ist möglich, den im Quest-Ablauf besprochenen *Quest-Timer* auf einen festen Wert zu setzen, den der Bearbeiter nicht mehr beeinflussen kann. Außerdem ist es möglich, die Quests mit besonderen gestalterischen Mitteln zu erstellen. Hier ist auch nochmal die Möglichkeit angedacht, eine Quest mit einer Automatik erstellen zu können, deren Umsetzung jedoch nicht im Rahmen dieser Arbeit möglich ist.

Auch während des Quest-Ablaufs gibt es mehrere Möglichkeiten, *Coins* zu investieren. Wenn beispielsweise bereits viele offene Quests existieren, kann ein Benutzer damit noch mehr Quests einreichen. Das gleiche gilt für das Zuweisen von Quests, wenn der Benutzer aufgrund seiner Stufe mehrere Quests gleichzeitig erledigen darf. Während des Bearbeitens kann er zudem gegen Zahlung einen nicht festgelegten *Quest-Timer* verlängern. Nach dem Bearbeiten ermöglichen *Coins* es, dass ein Nutzer einen Teil der Reviewer nach seinen Wünschen auswählen kann.

3.3 Mockup

Basierend auf diesen Konzepten wurden einige Mockups erstellt, um die grafische Benutzerschnittstelle zu planen und die Umsetzung anzuleiten. Alle Abbildungen wurden mit Balsamiq Mockups [Bal] erstellt.

3.3.1 Web-System

Die Umsetzung soll auf einer Weboberfläche basieren, in der alle Daten verwaltet werden können. Diese soll dabei ähnlich zu anderen webbasierten Werkzeugen wie z. B. dem Review-Werkzeug Gerrit [Ger] gestaltet werden, da Softwareentwickler so bereits mit einer ähnlichen Benutzungsschnittstelle vertraut sind. Das für diesen Kontext professionelle Aussehen macht es auch unwahrscheinlicher, dass die Entwickler die Plattform von vornherein ablehnen werden.

Dabei sollen mehrere Seiten entstehen:

- eine Seite für die Projektübersicht
- eine Seite für die Ranglisten
- eine Seite für die Errungenschaften
- eine Seite um neue Quests zu erstellen
- eine Seite um alle Quests aufzulisten
- eine Seite für an die Benutzer gerichtete Nachrichten

Der Mockup für die Seite um neue Quests zu erstellen ist in Abbildung 3.4 zu sehen. Er zeigt zudem einige Elemente, die auch in den folgenden Abbildungen vorhanden sind. Dazu zählt eine Titelzeile und ein Menü, das es ermöglicht zwischen den einzelnen Seiten zu navigieren. Ebenfalls im Menübereich befindet sich eine Anzeige diverser Informationen über das eigene Benutzerkonto. Hier werden das eventuell gewählte Benutzerbild, der Benutzername und eventuelle Titel, die eigene Stufe sowie die erlangten Erfahrungs- und Währungspunkte angezeigt. Des Weiteren wird hier auch darüber informiert, wie viele Erfahrungspunkte für das Erreichen der nächsten Stufe fehlen. Dies wird zusätzlich mit einem Fortschrittsbalken angezeigt.

In der konkreten Maske für das Bearbeiten der Quest finden sich mehrere Gruppierungen an Elementen. Zunächst finden sich die einstellbaren Eigenschaften der Quest. Darunter finden sich die Aufgaben, sprich die FindBugs-Meldungen, die in dieser Quest erledigt werden sollen. Rechts oben befinden sich eine Funktion zum Speichern sowie die Möglichkeit, die Übergänge für das Löschen und das Einreichen direkt auszuführen.

Die Quest hat zunächst einmal grundlegende Einstellungen wie Titel und Beschreibung. Des Weiteren kann hier für die Belohnung in Erfahrungs- und Währungspunkten ein Vorschlag gemacht werden, der den Rahmen für die spätere Abstimmung vorgibt. Weiterhin kann hier der in Abschnitt 3.2.1 beschriebene *Quest-Timer* festgelegt werden. Zudem würde sich hier eine Auswahl für die Funktion befinden, die automatisch eine ansprechende Quest aus den ausgewählten Aufgaben erstellen kann.

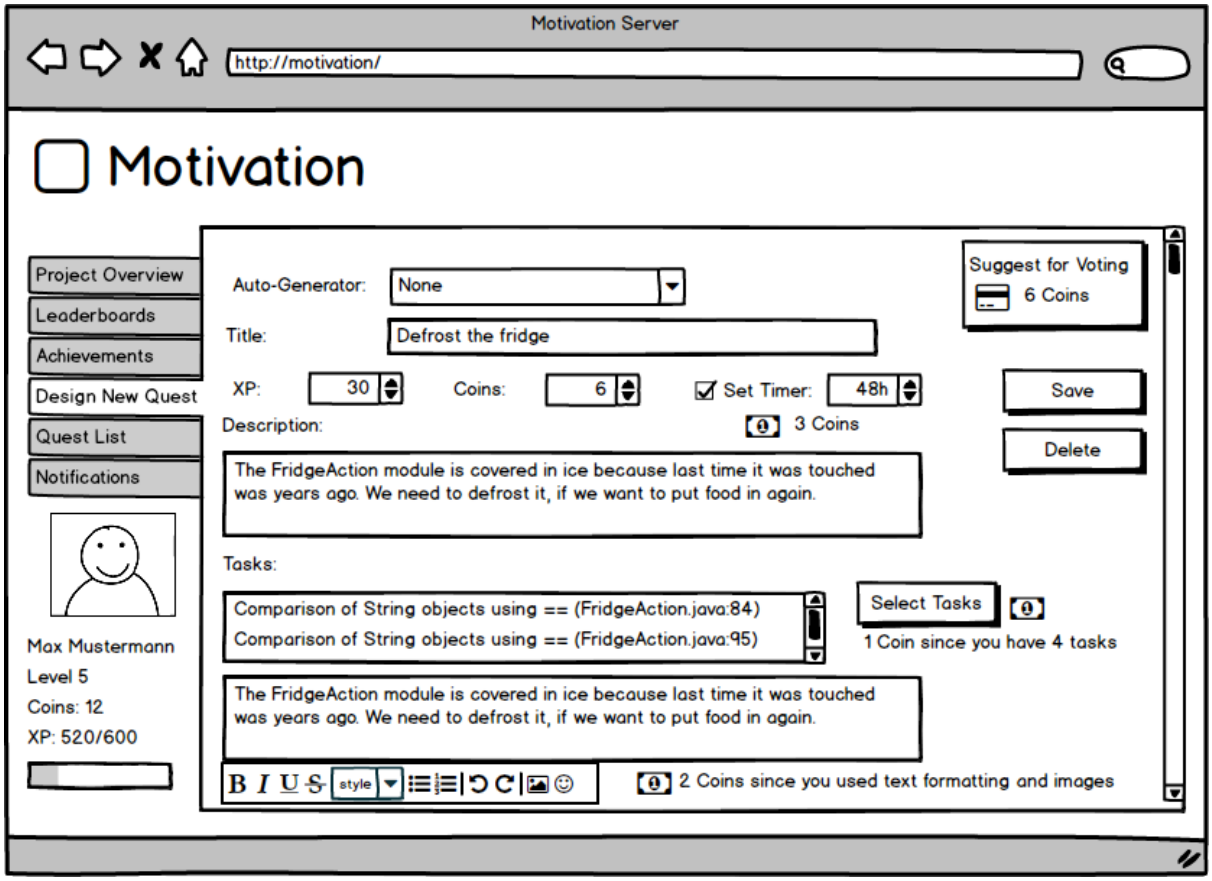

Abbildung 3.4: Mockup für die Seite der Weboberfläche in der neue Quests erstellt werden

Im unteren Teil lassen sich dann die einzelnen Aufgaben auswählen und auf Wunsch mit einer Beschreibung gestalten. Jede Aufgabe entspricht dabei dem Lösen genau einer FindBugs-Meldung. Dabei soll das Hinzufügen, Löschen und Umsortieren von Aufgaben jederzeit beliebig möglich sein.

Abgesehen davon soll versucht werden, die einzelnen Elemente hervorzuheben, deren Einstellung einen Einfluss auf die Anzahl an *Coins* hat, die beim Einreichen zu zahlen sind. Dadurch kann der Nutzer gezielt die kostenpflichtigen Funktionen verwenden, die er für lohnenswert erachtet.

Der Mockup für die Seite um alle Quests aufzulisten ist in Abbildung 3.5 zu sehen. Diese listet alle Quests in einer Baumansicht auf. Die obersten Kategorien entsprechen dabei den in Abschnitt 3.2.1 besprochenen Zuständen. Unter den Kategorien werden alle Quests dieser Kategorie aufgelistet. Dabei soll den Nutzern möglichst viele Informationen über den Zustand der Quests zur Verfügung gestellt werden, wie z. B. die zu erwartende Belohnung.

Zu sehen ist außerdem, dass eine Quest durch ein fettes Schriftbild und auffallende Farbe hervorgehoben wurde.

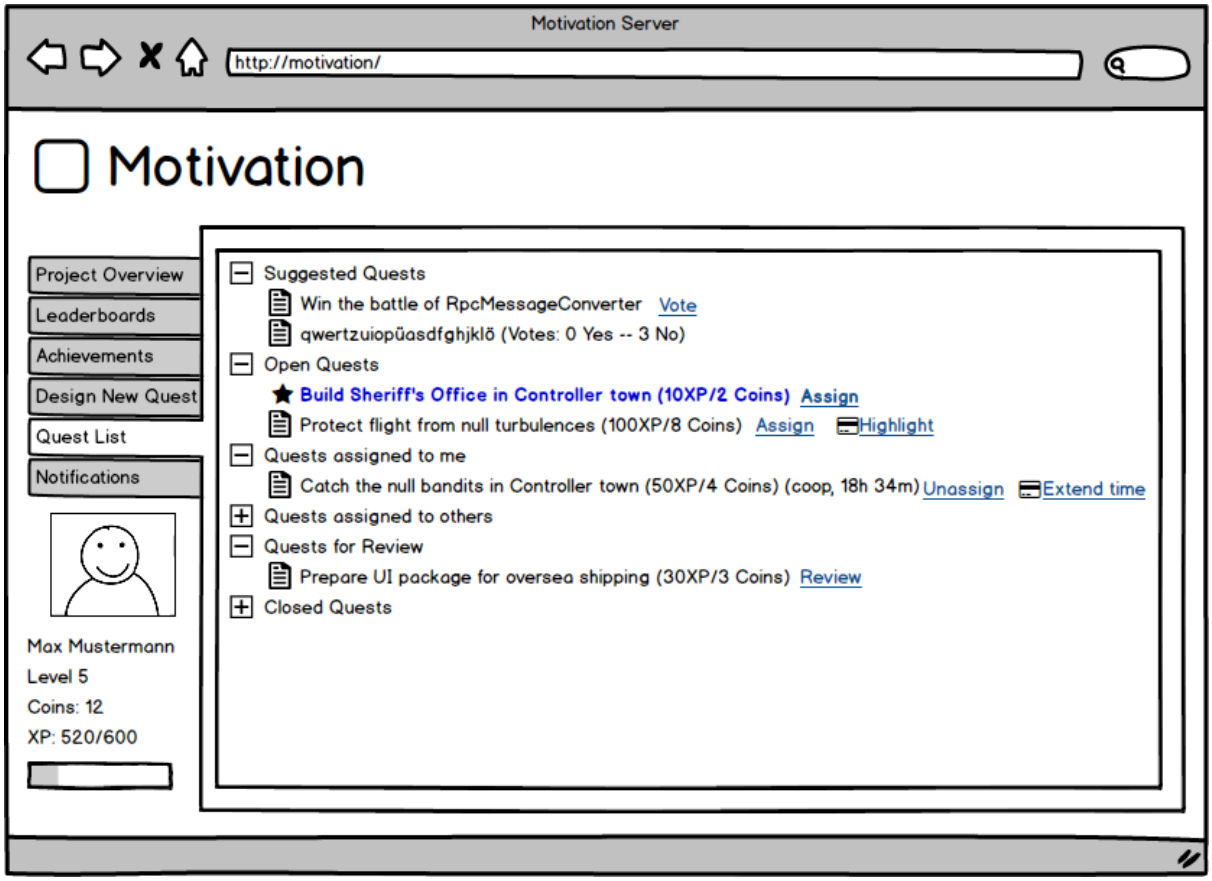

Abbildung 3.5: Mockup für die Seite der Weboberfläche in der alle Quests aufgelistet werden

Eine weitere Detailansicht der Quests mit Beschreibung und Auflistung der zu lösenden Aufgaben sollen per Klick auf den Questtitel erreichbar sein. Daneben werden neben jeder Questtitel die in Abschnitt 3.2.1 besprochenen möglichen Übergänge angezeigt, die in dieser Kategorie für den angemeldeten Benutzer möglich sind. Dabei werden Aktionen, die eine Zahlung von *Coins* erfordern, gesondert hervorgehoben.

Der Mockup für das Zuweisen einer Quest ist in Abbildung 3.6 zu sehen. Wenn das System eine konkrete Eingabe für den Übergang verlangt, so soll für die Eingabe ein sogenanntes *Popover* dargestellt werden. Dadurch wird die Navigation des Benutzers nicht gestört und es gibt einen größeren Bereich, in dem ihm weitere Informationen und Eingabemöglichkeiten angezeigt werden können.

Im konkreten Fall wird die ausgewählte Quest präsentiert und es werden diverse Informationen für die Zuweisung benötigt. Zunächst ist eine kooperative Zuweisung geplant, das bedeutet der Zweitbearbeiter muss dem System vorgeschlagen werden. Weiterhin muss festgelegt werden, wie die Belohnung aufgeteilt werden soll.

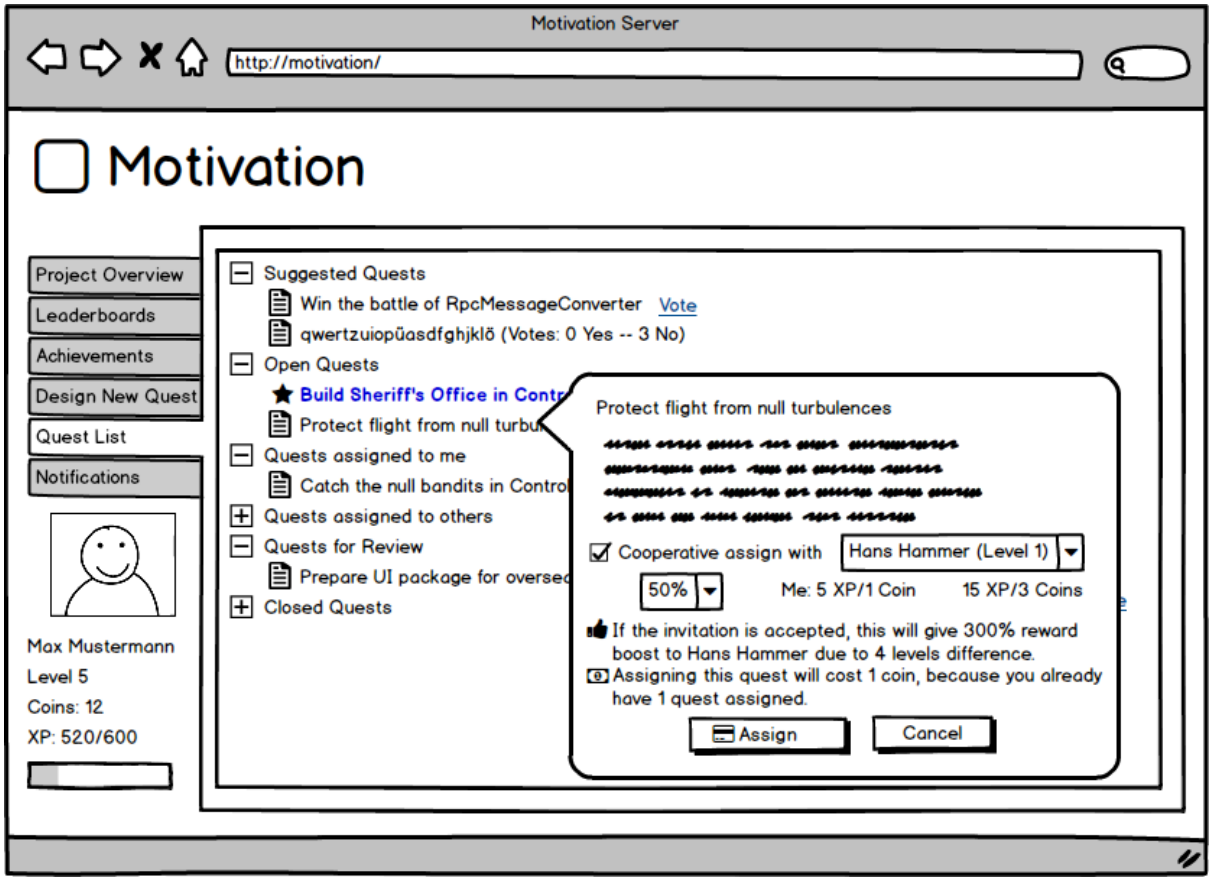

Abbildung 3.6: Mockup für die Seite der Weboberfläche in der ein offenes Quest zugewiesen wird

Das System gibt daraufhin in diesem Fall eine Vielzahl an Rückmeldungen. Dazu zählen die konkreten Belohnungen und inwieweit die Boni durch die Kooperation sie beeinflussen. Darüber hinaus weist das System darauf hin, dass der angemeldete Benutzer bereits eine Quest bearbeitet und daher eine gewisse Menge an *Coins* entrichten muss, um die Quest trotzdem zuzuweisen, was grafisch nochmals gesondert hervorgehoben wird.

Der Mockup für das Abstimmen über einen Quest-Vorschlag ist in Abbildung 3.7 zu sehen. Wie bereits beschrieben kommt auch hier ein *Popover* zum Einsatz. Ähnlich wie beim Zuweisen der Quest wird auch hier die Quest zusammengefasst.

Anschließend erhält der Benutzer die Möglichkeit seine Stimme für oder gegen den Entwurf abzugeben. Falls er mit seiner Stimme den Questentwurf akzeptieren will, so muss er zusätzlich einen Vorschlag für die Questbelohnung abgeben.

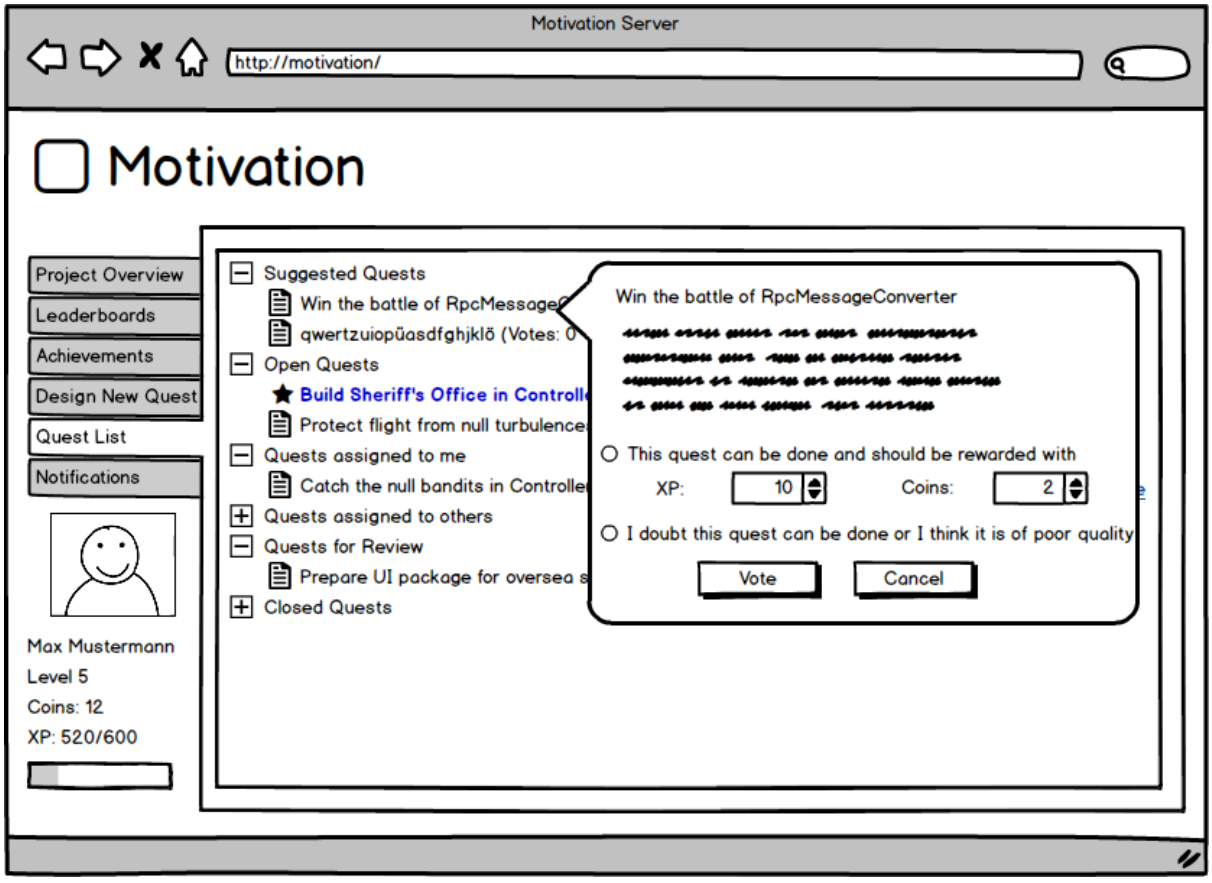

Abbildung 3.7: Mockup für die Seite der Weboberfläche in der für oder gegen ein Quest-Vorschlag abgestimmt wird

3.3.2 Eclipse-Plugin

FindBugs wird heutzutage hauptsächlich als Plugin für Eclipse [Ecl] oder direkt in Werkzeugen für die kontinuierliche Integration benutzt.

Daher soll sich die Umsetzung auch in die Eclipse-Perspektive des vorhandenen FindBugs-Plugins integrieren. Dafür soll es möglich sein, sich einzuloggen und Basisinformationen abzufragen. Dazu zählt eine Übersicht über die Quests, ihren Tasks und den verknüpften Bugs. Des Weiteren soll es möglich sein, die wichtigsten Aktionen auch im Eclipse-Plugin durchzuführen.

Der Mockup für das Eclipse-Plugins ist in Abbildung 3.8 zu sehen. Der Benutzer soll sich hier mit seinen Benutzerdaten einloggen können. Danach sollen dem Benutzer ähnlich wie im Web-System der aktuelle Stand seines Benutzerkontos angezeigt werden. Daneben soll es eine Anzeige der erreichten Badges geben.

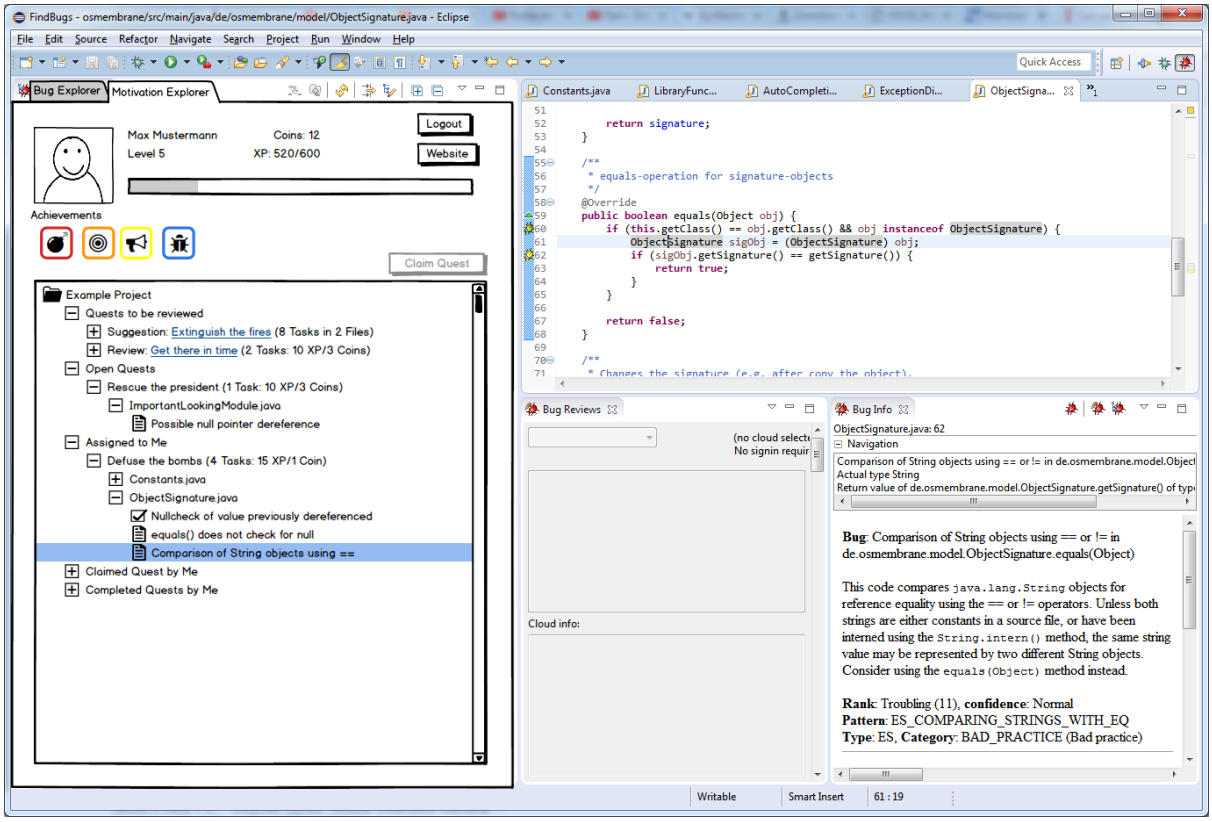

Abbildung 3.8: Mockup für das Eclipse-Plugin

Dabei soll der *Motivation Explorer* den von der FindBugs-Perspektive zur Verfügung gestellten *Bug Explorer* nach Möglichkeit ersetzen. Die Baumsicht soll eine Übersicht über die Questkategorien zeigen. Darunter befinden sich dann die einzelnen Quests mit ihren jeweiligen Aufgaben. Diese wiederum zeigen den Fehler nach dem Ort des Auftretens. Dabei sollten dem Benutzer stets so viele Informationen wie möglich zur Verfügung gestellt werden, z. B. die Quest-Belohnungen.

4 Implementierung

Diese Kapitel beschreibt die durchgeführte Implementierung. Diese fand in einem Repository statt, wie es auch für das Forschungsprojekt HaST [OW16] verwendet wird. Das entstandene neue System wurde auf den Namen „Frame“ (Findbugs enRiched with Applied Motivation Extension) getauft und wird nachfolgend auch so bezeichnet.

4.1 Verbesserung des Erstellungsprozess

Zunächst gab es keinen automatisierten Buildprozess, die zugrundeliegende Projektstruktur war sehr undurchsichtig und die Bibliotheken wurden als Binärartefakte (JAR-Dateien) importiert. Wurde das Datenformat angepasst, so musste es zunächst als Binärartefakt gebaut und in unterschiedlichen Projekten ersetzt werden. Daher bestand der erste Teil der Umsetzung darin, diese Situation zu verbessern.

Da es sich bei den Projekten um Eclipse-Plugins handelt, bietet sich die Verwendung des Buildverwaltungswerkzeugs Maven [Mav] in Verbindung mit der Erweiterung Tycho [Tyc] an, die speziell für diesen Zweck entwickelt wurde. Dazu wurde für jedes Unterprojekt eine *Project-Object-Model*-XML-Datei (`pom.xml`) geschrieben, die jeweils die spezifische Konfiguration des Projekts während eines automatischen Builds übernimmt.

Hierbei gibt es allerdings eine Besonderheit: Eclipse-Plugins können aufgrund ihrer Natur keine Maven-Abhängigkeiten direkt einbinden. Stattdessen ist es nötig, dass die Abhängigkeiten als binäre Artefakte importiert werden. Dafür können Plugins aber Abhängigkeiten zu anderen Plugins angeben. Da ein wichtiges Ziel der Änderung aber war, dass die Abhängigkeiten automatisch eingebunden und ausgeliefert werden, ist hier eine Sonderkonfiguration für Maven nötig.

Es gibt zunächst ein Java-Projekt (`maven-dependencies`), das alle von Eclipse-Plugins verwendeten Maven-Abhängigkeiten als JAR-Dateien exportiert. Diese werden wiederum von einem Plugin-Projekt (`maven-dependencies-plugin`) importiert, welches dann die eigentlichen Abhängigkeiten auf der Plugin-Ebene anderen Plugins zur Verfügung stellt. Eine Erläuterung, wie man weitere Abhängigkeiten hinzufügt, findet sich in einer dort abgelegten Readme-Datei.

Dabei wurde das gesamte Projekt so umstrukturiert, dass es zunächst einen `releng`-Ordner (Release Engineering) gibt, der die Hauptkonfiguration von Maven Tycho und die *Update-Site* enthält. Das Aggregator-Projekt liegt im Hauptordner und ist in der Projektstruktur unterhalb

der Hauptkonfiguration eingebunden. Es listet alle nachfolgenden Projektsammlungen in Unterordnern auf.

Einerseits gibt es jetzt den `dependencies`-Ordner, der alle reinen Java-Projekte enthält. Hierzu zählen nicht nur das erwähnte Projekt, das alle Maven-Abhängigkeiten für Plugins zur Verfügung stellt, sondern auch das Hauptprojekt von FindBugs, das mit HaST erweitert wurde. Ebenfalls befindet sich hier das Projekt für das HaST-Datenaustauschformat.

Daneben gibt es nun den `bundles`-Ordner, der alle Eclipse-Plugins enthält. Dort findet sich das FindBugs-Eclipse-Plugin und der HaST-Server, der zum damaligen Stand nur als Eclipse-Plugin vorlag. Ebenfalls zählt dazu auch das Eclipse-Plugin, das die Maven-Abhängigkeiten importiert. Das Frame-Eclipse-Plugin wurde hier hinzugefügt.

Weiterhin wurde noch ein `features`-Ordner angelegt, der derzeit das *Eclipse Feature* enthält, das alle Eclipse-Plugins zusammenfasst. Dieses Feature ist notwendig, um die *Update-Seite* zu bauen. Ebenso gibt es noch einen `tests`-Ordner, der die Eclipse-Plugin-Tests enthält.

Mit Frame neu hinzugekommen ist ein `serverprojects`-Ordner, der alle reinen Serverprojekte enthält. Der Frame-Server wurde hier entwickelt.

Alle erwähnten Überordner sind ihrerseits wieder Maven-Projekte, die ihre Unterordner als Unterprojekte enthalten.

Wird nun Maven mit Hilfe des Konsolenbefehls `mvn clean package` auf das Aggregator-Projekt ausgeführt, so werden alle Projekte automatisch gebaut, ggf. gegenseitig eingebunden und eine *Update-Site* erzeugt. Diese kann auf einem beliebigen Webserver installiert werden und von dort von beliebigen Eclipse-Installationen genutzt werden, um das Projekt in die lokale Installation zu übernehmen. Mit Hilfe der Maven-Integration M2Eclipse [M2E] ist es auch möglich, diesen Prozess direkt im für die Entwicklung verwendeten Eclipse durchzuführen.

4.2 Web-System

Die entstandene Web-Oberfläche des Frame-Servers kann in Abbildung 4.1 betrachtet werden. Der Frame-Server verwendet hierbei SpringMVC [Spr] um die Anfragen zu verarbeiten und JSP [JSP] mit JSTL[1] für die Darstellung der Webseiten. Für die Funktionalität und das Aussehen der Seiten werden das Frameworks Bootstrap [Boo] in Kombination mit JQuery [jQu] und Font Awesome [Fon] verwendet.

Die Daten werden im Hintergrund mittels Hibernate [Hib] in einer HSQL-Datenbank [HSQ] gespeichert. Dies hat gegenüber objektrelationalen Abbildungen den Vorteil, das die gesammelten Daten auch von anderen Systemen aus der Datenbank ausgelesen werden können.

[1] JavaServer Pages Standard Tag Library

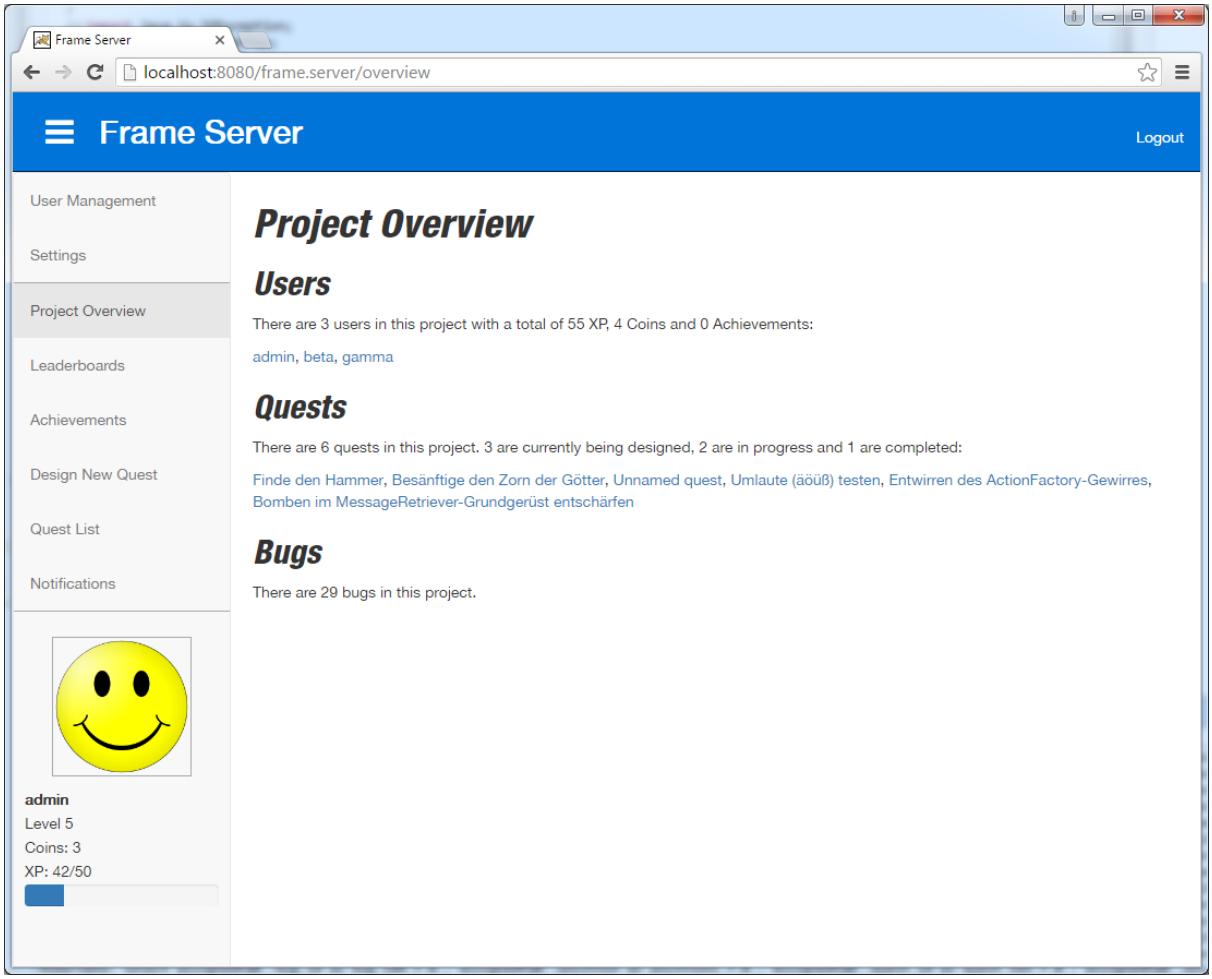

Abbildung 4.1: Beispielhafte Projektübersichtsseite der implementierten Web-Oberfläche

Für die externe Kommunikation steht eine REST[2]-Schnittstelle zur Verfügung, deren Antworten im JSON[3]-Format mittels Jackson [Jac] generiert werden. Die XML[4]-Schnittstelle zu FindBugs wird über einen JAXB-Parser [JAX] realisiert, der automatisch aus einem XSD[5]-Exportschema von FindBugs generiert wird.

[2]Representational State Transfer
[3]JavaScript Object Notation
[4]Extensible Markup Language
[5]XML Schema Definition

4.2.1 Aufbau

Der Web-Server besteht aus über 16 000 Zeilen Java-Code, 1 000 Zeilen JSP-Template-Code und 350 Zeilen JavaScript-Code.

Dabei wurde konsequent das *Model-View-Controller*-Entwurfsmuster eingesetzt. Darüber wird die Datenhaltung von der Verarbeitungslogik und der Anzeigeschicht getrennt. Außerdem kam das *Dependency-Injection*-Entwurfsmuster zur Anwendung. Hierbei wird im Stile der *Inversion of Control* eine Abhängigkeit nur dort eingebunden, wo sie auch tatsächlich benutzt wird. Beides wurde durch die Verwendung von SpringMVC stark unterstützt und erlaubt eine verbessertes Wiederverwenden, Testen und Erweitern der Software.

Die *View*-Schicht wird völlig von der Web-Oberfläche ausgefüllt, die mittels JSP-Templates und JSTL erzeugt wird. Hierzu wurde für den äußeren Rahmen, das Menü und jede Unterseite ein eigenes Template angelegt. Die einzelnen Seiten sind mittels Bootstrap gestaltet worden und Icons wurden der Font-Awesome-Bibliothek entnommen. Komplexere Funktionalität, wie die Eingabemasken und die *Popover* wurden über JavaScript mit jQuery realisiert. Die Seite, auf der neue Quests angelegt werden können, kann beispielhaft in Abbildung 4.2 betrachtet werden.

Daneben gibt es noch einige Hilfsklassen wie `ActionPresentation` oder `UserRuntimeInfos`, die es ermöglichen einen größeren Satz an Werten an die JSP-Visualisierung auf einmal zu übergeben.

In der *Controller*-Ebene befinden sich hauptsächlich die SpringMVC-Controller, die auf Web-Requests reagieren und die Visualisierung entsprechend konfigurieren. Hierbei gibt es für jede Unterseite sowie für die REST-Schnittstelle einen eigenen Controller. Weiterhin finden sich hier auch ein `PostStartupService`, der sicherstellt, dass es ein aktives Administrationskonto gibt. Der `PreShutdownService` sorgt für ein sauberes Beenden der Datenbankschnittstelle, wenn der Server beendet wird.

Auf der *Model*-Ebene finden sich alle Datenhaltungsklassen, die jeweils einen eigenes `Service`-*Interface* mit entsprechender Implementierung besitzen. Ein UML-Klassendiagramm hiervon findet sich in Abbildung 4.3. Der *Service* kommuniziert dabei mit der Persistenzschicht, also in diesem Falle Hibernate als JPA[6]-Implementierung. Er ist in der Lage, Daten-Objekte zu laden, zu suchen, zu ändern, zu bearbeiten und zu speichern.

Dazu enthält jeder `Service` die Behandlung von Spezialfällen abhängig von der behandelten Klasse. Zum Beispiel im Falle der `Bug`-Klasse kümmert er sich auch um die Synchronisation mit der FindBugs-XML-Datei, die in Abschnitt 4.2.3 beschrieben ist. Der `QuestService` kann beispielsweise Quests abhängig vom zugewiesenen Benutzer suchen. Der `UserService` ist in der Lage, die Anzahl der abstimmungsberechtigen Konten aus der Datenbank auszulesen, während der `TaskService` Aufgaben neu ordnen kann. Er ist weiterhin fähig, einen `Bug` nur zu

[6]Java Persistence API

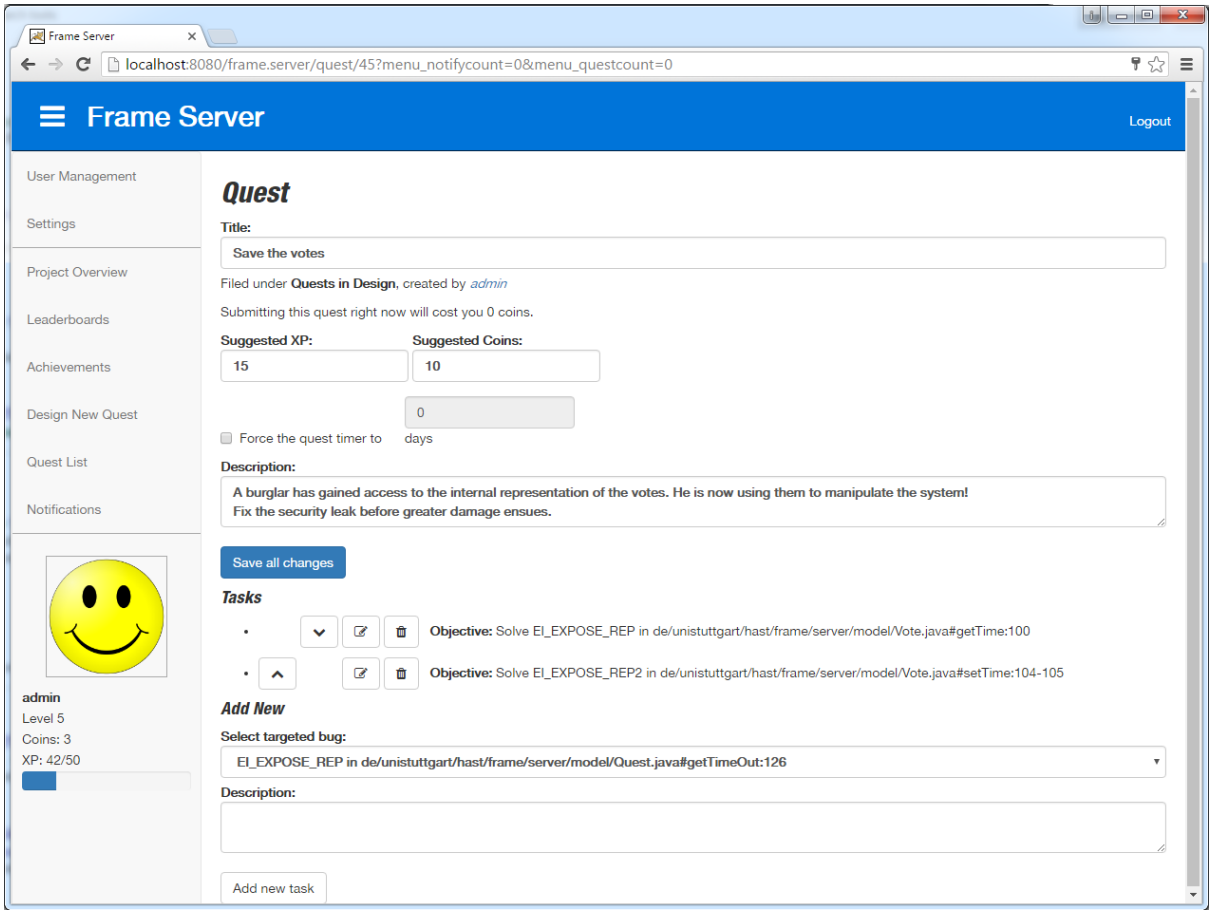

Abbildung 4.2: Seite zum Anlegen neuer Quests der implementierten Web-Oberfläche

einem `Task` zu verknüpfen und aus allen anderen Questvorschlägen zu entfernen, sobald ein Questvorschlag eingereicht wurde. Der `VotingService` enthält Routinen um Abstimmungen in der Datenbank zu verknüpfen.

Nachfolgend findet sich eine Liste der Klassennamen mit Erläuterung ihrer Bedeutung in Frame:

User repräsentiert einen Benutzer im System. Neben seinem Benutzernamen, Passwort-Hash und Zugriffsrechten werden auch die gesammelten *XP*, *Coins* und das eingestellte Benutzerbild gespeichert.

Quest stellt eine Aufgabe im System dar. Diese kennt ihren Ersteller, ihre Unteraufgaben (`Tasks`) und ihren aktuellen Status. Darüber werden hier die Beschreibungen und Darstellungsvorgaben, die Belohnungen und *Quest-Timer*-Einstellungen sowie die Zuweisungen an Benutzer gespeichert.

Task ist die konkrete Aufgabenstellung, einen `Bug` in einer `Quest` zu lösen.

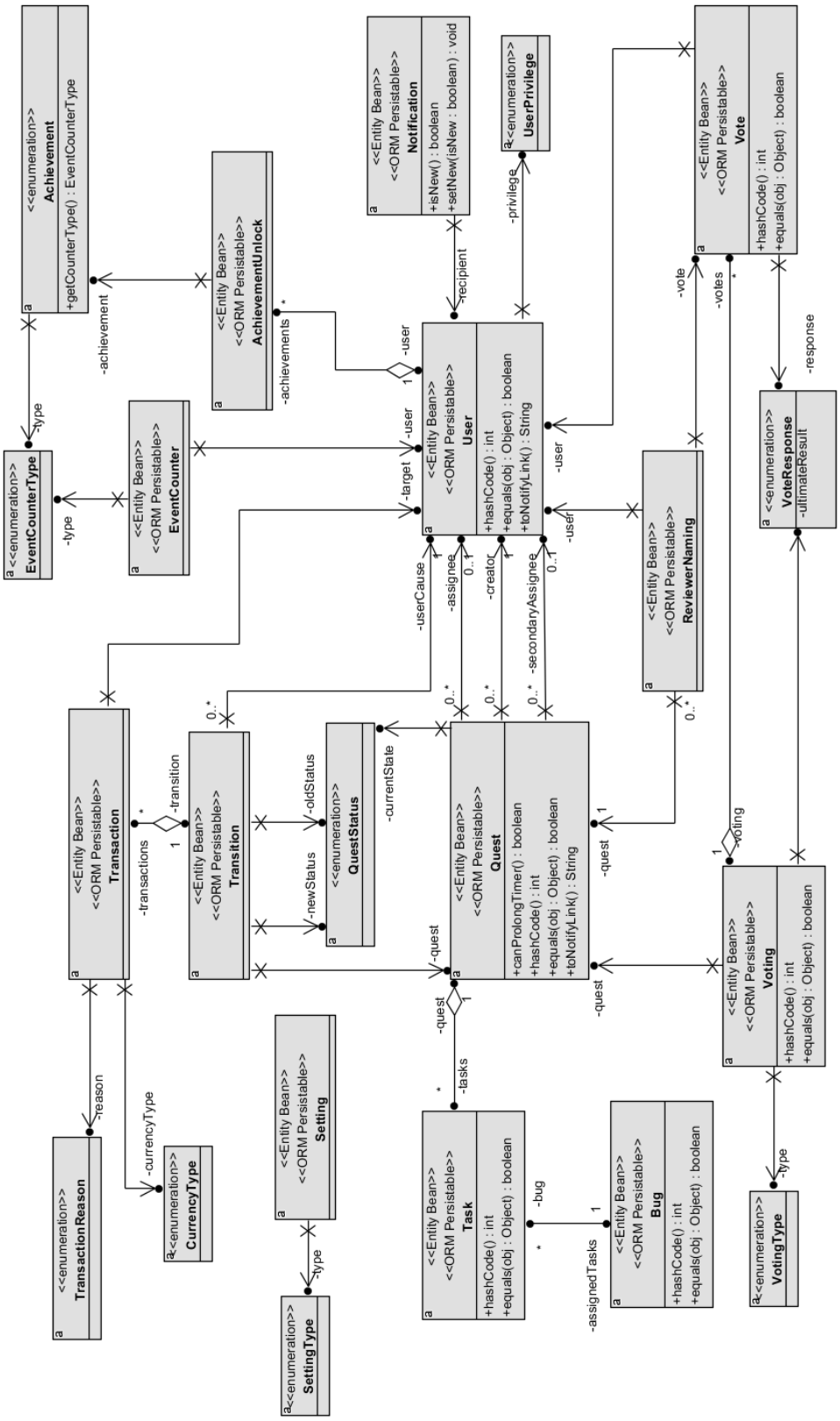

Abbildung 4.3: UML-Klassendiagramm der zentralen *Model*-Klassen, durch *Reverse Engineering* mit Visual Paradigm generiert und überarbeitet

Bug speichert, welche FindBugs-Meldungen in welchen Klassen aufgetreten sind. Darüber hinaus wird erfasst, ob die Meldung in der letzten Analyse noch vorhanden war.

Voting enthält die Review-Abstimmungen einer Quest. Hierfür gibt es einen Typen sowohl für das Einreichen als auch für das Beanspruchen einer Quest. Es werden die einzelnen Stimmen verknüpft und das endgültige Ergebnis gespeichert.

Vote ist eine abgegebene Stimme in einer Abstimmung. Sie speichert, welcher Benutzer wann dafür oder dagegen gestimmt hat. Im Falle des Einreichen eines Questvorschlags werden auch die vorgeschlagenen Belohnungen gespeichert.

Transition protokolliert, zu welchem Zeitpunkt und durch welchen Benutzer eine Quest ihren Zustand geändert hat.

Transaction hält fest, aus welchem Grund, zu welchem Zeitpunkt und wie viel Währung wegen einer Questaktion geflossen ist.

AchievementUnlock protokolliert, wann ein Benutzer ein bestimmten Achievement-Typ freigeschalten hat.

EventCounter zählt die Anzahl von bestimmten Ereignistypen, die ein Benutzer getätigt hat.

Notification bildet eine Text-Benachrichtigung an einen Benutzer im System ab.

ReviewerNaming hält jedes Mal fest, wenn ein Benutzer für das Beanspruchen einer Quest kostenpflichtig zum Reviewer ernannt wurde.

Setting ist dafür zuständig, die Werte bestimmter Einstellungstypen zu speichern.

Wenn nötig werden Hilfsklassen wie `RandomStringBuilder` (Generierung zufälliger Passwörter), `LevelCalculator` (Berechnung der Stufen dynamisch aus den Erfahrungspunkten des Kontos) oder `BugCollectionWrapper` (Schnittstelle zum JAXB-Parser) durch Spring-Konfigurationen für die *Dependeny Injection* bereitgestellt.

Des Weiteren gibt es die Hierarchie der `Action`-Klassen, die die Verarbeitungslogik für alle in Abschnitt 3.2.1 erwähnten Übergänge anbietet. Zunächst wurden alle möglichen Übergänge in Tabellenform mitsamt Name, Ursprungs- und Zielzustand, Eingabeparameter, Vorbedingungen, *Coin*- und *XP*-Transaktionen, andere Effekte und ausgelöste Benachrichtigungen dokumentiert. Die Tabelle wurden zur Implementierung konsistent gehalten und in der `Transitions_Semantics.html` im `model`-Paket hinterlegt. Die abstrakte `Action`-Klasse stellt dann eine Grundfunktionalität für alle Übergänge bereit. Jeder einzelne Übergang wird dann durch eine eigene abgeleitete Klasse realisiert.

Jede `Action` weiß, ob sie automatisch oder vom Benutzer ausgeführt wird (`canBeDoneOnUI()`), ob es grade für das gegebene Benutzer- und Quest-Paar möglich ist, sie durchzuführen (`isPossible()`). Sie kennt ihre Vorbedingungen (`getRequiredLevel()`) und ihre Kosten (`getRequiredCoins(), getCostTransactions()`) sowie Belohnungen (`getRewardTransactions()`).

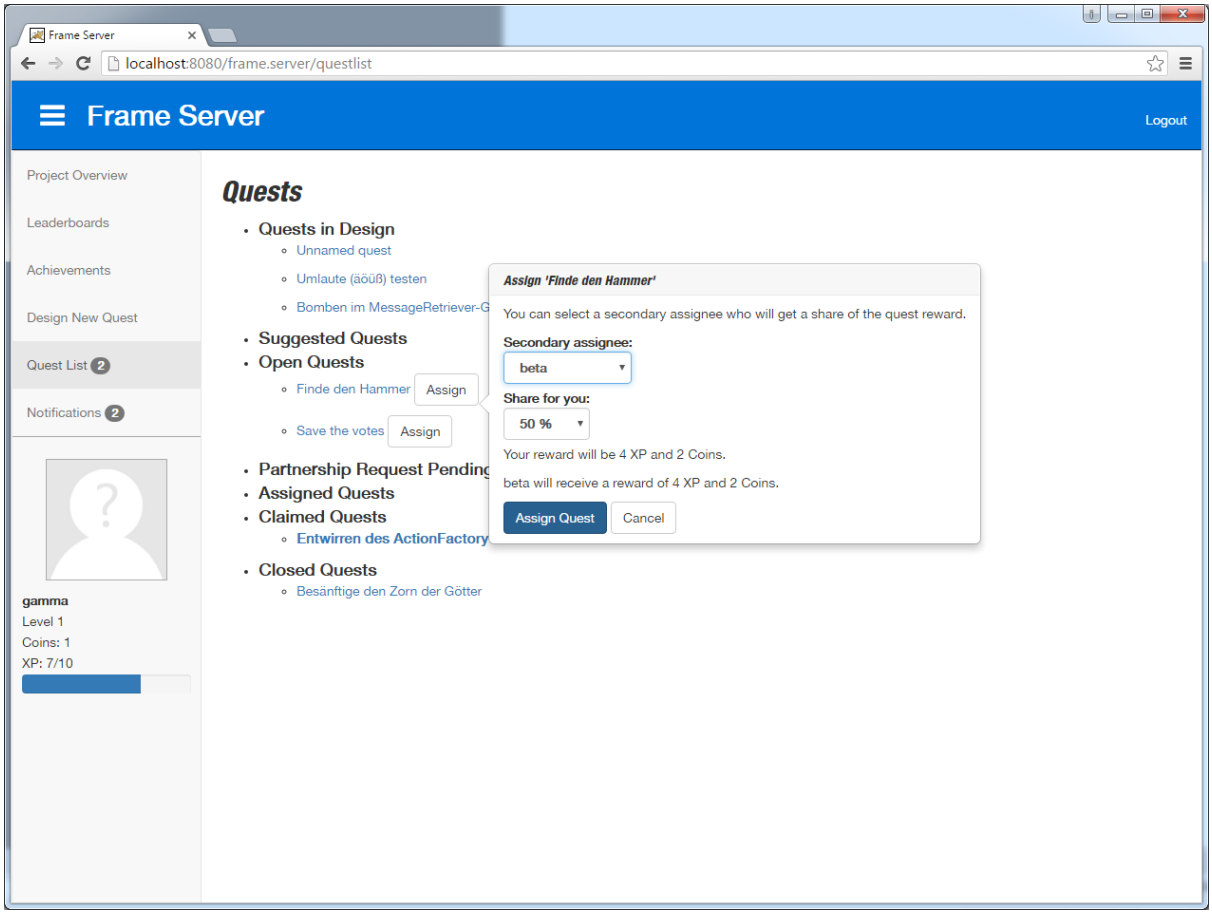

Abbildung 4.4: Übersichtsliste aller Quests mit Eingabe für Zuweisung auf der implementierten Web-Oberfläche

Dadurch kann die Übersichtsliste der Quests einfach entscheiden, ob eine Interaktionsmöglichkeit angezeigt werden soll oder nicht. Die Übersichtliste ist beispielhaft in Abbildung 4.4 zu sehen.

Beim Durchführen der Interaktion per `perform()` von außen geschieht die konkrete Verarbeitung der *Coin*- und *XP*-Transaktionen und Transitionen (Protokoll über die Zustandsänderungen der Quest) in der abstrakten Klasse, sodass die Kindklassen nur die nötigen Werte und einen Durchführungsalgorithmus in der `transit()`-Methode angeben müssen und übersichtlich gehalten werden können. Die Kindklassen fügen dann eigene Konfigurationsmöglichkeiten für Benutzereingaben hinzu. Dabei folgt jede instantiierte `Action` der Semantik, dass sie höchstens einmal ausgeführt werden kann. Wenn sie ausgeführt wird, sendet sie allen beteiligten Benutzern eine *Notification* über die resultierenden Ergebnisse.

Daneben gibt es den `ActionEnum`, der Bezeichner für die Aktionen mit konkreten Klassen und den erlaubten Ausgangszuständen der Quest verknüpft. Zugriff auf die `Action`-Klassen geschieht zur Entkopplung nur über die `ActionFactory`. Dabei werden die Aktionen als

Prototype Scope von Spring instantiiert. Die `ActionFactory` kann auch direkt die möglichen Übergänge für einen Questzustand zurückliefern.

4.2.2 Ablauf

Die konkreten Abläufe finden wie erwähnt in den `Action`-Klassen statt. Diese sind dazu in vier Pakete unterteilt: `design` (Erstellen und Bearbeiten von Questvorschlägen), `submit` (Einreichen und Review von Questvorschlägen), `work` (Zuweisen und Erledigen von Arbeitsschritten) und `review` (Beanspruchen von Quests und Review der Lösung). Im Nachfolgenden wird der individuelle Ablauf aller `Action`-Klassen erläutert:

CreateQuestAction erstellt eine neue Quest, wenn der Benutzer diese Option wählt.

EditQuestAction existiert nur als visueller Hinweis auf der Quest-Liste, dass die Questseite das Bearbeiten ermöglicht. Das eigentliche Ändern der Questwerte wird vom `QuestController` übernommen.

DeleteQuestAction löscht eine vorhandene Quest.

ForceReworkQuestAction setzt den Zustand einer Quest zurück in den Entwurf und informiert den Ersteller hierüber.

SubmitQuestAction reicht einen Questentwurf zur Abstimmung ein. Dabei werden alle Bugs, die behoben werden sollen, aus den anderen Questvorschlägen entfernt. Zudem werden alle Benutzer informiert, dass sie nun für oder gegen den Entwurf abstimmen können.

QuestSuggestionVoteAction führt das Speichern einer einzelnen Stimme und das Aktualisieren der Entwurfsabstimmung durch.

QuestSuggestionCloseAction wird ausgelöst, wenn genug Stimmen eingegangen sind. Hier werden die Stimmen gezählt und auf das $\pm 50\%$ Plausabilitätsintervall hin untersucht. Das Abstimmungsergebnis wird dann festgelegt. Entweder wird das Quest zurück in Entwurf oder in den offenen Zustand gesetzt. In letzterem Fall wird die Belohnung als Median aller abgegebenen Stimmen inklusive dem Questersteller-Vorschlag festgelegt. Außerdem werden die Belohnungen für die Reviewer und ggf. den Questersteller ausgezahlt und jeder mit einer `Notification` hierüber unterrichtet.

HighlightQuestAction speichert, dass die Quest optisch hervorgehoben werden soll. Der Questersteller wird hierüber informiert.

AssignQuestAction weist eine offene Quest einem oder zwei Benutzern zu. Falls es einen Zweitbearbeiter gibt, wird dieser zunächst gefragt, ob er dieser Zuweisung zustimmt. Bei einem Bearbeiter wird die Quest zugewiesen und der Questersteller informiert.

QuestPartnershipRespondAction wird bei der Antwort eines Zweitbearbeiters zur kooperativen Zuweisung ausgelöst. Hier wird das Quest entweder zugewiesen oder wieder eröffnet. Die beteiligten Bearbeiter und im Erfolgsfall auch der Quest-Ersteller werden hierüber informiert.

ForceQuestOnUserAction ermöglicht es Administratoren, Quests an Benutzer zuzuweisen.

UnassignQuestAction trägt einen oder beide Bearbeiter aus einer Quest aus. Die Bearbeiter und der Questersteller werden hierüber informiert. Im Hintergrund wird die Sperre für die erneute Zuweisung geschrieben.

ProlongQuestTimerAction speichert, dass der *Quest-Timer* verlängert wurde. Der Questersteller wird hierüber informiert.

QuestTimeoutAction wird ausgelöst, wenn der *Quest-Timer* abgelaufen ist. Die Bearbeiter werden daraufhin ausgetragen, das Quest wieder als offen geführt und sowohl Bearbeiter als auch der Questersteller über den Vorgang unterrichtet.

ClaimQuestAction reicht die Lösung zur Abstimmung ein und beansprucht das Quest. Dabei bestimmt das System zufällige Reviewer, wenn der Benutzer keine ausreichende Anzahl angegeben hat. Kostenpflichtige Ernennungen werden protokolliert. Die Bearbeiter, der Questersteller und die bestimmten Reviewer werden über den Vorgang in Kenntnis gesetzt.

QuestAcceptanceVoteAction führt das Speichern einer einzelnen Stimme und das Aktualisieren der Lösungsabstimmung durch.

QuestAcceptanceCloseAction wird ausgelöst, wenn genug Stimmen eingegangen sind. Die Stimmen werden gezählt und das Abstimmungsergebnis wird festgelegt. Die Belohnungen an die Reviewer werden ausgeschüttet. Entweder wird das Quest zurück in den Bearbeitungszustand gesetzt und der *Quest-Timer* so eingestellt, dass keine Zeit für die Bearbeitung verloren gegangen ist, oder das Quest wird geschlossen. In diesem Fall werden auch die Quest-Belohnungen ggf. berechnet und ausgezahlt. Die Bearbeiter und der Questersteller werden mittels einer `Notification` hierüber unterrichtet.

4.2.3 Konfiguration

Das System wurde darauf ausgelegt in Verbindung mit einem zentralen Server für die Versionskontrolle oder kontinuierlichen Integration verwendet zu werden. Dadurch steht immer ein aktueller Codestand zur Verfügung, der repräsentativ für das zu untersuchende Projekt ist. Hiermit wird sichergestellt, dass nur Code-Änderungen vom System analysiert werden, die auch tatsächlichen im Entwicklungsprozess stattgefunden haben und beispielsweise durch ein *Code Review* akzeptiert wurden.

Das System lässt sich dabei auf zwei Arten konfigurieren, um die von FindBugs gefunden Fehler anzupassen. Die Konfigurationsmaske ist in Abbildung 4.5 zu sehen. Die gefundenen

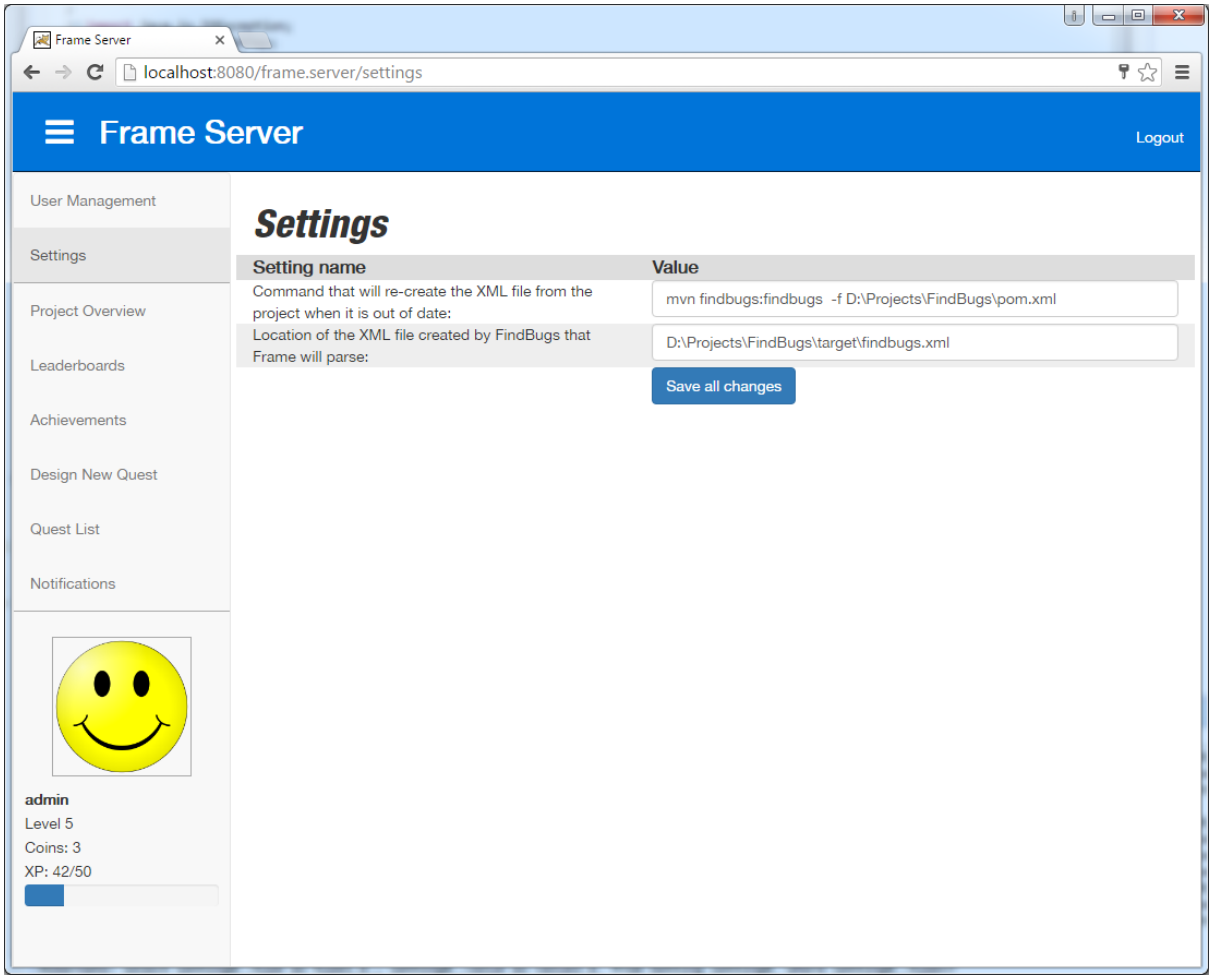

Abbildung 4.5: Konfigurationsseite der implementierten Web-Oberfläche

Fehler werden aus einer XML-Datei gelesen, die von FindBugs als Ausgabe produziert wird. Dabei lässt sich in Frame der Ort einstellen, von dem die XML-Datei gelesen werden soll. Wird diese Datei bereits während des Build-Prozesses in einer kontinuierlichen Integration erstellt, so kann sie direkt verwendet werden.

Gibt es keine kontinuierliche Integration, lässt sich das System auch so konfigurieren, dass es Fehler kontinuierlich erfassen kann. Es gibt die Möglichkeit eine Kommandozeile einzustellen, die aufgerufen wird, wenn die Datei älter als 5 Minuten ist. Hier kann beispielsweise das FindBugs-Plugin von Maven mittels der Kommandozeile `mvn findbugs:findbugs` aufgerufen werden, das die gewünschte XML-Datei produziert.

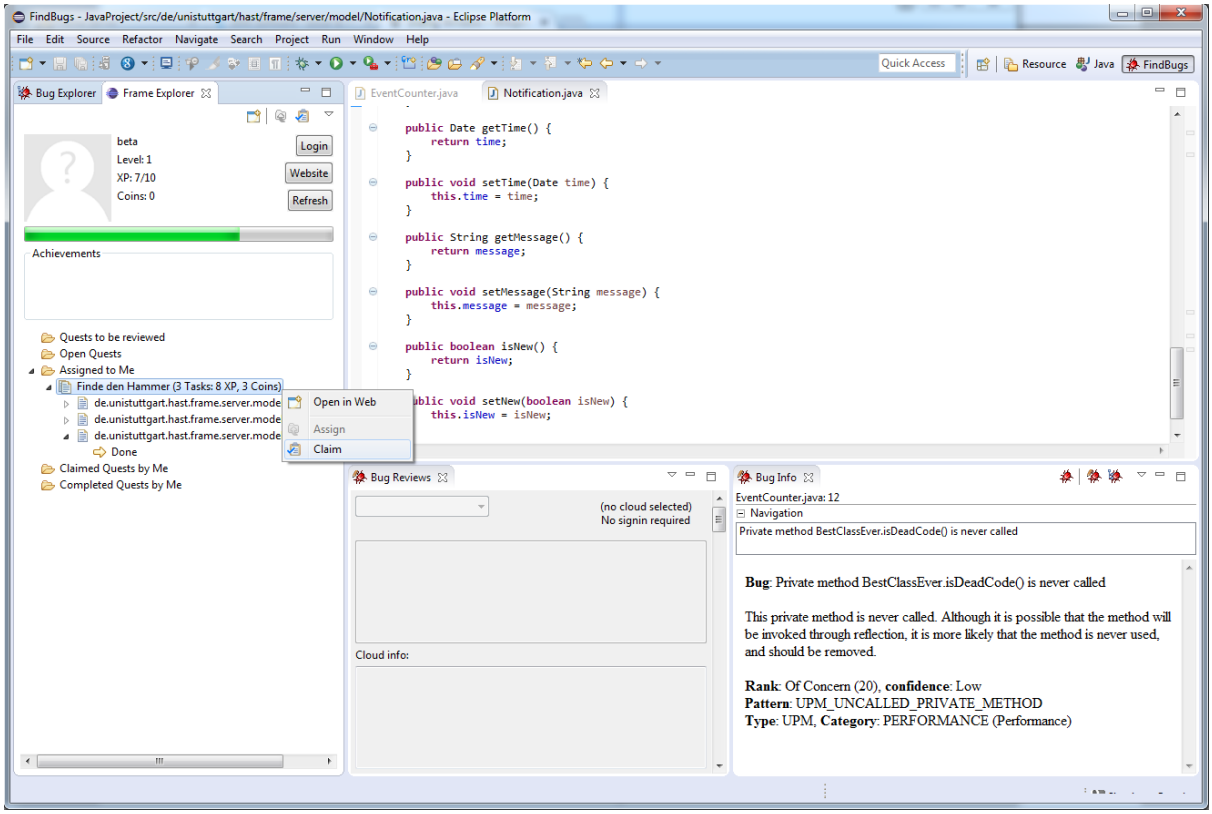

Abbildung 4.6: Implementiertes Eclipse-Plugins

4.3 Eclipse-Plugin

Die Umsetzung des Eclipse-Plugins für Frame ist in Abbildung 4.6 zu sehen. Für die Kommunikation per REST mit dem Frame-Server verwendet das Plugin JavaLite [Jav] und liest die Antworten im JSON-Format mittels Jackson [Jac] aus.

4.3.1 Aufbau

Das Eclipse-Plugin ist eher simpel aufgebaut, versucht aber auch dem MVC-Schema zu folgen. Es besteht im Kern aus etwa 1 500 Zeilen Java-Code.

Es existiert auf der *Model*-Ebene eine `FrameConnection`-Klasse, die mit dem Server kommuniziert. Alle Aktionen, die der Nutzer ausführen kann, werden über eine `ButtonController`-Klasse realisiert.

Der *View*-Teil ist dabei in mehrere Elemente aufgeteilt. Die Darstellung der Benutzerinformationen werden von der `FrameInfoBoardView`-, die Anzeige des Questbaums von der

`FrameQuestView`-Klasse übernommen. Darüber hinaus gibt es noch mehrere Hilfsklassen für das Layout der Elemente, die Verwaltung der Baumansicht und den Login-Dialog.

Die einzelnen Aktionen, wie Aufrufen der Quest auf der Frame-Webseite sowie das simple Zuweisen und Beanspruchen der Quests sind als eigene `Action` in Eclipse implementiert und werden wie das Aktualisieren der Daten im `ButtonController` durchgeführt. Im Falle des Aktualisierens leitet dieser die Daten an die *View*-Komponenten weiter, die darauffolgend ihre Darstellung anpassen.

4.3.2 Ablauf

Meldet sich ein Benutzer im Plugin an, so wird zunächst versucht per REST-Schnittstelle seine Daten abzufragen. Gelingt dies, so werden die Daten gespeichert und für weitere Anfragen benutzt. Dies ist notwendig, da REST ein zustandsloses Protokoll ist und somit keine Sitzung aufrecht halten kann.

Neben dem REST-Aufruf `/rest/user`, um die Benutzerdaten als JSON abzufragen, gibt es auch noch die Abfrage der Questliste mittels `/rest/quests`. Aktionen können auf dem Web-Server durch den Aufruf von `/rest/quest` mit zusätzlicher Angabe der Quest-Id und Aktionsbezeichnung durchgeführt werden.

5 Zusammenfassung und Ausblick

Dieses Kapitel beendet diese Arbeit und rekapituliert den berichteten Inhalt. Darüber hinaus werden diverse Anknüpfungspunkte für zukünftige Arbeiten dargelegt.

5.1 Zusammenfassung

Das konsequente Anwenden von Werkzeuge zur statischen Codeanalyse stellt Entwickler vor Schwierigkeiten. Einerseits können mit vergleichsweise geringem Aufwand unzählige Fehler und Probleme gefunden werden. Andererseits ist das Beheben dieser Fehler ein eher schleppender Prozess, wenn er nicht rigoros durchgeführt wird. Ein vollständiges Beheben der Fehler führt zu erhöhter Qualität der Software, aber damit ist oft umfangreiche Arbeit verbunden. Viele Fehler müssen im Kontext des Systems analysiert und eine passende Behebungsstrategie ausgewählt werden. Des Weiteren müssen Falschmeldungen aussortiert werden, ohne dabei erwünschte Meldungen auszuschließen. Diese Faktoren beeinträchtigen die Motivation der Entwickler.

Die Methode der *Gamification* wurde vorgestellt als Mittel, um die Entwickler zu motivieren, sich mit den Meldungen der statischen Codeanalyse auseinander zu setzen. Daher wurden unterschiedlichen Ansätze der *Gamification*, u. a. Punkt- und Aufgaben-Systeme vorgestellt und ihre Wirksamkeit begründet. Darauf aufbauend wurden die Ansätze in einen Entwurf für FindBugs als konkretes Beispiel der Werkzeuge zur statischen Code-Analyse überführt. Für diesen wurde ein zustandsbasiertes Aufgaben-System und mehrere Mockups erstellt.

Der Entwurf wurde anschließend prototypisch umgesetzt. Hierbei wurde für den Server eine zeitgemäße Web-Oberfläche entworfen und implementiert, die auf industrieüblichen Technologien basiert. Dazu wurde ein Eclipse-Plugin entwickelt, das über eine Schnittstelle mit dem Server kommuniziert und es den Entwicklern in der Praxis ermöglicht, direkt auf Informationen im System zuzugreifen und Aktionen auszulösen.

5.2 Ausblick

Es gibt mehrere offene Fragestellungen und Weiterentwicklungspotential des umgesetzten Systems. Hierzu zählt zunächst die Weiterentwicklung, da die ambitionierte Umsetzung durch die

begrenzte Zeit nicht alle erwünschten Funktionalitäten enthält, aber auch generell verbessert werden kann.

Eine Möglichkeit besteht darin, Frame so zu gestalten, dass es mehrere Projekte und individuelle Projektteams verwalten kann. Dadurch könnten beispielsweise verschiedene Projekt-Manager die Erfahrung einzelner Entwickler mit dem System und der Behebung von FindBugs-Fehlern einschätzen. Ebenso ergeben sich Fragen, ob und inwieweit Erfahrungspunkte und Währungspunkte zwischen Projekten übertragbar sein sollen, um die Motivation aufrecht zu erhalten. In diesem Zuge wäre es auch nötig, die Belohnungen und Kosten pro Projekt konfigurierbar zu machen. Dort wäre auch zu entscheiden, ob die Reviews nur innerhalb eines Projekts oder projektübergreifend stattfinden sollen. Weiterhin wäre auch denkbar, dass es eine Art firmenweiten Marktplatz gibt, in dem gegen ausreichend *Coins* echte Gegenleistungen, wie z. B. ein besonders angenehmer Sitzplatz in Meetings, eine technische Spielerei oder ein kostenloses Kantinen-Mittagessen erhältlich sind.

Des Weiteren kann das Eclipse-Plugin erweitert werden, sodass es Frame noch weitergehend bedienen kann. Hierzu könnten weitere Aktionen implementiert werden und das Erstellen von neuen Quests direkt aus der FindBugs-Ansicht in Eclipse ermöglicht werden. Das Plugin könnte die Anzeige, die auf den Server-Daten basiert, mit dem derzeit im Eclipse bearbeiteten Stand abgleichen. Außerdem ist es denkbar, dass direkt aus dem Plugin mit anderen Frame-Nutzern kommuniziert werden kann.

Auch eine Überarbeitung der Web-Oberfläche bietet Vorteile. So konnten weniger integrale Funktionalitäten wie das Wechseln des Farbschemas, Benutzer-Titel und die umfangreiche Anzeige aller Gestaltungsmerkmale an jeder erdenklichen Stelle in dieser Arbeit nicht umgesetzt werden. Weiterhin ist die beschriebene, aber nicht umgesetzte Auto-Generierung von Quests ein vielversprechendes Thema um die Immersion zu steigern. Ebenfalls könnte das Ausrollen von Frame mit einem eingebetteten Webserver verbessert werden. Auch kann eine Interoperabilität mit Versionskontrollsystemen, kontinuierlicher Integration und Mail-Servern hergestellt werden.

Ansonsten besteht die Möglichkeit das System in einem echten Einsatzszenario zu erproben und ein umfassendes Experiment durchzuführen. Dabei kann versucht werden, die intuitiv aber begründet gewählten Werte für Kosten und Belohnungen zu verbessern. Auch kann explizit versucht werden, das System zu untergraben, zu betrügen und zu stören. Die gewonnen Daten können auch von anderen Systemen einfach aus der Datenbank des Servers gelesen und weiterverarbeitet werden.

Darüber hinaus kann versucht werden, die Anwendung auf andere Werkzeuge und Teilbereiche des Software-Entwicklungsprozesses zu übertragen. Hierbei können Bereiche untersucht werden, zu denen Entwickler eher weniger motiviert sind, wie z. B. viele Formen des Testens. Abgesehen davon kann ebenso untersucht werden, ob die Anwendung der *Gamification* auf üblicherweise motivierende Aufgabenfelder wie die Implementierung ebenfalls positive Auswirkungen zeigt.

Literaturverzeichnis

[Bal] Balsamiq Studios, LLC. *Balsamiq Mockups*. URL: https://balsamiq.com/products/mockups/ (zitiert auf S. 27).

[Bar04] R. A. Bartle. *Designing virtual worlds*. New Riders, 2004 (zitiert auf S. 13).

[Bar96] R. Bartle. „Hearts, clubs, diamonds, spades: Players who suit MUDs“. In: *Journal of MUD research* 1.1 (1996), S. 19 (zitiert auf S. 13).

[Boo] M. Otto, J. Thornton. *Bootstrap*. URL: http://getbootstrap.com/ (zitiert auf S. 36).

[BSK11] J. Bell, S. Sheth, G. Kaiser. „Secret ninja testing with HALO software engineering“. In: *Proceedings of the 4th international workshop on Social software engineering*. ACM. 2011, S. 43–47 (zitiert auf S. 20).

[BT13] K. Berkling, C. Thomas. „Gamification of a Software Engineering course and a detailed analysis of the factors that lead to it's failure“. In: *Interactive Collaborative Learning (ICL), 2013 International Conference on*. IEEE. 2013, S. 525–530 (zitiert auf S. 19).

[CB12] E. Coelho, A. Basu. „Effort estimation in agile software development using story points“. In: *International Journal of Applied Information Systems (IJAIS)* 3.7 (2012) (zitiert auf S. 21).

[CC92] M. Csikszentmihalyi, I. S. Csikszentmihalyi. *Optimal experience: Psychological studies of flow in consciousness*. Cambridge university press, 1992 (zitiert auf S. 18).

[DD80] S. E. Dreyfus, H. L. Dreyfus. *A five-stage model of the mental activities involved in directed skill acquisition*. Techn. Ber. DTIC Document, 1980 (zitiert auf S. 14).

[DT13] D. J. Dubois, G. Tamburrelli. „Understanding gamification mechanisms for software development“. In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM. 2013, S. 659–662 (zitiert auf S. 19).

[Ecl] The Eclipse Foundation. *Eclipse*. URL: https://www.eclipse.org/ (zitiert auf S. 32).

[Fec12] M. Fecher. „Gamification in der Softwareentwicklung: Chancen und Möglichkeiten.“ In: *IEEE GSC*. 2012 (zitiert auf S. 19, 21).

[Fin] University of Maryland. *FindBugs – Find Bugs in Java Programs*. URL: http://findbugs.sourceforge.net/ (zitiert auf S. 9).

[Fon] D. Gandy. *Font Awesome*. URL: http://fontawesome.io/ (zitiert auf S. 36).

[Ger] Google. *Gerrit Code Review*. URL: https://www.gerritcodereview.com/ (zitiert auf S. 28).

[HH12] K. Huotari, J. Hamari. „Defining gamification: a service marketing perspective“. In: *Proceeding of the 16th International Academic MindTrek Conference*. ACM. 2012, S. 17–22 (zitiert auf S. 11).

[Hib] Red Hat. *Hibernate*. URL: http://hibernate.org/ (zitiert auf S. 36).

[HKS14] J. Hamari, J. Koivisto, H. Sarsa. „Does gamification work?–a literature review of empirical studies on gamification“. In: *2014 47th Hawaii International Conference on System Sciences*. IEEE. 2014, S. 3025–3034 (zitiert auf S. 11).

[HSQ] The HSQL Development Group. *HSQLDB*. URL: http://hsqldb.org/ (zitiert auf S. 36).

[Jac] FasterXML. *Jackson*. URL: http://wiki.fasterxml.com/JacksonHome (zitiert auf S. 37, 46).

[Jav] I. Polevoy. *JavaLite*. URL: http://javalite.io/ (zitiert auf S. 46).

[JAX] Oracle. *Java Architecture for XML Binding*. URL: http : / / www . oracle . com / technetwork/articles/javase/index-140168.html (zitiert auf S. 37).

[JIR] Atlassian. *JIRA*. URL: https://www.atlassian.com/software/jira (zitiert auf S. 24).

[jQu] The jQuery Foundation. *jQuery*. URL: https://jquery.com/ (zitiert auf S. 36).

[JSP] Oracle. *JavaServer Pages Technology*. URL: http://www.oracle.com/technetwork/java/jsp-138432.html (zitiert auf S. 36).

[Kap12] K. M. Kapp. *The gamification of learning and instruction: game-based methods and strategies for training and education*. Pfeiffer, 2012 (zitiert auf S. 9, 11, 13, 14).

[KZ08] J. R. Kiniry, D. M. Zimmerman. „Secret ninja formal methods“. In: *International Symposium on Formal Methods*. Springer. 2008, S. 214–228 (zitiert auf S. 20).

[LL13] J. Ludewig, H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. dpunkt. verlag, 2013 (zitiert auf S. 21, 25).

[M2E] The Eclipse Foundation. *M2Eclipse*. URL: http://www.eclipse.org/m2e/ (zitiert auf S. 36).

[Mav] The Apache Software Foundation. *Apache Maven*. URL: https://maven.apache.org/ (zitiert auf S. 35).

[OW16] J.-P. Ostberg, S. Wagner. „At ease with your warnings: the principles of the salutogenesis model applied to automatic static analysis“. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Bd. 1. IEEE. 2016, S. 629–633 (zitiert auf S. 35).

[SBK11] S. Sheth, J. Bell, G. Kaiser. „Halo (highly addictive, socially optimized) software engineering“. In: *Proceedings of the 1st International Workshop on Games and Software Engineering*. ACM. 2011, S. 29–32 (zitiert auf S. 19).

[Sch14] J. Schell. *The Art of Game Design: A book of lenses*. CRC Press, 2014 (zitiert auf S. 14).

[Son] SonarSource SA. *SonarQube*. URL: http://www.sonarqube.org/ (zitiert auf S. 19).

[Spr] Pivotal Software. *Spring*. URL: https://spring.io/ (zitiert auf S. 36).

[Tyc] The Eclipse Foundation. *Tycho*. URL: https://eclipse.org/tycho/ (zitiert auf S. 35).

[Yee06] N. Yee. „Motivations for play in online games“. In: *CyberPsychology & behavior* 9.6 (2006), S. 772–775 (zitiert auf S. 13, 14).

[ZC11] G. Zichermann, C. Cunningham. *Gamification by design: Implementing game mechanics in web and mobile apps*. O'Reilly Media, Inc., 2011 (zitiert auf S. 10, 13, 14).

Alle URLs wurden zuletzt am 17. 07. 2016 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift