Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 6

Erweiterung der automatischen statischen Codeanalyse um Social Coding

Kai Mindermann B.Sc.

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Stefan Wagner
Betreuer/in:	Dipl.-Ing. Jan-Peter Ostberg
Beginn am:	2014/06/02
Beendet am:	2014/12/02
CR-Nummer:	D.2.4, D.2.5, H.3.4 , H.3.5 , H.5.2 , H.5.3

UNIVERSITÄT STUTTGART

Kurzfassung

Institut für Softwaretechnologie
Abteilung Software Engineering

Master of Science

Erweiterung der automatischen statischen Codeanalyse um Social Coding

von Kai Mindermann B.Sc.

In dieser Masterarbeit wird zunächst eine Definition für Social Coding hergeleitet. Danach werden verschiedenen Ansätze für Social Coding in die drei Kategorien, Kommunikation, Kooperation und Koordination des 3C-Modells sowie nach der grundlegenden Art des Ansatzes eingeteilt. Zu den analysierten Ansätzen gehören Online-Plattformen wie Stack Overflow und GitHub sowie Entwicklungsumgebungen und Erweiterungen davon wie Cloud9 und Visual Studio Anywhere. Im Weiteren werden zwei Ansätze zur Erweiterung der statischen Code-Analyse Software FindBugs um Social Coding vorgestellt. Die erste Erweiterung bietet dem Benutzer die Möglichkeit gefundene Bugs zu Online-Plattformen zu exportieren während die zweite Erweiterung ein eigenes Bug-Tracking-System mit dem Hauptaugenmerk auf einem Kommentarsystem im Quellcode-Repository des Projekts abbildet und mit einer modernen Oberfläche präsentiert.

English abstract:

A definition for the term social coding is derived first in this thesis. Afterwards different approaches for social coding are put in the three categories, communication, cooperation and coordination of the 3C-Model as well as grouped by the kind of their approach. The analyzed approaches consist of online platforms like Stack Overflow and GitHub and also development environments and extensions of them like Cloud9 and Visual Studio Anywhere. Apart from that two approaches that add social coding to the static code analysis software FindBugs are being presented. The first approach offers an export possibility to an online platform for bugs whereas the second approach implements social coding itself in form of a bug-tracking-system with focus on a commenting-system and presenting that through a modern user interface.

Danksagung

Als erstes möchte ich meinem Betreuer, Dipl.-Ing. Jan-Peter Ostberg, dafür danken, dass er mir die 6 Monate über in regelmäßigen Treffen mit konstruktiver Kritik und Gesprächen bereit stand. Als zweites möchte ich meiner gesamten Familie meinen Dank aussprechen. Ohne die mentale Unterstützung und die einen oder anderen motivierenden Leckerei wäre diese Arbeit nicht möglich gewesen. Weiterhin gebührt meinem Kommilitone und Kollege Daniel Maurer M.Sc. ein besonderer Dank für die tatkräftige Unterstützung durch das umfangreiche Korrekturlesen und die vielen weiteren Anregungen.

Ich möchte auch all denen Danken, die mich während der letzten 6 Monate in anderen Formen unterstützt haben. Dazu zähle ich Ablenkungen, Pausen und Unternehmungen die für eine gesunde Abwechslung gesorgt haben. Dadurch kam ich oft auf neue Gedanken und andere Lösungsansätze. Somit haben auch die, die gar nicht direkt mit der Arbeit in Berührung gekommen sind, doch alle bei ihrer Entstehung mitgewirkt. Vielen Dank dafür.

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Prolog

In den letzten Jahren haben sich Web-Plattformen wie GitHub zu riesigen Beschleunigern der Open-Source-Szene entwickelt. Viele Projekte haben nicht nur ihre Quellcode-Verwaltung dort hin migriert, sondern auch ihre Issue-Tracker und Dokumentation. Dies liegt einerseits an der einsteigerfreundlichen und trotzdem funktionsreichen Gestaltung und den Interaktionsmöglichkeiten der Webseite als Front-End für das Versionsverwaltungsprogramm git. Andererseits an der großen Anzahl an registrierter Entwickler die für jedes Open-Source-Projekt und die Interaktion erforderlich sind. Die Webplattformen können mittlerweile als soziale Netzwerke für Entwickler angesehen und verwendet werden. In diesem Zusammenhang gibt es auch viele weitere entwickelte und sich in der Entwicklung befindliche Technologien für die Unterstützung der Programmierer durch soziale Interaktion miteinander und dem Quellcode. Insgesamt sind diese Technologien unter dem Begriff „Social Coding“ im Sprachgebrauch geläufig.

Ein ganz anderes Gebiet ist die statische Code-Analyse. Damit wird, ohne das Programm auszuführen, versucht, nur auf Basis des Quellcodes bestimmte Arten von Fehlern in Programmen zu finden. Das Ergebnis der statischen Code-Analyse ist eine Liste von möglichen Fehlern. Es ist nicht für jeden Entwickler gleich einfach zu beurteilen, wie mit den gefundenen möglichen Fehlern umzugehen ist. Insbesondere befinden sich in der Liste heutzutage teilweise noch eine große Anzahl an sogenannten *false positives*. Um unter anderem diese *false positives* schneller erkennen zu können und das Verständnis über die gefundenen möglichen Fehler zu erhöhen, könnte es sinnvoll sein, sich der Techniken des Social Codings für die automatische statische Code-Analyse zu bedienen und diese damit um Interaktionsmöglichkeiten für die Entwickler zu erweitern. Genau die Idee wird in dieser Masterarbeit behandelt.

1.2 Aufgabenstellung

Die Hauptaufgabe ist die Erörterung der Anwendung von Social Coding für die Bearbeitung der Analyseergebnisse von statischen Code-Analysewerkzeugen. Diese Aufgabe soll dabei iterativ

durch die Bearbeitung der folgenden Teilaufgaben abgearbeitet werden. Dazu gehört zunächst die Schaffung eines Überblicks über die Ansätze des Social Coding und die Katalogisierung der vorhandenen Ansätze. Des Weiteren sollen diese Ansätze auf die Eignung für die Erweiterung der statischen Code-Analyse untersucht werden. Danach gilt es mindestens einen Ansatz zur Erweiterung eines Code-Analyseprogramms zu Implementieren.

1.3 Verwandte Arbeiten

Aufgrund des erst vor wenigen Jahren entstandenen Forschungsbereich des Social Software Engineering (SSE) (siehe Unterunterabschnitt 1.4.1.5) gibt es noch keine komplett gefestigte Literaturbasis. Die meisten Arbeiten sind nur als wissenschaftliche Artikel und erst vereinzelt als Bücher oder ähnliche umfangreiche Werke verfügbar. Des Weiteren ist das Thema dieser Arbeit, vor allem wegen Kombination zwei unterschiedlicher Gebiete, in der Literatur bis jetzt kaum behandelt worden. Dennoch gibt es einige nennenswerte Arbeiten, die hier vorgestellt werden und eine umfassende Einführung in das Thema der sozialen Medien und ihren Beitrag zur Softwareentwicklung bieten.

Mit „The (R) Evolution of Social Media in Software Engineering“ [1] haben Storey et al. einen guten und aktuellen (2014) Überblick über die Rolle von sozialen Medien in der Softwareentwicklung geschaffen.

Ausgehend davon, dass Entwickler verschiedene Medien zur Kommunikation, Zusammenarbeit, Koordination und zum Lernen verwenden, sagen sie eine grundsätzliche Änderung der Denkweise (*paradigm shift*) in der Softwareentwicklung voraus, welche bereits jetzt jeden Tag online miterlebt werden können. Danach formen sich (i) sogenannte **soziale Programmierer** die sich aktiv in Online-Entwicklergemeinden beteiligen. Weiterhin soll es (ii) einen rasanten Anstieg von neu erschaffenen und sich verteilenden Technologien sowie durch Crowdsourcing erstellte Inhalte geben. Als letztes (iii) werden sich Ökosysteme um diese Technologien, Medien, Inhalte und Entwickler formen, die günstige und hindernissfreie Möglichkeiten zum Teilen eben dieser bieten. Dies führt zu einer neuen, **sozialen, Ära**.

Weitere wichtige Erkenntnisse zur Entstehung der sozialen Ära sind, dass die Diskussion von Programmierproblemen mittlerweile häufig direkt innerhalb der Diskussionssysteme der Code Repositories stattfindet, anstatt per E-Mail o. ä. [2]. Nichtsdestoweniger sind wichtige Informationen zum Quellcode manchmal über mehrere verwendete Kommunikationskanäle verteilt. Nach einer zwar schon etwas älteren (2004) Studie von Gutwin, Penner und Schneider war es damals schwierig sicherzustellen, dass Informationen von den richtigen Leuten rechtzeitig gelesen wurden. Durch die vielen Folge- und Hinweis-Möglichkeiten der Online-Plattformen die wir in der Analyse kennen lernen, sehen wir dies für viele Projekte nicht mehr als komplett korrekt an. Dennoch gibt es nun noch mehr Kommunikationskanäle die beobachtet werden müssen.

In [4] wurde untersucht, inwiefern mit Quellcodebeiträgen zu Open-Source-Projekten auf GitHub in Bezug zu Softwaretests umgegangen wird. Dabei haben Pham et al. durch eine Umfrage herausgefunden, dass das unterschiedliche Vertrauen der Projektleiter in den von verschiedenen Benutzern geschriebenen Code ein wichtiges Kriterium für den zu treibenden Aufwand bei der Integration und dem Review solcher Beiträge ist. Weiterhin geben sie

Vorschläge, wie mehr und bessere Tests bei Quellcodebeiträgen gefördert werden können. Der Vorschlag ist hauptsächlich generell anwendbar und lautet, die Hindernisse für Benutzerbeiträge zu entfernen. Umgesetzt werden kann dies durch das Anbieten von Testvorgaben, Beispiel-Tests und einer einfach einzurichtenden Test-Infrastruktur. Genau solche verringerten Hindernisse sind auch der Grund für die sogenannten *Drive-by-commits*[1] auf GitHub. Viele Benutzer gaben in der Umfrage an, dass sie einen *Pull-Request* (Quellcodebeitrag) nur deshalb machten, weil es so einfach und schnell möglich ist („Since it is so easy to send a pull request, I contribute more changes that I would not have engaged in otherwise."). Ein weiterer wichtiger Punkt ist, dass die soziale Transparenz auf GitHub die Aktionen der Entwickler leichter nachvollziehbar macht. Genau diese Transparenz motiviert laut [5] andere etwas zum Projekt beizutragen, trotzdem ist es gerade für nicht technische Benutzer nachweislich schwierig, manche der Funktionen der Plattform zu verwenden [6] . Trotz der Abstraktion ist ein gewisses Grundwissen über die Funktionsweise der Quellcodeverwaltung mit git nötig.

Ähnliches wurde von Tsay, Dabbish und Herbsleb in [7] herausgefunden. Projektleiter stützen sich laut der durchgeführten statistischen Analyse sowohl auf die bisherige Interaktion mit dem Projekt als auch auf die Stärke der sozialen Verbindungen des beitragenden Entwicklers (Anzahl an anderen Entwicklern die diesem Entwickler auf GitHub folgen sowie Verbindung zum Projektleiter), um zu Entscheiden einen Quellcodebeitrag zu akzeptieren. Des Weiteren wurde herausgefunden, dass weniger kommentierte/diskutierte Beiträge tendenziell mit einer höheren Wahrscheinlichkeit akzeptiert werden.

Kumar und Gupta versuchen in „Evolution of Developer Social Network and Its Impact on Bug Fixing Process" [8] anhand der zeitlichen Veränderung des sogenannten sozialen Netzwerks der Entwickler (*Developers Social Network (DSN)*) der Eclipse Softwareentwickler herauszufinden, wie diese mit der Effizienz des Bug-Fix-Prozess korreliert. Herausgekommen ist, dass die durchschnittliche Zeit einen Fehler zu beheben stark mit der Anzahl an Cliquen[2] des DSN korreliert. Die Schlussfolgerung ist, dass je stärker die Kommunikationsstruktur modular, also aufgeteilt in Module die in sich stark verbunden und untereinander wenig verbunden sind, aufgebaut ist, desto mehr und schneller werden Fehler behoben.

Eine Untersuchung „Warum Entwickler Automatische Statische Code-Analyse (ASA) Tools nicht verwenden" findet sich in [9] von Johnson et al. Die durchgeführte Umfrage ergab, dass sich Entwickler wünschen, dass sie nicht in ihrem Arbeitsablauf gestört oder unterbrochen werden und/oder, dass sie die Ergebnisse schneller mitgeteilt bekommen. Eine Möglichkeit dafür ist der Vorschlag die Code-Analyse als Hintergrundprozess oder nur an vorhandenen Unterbrechungspunkten des Arbeitsablaufs wie beim Kompilieren laufen zu lassen. Eine weitere Anmerkung war, dass sogenannte *quick fixes* vorgeschlagen werden sollen inklusive einer Ansicht, was sich dadurch im Code ändern würde. 19 von 20 Teilnehmer merkten weiterhin an, dass die gefundenen Bugs nicht so präsentiert werden um ablesen zu können was die Ursache des Problems ist und/oder was geändert werden muss um das Problem zu beheben. Insgesamt liegt dies aber auch an vielen *false positives* bei denen eben nicht sicher ist, ob es sich um einen Fehler handelt und somit schlechter beschrieben werden

[1] Mit Drive-by-commits werden Quellcodebeiträge bezeichnet, die von eigentlich nicht involvierten oder sich normalerweise mit anderen Projekten beschäftigenden Entwicklern nebenher beigetragen werden.

[2] Eine Clique beschreibt in einem ungerichteten Graphen eine Teilmenge von Knoten, die jeweils mit einer Kante verbunden sind. Anzahl der Cliquen bedeutet hier für einen Knoten, in wie vielen Cliquen er enthalten ist. Knoten entsprechen Entwicklern und Kanten bedeuten, dass sich diese beiden Entwickler kennen oder miteinander zu tun haben.

kann was das Problem sein könnte und wann es kein Problem ist. Als Ziel für die zukünftige Entwicklung von ASA-Tools soll eine bessere Integration in die Arbeitsabläufe von Entwicklern und eine bessere Präsentation der gefundenen Fehler angestrebt werden. Mehr zur ASA gleich in Unterabschnitt 1.4.3.

Nach [10] sind viele Benutzer von ASA-Tools auch an den Fehlern mit einer kleinen *severity* (Schadensausmaß, Gewichtung) interessiert. Weiterhin ist es laut [11] nicht nur wichtig, wem ein Bug zum Beheben zugewiesen wird, sondern auch wer die Zuweisung vornimmt.

Durch eine systematische Überprüfung vorhandener Literatur wurde von [12] herausgefunden, dass sich die meisten Arbeiten aus dem Bereich Distributed Software Development (DSD) zwischen 2006 und 2010 mit der Koordination (siehe 3C-Modell Unterunterabschnitt 1.4.1.2) beschäftigen und die wenigsten mit der Kommunikation.

Neben den gerade vorgestellten Arbeiten, gibt es noch weitere die sich um noch detailliertere Probleme kümmern. Diese werden aber in den im Zusammenhang stehenden entsprechenden Abschnitten behandelt. Nachdem wir nun bereits einen guten Eindruck von den behandelten Themen haben, werden im nächsten Kapitel wichtige Grundlagen dazu erläutert.

1.4 Grundlagen

Diese Arbeit verwendet viele Fachbegriffe aus dem Gebiet der Softwareentwicklung. Viele dieser Fachbegriffe werden auch nicht übersetzt, weil es im Deutschen keinen, den gleichen Sinn wiedergebendes, Wort gibt oder das Wort nicht gebräuchlich ist. Für Begriffe und Ausdrücke für die eine im Sprachgebrauch verwendete deutsche Bezeichnung geläufig ist, werden diese aber auch verwendet.

Einige der wichtigen verwendeten Fachbegriffe und Fachgebiete werden in diesem Abschnitt näher erläutert. Insbesondere werden die umschließenden Forschungsgebiete und Themen für den Begriff Social Coding hergeleitet. Diejenigen die sich aber schon mit Social Software Engineering (SSE) und Automatische Statische Code-Analyse (ASA) auskennen, können den Rest von diesem Kapitel überspringen und direkt in Kapitel 2 bei der Analyse vorhandener Ansätze wieder einsteigen.

1.4.1 Social Coding und Kollaboratives Arbeiten

Um den Begriff Social Coding einführen zu können, werden zunächst weitere Begriffe die mit Social Coding im Zusammenhang stehen erklärt. Die vorhandenen Begriffe überschneiden sich thematisch stark. Trotzdem ist es wichtig hier darauf einzugehen, um später die richtige Terminologie verwenden zu können.

1.4.1.1 Computer Supported Cooperative Work

Computer Supported Cooperative Work (CSCW) (dt. Rechnergestützte Gruppenarbeit) ist „ein interdisziplinäres Forschungsgebiet aus Informatik, Soziologie, Psychologie, Anthropologie,

[...], Wirtschaftswissenschaften, Medienwissenschaft und verschiedenen weiteren Disziplinen [...], das sich mit Gruppenarbeit und Zusammenarbeit und den die Gruppenarbeit unterstützenden Informations- und Kommunikationstechnologien befasst“ ([13]). Dieses schon sehr lange existierende Forschungsgebiet entspricht der äußeren Abgrenzung des Themengebiets innerhalb dem sich die Arbeit nun im Folgenden weiter bewegt.

1.4.1.2 Collaborative Software/3C-Modell

Die Anwendung des Forschungsgebiets CSCW in konkreter Software wird als *Groupware* oder *collaborative software* bezeichnet. Groupware wird oft anhand der drei Kategorien Kommunikation (communication), Kooperation (cooperation) und Koordination (coordination) (3K- bzw. **3C-Modell**) klassifiziert [14]. Bei einer Einteilung in die Kategorien muss beachtet werden, dass diese sich gegenseitig beeinflussen, siehe auch Abbildung 1.1. Dieses Modell wird auch später in der Analyse (Kapitel 2) zur Einordnung der vorhandenen Ansätze verwendet.

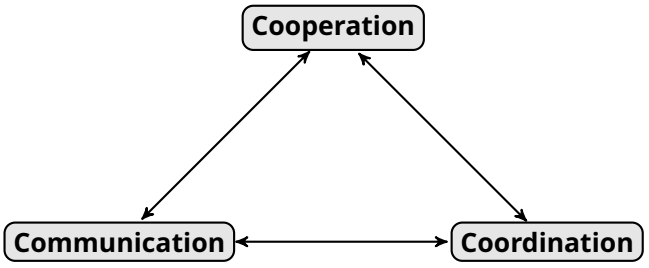


ABBILDUNG 1.1: 3C (Kollaborations-)modell

1.4.1.3 Soziale Medien und soziale Netzwerke

Soziale Medien (social media) ist nach Kaplan und Haenlein: „eine Gruppe von Internetanwendungen, die auf den technologischen und ideologischen Grundlagen des Web 2.0 aufbauen und die Herstellung und den Austausch von *User Generated Content* ermöglichen“ [15]. Sozial bedeutet in diesem Zusammenhang, dass Möglichkeiten zur Teilnahme durch die Transparenz der Identität, Inhalte und Interaktion unterstützt und gefördert werden [5].

Mit den heutzutage verbreiteten Techniken, Webseiten und Apps einfach zugänglich zu Gestalten, ist es möglich damit neue eigene Medien zu schaffen, welche sich mehr auf den Benutzer konzentrieren und diesen stärker einbinden können. Per Browser als Webseite oder als App auf mobilen Geräten verfügbar, bieten sie die unterschiedlichsten Möglichkeiten für Kommunikation, Zusammenarbeit, Unterhaltung und Wissensmanagement. Sie zeichnen sich vor allem durch eine hohe Benutzerfreundlichkeit und Zugänglichkeit sowie Multimedialität aus. Als bekannte Beispiele für soziale Medien lassen sich soziale Netzwerke wie Facebook, Twitter und Kollektivprojekte wie Wikipedia aufführen.

1.4.1.4 Social Collaboration und Social Software

Stark für die Unterstützung der Kommunikation zwischen Menschen ausgeprägte *Groupware*, die vorrangig auf Basis des World Wide Web und im Zusammenhang mit sozialen Medien

verwendet wird, wird als Social Software bezeichnet [16]. Die Arbeit mit Social Software wird weiterhin als *Social Collaboration* bezeichnet.

1.4.1.5 Social Software Engineering (SSE)

SSE ist ein Forschungsbereich der Softwareentwicklung, welcher sich mit den sozialen Aspekten der Softwareentwicklung und der Software befasst [17]. Dabei wird besonders die soziale Integration von Entwicklern, soziale Aktivitäten sowie die Kommunikation, Kooperation und Koordination (3C) betrachtet. Dazu fand bereits im September 2008 der erste internationale Workshop für Social Software Engineering and Applications (SoSEA) statt. Mittlerweile finden Konferenzen aber nur noch unter dem Namen SSE gemeinsam statt.

1.4.1.6 Social Coding

Der Begriff Social Coding wurde erstmals Ende 2007 mit der Veröffentlichung von Kogbox[3] im Zusammenhang mit Softwareentwicklung verwendet [18]. Deutlich populärer wurde der Begriff durch die den Begriff prägende, im Jahr 2008 veröffentlichte Plattform, GitHub[4]. GitHub erzeugte eine Art soziales Netz um die dort verwalteten Open-Source-Projekte und damit involvierten Entwickler. Es vereinfachte die häufigen Aufgaben der Softwareentwicklung mit git und machte diese für jeden interessierten Entwickler direkt verfügbar. Insbesondere das schnelle Erzeugen und Verwalten von Forks (Abspaltungen) wurde sehr benutzerfreundlich umgesetzt und damit wurde auch die Basis für die heutige starke Verbreitung von GitHub und ähnlichen Plattformen geschaffen.

Eine konkrete Definition des Begriffs Social Coding wurde 2012 im Netz veröffentlicht:

Definition 1.1 *Social Coding nach Rouse: „Social coding is an approach to software development that places an emphasis on formal and informal collaboration.[. . .] the term can be used to describe any development environment that encourages discussion and sharing.“ [19]*

Nach Definition 1.1 ist Social Coding ein auf formeller und informeller Zusammenarbeit basierender Ansatz für die Softwareentwicklung. Im Folgenden möchte ich den Begriff etwas genauer definieren und gleichzeitig einordnen:

Definition 1.2 *Social Coding ist ein Modewort für aktuelle Quellcode-nahe Technologien und Entwicklungen aus dem Bereich Social Software Engineering (SSE).*

Damit kann Social Coding als Unterbegriff für SSE eingeordnet werden.

1.4.1.7 Social Coding Software

Nach Definition 1.2 ist Software, welche Social Coding anwendet oder unterstützt, damit eine Art Social Software für die Unterstützung von Softwareentwicklern. Dazu zählen auch Social Coding Seiten und -Plattformen wie GitHub und Bitbucket.

[3] `http://kogbox.com` (Besucht am 24.11.2014)
[4] `https://github.com/` (Besucht am 24.11.2014)

Um die gerade eingeführten Begriffe schnell einordnen zu können, sind sie in Abbildung 1.2 dargestellt.

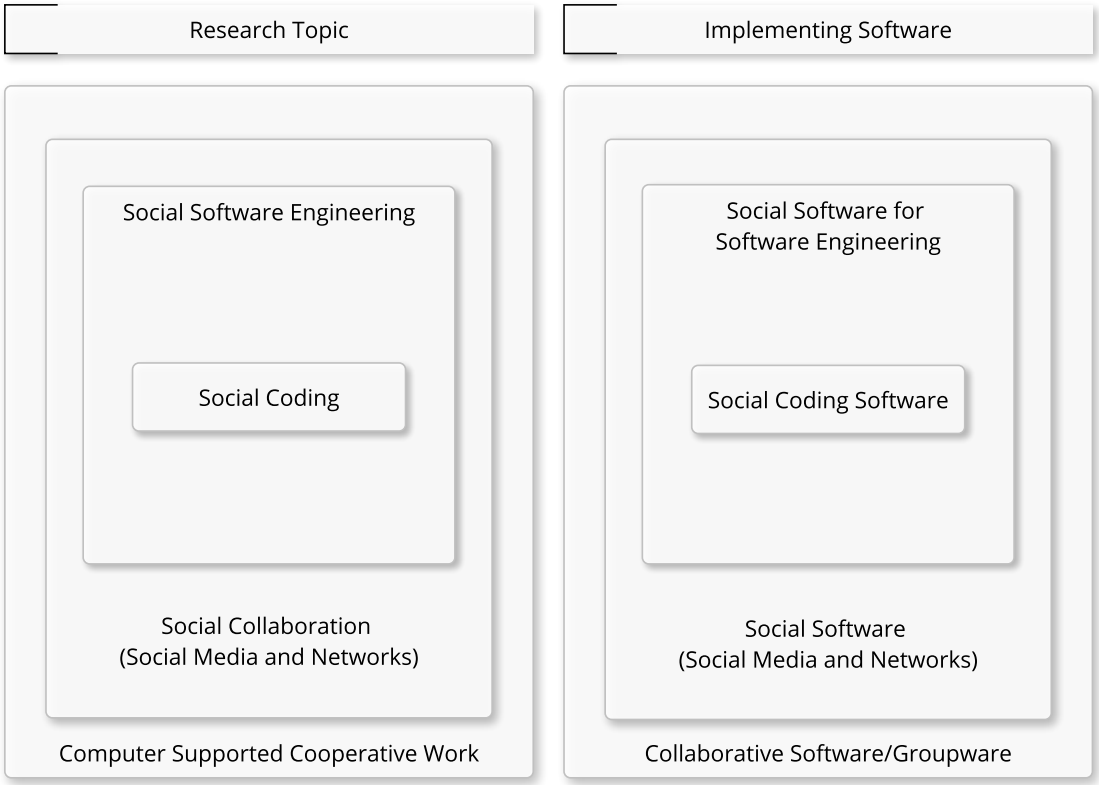


ABBILDUNG 1.2: Visualisierung für die Forschungsbereiche um Social Coding und die dazugehörige Anwendung in Software.

1.4.1.8 Kommunikationskanal

Mit Kommunikationskanal werden hier bestimmte Arten von direkter als auch indirekter, die Zusammenarbeit in irgendeiner Weise unterstützenden, Kommunikationstechnologien bezeichnet. Das heißt also auch, dass ein Kommunikationskanal nicht nur für die Kommunikation verwendet wird, sondern auch die Kollaboration und Koordination nach dem 3C-Modell unterstützt. Beispielsweise ist ein Chat ein normalerweise direkter und eine Kommentarliste ein indirekter Kommunikationskanal. Dies hängt im Allgemeinen aber von der Implementierung ab. Des Weiteren zählt auch die Darstellung und damit die Kommunikation von Informationen durch die Software/Plattform für den Entwickler als Kommunikationskanal.

1.4.2 Softwareorganisation

1.4.2.1 Bug-Tracker/Issue-Tracker

Für die in Kapitel 3 vorgestellte Implementierung wird hauptsächlich auf den Begriff **Bug-Tracker** verwiesen. In einem Bug-Tracker werden im Allgemeinen alle zu einem Softwareprojekt gehörenden und gefundenen Bugs eingetragen. Je nach verwendeter Bug-Tracker-Software haben die eingetragenen Bugs verschiedene zugehörige Statusinformationen. Dazu gehört häufig der allgemeine Bearbeitungsstatus wie „Bearbeitet/Nicht bearbeitet“, „Unbestätigt/Bestätigt“, eine Zuordnung zu einem verantwortlichen Entwickler sowie die Angabe der Schwerwiegendheit des Bugs. Weiterhin werden im Rahmen von Social Coding Plattformen häufig nicht nur Bugs im Quellcode, sondern auch (neue) Anforderungen und Fehler in der Dokumentation im selben, in dem Zusammenhang dann als Issue-Tracker bezeichneten, Bug-Tracker verwaltet. Dazu kommt, dass neben der allgemeinen Beschreibung normalerweise auch eine Diskussionsmöglichkeit im Rahmen von Kommentaren für einen Bug vorgesehen ist.

1.4.2.2 Versionsverwaltung

Für die zweite Implementierung im Kapitel 4 werden viele Begriffe aus dem Bereich der Versionsverwaltung verwendet. Ein grundsätzliches Verständnis wird im weiteren dann vorausgesetzt.

Um einerseits den Entwicklungsverlauf zu dokumentieren und andererseits verschiedene Versionen und Varianten einer Software wiederherstellen zu können und an diesen auch in Zukunft Verbesserungen und Änderungen durchführen zu können, werden Versionsverwaltungssysteme eingesetzt. In diese wird normalerweise nach erfolgten Änderungen manuell durch die Entwickler der neue Stand in Form vom Quellcode und zum Quellcode gehörenden Dateien eingetragen. Das Versionsverwaltungssystem speichert die Änderungen und eventuelle Meta-Daten wie Datum, Autor und Kommentare für jede Eintragung mit ab. Die gesamten Informationen werden in einem sogenannten **Repository** abgelegt.

Es gibt verschiedene Arten von Versionsverwaltungssystemen. Ein häufig eingesetztes System ist git[5]. Dieses ist dezentral aufgebaut und jeder Entwickler hat eine vollständige oder teilweise Kopie des Repositories. Durch den dezentralen Aufbau können Entwickler ohne Zugriff auf eine zentrale Instanz lokal ihre Änderungen eintragen und mit beteiligten einzelnen Entwicklern austauschen. Normalerweise gibt es aber immer ein als Zentral deklariertes Haupt-Repository in das alle Änderungen eingetragen werden. Im Folgenden wird das Eintragen in das Repository als **Commit** bezeichnet. Um verschiedene Entwicklungszweige zu unterstützen wird dies mit Abzweigungen in der Quellcodegeschichte realisiert. Diese Entwicklungs-Zweige, im folgenden **Branches** genannt, können bei git zu jedem Zeitpunkt wieder zusammengeführt werden.

[5] `http://git-scm.com/` (Besucht am 24.11.2014)

1.4.3 Statische Code-Analyse

Automatische Statische Code-Analyse (ASA) ist ein Teil der allgemeinen Programmanalyse bei der für verschiedene Anforderungen Computerprogramme automatisch analysiert werden. Mit statischer Code-Analyse werden Software-Testverfahren bezeichnet, die ein, als Quelltext vorliegendes, Programm anhand eben dieses Quelltextes auf bestimmte Fehler hin untersuchen. Im statischen Fall ohne das Programm auszuführen bzw. ohne Informationen zu verwenden die erst zur Laufzeit zur Verfügung stehen würden. Die Code-Analyseverfahren können teilweise von Hand durchgeführt werden, der Fokus liegt aber auf der automatischen Code-Analyse durch Analysesoftware. Einen ausführlichen Einblick in das Thema der Programmanalysen bietet das Buch *Principles of Program Analysis* von Nielson, Nielson und Hankin [20].

Je nach Sprache gibt es mehr oder weniger Programme um die Automatische Statische Code-Analyse (ASA) ausführen zu können. Einige Vertreter sind hier aufgelistet:

- FindBugs für Java. FindBugs hat sogar bereits eine Integration für Kommentare für die Ergebnisse, siehe Unterabschnitt 2.3.2.
- Secure Programming Lint (Splint) für C
- Cppcheck für C++
- OClint für C, C++ und Objective-C

Viele Programme werden dabei noch als *Linter* (engl. für Fussel) bezeichnet, da das erste Programm zur statischen Code-Analyse **Lint** genannt wurde. Eine umfangreiche Liste von verfügbaren ASA-Programmen findet sich unter [21].

1.4.3.1 Code-Analyseverfahren

Je nach Programm werden unterschiedliche Code-Analyseverfahren unterstützt. Dies wirkt sich auch in der Anzahl und Art der auffindbaren Fehler sowie in der Laufzeit aus. Die Code-Analyseverfahren reichen von einfachem Style-Checking und Einhalten von Programmierrichtlinien bis zu formalen, mathematische Beweise nutzenden, Methoden. Die einfachen Verfahren werden normalerweise von allen Programmen unterstützt und die Erklärung der Fehler bzw. wie diese gefunden werden können sind einfach. Um weitere Fehler finden zu können, bedarf es komplexerer Methoden welche im Folgenden angesprochen werden:

- **Muster-basierte Code-Analyse**: Für diese werden für eine bestimmte Art von Fehler (im Sinne von schlechter Code, langsamer Code, falscher Code oder usw.) bestimmte Erkennungsmuster festgelegt. Auf Basis dieser wird der Code nach dem Muster untersucht und falls es gefunden wird, ein Fehler gemeldet. Hier können *false positives* dadurch entstehen, dass das Muster nicht genau genug ist oder der Typ des Fehlers an sich eine gewisse Ungenauigkeit hat.

- **Datenflussanalyse**: Aufbauend auf dem Kontrollflussgraphen, der die zeitliche Abarbeitungsreihenfolge der Befehle des Programms abbildet, werden Fakten über Werte durch diesen Flussgraphen überall dorthin propagiert wo sie Gültigkeit besitzen. Dabei werden diese Fakten in der Praxis häufig nur am Ein- und Ausgang der Grundblöcke, in denen keine Verzweigungen sondern nur ein sequentieller Fluss möglich ist, des Kontrollflussgraphen gespeichert. Das grundlegende Vorgehen bei der Datenflussanalyse ist die Erzeugung von einer *Erzeuge*- und einer *Lösche*-Menge pro Grundblock. Anhand diesen Mengen kann nachvollzogen werden, welche Fakten von einem Grundblock erzeugt oder gelöscht werden und damit je nach Fehlertyp ob die Fakten zu einem bestimmten Fehler führen oder nicht. Aufmerksame Leser haben bemerkt, dass von Mengen geredet wurde. Aufgrund dieser Tatsache kann es auch mehrere mögliche Fakten auf Basis der statischen Analyse geben. Dies wird vor allem durch die gleich betrachtete Zeigeranalyse deutlich. Bei manchen Fehlertypen führt dies dazu, dass ein Fehler *wahrscheinlich* ist, aber nicht zu 100 % beim Ausführen des Programms auftreten muss.
- **Zeigeranalyse**: Ein wichtiges Konstrukt in Programmen sind Zeiger. Zeiger-Variablen enthalten als Wert eine Speicheradresse an der sich das referenzierte Objekt befindet. Dies ermöglicht theoretisch eine Zuweisung von beliebigen Speicheradressen/-bereichen. Um konkrete Aussagen treffen zu können gilt es herauszufinden auf welche Objekte ein Zeiger tatsächlich zeigen kann. Im Optimalfall wäre das Ergebnis ein einzelnes Objekt. In der Praxis ist es aber eine Menge von möglichen Objekten. Die eingesetzten Verfahren können dazu verschiedene Kriterien erfüllen. Dazu gehören hauptsächlich ob der Kontrollfluss, die Objektstrukturen und/oder der Kontext beachtet werden. Des Weiteren gibt es Verfahren deren Ergebnisse nicht immer zutreffend sind und *false positives* enthalten. Danach ergibt sich auch die Laufzeitkomplexität der Verfahren. Wichtige grundlegende Verfahren sind zum Beispiel das Verfahren von Steensgaard [22] oder das von Wilson und Lam [23].
- **Abstrakte Interpretation**: Bereits 1977 von Cousot und Cousot [24] beschrieben, geht es bei der abstrakten Interpretation darum, durch gezieltes Weglassen von Informationen (Abstraktion) eine Näherung an die Semantik des später ablaufenden Programms zu erhalten. Beispielsweise kann ohne die konkreten Werte für die spätere Ausführung der Variablen zu kennen, mit dem Typ der Variablen weiter gerechnet werden um den Rückgabetyp einer Funktion zu ermitteln.

Eine wichtige Erkenntnis in diesem Zusammenhang ist, dass mit statischer Code-Analyse nur die **Anwesenheit von Fehlern** bestätigt werden kann. Das bedeutet, dass nur solche Fehler(-Klassen) gefunden werden, die man vorher allgemein definiert hat. Noch unbekannte Fehler werden nicht gefunden.

Der Grund warum es separate Programme für die statische Code-Analyse gibt und die Verfahren nicht alle in vollem Umfang in Compiler eingebaut werden ist folgender. Compiler müssen zwischen der Zeit die für das Kompilieren benötigt wird und den gemachten Optimierungen abwägen. Sie dürfen nicht zu lange brauchen aber müssen trotzdem ein gut optimiertes Programm abliefern. Des Weiteren enthalten die Ergebnisse der statischen Code-Analyse *false positives*. Die Fehlerausgabe eines Compilers kann zwar angepasst werden, aber hier ist es für den Benutzer möglicherweise verwirrend wenn Fehler angezeigt werden, die in Wirklichkeit

keine sind. Deshalb findet man die Code-Analyseverfahren die *false positives* liefern können hauptsächlich in separaten Code-Analyseprogrammen.

1.4.3.2 Fehlerarten

Die von statischen Code-Analyse Programmen gefundenen Fehler lassen sich je nach verwendeter und kombinierter Code-Analyseverfahren oder Programm verschiedenen Kategorien zuordnen. Eine mögliche Einteilung nimmt FindBugs, welches auch in der Implementierung verwendet wird, wie folgt vor:

- Korrektheit
- Schlechte Angewohnheiten
- Experimentell
- Internationalisierung
- Bösartige Schwachstellen
- Multithread Korrektheit
- Performanz
- Sicherheit
- Verdächtiger Code

Weiterhin können einzelne Fehler nach ihrer *severity* (Schadensausmaß, Gewichtung) eingeteilt werden um schwerwiegende Fehler zuerst beheben zu können.

Einige Beispiele für von ASA gefundenen Fehlertypen sind Null Pointer Dereferenzierung (Korrektheit), Undefinierte Variablen (Korrektheit), Fehlende Typüberprüfungen (Sicherheit), Speichermanagementfehler (Sicherheit, Performanz), Aliase (Zwei oder mehr Namen/Zeiger auf die gleiche Speicherstelle) (Performanz, Sicherheit), Pufferüberläufe (Sicherheit) sowie unbenutzter/unerreichbarer Code (Performanz, Schlechte Angewohnheiten).

1.4.3.3 Ergebnisse/Gefundene Bugs

Nachdem die ASA des Programms abgeschlossen ist, wird eine Liste mit möglichen Fehlern vom Programm in irgendeiner Art dem Benutzer präsentiert. An diesem Punkt setzt diese Arbeit an. Die Anzeige, und die damit verbundenen Interaktionsmöglichkeiten, dieser Ergebnisse unterscheidet sich von Programm zu Programm stark. Als Beispiel wird das Eclipse Plugin FindBugs verwendet. In Abbildung 1.3 sieht man eine teilweise ausgeklappte Liste verschiedener möglicher Bugs der Literaturverwaltungssoftware JabRef. Man sieht wie die Bugs per Voreinstellung nach ihrer *severity* und dann nach der Fehlerklasse sortiert sind. Die Detailansicht eines der Bugs ist in Abbildung 1.4 dargestellt. Im linken Bereich werden allgemeine Informationen zum Bug einfach dargestellt, im rechten Bereich gibt es alle weiteren, auf den verfügbaren Variablen der Java-Klassen-Repräsentation der Fehler basierenden,

Informationen zum Bug. Die Benutzeroberflächen, falls existent, anderer Programme sind vom Funktionsumfang vergleichbar und werden deshalb nicht weiter vorgestellt.

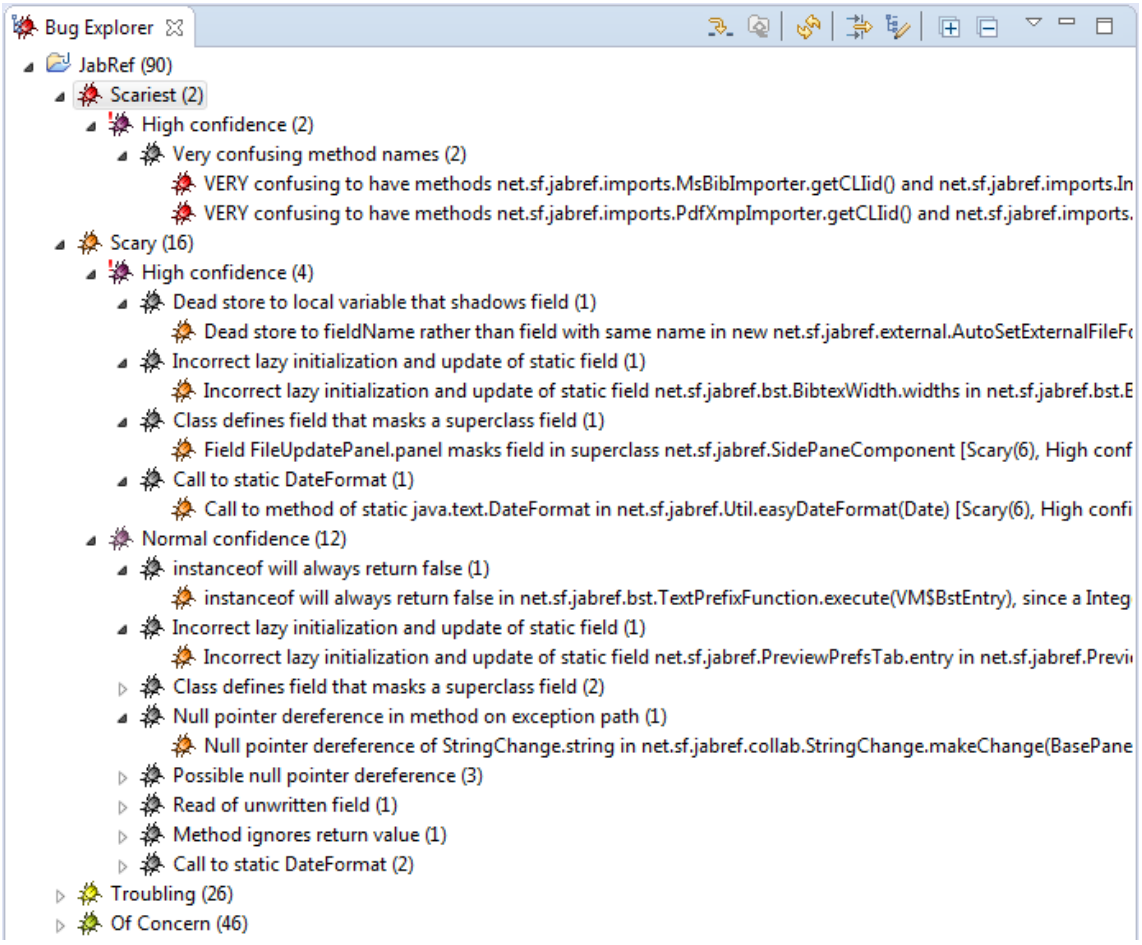


ABBILDUNG 1.3: Beispiel der Liste von möglichen Fehlern im Eclipse Plugin von FindBugs. Diese Bugs wurden in der aktuellsten (03.11.14) Version von JabRef gefunden.

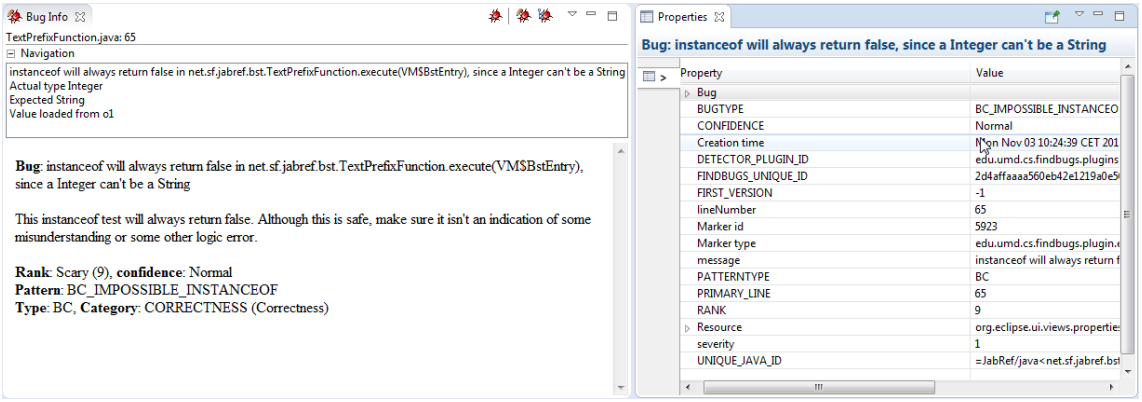


ABBILDUNG 1.4: Beispiel der Detailansicht eines möglichen Fehlers im Eclipse Plugin von FindBugs. Dieser Bug wurden in der aktuellsten (03.11.14) Version von JabRef gefunden.

Die bereits in den verwandten Arbeiten genannte Publikation „Why Don't Software Developers Use Static Analysis Tools to Find Bugs?" [9] gibt Hinweise was verbessert werden muss.

Nun soll herausgefunden werden, inwiefern mit Social Coding eine effizientere Be- und Verarbeitung der Ergebnisse möglich ist. Dazu werden im nächsten Kapitel zunächst vorhandene Ansätze für Social Coding untersucht. Danach werden zwei geeignete Ansätze als Erweiterung eines vorhandenen ASA-Programms auf Basis der Erkenntnisse aus der Analyse implementiert.

Kapitel 2

Analyse der Ansätze für Social Coding

In diesem Kapitel werden vorhandene Ansätze für Social Coding bzw. Social Software Engineering analysiert. Zunächst werden dazu die Kriterien, anhand denen die Analyse vorgenommen wird, vorgestellt. Danach folgen die Abschnitte in denen die Programme, Tools und Plattformen analysiert werden.

2.1 Betrachtungspunkte bei der Analyse

2.1.1 Auswahl Unterstützter Kommunikationskanäle

Es werden nicht alle unterstützten Kommunikationskanäle untersucht, sondern nur diejenigen die für eine soziale Interaktion relevant sind.

2.1.2 Bewertung der Umsetzung einzelner Kanäle

Um die Kommunikationskanäle und ihre Interaktionsmöglichkeiten möglichst objektiv bewerten zu können, werden die drei Kategorien des 3C-Modells, Kommunikation, Kooperation und Koordination verwendet. Für eine genauere Einordnung werden pro Kategorie eine Einteilung in fünf Level eingeführt. Level 1 wird dabei eine sehr geringe Unterstützung und Level 5 eine sehr umfangreiche und gute Unterstützung der jeweiligen Kategorie bedeuten. Um zu entscheiden auf welchem Level die Umsetzung des Kanals liegt wird ein Punktemodell verwendet. Für jede Frage gibt es zwischen 0 und 4 Punkten. Diese Punkte entsprechen der Güte der Umsetzung entsprechend einer konkreten Frage. Es gibt 5 Fragen pro Kategorie, somit gibt es maximal 20 Punkte. Damit werden die Level wie folgt aus der Summe aller Punkte pro Kategorie berechnet:

0–4 Punkte	5–8 Punkte	9–12 Punkte	13–16 Punkte	17–20 Punkte
Level 1	Level 2	Level 3	Level 4	Level 5

Folgende Fragen werden verwendet:

Kommunikation:

Frage 1 (KommQ1): Ist Kommunikation möglich?

Frage 2 (KommQ2): Werden Metadaten zur Kommunikation dargestellt und wird die Kommunikation formatiert präsentiert (z. B. Datum, Autor sowie entsprechendes Layout)?

Frage 3 (KommQ3): Ermöglicht die Darstellung eine schnelle Erfassung des Inhalts (z. B. mittels Syntaxhervorhebung)?

Frage 4 (KommQ4): Gibt es Interaktionsmöglichkeiten innerhalb der Kommunikation (z. B. Anklicken von Nutzern/Ressourcen)?

Frage 5 (KommQ5): Ist direkte Kommunikation möglich (z. B. mittels Instant Messaging/Chat)?

Kooperation:

Frage 1 (KoopQ1): Ist eine Unterstützung der Kooperation vorhanden (z. B. grundsätzlich für mehrere Benutzer ausgelegt)?

Frage 2 (KoopQ2): Wird die Kooperation gefördert (z. B. Hürden werden reduziert um Beiträge von anderen Benutzern zu erleichtern)?

Frage 3 (KoopQ3): Ist die Kooperation effizient möglich?

Frage 4 (KoopQ4): Gibt es Interaktionsmöglichkeiten innerhalb der Kooperation (z. B. Chat direkt aufrufbar)?

Frage 5 (KoopQ5): Ist direkte Kooperation möglich (z. B. mittels Live-Editing mit mehreren Benutzern)

Koordination:

Frage 1 (KoorQ1): Ist eine Unterstützung der Koordination vorhanden?

Frage 2 (KoorQ2): Wird die Koordination gefördert (z. B. Hürden werden reduziert, Koordinationsmöglichkeiten werden vorgeschlagen)?

Frage 3 (KoorQ3): Können Koordinationsmöglichkeiten von Betroffenen schnell erkannt werden?

Frage 4 (KoorQ4): Gibt es Interaktionsmöglichkeiten innerhalb der Koordination (z. B. mittels Nennen von Benutzern)?

Frage 5 (KoorQ5): Ist direkte Koordination möglich (z. B. mittels direkter Zuweisung von Aufgaben an Benutzer)?

Jede Frage kann mit **Ja** oder **Nein** beantwortet werden, was jeweils 4 oder 0 Punkten entspricht. Es ist aber auch möglich, falls die Frage nur teilweise erfüllt wird, einen, zwei oder drei Punkte zu vergeben.

Hierbei wird zur Visualisierung eine Zuordnung, wie in Abbildung 2.1 dargestellt, verwendet. Dabei wird das in Abbildung 1.1 dargestellte 3C-Modell um die hier definierten Level ergänzt. Für jedes Level gibt es einen Kreis, der innerste und kleinste Kreis ist für das Level 1 und der größte Kreis ist für das Level 5. Durch die Platzierung auf den nicht dargestellten Achsen für jede Kategorie, kann eine einfach abzulesende Visualisierung der Unterstützung für jede Kategorie erreicht werden.

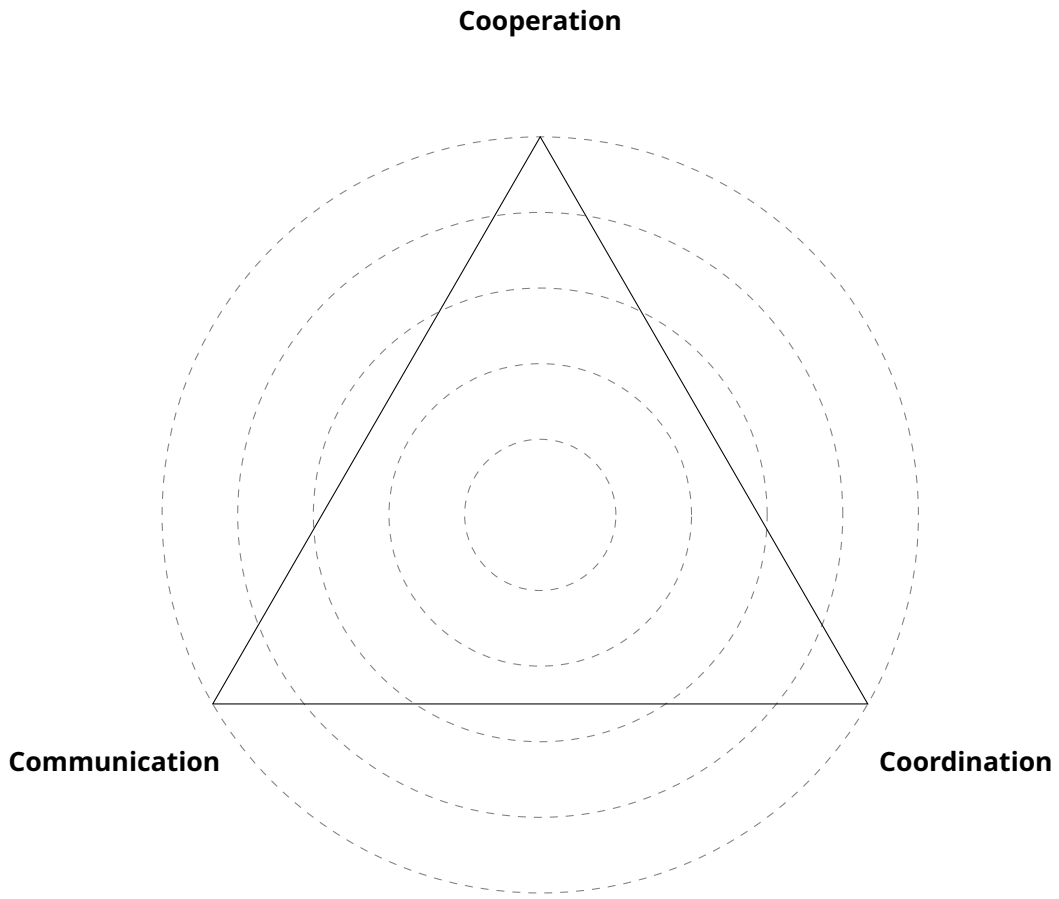


ABBILDUNG 2.1: Visualisierung des Bewertungsergebnisses durch die Markierung der drei Kategorien auf dem, dem Level entsprechenden, Kreis. Der innerste Kreis entspricht dem kleinsten Level 1.

Die im Folgendenen analysierten Ansätze haben wurden aufgrund ihrer Bekanntheit und Verbreitung sowie aufgrund unseren Erfahrungen ausgewählt. Diese stellen nur einen kleinen Teil der verfügbaren Ansätze dar, geben aber trotzdem einen guten Eindruck über die verfügbaren Technologien. Insbesondere ähneln sich viele Ansätze stark und unterscheiden sich nur in den dahinterstehenden Firmen.

2.2 Vorhandene Ansätze in kommerzieller Software

2.2.1 Stack Overflow

Stack Overflow[1] ist eine nahezu allen Softwareentwicklern bekannte Frage und Antwort Plattform. Zu Beginn nur für Softwareentwickler gedacht, wurden bald weitere gleich aufgebaute Seiten wie Server Fault oder Super User und viele weitere[2] vom StackExchange Netzwerk erstellt. Der Erfolg liegt in der einfachen Handhabung, in der hauptsächlich durch die Benutzer der Seite regulierten und bewerteten Antworten und Kommentare sowie der schnellen Beantwortung von Fragen (92% nach durchschnittlich 11 Minuten [25]). Dies wird durch ein einfaches Bewertungssystem umgesetzt, nach dem höher bewertete Antworten auch weiter oben und damit zuerst angezeigt werden. Das führt bei genug Aktivität zu sehr vielen qualitativ hochwertigen Antworten auf häufig gestellte Fragen. Ebenso reizt es Benutzer gute Fragen zu stellen und gute Antworten zu geben, da durch die Bewertung die eigenen Reputationspunkte steigen. Es gilt aber auch ebenso für nicht so häufig gestellte Fragen und die Antworten dafür, da die Benutzer in einem Review-Prozess auch solche Fragen bearbeiten und die Qualität steigern. Mittlerweile kann die Kombination aus Fragen und Antworten sogar für manches Application Programmable Interface (API) als gute Dokumentation inkl. guter Beispiele dienen. Ein Beispiel für eine Übersicht vorhandener Fragen ist in ausschnittsweise in Abbildung 2.2 dargestellt. Je nach Plattform gibt es von der Gemeinschaft der Benutzer selbst auferlegte Regeln für die Ansprüche sowohl an Fragen als auch an Antworten. Es kann dabei passieren, dass falls mehrere Benutzer dafür abstimmen, eine Frage gesperrt wird auch wenn sie relativ populär ist (Abbildung 2.3). Nach [1] gibt es sogar Experten, die häufig gestellte Fragen auf Stack Overflow beantworten um weniger E-Mails dazu zu bekommen.

Eine wichtige Funktion ist weiterhin die Unterstützung der Erstellung von Schlagworten und Verschlagwortung durch die Benutzer. Dabei können Benutzer ab einer gewissen Reputation Fragen zusätzliche Schlagwörter (*Tags*) zuweisen, über die andere Benutzer und Suchmaschinen diese Frage besser finden können. Mit noch mehr Reputationspunkten können Benutzer auch neue, noch nicht existierende Schlagwörter vorschlagen. Hierbei setzen die Seiten wieder auf eine automatische Verwaltung, inklusive der Vereinigung und dem Löschen von Schlagwörtern, durch die Gemeinschaft der Benutzer.

Bewertung von Stack Overflow

Die Frage und Antwort Plattform Stack Overflow bietet offensichtlich eine Grundlage für Kommunikation (KommQ1:4P). Dabei werden Meta-Daten und die Fragen, Antworten und Kommentare formatiert dargestellt (KommQ2:4P). Die schnelle Erfassung des Inhalts ist durch eine große Überschrift, Tags und Syntax-Highlighting gut möglich. Dabei kann es aber sein, dass die richtige Antwort eventuell noch nicht an oberster Position ist da sie erst später dazugekommen ist und noch nicht so viele Bewertungen erhalten hat oder sie noch nicht akzeptiert wurde (KommQ3:3P). Es gibt die Möglichkeit mit Benutzern über Kommentare und

[1] `http://stackoverflow.com` (Besucht am 24.11.2014)
[2] `http://stackexchange.com/sites` (Besucht am 24.11.2014)

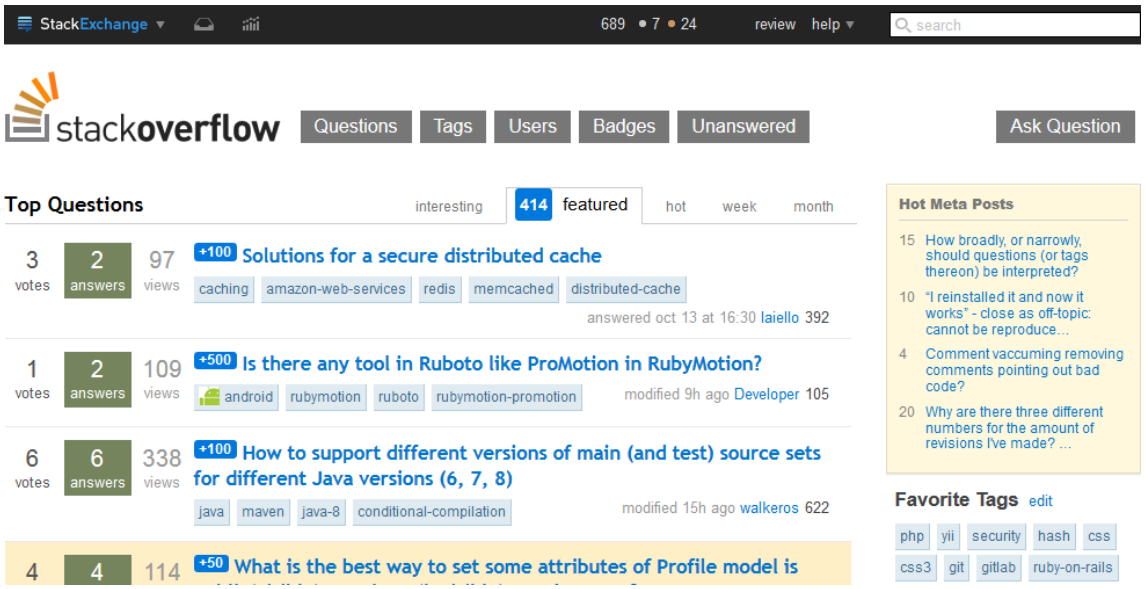
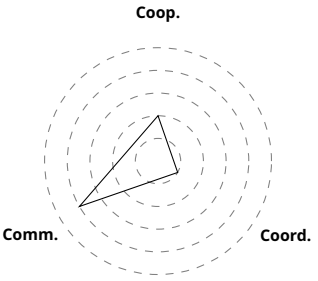


ABBILDUNG 2.2: Übersicht hervorgehobener Fragen auf Stack Overflow

einen allgemeinen Chatraum für bestimmte Themenbereiche aber nicht einzelne Fragen oder durch anklicken zu interagieren (KommQ4:3P, KommQ5:2P).

Die Kooperation wird indirekt durch die Kommunikation begünstigt und die Plattform ist auch für mehrere Benutzer ausgelegt (KoopQ1:4P). Es ist einfach möglich Antworten inkl. Quellcode zu Fragen beizutragen (KoopQ2:2P). Die Effizienz der Zusammenarbeit ist nicht so einfach feststellbar und kommt auf weitere Faktoren an (KoopQ3:2P). Im Weiteren beziehen sich die Fragen auf die Zusammenarbeit und nicht mehr auf die Kommunikation, deshalb gibt es auch keine weiteren Interaktionsmöglichkeiten die eine Zusammenarbeit unterstützen und noch nicht bei den Fragen zur Kommunikation einsortiert wurden (KoopQ4:0P, KoopQ5:0P).

Für die Koordination ist die Plattform ungeeignet, außer man weist Aufgaben im Chat-Gespräch einander zu (KoorQ1:0P, KoorQ2:0P, KoorQ3:0P, KoorQ4:0P, KoorQ5:0P).



Stack Overflow Q&A Plattform

	Q1	Q2	Q3	Q4	Q5	Summe
Kommunikation	4	4	3	3	2	16: **Level 4**
Kooperation	4	2	2	0	0	8: **Level 2**
Koordination	0	0	0	0	0	0: **Level 1**

2.2.2 Coderwall

Coderwall[3] ist eine Webplattform auf der man Tipps (siehe Beispiel in Abbildung 2.4) für andere Softwareentwickler veröffentlichen kann. Im Format eines einzelnen Blog-Posts oder Artikels wird ein bestimmter Sachverhalt erklärt. Ansprechend formatiert inkl. Syntax-Highlighting für

[3] `https://coderwall.com` (Besucht am 24.11.2014)

PHP/MySQL select a database? [closed]

-4

Think this should be an easy one. I understand how to establish a connection, eg:

```
$con = mysql_connect("localhost","peter","abc123");
```

Where `localhost` is the database type, `peter` is the user and `abc123` the password but how do you specify the actual database? For example, I have 3 databases on my server. How do I specify database2?

php | mysql | database | connection

share | edit | flag

asked Sep 18 '12 at 7:25

2,334 ●4 ●47 ●103

closed as not constructive by
Sep 18 '12 at 9:20

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

If this question can be reworded to fit the rules in the help center, please edit the question or leave a comment.

1 And what if you started by reading the PHP manual first? –

1 a simple www search would have given you the answer :(–

thats what I did first off, of course, and it didnt give me a satisfactory one, hence the question – Sep 18 '12 at 7:31

3 google.de/search?q=php+mysql+select+database was not satisfactory, okey ... – 7:32

The PHP manual is a complete and organized reference for all PHP functions

add a comment | show 3 more comments

8 Answers

active oldest votes

2

As there are so many answers that suggest `mysql_select_db('database1');` and its getting more upvotes I'll try to create a complete answer:

Don't use `mysql_select_db('database1');` **!** (You should not use it in new development projects because it is in deprecation process)

Use one of the following ways:

Either you use PDO like this:

```
<?php
    try {
        $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
        foreach ($dbh->query('SELECT * from FOO') as $row) {
            print_r($row);
        }
        $dbh = null;
    } catch (PDOException $e) {
        print "Error!: " . $e->getMessage() . "<br/>";
        die();
    }
?>
```

Or you could use mysqli:

```
<?php
   $mysqli = new mysqli("127.0.0.1", "user", "password", "database", 3306);
    if ($mysqli->connect_errno) {
        echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $my
    }
?>
```

share | edit | delete | flag

answered Sep 18 '12 at 8:35

689 ●7 ●24

add a comment

asked **2 years ago**
viewed **3954 times**
active **2 years ago**

Hot Meta Posts

15 How broadly, or narrowly, should questions (or tags thereon) be interpreted?
10 "I reinstalled it and now it works" - close as off-topic: cannot be reproduce...
4 Comment vaccuming removing comments pointing out bad code?
20 Why are there three different numbers for the amount of revisions I've made? ...

92 People Chatting

JavaScript

Lounge<C++>

Related

2794 How can I prevent SQL-injection in PHP?
2 Use PHP to authenticate information in a MySQL database from an external server
0 php mysql query connection error cannot select database
0 Creating a secure mySQL database connection with WebMatrix in php
2 MySQL connection works in PHP but not ASP

ABBILDUNG 2.3: Beispiel einer relativ häufig angesehenen Frage auf Stack Overflow, welche die Kriterien nicht erfüllt, aber trotzdem zufriedenstellend beantwortet wurde.

Quellcode können am unteren Ende Kommentare von Besuchern dazu abgegeben werden um den Tipp zu diskutieren oder auf verwandte Beiträge hinzuweisen. Der Beitrag/Tipp kann mit Tags ähnlich wie bei Stack Overflow versehen werden. Insgesamt kann eine große Ähnlichkeit zu Stack Overflow festgestellt werden. Was fehlt ist die Fragestellung und mehrere mögliche Antworten. Man kann Tipps für einzelne Tags oder in zu *Netzwerken* (Bezeichnung von Coderwall) zusammengefasste Technologien und Benutzer der Technologien ansehen oder abonnieren um über neue Tipps informiert zu werden. Benutzerprofile werden ausgehend von den Inhalten der Tipps generiert und mit den verwendeten Technologien markiert. Dafür erhalten die Benutzer wie auch auf Stack Overflow Auszeichnungen nach festgelegten Kriterien (siehe Beispiel eines Profils in Abbildung 2.5).

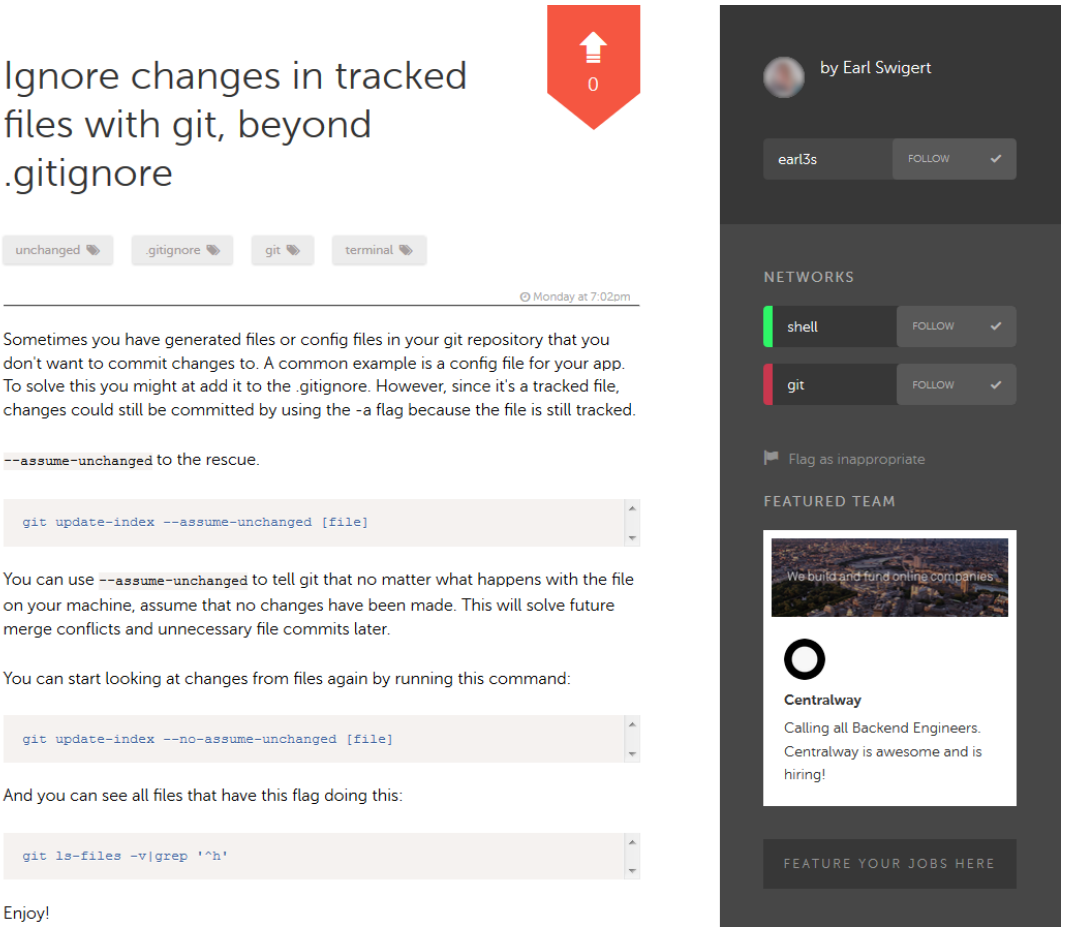


ABBILDUNG 2.4: Coderwall Tipp eines Benutzers für das Ignorieren von Änderungen an einer Datei unabhängig von der .gitignore-Datei. Quelle: Earl Swigert auf coderwall.com

Bewertung von Coderwall

Coderwall ist grundsätzlich gesehen ein indirektes Kommunikationsmedium (KommQ1:4P). Neben der zusätzlichen formatierten Darstellung von Meta-Daten (KommQ2:4P) und der schnellen Erfassbarkeit (KommQ3:4P) wird eine weitere Interaktion aber nur durch Kommentare ermöglicht (KommQ4:1P, KommQ5:0P). Bezüglich unserer beiden anderen Kategorien des 3C-Modells, Kooperation und Koordination werden keine Punkte erreicht.

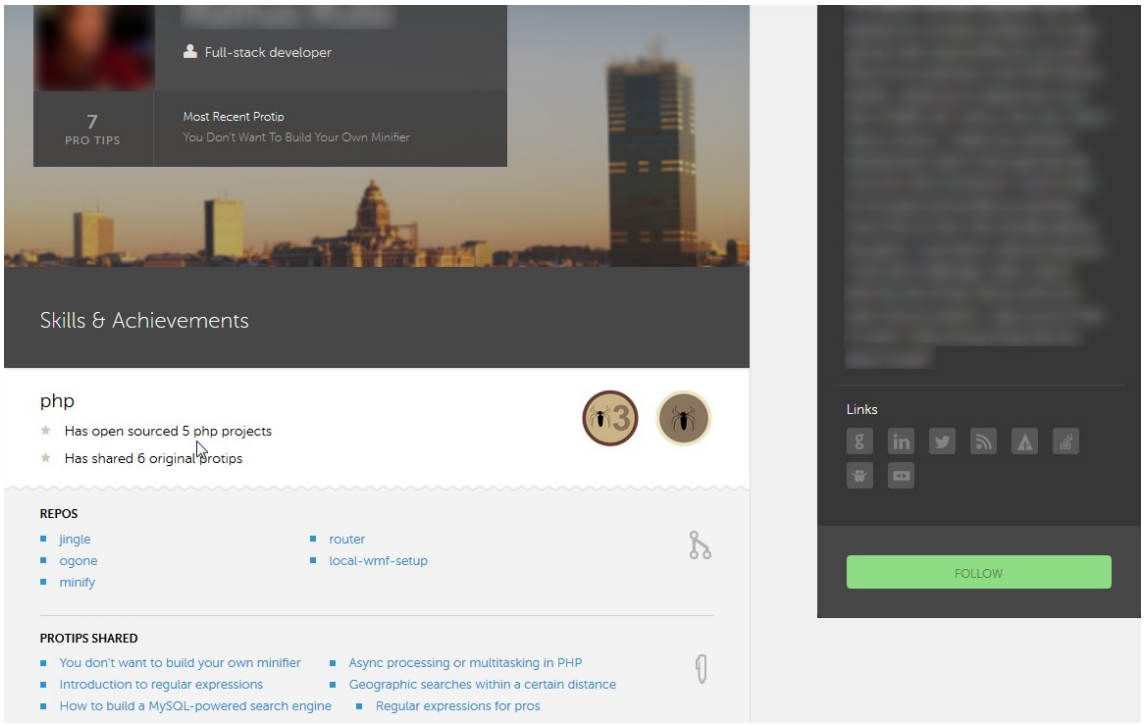
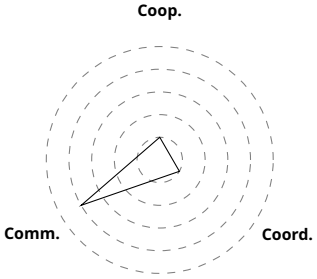


ABBILDUNG 2.5: Coderwall Profil eines Benutzers der 2 Auszeichnungen für seine PHP Projekte erhalten hat. Quelle: coderwall.com



Coderwall Protips (Blog)

	Q1	Q2	Q3	Q4	Q5	Summe
Kommunikation	4	4	4	1	0	13: **Level 4**
Kooperation	0	0	0	0	0	0: **Level 1**
Koordination	0	0	0	0	0	0: **Level 1**

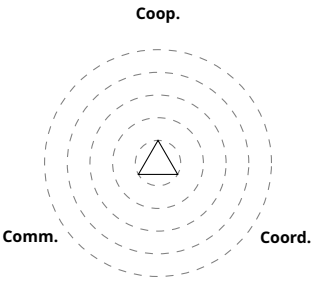
2.2.3 Masterbranch

Masterbranch[4] fasst Informationen aus Quellen wie GitHub, Stack Overflow, Sourceforge, CodePlex, Bitbucket, Google Code, java.net und vielen weiteren zusammen. Aus den gesammelten Informationen werden verschiedene Kennzahlen abgeleitet. Dazu zählt zum Beispiel die Anzahl der Projekte pro Programmiersprache, oder wie viel Prozent der Commits eines Projekts von diesem Entwickler gemacht wurden. Des Weiteren gibt es eine automatisch generierte Liste von Programmiersprachen und Technologien die der Entwickler hauptsächlich verwendet. Über die Zusammenfassung hinausgehende Interaktionsmöglichkeiten gibt es nicht. Masterbranch dient eher der Selbstdarstellung des eigenen Entwicklerprofils als der Kommunikation, Kooperation oder Koordination.

[4] `https://masterbranch.com` (Besucht am 24.11.2014)

Bewertung von Masterbranch

Aus dem gerade genannten Grund werden auch kaum Punkte in den von uns verwendeten Fragen erreicht (KommQ1:0P, KommQ2:0P, KommQ3:0P, KommQ4:0P, KommQ5:0P; KoopQ1:0P, KoopQ2:0P, KoopQ3:0P, KoopQ4:0P, KoopQ5:0P; KoorQ1:0P, KoorQ2:0P, KoorQ3:0P, KoorQ4:0P, KoorQ5:0P).



Masterbranch

Automatische Profilerstellung durch Aggregation vorhandener Informationen

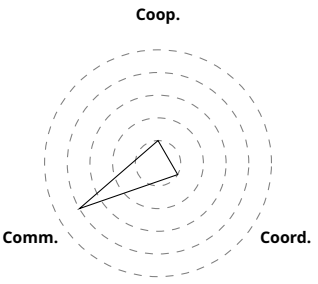
	Q1	Q2	Q3	Q4	Q5	Summe
Kommunikation	0	0	0	0	0	0: **Level 1**
Kooperation	0	0	0	0	0	0: **Level 1**
Koordination	0	0	0	0	0	0: **Level 1**

2.2.4 CodeProject

CodeProject[5] bietet online die Möglichkeit, einer großen Gemeinde an Software-Entwicklern beizutreten. Die Online-Plattform ist eine Kombination aus einem Forum zur Diskussion, einem Blog auf dem jeder Benutzer eigene Artikel veröffentlichen kann (ähnlich wie auf Coderwall) sowie einer Frage- und Antwort-Plattform wie Stack Overflow.

Bewertung von CodeProject

Durch die Kombination der Ansätze mehrerer bereits vorhandener Plattformen wird die Kommunikation zentralisiert. Weitergehende Unterstützung für die Zusammenarbeit oder die Koordination bietet diese Plattform aber trotzdem nicht. Dadurch fällt auch CodeProject nur in den Bereich der Kommunikation (KommQ1:4P, KommQ2:0P, KommQ3:4P). Die Verlinkung der einzelnen Plattformbestandteile ist aber erhöht und gewünscht. So gibt es ein erhöhtes Maß an Interaktionsmöglichkeiten innerhalb der Plattform (KommQ4:3P). Eine direkte Kommunikation wird nicht unterstützt (KommQ5:0P).



CodeProject

Kombination aus Blog, Forum und Q&A-Plattform

	Q1	Q2	Q3	Q4	Q5	Summe
Kommunikation	4	4	4	3	0	15: **Level 4**
Kooperation	0	0	0	0	0	0: **Level 1**
Koordination	0	0	0	0	0	0: **Level 1**

[5] `https://codeproject.com` (Besucht am 24.11.2014)

2.2.5 Cloud9

Die Online-Plattform Cloud 9[6], basierend auf der freien Cloud9 Integrierte Entwicklungsumgebung (IDE)[7] welche selbst wiederum auf Bespin/Mozilla Skywriter basiert, enthält einen Quellcode-Editor mit Syntaxhervorhebung, Code-Vervollständigung, gleichzeitigem Editieren (Echtzeit) für mehrere Benutzer, siehe Abbildung 2.6, und Chat-Funktion. Des Weiteren kann man verschiedene Arbeitsbereiche erstellen, die bei Bedarf vorkonfigurierte Softwarepakete wie Webserver oder andere Programmierumgebungen bereitstellen. Dies reduziert den Konfigurationsbedarf, sofern alles zur Genüge vorkonfiguriert ist, enorm. Das Ziel ist, möglichst effizient und schnell programmieren zu können. Es ist auch möglich einen Arbeitsbereich zu erstellen der per Secure Shell (SSH) auf einen eigenen Server zugreift und die Umgebung dort nutzt. Weiterhin wird das direkte Klonen von git-Projekten unterstützt sowie vereinfacht für Bitbucket- und GitHub-Projekte, falls man sein Konto mit dem jeweiligen Dienst verbunden hat.

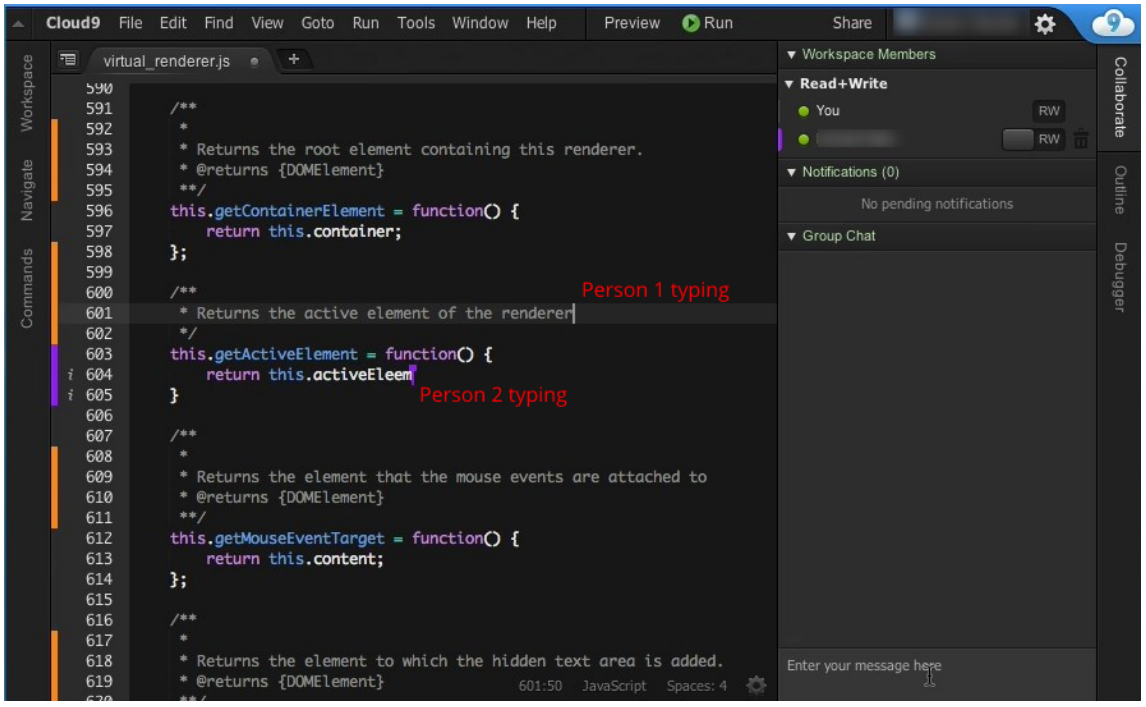


ABBILDUNG 2.6: CLOUD9 IDE mit gleichzeitigem Editieren durch zwei Benutzer. (Hervorhebung in Rot für Markierung der Teilnehmer)

Neben dem Quellcode-Editor gibt es verschiedene zusätzliche Bereiche wie einen Debugger und Terminal für Kommandozeilen-Operationen wie zum Beispiel `git commit` und `git push`. Die Bereiche des Quellcode-Editors können beliebig aufgeteilt (Split View) werden.

Ein ähnliches aber nicht mehr aktives Projekt ist der Space Editor[8]. Weitere Funktionen der Cloud9 Online-Plattform werden in der folgenden Bewertung genannt:

[6] `https://c9.io/` (Besucht am 24.11.2014)
[7] `https://github.com/ajaxorg/cloud9/` (Besucht am 24.11.2014)
[8] `https://github.com/chaoscollective/Space_Editor` (Besucht am 24.11.2014)

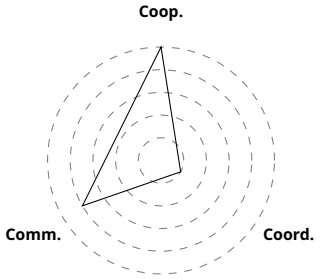
Bewertung der Cloud9 Online-Plattform

Die Bewertung basiert auf der kompletten Plattform und ihren Möglichkeiten und nicht nur auf einzelnen Komponenten wie der Cloud9 IDE oder deren Komponenten.

Cloud9 unterstützt eine **direkte Kommunikation** durch eine Chat-Funktion für alle beteiligten (anwesenden) Entwickler (KommQ1:4P, KommQ5:4P). Im Chat ist es aber nicht üblich längere Texte zu schreiben bzw. Code zu verschicken. Dies könnte sowieso direkt im Echtzeit-Quellcode-Editor gemacht werden. Meta-Daten werden formatiert und ordentlich dargestellt inklusive Umwandlung des Zeitstempels in eine Angabe wie *vor 5 Minuten* (KommQ2:4P, KommQ3:1P). Zusätzlich existieren Interaktionsmöglichkeiten wie die automatische Umwandlung von URLs in anklickbare Hyperlinks (KommQ4:2P).

Die offensichtlich **hauptsächlich auf Kooperation ausgelegte Entwicklungsumgebung** bietet gleichzeitiges Editieren von den Quellcode-Dateien(KoopQ1:4P). Benutzer können direkt sehen, was der andere tippt und darauf reagieren und es kommentieren oder anpassen. Dies ermöglicht zum Beispiel effiziente Paarprogrammierung. Mit der direkten Integration des Chats ist auch für weitere Interaktion gesorgt (KoopQ2:4P, KoopQ3:4P, KoopQ4:4P, KoopQ5:4P).

Dafür sieht es im Bereich der Koordination eher schlecht aus. Es können zwar andere Benutzer eingeladen werden und diese mit Lese- und/oder Schreibrechten ausgestattet werden, aber das war es auch schon(KoorQ1:1P, KoorQ2:1P). So können Benutzer nur indirekt koordiniert werden (KoorQ3:0P, KoorQ4:0P, KoorQ4:0P).



Cloud9 Online-Plattform

Online-IDE mit Live-Editing und Chat

	Q1	Q2	Q3	Q4	Q5	Summe
Kommunikation	4	4	1	2	4	15: **Level 4**
Kooperation	4	4	4	4	4	20: **Level 5**
Koordination	1	1	0	0	0	2: **Level 1**

2.2.6 GitHub

2.2.6.1 Einführung

GitHub[9] ist eine webbasierte Plattform für mit git versionierte Softwareprojekte. Es ist kostenlos und jeder kann sich ein Konto erstellen. Für private Repositories, auf die dann nur ausgewählte Benutzer Zugriff haben, kann man bezahlen oder eine eigene Enterprise Version von GitHub in seinem Unternehmensnetzwerk auch kostenpflichtig betreiben. GitHub bietet eine Reihe von Kommunikationskanälen für die Interaktion mit dem Code und der Dokumentation dazu an. Diese beschreibe ich in den nächsten Unterabschnitten.

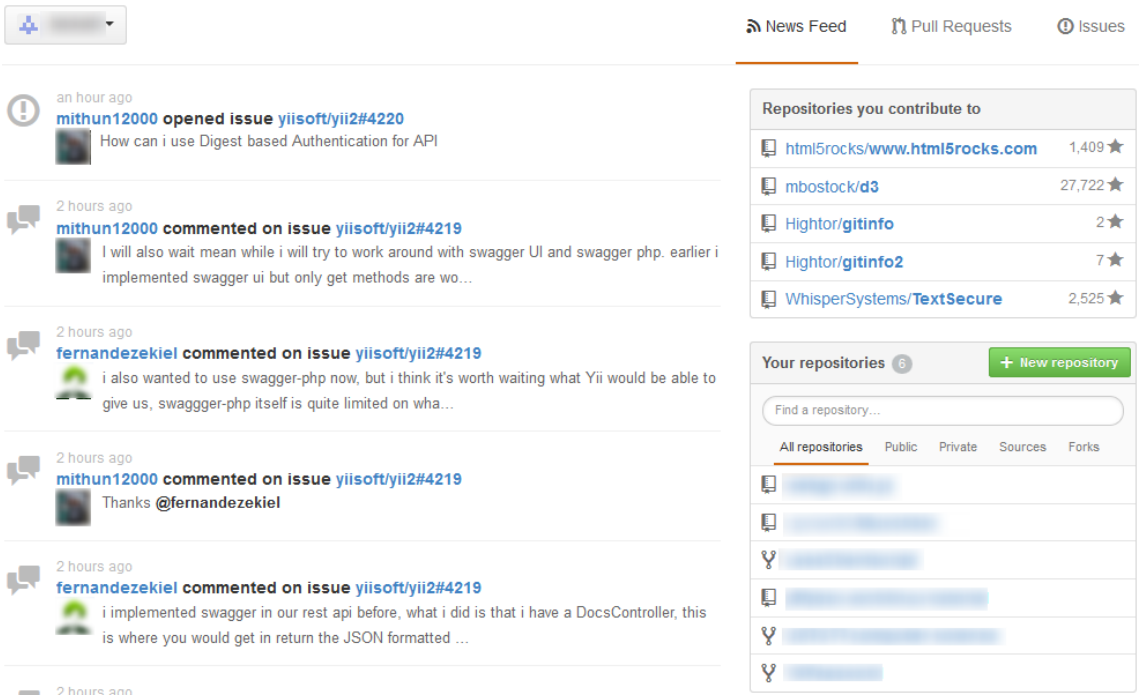


ABBILDUNG 2.7: GitHub Dashboard

2.2.6.2 Dashboard

Die in Abbildung 2.7 dargestellte Übersichtsseite zeigt einem einen personalisierten Überblick über aktuelle Geschehnisse wie Commits, Issues oder Kommentare zu Repositories welche man beobachtet. Diese sich automatisch aktualisierende Liste ist vergleichbar mit dem Newsstream von Facebook.

2.2.6.3 Profil

Das für Interaktion erforderliche Benutzerkonto bietet eine Profilansicht öffentlicher Informationen an. In Abbildung 2.8 sieht man eine Übersicht über die Aktivitäten eines einzelnen Benutzers inklusive beliebter Repositories und Repositories zu denen der Benutzer etwas beigetragen hat. Des Weiteren gibt es eine farb-codierte Übersicht über allgemeine Beiträge im Zeitverlauf und der letzten Beiträge je nach Aktivität. Zusätzlich kann man hier sehen wie viele andere Benutzer diesem Benutzer folgen, sprich über Aktivitäten informiert werden und welche Repositories der Benutzer beobachtet (*starred*). Das Profil ist vergleichbar mit der Chronik von Facebook.

2.2.6.4 Repository/Projekt Übersicht

Projekte werden in, mit git versionierten, Repositories verwaltet. In Abbildung 2.9 sieht man die Übersichtsseite eines Repositories. Für wichtige soziale Aktionen sind drei Knöpfe, siehe

[9] `https://github.com/` (Besucht am 24.11.2014)

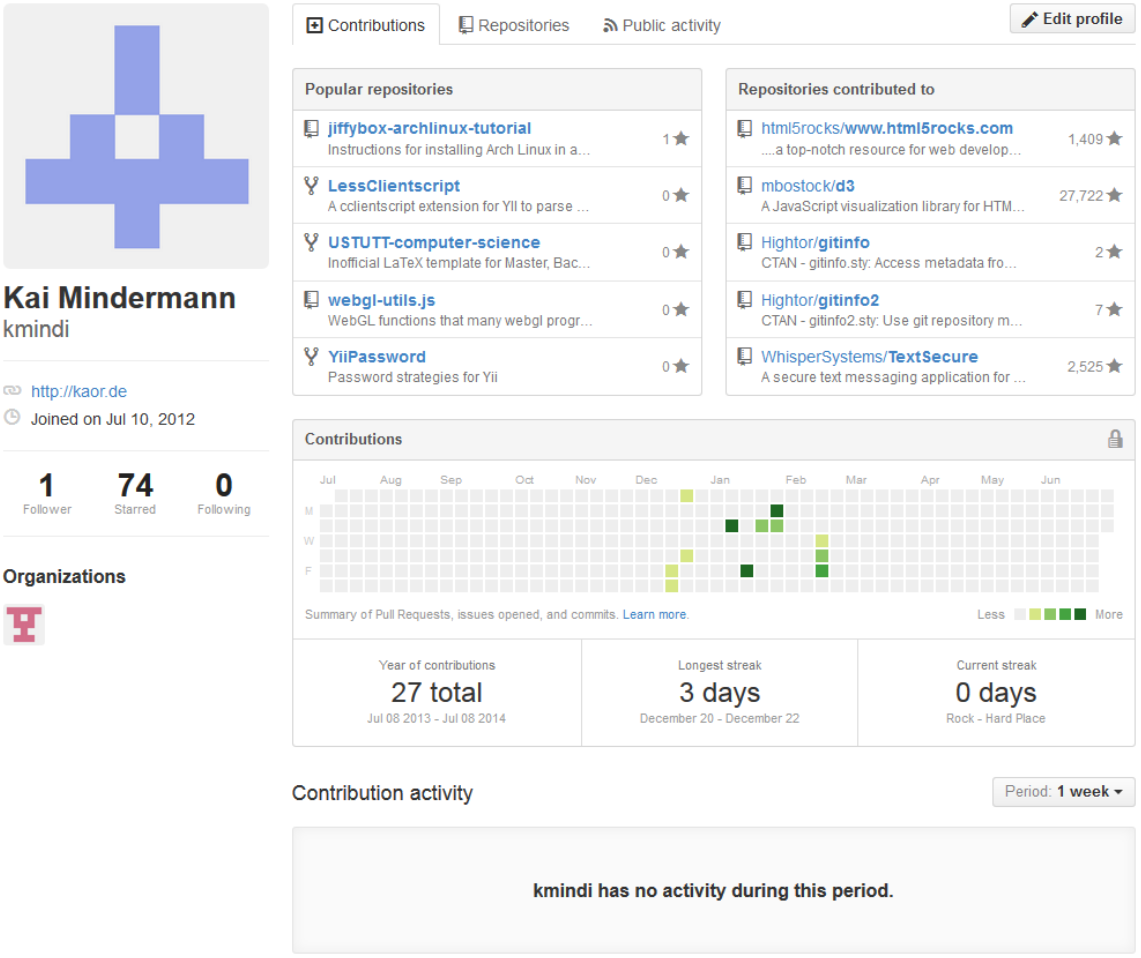


ABBILDUNG 2.8: GitHub Profilansicht mit Übersicht beliebter Repositories und der Übersicht der letzten Beiträge, aufgetragen über die Zeit.

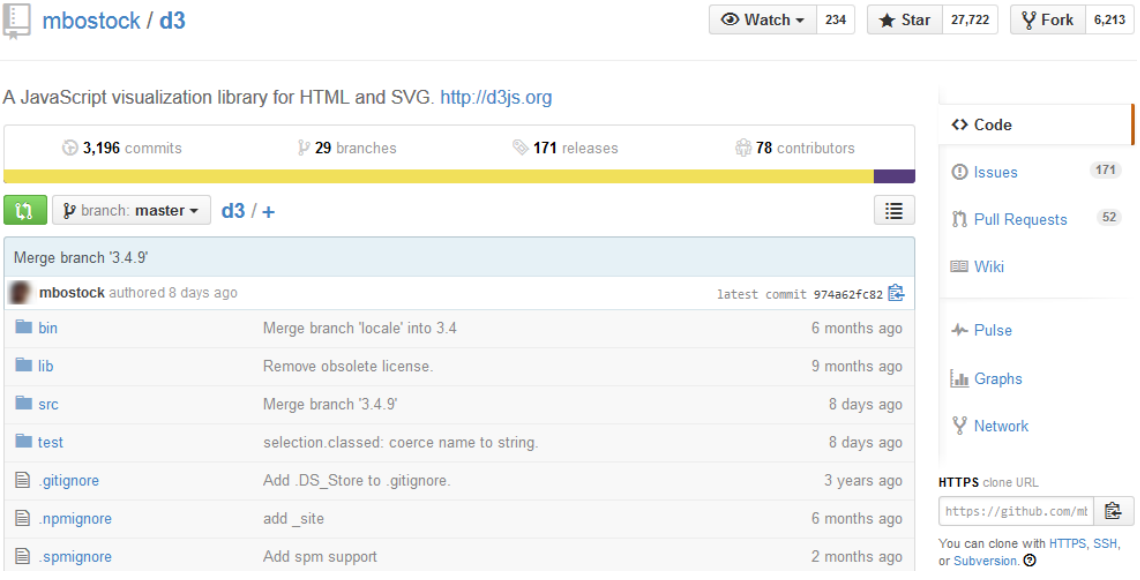


ABBILDUNG 2.9: GitHub Repository Übersicht

Abbildung 2.10, rechts oben verfügbar. Dazu gehört die Aktion *Watch* mit der man über alle Beiträge zu dem Repository informiert wird. Im Gegensatz dazu ist die Aktion *Star* nur dafür gedacht, dieses Repository auf eine Liste für eine spätere Referenz zu setzen oder einfach nur Dankbarkeit dafür auszudrücken wobei aber nicht alle Beiträge abonniert werden. Die *Fork*-Aktion wird benötigt wenn man Änderungen an den Dateien und somit auch dem Quellcode eines Repositories vornehmen möchte.

Die weiteren Elemente geben die Anzahl der Commits, Branches, Releases und Mitwirkender Benutzer an. Nach einem klick auf die Zahlen kommt man zu einer Auflistung der Inhalte des entsprechenden Bereichs. Danach folgt eine interaktive Ordner- und Dateiansicht des Repositorys inklusive letzter Commit-Nachricht und wie lange die Änderung her ist. Weiter unten wird der Text aus einer Readme-Datei schön formatiert dargestellt. Hier bieten die meisten Projekte erste Beschreibungen und allgemeine Informationen über ihr Projekt an.

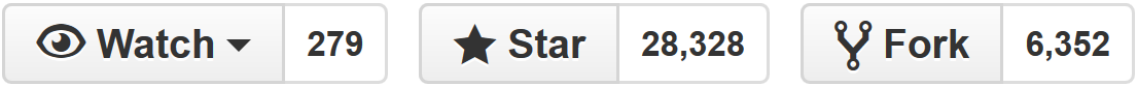


ABBILDUNG 2.10: GitHub Repository soziale Aktionen: *Watch*, *Star* und *Fork*

2.2.6.5 Repository Bug-Tracking System

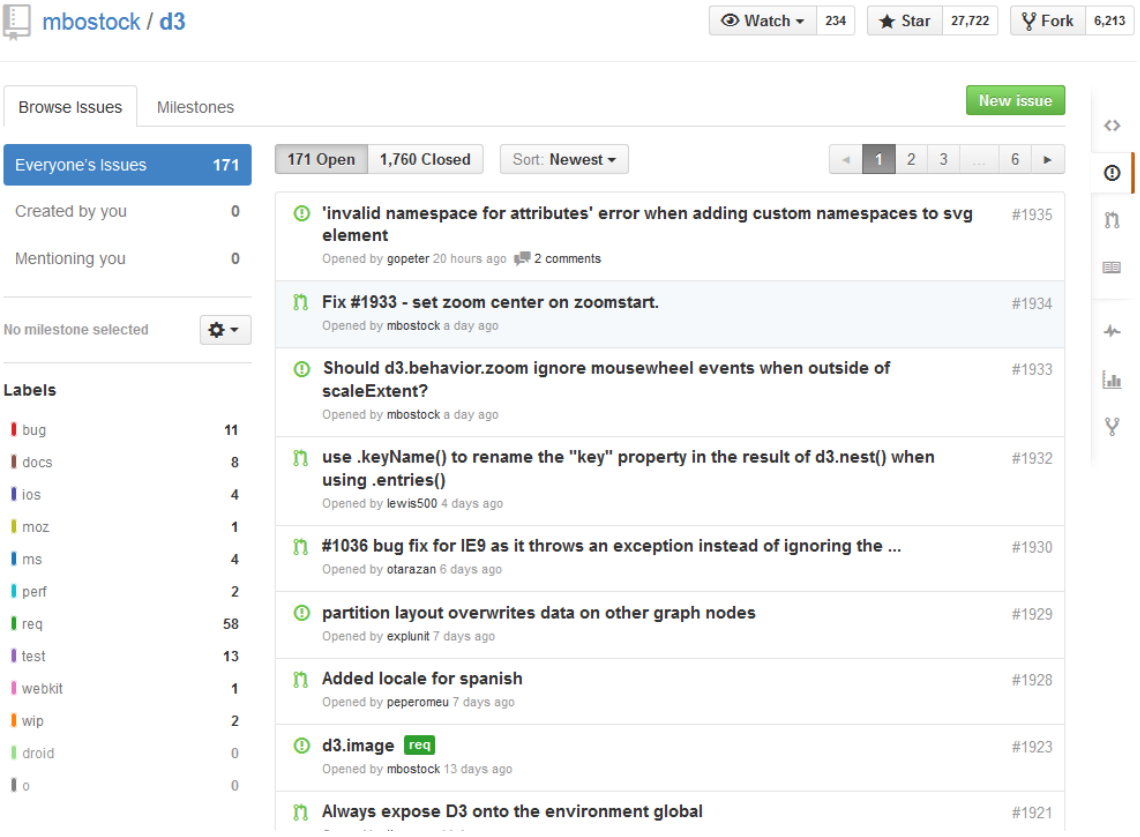


ABBILDUNG 2.11: GitHub Repository Issues

Für die Meldung von Problemen oder die Anfrage von neuen Features gibt es ein integriertes Bug-Tracking System (Issues). In Abbildung 2.11 sieht man eine Liste der noch geöffneten

Issues. Ein Issue erhält eine Nummer mit der er überall referenziert werden kann (und auch automatisch verlinkt wird), einen aussagekräftigen Titel sowie eine ausführliche Beschreibung. Wahlweise können Issues noch Label zugeordnet werden mit denen diese Kategorisiert werden können. Des Weiteren kann jeder Benutzer von GitHub Kommentare zu einem Issue abgeben oder Vorschläge für Code der das Problem löst, siehe Abbildung 2.12. Vorgeschlagene Codeänderungen, welche in Form von Diffs dargestellt werden, können auch zeilenweise kommentiert werden, siehe Abbildung 2.13.

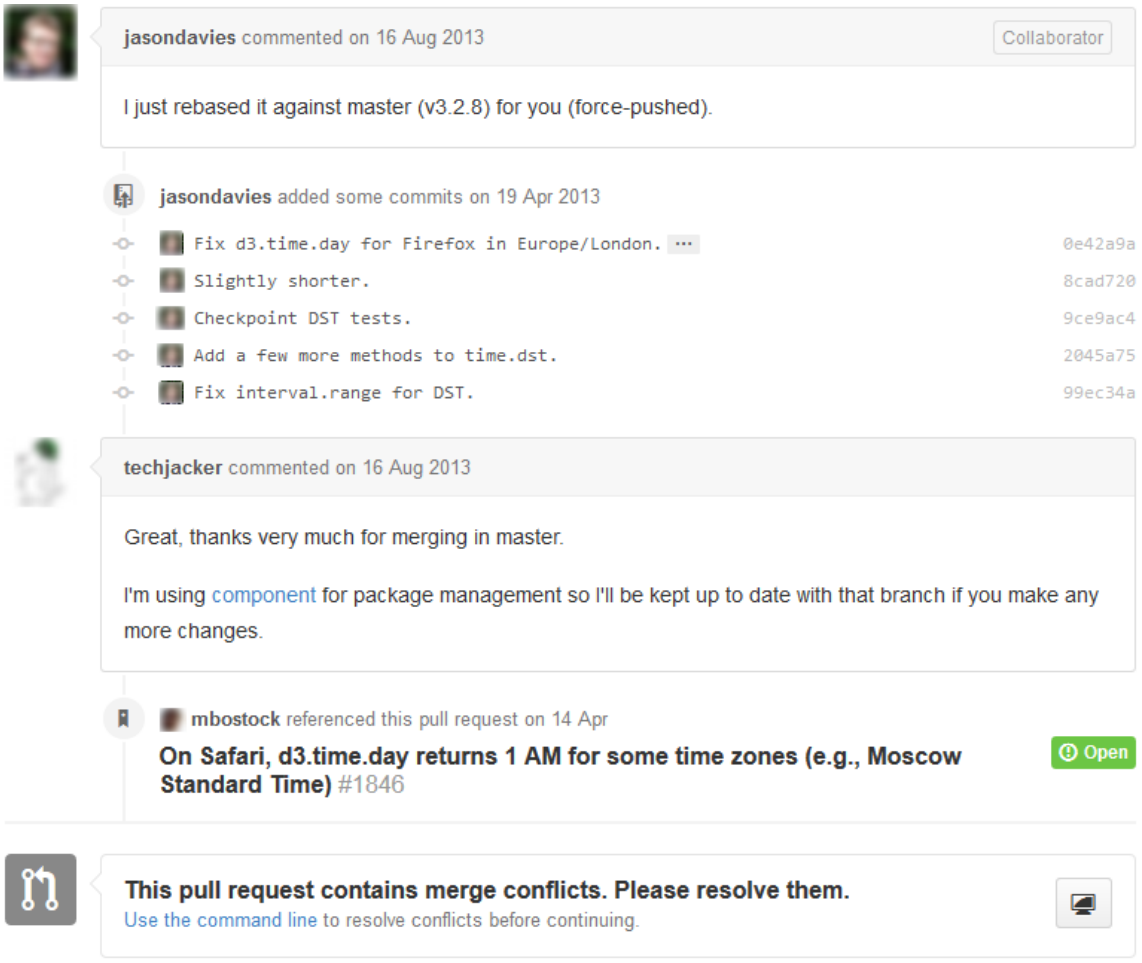


ABBILDUNG 2.12: GitHub Issue Kommentare und weitere Features

2.2.6.6 Labels/Schlagworte

Von Projektmitarbeitern können beliebige Begriffe zur Klassifizierung der Bugs verwendet werden, diese funktionieren wie Schlagworte, sind aber eben nur für ein Projekt und nicht auf der gesamten Plattform gültig.

2.2.6.7 Kommentarfunktion für Issues

Die gerade eben allgemein beschriebene Kommentarfunktion für Issues ist der wichtigste indirekte Kommunikationskanal auf GitHub. Ein wichtiges Merkmal der Kommentare ist die

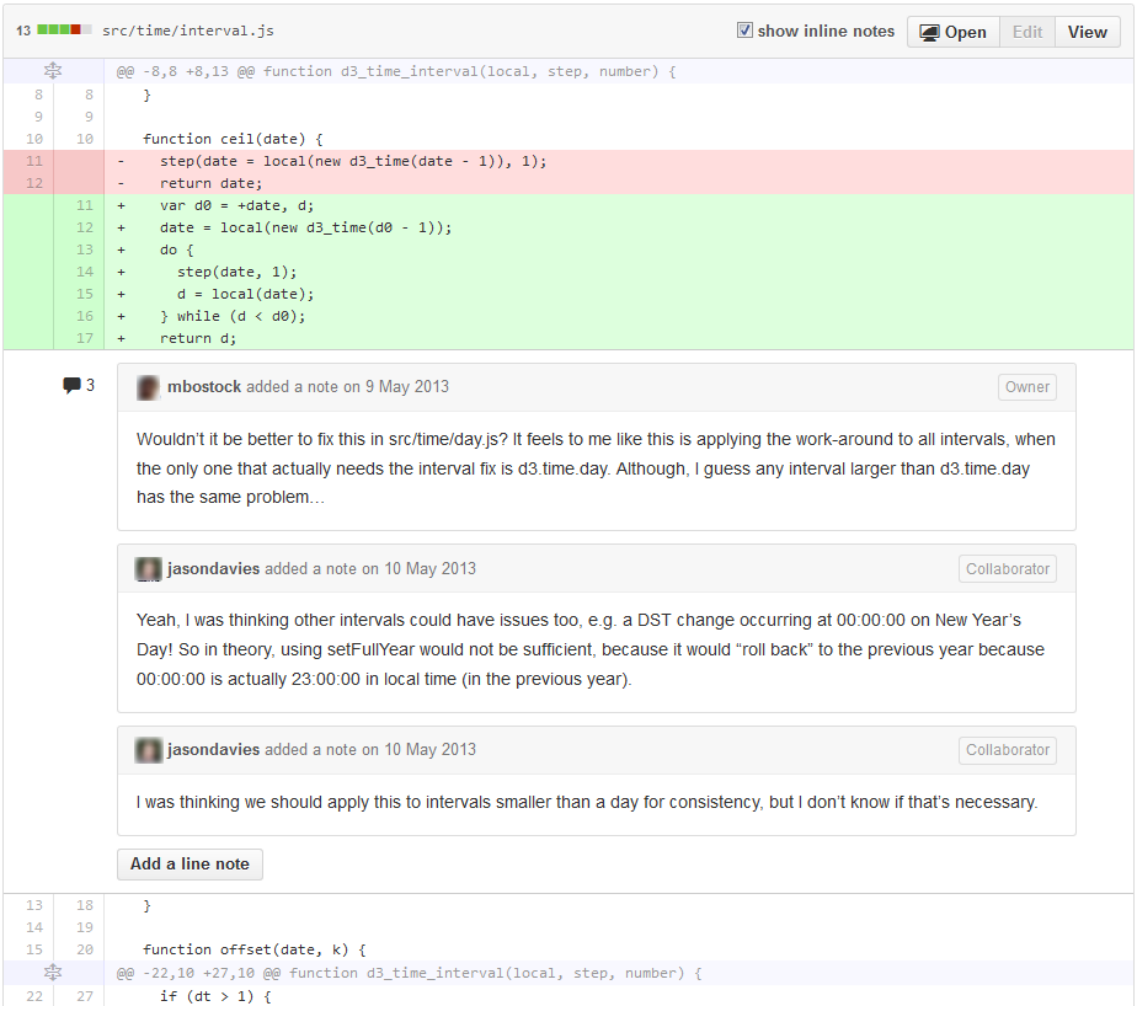


ABBILDUNG 2.13: GitHub Diff Kommentare

Integration von **Markdown** [26] [27]. Mit Markdown ist es möglich Text mit intuitiven Mitteln gestalterisch hervorzuheben. Markdown ist ohne, dass der Text umgewandelt wird noch lesbar. Dies ermöglicht es ohne Vorkenntnisse oder spezielle Programme den Text lesen und bearbeiten zu können. Markdown muss von neuen Benutzern nicht gekannt werden um Kommentare abzugeben, aber bietet erfahrenen Benutzern die Möglichkeit Kommentare oder Issues selbst übersichtlich zu gestalten. Weiterhin ist es möglich in Kommentaren oder Issues selbst mit **@username** einen anderen Benutzer zu nennen, zu referenzieren (mittels automatischen Hyperlinks) und gleichzeitig auch zu informieren bzw. zur Diskussion einzuladen (der Benutzer erhält automatisch eine Nachricht, dass er in einem Kommentar genannt wurde). Darüber hinaus ist es nicht nur möglich Benutzer zu referenzieren, sondern auch Issues bzw. Pull Requests mit **#999** über ihre im Repository eindeutige Nummer. Der referenzierte Issue/Pull Request wird damit verlinkt und auf der entsprechenden Seite wird der aktuelle Issue in dem die Referenz geschrieben wurde, dort auch referenziert.

Eine sehr ähnliche Umsetzung der Kommentarfunktion an sich, kann bei weiteren populären Seiten wie Stack Overflow und Bitbucket[10] betrachtet/verwendet werden.

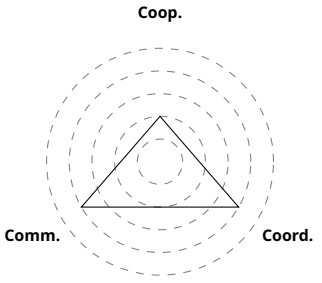
[10] `http://bitbucket.org` (Besucht am 24.11.2014)

2.2.6.8 Bewertung des Bug-Trackers

Da die Kommentarfunktion immer in Kombination innerhalb eines Bug-Reports oder Pull-Requests, also des gesamten Bug-Trackers, verwendet wird, werden die dadurch erweiterten Funktionen wie Label, Milestones, usw. in die Bewertung miteinbezogen.

Es ist offensichtlich, dass Kommunikation möglich ist (KommQ1:4P). Weiterhin ist die Darstellung sehr ansprechend und zum schnellen erfassen optimiert (KommQ2:4P, KommQ3:4P). Es gibt Interaktionsmöglichkeiten durch das Anklicken von **@username** und referenzieren von anderen Issues durch **#999** (KommQ4:3P). Direkte Kommunikation ist nicht möglich (KommQ5:0P). Grundsätzlich ist der Bug-Tracker und die Kommentarfunktion für mehrere Benutzer ausgelegt (KoopQ1:3P) und es ist sehr einfach neue Kommentare zu veröffentlichen (KoopQ2:4P). Eine weitere direkte und effiziente Kooperation ist darüber aber nicht möglich. Die Kooperation wird nur indirekt durch die Diskussion des Quellcodes gefördert (KoopQ3:1P, KoopQ4:0P, KoopQ5:0P). Die hauptsächliche Kooperation findet aber auch im Rahmen der Bearbeitung des Quellcodes statt. Der Bug-Tracker unterstützt durch die Kommunikation natürlich auch eine Koordination (KoorQ1:2P, KoorQ2:1P). Des Weiteren ist es aber auch möglich Benutzer zu nennen und diese werden darüber informiert (KoorQ3:4P, KoorQ4:4P). Zusätzlich ist es möglich den Issues Label zuzuordnen und Milestones festzulegen. So ist sogar eine leichte direkte Koordination möglich (KoorQ5:2P).

Der GitHub Bug-Tracker dient also hauptsächlich der Kommunikation und Koordination.



GitHub Bug-Tracker

Mit Markdown-Hervorhebung, Verlinkung und Benachrichtigungen

	Q1	Q2	Q3	Q4	Q5	Summe
Kommunikation	4	4	4	3	0	15: **Level 4**
Kooperation	3	4	1	0	0	8: **Level 2**
Koordination	2	1	4	4	2	13: **Level 4**

2.2.7 Visual Studio 2013 (Ultimate) CodeLens

In der Ultimate Version von Microsofts Visual Studio 2013 gibt es im Quelltext-Editor die sogenannte CodeLens (Code Information Indicator). Diese hat verschiedene Indikatoren die über Funktionen angezeigt werden können. Zu den Indikatoren zählen Referenzen, Änderungen, Autoren, Code Reviews, *Work Items*, Bugs & *Incoming Changes*. Wenn das Projekt mit einem Team Foundation Server verwaltet wird, sind alle Indikatoren verfügbar, ansonsten nur der Indikator für Referenzen auf diese Funktion. Seit dem Update 3 für VS 2013 Ultimate werden auch die Indikatoren Autoren, Änderungen und *Work Items* in nur mit git verwalteten Projekten angezeigt, siehe Abbildung 2.14. *Work Items* werden zwar angezeigt und beziehen sich vermutlich auf per #nummer bezeichnete Issues, aber diese Issues werden ohne einen Team Foundation Server nicht angezeigt. Die im Tooltip angezeigten Autoren können ohne Team Foundation Server nur per E-Mail kontaktiert werden. Mit Team Foundation Server stehen weitere Optionen zur Verfügung.

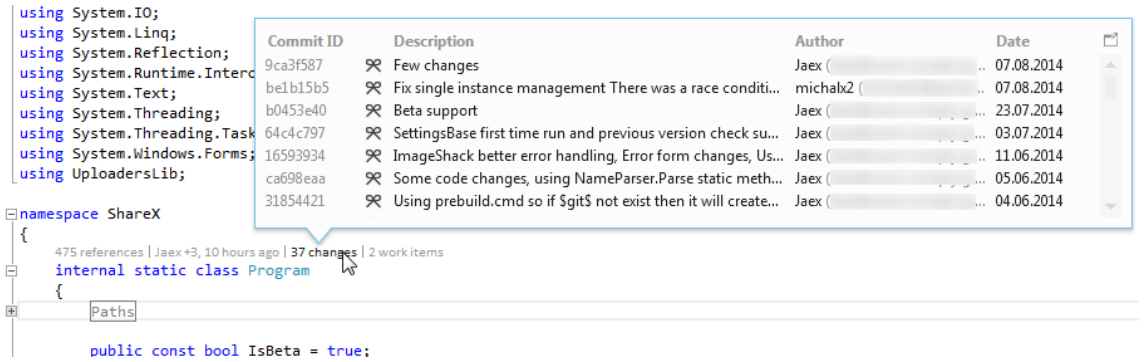
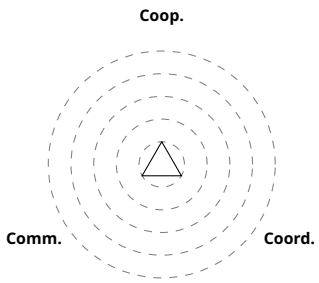


ABBILDUNG 2.14: Visual Studio 2013 (Ultimate) CodeLens für mit git versionierte Projekte

Bewertung der Visual Studio CodeLens

Hier wird nur die Funktionalität der CodeLens an sich bewertet. Weitere Funktionen von Visual Studio werden außen vor gelassen.

Zur Kommunikation trägt die CodeLens nur in der Form bei, dass Beteiligte evtl. indirekt kontaktiert werden können (KommQ1:0P, KommQ2:0P, KommQ3:0P, KommQ4:2P, KommQ5:0P). Ähnlich spärlich sieht es bei der Kooperation aus. Eine Unterstützung für mehrere Benutzer ist nur im Ansatz vorhanden (KoopQ1:1P, KoopQ2:0P, KoopQ3:0P). Die indirekte Kontaktmöglichkeit führt hier bei Frage KoopQ4 zu 2 Punkten (KoopQ5:0P). Auch die Koordinationsmöglichkeiten sind praktisch nicht vorhanden. Man könnte aus den bisherigen Bearbeitern und vorhandenen *Work Items* ableiten, wer zuständig ist (KoorQ1:1P, KoorQ2:0P, KoorQ3:0P, KoorQ4:1P, KoorQ5:0P). Zusammenfassend bietet die CodeLens nur Ansatzpunkte für die 3 Kategorien und kann höchstens als schnelle Referenz dafür dienen, wer an der entsprechenden Funktion im Quelltext mitgearbeitet hat und welche Issues damit verbunden sind.



Visual Studio 2013 (Ultimate) CodeLens

	Q1	Q2	Q3	Q4	Q5	Summe
Kommunikation	0	0	0	2	0	2: **Level 1**
Kooperation	1	0	0	2	0	3: **Level 1**
Koordination	1	0	0	1	0	2: **Level 1**

2.2.8 Visual Studio Anywhere

Visual Studio Anywhere (VSAnywhere)[11] ist eine Echtzeit-Kollaborationsplattform als Erweiterung für Microsoft Visual Studio mit direktem Teilen von geöffneten Quellcode Tabs u. ä. mit Bearbeitungsfunktion und Chatfunktionalität. Dabei ergänzt VSAnywhere die Visual Studio IDE um ähnliche Funktionen wie die bereits beschriebene Cloud9 Plattform. VSAnywhere bietet eine einfache Oberfläche für die Verwaltung von derzeitigen Online-Kontakten auf Basis des Team Foundation Servers für Visual Studio Projekte an. Mit den Benutzern kann dann zusammen über Live-Editing gleichzeitig am Code gearbeitet werden. Hierbei werden zusätzlich auch

[11] `https://vsanywhere.com/web/` (Besucht am 24.11.2014)

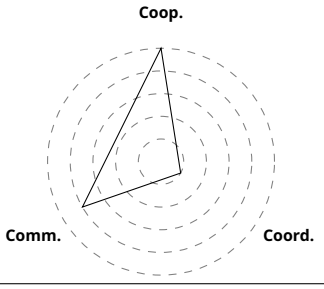
die Design-Elemente wie WinForms oder Extensible Application Markup Language (XAML) unterstützt. Neben dem gleichzeitigen Bearbeiten kann mit den anderen Benutzern auch gechattet werden. Es ist weiterhin möglich Benutzern die kein Visual Studio haben einen Link zu schicken um per Browser zuschauen zu können. Dabei können die Browser-Nutzer aber nicht weiter mit dem Code interagieren.

Bewertung von Visual Studio Anywhere

Hauptsächlich auf Paarprogrammierung ausgelegt bietet die Erweiterung von Visual Studio hauptsächlich kooperative Unterstützung. Durch die Erweiterung wird es ermöglicht, dass mehrere Benutzer am selben Code arbeiten können (KoopQ1:4P, KoopQ2:4P). Durch die Umsetzung von Live-Editing ist auch ein effizientes gleichzeitiges Arbeiten möglich (KoopQ3:4P, KoopQ5:4P). Weitere Interaktion wird durch die direkte Verfübarkeit des Chats ermöglicht (KoopQ4:4P).

Durch die Chat-Funktion wird direkte Kommunikation unterstützt (KommQ1:4P, KommQ5:4P). Ebenso werden im Chat Meta-Daten dargestellt und es existieren viele Möglichkeiten zur Hervorhebung der Kommunikationsinhalte (KommQ2:4P, KommQ3:4P). Weitere Interaktionsmöglichkeiten gibt es innerhalb des Chats nicht (KommQ4:0P).

Eine in-/direkte Koordination ist durch die Kommunikation möglich (KoorQ1:1P, KoorQ5:1P). Eine explizite Unterstützung zur Koordination ist aber nicht vorhanden (KoorQ2:0P, KoorQ3:0P, KoorQ4:0P).



Visual Studio Anywhere

Gleichzeitiges Editieren und Chat

	Q1	Q2	Q3	Q4	Q5	Summe
Kommunikation	4	4	0	4	4	16: **Level 4**
Kooperation	4	4	4	4	4	20: **Level 5**
Koordination	1	0	0	0	1	2: **Level 1**

2.3 Vorhandene Ansätze in freier Software

2.3.1 GitLab

GitLab[12] ist eine Open-Source-Nachbildung von GitHub. Die Funktionen sowie die Darstellung sind sehr ähnlich. Manche Funktionen werden aber zum Beispiel, vermutlich aus Markenschutzgründen, anders bezeichnet. Dazu zählt zum Beispiel ein *Merge Request* der bei GitHub *Pull Request* genannt wird. Der Vorteil von GitLab ist, dass man es selber hosten, verwalten sowie eigene Anpassungen am Quellcode vornehmen kann. Dies ist zwar auch mit GitHub Enterprise möglich, aber dieses ist kostenpflichtig. Die Bewertung entspricht der von GitHub.

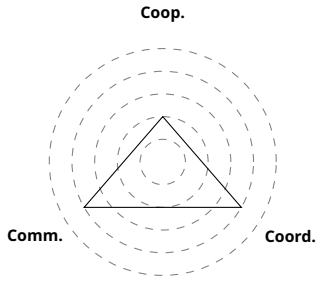
[12] `https://about.gitlab.com` (Besucht am 24.11.2014)

Bewertung von GitLab

Genau wie auch schon bei der Bewertung von GitHub, wird hier der gesamte Bug-Tracker-Bereich von GitLab betrachtet. Dazu gehören neben der Kommentarfunktion für Bug- und Merge-Requests damit auch Funktionen wie Labels und Meilensteine.

Es ist offensichtlich, dass Kommunikation möglich ist (KommQ1:4P). Weiterhin ist die Darstellung sehr ansprechend und zum schnellen erfassen optimiert (KommQ2:4P, KommQ3:4P). Es gibt Interaktionsmöglichkeiten durch das Anklicken von **@username** und referenzieren von anderen Issues durch **#999** (KommQ4:3P). Direkte Kommunikation ist nicht möglich (KommQ5:0P). Grundsätzlich ist die Kommentarfunktion für mehrere Benutzer ausgelegt (KoopQ1:3P) und es ist sehr einfach neue Kommentare zu veröffentlichen (KoopQ2:4P). Eine weitere direkte und effiziente Kooperation ist darüber aber nicht möglich. Die Kooperation wird nur indirekt durch die Diskussion des Quellcodes gefördert (KoopQ3:1P, KoopQ4:0P, KoopQ5:0P). Die hauptsächliche Kooperation findet aber auch im Rahmen der Bearbeitung des Quellcodes statt. GitLabs Bug-Tracker unterstützt durch die Kommunikation natürlich auch eine Koordination (KoorQ1:2P, KoorQ2:1P). Des Weiteren ist es aber auch möglich Benutzer zu nennen und diese werden darüber informiert (KoorQ3:4P, KoorQ4:4P). Zusätzlich ist es möglich den Issues Label zuzuordnen und Milestones festzulegen. So ist sogar eine leichte direkte Koordination möglich (KoorQ5:2P).

Damit entspricht die Bewertung genau der Von GitHub. Somit kann der Bug-Tracker von GitLab auch im Bereich der Kommunikation und Koordination eingeordnet werden.



GitLab Bug-Tracker

Mit Markdown-Hervorhebung, Verlinkung und Benachrichtigungen

	Q1	Q2	Q3	Q4	Q5	Summe
Kommunikation	4	4	4	3	0	15: **Level 4**
Kooperation	3	4	1	0	0	8: **Level 2**
Koordination	2	1	4	4	2	13: **Level 4**

2.3.2 FindBugs Cloud

FindBugs bietet mit dem Plugin FindBugs Cloud[13] eine Kommentarfunktion für die gefundenen Bugs (siehe Abbildung 2.15).

Ebenso können damit die Bugs nach vorgegebenen Klassifizierungen eingeordnet/als bearbeitet markiert werden. Dazu gehören „Needs further study, Not a bug, Mostly harmless, Should fix, Must fix, I will fix, Bad analysis, unclassified, Obsolete/unused (will not fix)“ (siehe Abbildung 2.16). Es können aber nicht nur einzelne Bugs kommentiert und klassifiziert werden sondern auch komplette Bug-Gruppen. Dazu werden aber nur die vorgegebenen Gruppen für Bugs verwendet.

Bugs werden dabei über ihren `InstanceHash` identifiziert (genauere Beschreibung in Unterunterabschnitt 4.5.2.2).

[13] `https://code.google.com/p/findbugs/wiki/FindBugsCloudTutorial` (Besucht am 24.11.2014)

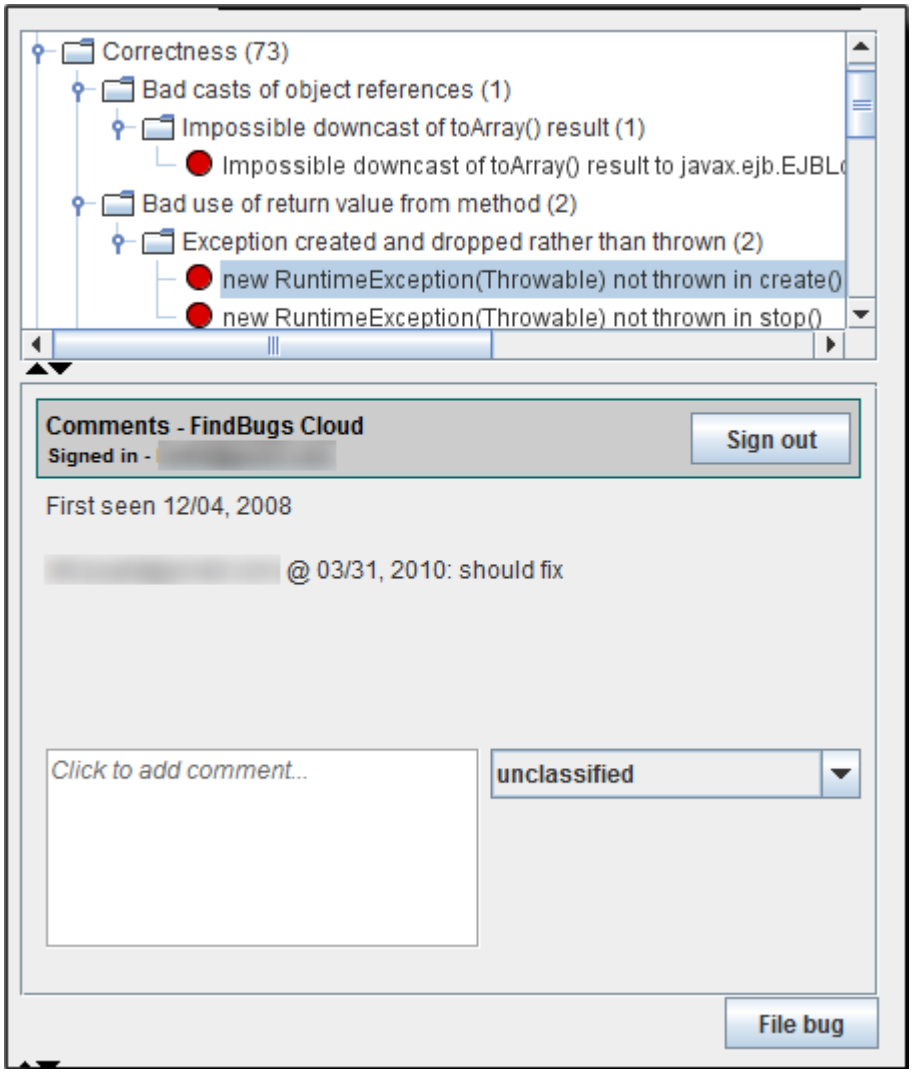


ABBILDUNG 2.15: FindBugs Cloud Kommentarfunktion GUI

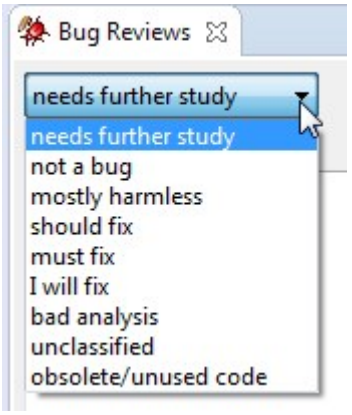
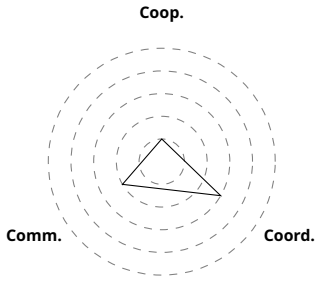


ABBILDUNG 2.16: FindBugs Cloud Klassifikationsauswahl für einen gefundenen Bug

Für die Benutzung dieser Erweiterung von FindBugs wird ein spezieller FindBugs Cloud Server benötigt. Man kann den bereits voreingestellten öffentlichen FindBugs Cloud Server verwenden oder einen solchen selber betreiben. Ein selber betriebener Server kann natürlich auch nur für eine begrenzte Anzahl an Leuten, privat/innerhalb eines Firmennetzwerkes, betrieben werden. Für die Anmeldung am Server wird nur eine OpenID[14] benötigt, bei der die E-Mail-Adresse weitergegeben wird.

Bewertung der FindBugs Cloud

Die durch die Kommentar- und Klassifikationsfunktion gebotenen Möglichkeiten für die Interaktion fallen einerseits in die Kategorie Kommunikation (Kommentarfunktion) und andererseits in die Kategorie Koordination (Klassifikationsfunktion). Dabei ist die Kommunikation durch die Kommentare möglich und Meta-Daten werden zwar dargestellt, aber alles nur wenig für den Benutzer aufbereitet (KommQ1:4P, KommQ2:2P). Daraus resultiert auch eine langsamere Erfassung des Inhalts (KoomQ3:0P). Interaktionsmöglichkeiten sind praktisch nicht vorhanden (KommQ4:0P, KommQ5:0P). Die Kooperation ist einzig durch den Status des Bugs möglich (KoopQ1:3P, KoopQ2:0P, KoopQ3:0P, KoopQ4:0P, KoopQ5:0P). Genau durch diese Funktionalität ist aber eine einfache und effiziente Koordination möglich (KoorQ1:4P, KoorQ2:1P, KoorQ3:4P KoorQ4:0P, KoorQ5:0P). Die Benutzbarkeit der gesamten FindBugs Erweiterung kann als gut und einfach bewertet werden, da die graphische Benutzeroberfläche keine weiteren Hürden für die Benutzung in den Weg stellt und die beschriebenen Funktionen in den Vordergrund stellt. Die Darstellung der Kommentare ist gestalterisch aber ungenügend. Hinzu kommt, dass die Erweiterung nur eine Art *Proof-of-concept* war und nicht weiter entwickelt wird.



FindBugs Cloud

Integrierte Kommentare für FindBugs Bugs

	Q1	Q2	Q3	Q4	Q5	Summe
Kommunikation	4	2	0	0	0	6: **Level 2**
Kooperation	3	0	0	0	0	3: **Level 1**
Koordination	4	1	4	0	0	12: **Level 3**

2.4 Vorhandene Ansätze in der Forschung

Neben den bis hier vorgestellten Ansätzen, die öffentlich verfügbar sind, gibt es noch nicht kommerziell und/oder wenig verbreitete Ansätze aus der aktuellen Forschung. Zwei der Ansätze werden hier vorgestellt.

[14] `https://openid.net` (Besucht am 24.11.2014)

2.4.1 Firmentinterne Expertensuche SmallBlue

Mit SmallBlue haben Lin et al. [28] eine Anwendung vorgestellt, die vorhandene Kommunikationsdaten innerhalb einer Firma nutzt, um daraus ein soziales Netzwerk von Experten abzuleiten. Es soll automatisch Antworten auf die Fragen „Wer weiß was?“ , „Wer kennt wen?“ sowie „Wer weiß was über wen?“ geben können. Das Ziel ist für ein bestimmtes Thema einen Experten zu suchen um mit diesem Kontakt aufnehmen zu können. Hierzu werden von SmallBlue, nach vorheriger Einwilligung, die Inhalte der persönlichen E-Mail Korrespondenz analysiert. Die aggregierten Informationen von möglichst vielen Benutzern werden über eine Weboberfläche zugänglich gemacht. Dabei kann ein Benutzer persönliche Informationen über sich selbst sehen, sowie verschiedene Suchanfragen über die Webseite formulieren. Dazu gehört die Suche nach anderen Leuten die vornehmlich Experten auf einem bestimmten in der Frage mit Stichworten formulierten Themengebiets sind. Das Ergebnis enthält mehrere Personen inklusive Kontaktinformationen die die höchste Übereinstimmung bezüglich der Stichwörter haben. Weiterhin wird, um die Wahrscheinlichkeit einer Kontaktaufnahme zu erhöhen, der kürzeste Pfad über bereits bekannte Personen angezeigt. Bei höchstens 3 zu kontaktierenden Personen werden diese direkt in einer Form wie *Frage <Person> nach <Experte>* angezeigt.

Aufgrund der nur für Firmen verfügbaren Software kann keine weitere Bewertung vorgenommen werden. Die Idee, vorhandene Kommunikation zu analysieren, bietet einerseits Vorteile, da diese sowieso stattfindet, andererseits auch Probleme im Sinne von Eingriffen in die Privatsphäre.

2.4.2 Jazz

Eine frühe Erweiterung der Eclipse-IDE um Funktionen die die Zusammenarbeit unterstützen und fördern sollten, wurde 2003 von Cheng et al. mit Jazz vorgestellt [29]. Jazz erweiterte Eclipse hauptsächlich um das sogenannte *Jazz Band*, Chat, Teilen des Bildschirminhalts sowie um *Concert Awareness*. Dabei ist das *Jazz Band* eine visuelle Erweiterung um einen View in dem andere Team-Mitglieder angezeigt werden. Diese Teams können von jedem zu jedem Zeitpunkt erstellt werden. Eine frühe Form des gleichzeitigen Arbeitens wird von *Concert Awareness* durch die Markierung von Ressourcen bezüglich des Bearbeitungsstatus vorbereitet. Durch die Möglichkeit des Bildschirmteilens konnte damals schon Paarprogrammierung an verschiedenen Arbeitsplätzen durchgeführt werden. Die Technik basierte aber noch auf Virtual Network Computing (VNC) und nur einer konnte jeweils tatsächlich Änderungen am Code vornehmen. Mittlerweile ist Jazz besser bekannt als IBM Rational Team Concert was als weiterentwickeltes aktuelles Programm kommerziell verkauft wird, nicht mehr nur in Eclipse sondern auch in Microsoft Visual Studio integriert ist und eine eigene Web-Oberfläche bietet.

2.4.3 Selbstbewusster Prompter für Eclipse

Ponzanelli et al. [30] haben eine Erweiterung für die Eclipse IDE entwickelt, die auf Basis des aktuell geöffneten Quellcodes möglichst relevante, dazu passende, Fragen und Antworten von Stack Overflow (siehe Unterabschnitt 2.2.1) sucht und bei hoher Relevanz dem Entwickler

anzeigt. Ausgehend von der Idee, dass ein Empfehlungssystem sich im besten Fall wie ein (Tele-)Prompter verhalten sollte: „Jederzeit bereit Vorschläge zu geben, wenn der Schauspieler sie braucht sowie bereit zu sein automatisch Vorschläge zu geben wenn er merkt, dass etwas falsch läuft“ wurde versucht dies umzusetzen. Die Erweiterung bietet dazu zwei neue Views in Eclipse an. Im *Notification Center* werden nur relevante (Schwellwert einfach über einen Schieberegler einstellbar) Fragen inklusive einer Anzeige, wie hoch die Übereinstimmung ist, angezeigt. Im zweiten View wird die Stack Overflow Diskussion angezeigt, sobald der Entwickler in im *Notification Center* die Frage anklickt. Um die Relevanz zu beurteilen werden verschiedene Kriterien angewendet. Dazu gehört die Textuelle Ähnlichkeit, Code Ähnlichkeit, API Typen- und Methoden-Übereinstimmung, Frage-Bewertung (*Question Score*), Antwort-Bewertung (*Answer Score*), Benutzer Reputation und *Tag* Ähnlichkeit. Diese Kriterien werden jeweils mit einem bestimmten Faktor gewichtet. Weiterhin wird nicht bei jeder Änderung des Codes eine Anfrage generiert, sondern auch nur wenn ein gewisser Schwellwert erreicht ist.

Es wurden zur Evaluation zwei Benutzerstudien durchgeführt. Das Ergebnis ist zweigeteilt. Als Stärken der Erweiterung werden die Genauigkeit und Relevanz der Vorschläge, die Benutzeroberfläche, die einfache Benutzbarkeit sowie die Möglichkeit, die Sensitivität für die Relevanz der angezeigten Fragen einzustellen, genannt. Für weitere Verbesserungen wird von den Studienteilnehmern gefordert, dass neben Stack Overflow weitere Quellen einbezogen werden sollen sowie ein Such-Feld für direktes Suchen mit eigenen Begriffen implementiert werden soll.

Eine weitere Bewertung kann hier aufgrund fehlende Zugriffs auf die Erweiterung nicht gemacht werden. Dennoch macht die vorgestellte Erweiterung einen sehr guten Eindruck was auch durch die Evaluationsergebnisse unterstützt wird.

2.5 Fazit

2.5.1 Relevanz der verwendeten Fragen und Visualisierung

Mache Plattformen und manche Software lässt sich mit den verwendeten Fragen kaum in eine Kategorie des 3C-Modells einordnen. Sie passen aber dennoch, auf Basis anderer Eigenschaften, in bestimmte Kategorien. Hierzu sollte der Fragenkatalog in Zukunft angepasst und evaluiert werden. Weiterhin wäre es sinnvoll die Level um ein Level 0 zu ergänzen. Damit kann besser zum Ausdruck gebracht werden, dass kaum bis keine Unterstützung für die entsprechende Kategorie des 3C-Modells vorhanden ist. Die Visualisierung unterstützt die Darstellung bereits indem für Level 0 der Kreismittelpunkt verwendet wird.

2.5.2 Ansätze für Social Coding

Wir haben nun verschiedene Ansätze vorgestellt und in die Kategorien des 3C-Modells eingeteilt (siehe dazu auch die zusammenfassende Tabelle in Abbildung A.1 im Anhang). Dabei ist aufgefallen, dass es prinzipiell unterschiedliche Ansätze gibt, die Social Coding ermöglichen:

- **Gleichzeitiges Editieren/Live Editing** als Erweiterung einer IDE.

  Durch moderne JavaScript-Technolgien und die weite Verbreitung der dafür nötigen Browser ist es möglich diese Technologien zur Unterstützung von Programmierern zu verwenden. In Form von neuen Online-IDEs, die gleichzeitiges Editieren durch mehrere Benutzer ermöglichen, ist effiziente Paarprogrammierung möglich. Ähnliche Ansätze, aber basierend auf anderen Technologien, halten auch Einzug in vorhandene IDEs um auch hier gleichzeitiges Editieren zu ermöglichen.
  (Cloud9, Visual Studio Anywhere)

- **Integration von Diskussion/Kommunikation** in eine IDE.

  Gerade im Zusammenhang mit gleichzeitigem Editieren bietet es sich an, die Beteiligten auch durch einen separaten Chat miteinander kommunizieren zu lassen.
  (Cloud9, Visual Studio Anywhere, Eclipse Prompter, FindBugs Cloud)

- **Online-Plattform zur Diskussion** des Codes und der Veränderungen.

  Durch die Verbreitung von verteilten Versionsverwaltungssystemen haben sich Plattformen um die davon verwalteten Repositories und der Abläufe davon geformt. Durch die hohe Code-Nähe können der Code und Änderungen daran direkt diskutiert werden und effizient an einer Stelle verwaltet werden. Des Weiteren bieten nicht auf einzelne Projekte fokussierte Plattformen weitere Diskussionsmöglichkeiten.
  (GitHub/GitLab, Stack Overflow, CodeProject)

- **Zusammenfassende Plattformen** zur Vernetzung der Entwickler.

  Einerseits durch die Aggregation verschiedener Kennzahlen die aus den Quellcode-Repositories oder anderen Beiträgen eines Entwicklers gewonnen werden können, andererseits durch die Darstellung von diesen Aktivitäten sind Entwickler besser dazu in der Lage einzuschätzen, an was andere arbeiten, diesen zu folgen, sich davon inspirieren zu lassen sowie bessere Entscheidungen zu treffen. Dies wurde auch durch eine Studie mit GitHub Entwicklern von Dabbish et al. [5] bestätigt. Dort wurde herausgefunden, dass Entwickler durch die Aktivitäten, teilnehmende Entwickler, verfügbare Dokumentation und allgemeine Transparenz entscheiden, welches Projekt für ein bestimmtes Problem die größten Zukunftsaussichten hat.
  (GitHub/GitLab, Masterbranch)

- **Social Tagging/Folksonomien**: Zur benutzergenerierten Verschlagwortung von Inhalten. Durch das Hinzufügen von klassifizierenden Begriffen können viele Inhalte besser und für die Entwickler relevanter klassifiziert werden. Im Rahmen von Social Coding geschieht die Verschlagwortung aber nicht automatisch anhand eines festgelegten Katalogs von Begriffen, sondern durch die Entwickler. (StackOverflow, GitHub/GitLab teilweise)

2.5.3 Möglichkeiten für die Implementierung

Mit den Erkenntnissen aus diesem Kapitel ergeben sich mehrere Möglichkeiten Social Coding für die ASA umzusetzen. Zunächst bietet es sich an, eine Anbindung an existierende Plattformen wie GitHub, Bitbucket oder Stack Overflow zu schaffen. Für Stack Overflow existiert bereits ein Ansatz. Deshalb wird im Folgenden eine **Export-Möglichkeit für gefundene Bugs**

vorgestellt. Weiterhin sind aber nicht alle Projekte, insbesondere Closed-Source- bzw. kommerzielle Projekte, dafür geeignet die öffentlichen Plattformen zu verwenden bzw. verwenden diese eben nicht. Für diese Projekte wird ein **Kommentarsystem** (Kapitel 4) beschrieben. Neben der einfachen Kommunikation kann ein Kommentarsystem bei geeigneter Wahl von Interaktions- und Darstellungsmöglichkeiten auch für eine effiziente Koordination sorgen. Für beide Ansätze werden aus den Analyseergebnissen und Erkenntnissen zunächst allgemeine Anforderungen abgeleitet. Danach werden verschiedene Vorgehensweisen das Umzusetzen diskutiert. Im Anschluss folgt die konkrete Implementierung mit der Erläuterung der getroffenen Entscheidungen, Darstellung von Abläufen und Vereinbarungen.

Kapitel 3

FindBugs Issue Export

3.1 Einleitung

Social Coding findet häufig direkt in den Bug-Tracker Einträgen auf den entsprechenden Plattformen statt [2]. Dort wird zu einem beschriebenen Fehler diskutiert ob er existiert, welche Auswirkungen er hat und wie er behoben werden kann. Diese Diskussion würde auch den von der ASA gefundenen möglichen Fehlern helfen, um herauszufinden ob die Fehler tatsächlich relevant sind und behoben werden müssen, oder nicht. In diesem Kapitel wird dazu eine Erweiterung eines ASA-Tools entwickelt. Zunächst wird entschieden, welches ASA-Tool erweitert werden soll. Danach wird die Implementierung im Allgemeinen vorgestellt. Darauf folgt eine Beschreibung von Anforderungen in Form von *User Stories* und dem jeweiligen Ansatz diese Anforderung zu erfüllen. Abschließend wird die Benutzeroberfläche vorgestellt und dieses Kapitel mit einer Bewertung zusammengefasst.

3.1.1 Zu erweiterndes ASA-Werkzeug

Ein häufig für Java-Projekte eingesetztes ASA-Tool ist **FindBugs**. Insbesondere wird hier in Verbindung mit Eclipse das entsprechende Plugin verwendet. Am Ende von Kapitel 1 wurde die Detail-Ansicht eines gefundenen möglichen Fehlers in Abbildung 1.4 bereits kurz vorgestellt. Diese Ansicht bietet eine kurze Beschreibung des für diesen Fehler verantwortlichen Fehlermusters sowie die Einstufung und weitere Merkmale. Nach einem Doppelklick auf einen möglichen Fehler im *Bug Explorer* wird der relevante Quellcode, an dem der Fehler auftritt, dargestellt. Hier ist es aufgrund der häufig über mehrere Pakete und Klassen aufgeteilten Codes schwierig für einen Entwickler, der sich gerade nicht mit diesem Bereich des Quellcodes auskennt, zu beurteilen, wie mit diesem möglichen Fehler weiter vorgegangen werden soll. Aufgrund der einfachen Verwendung und hohen Verbreitung von Java und Eclipse kann die FindBugs Eclipse Erweiterung als guter Kandidat für den Ausgangspunkt der Implementierung angesehen werden, die später auch evaluiert werden kann.

Des Weiteren wird die beschriebene Export-Möglichkeit auch von Benutzern gefordert[1,2] und wurde bis jetzt nicht umgesetzt.

3.2 Implementierung

3.2.1 Einführung

Social Coding Techniken versuchen dem Benutzer so wenig wie möglich Hürden in den Weg zu legen um seine Ziele zu erreichen. Diese Strategie wird auch mit dieser Erweiterung verfolgt. Das Ziel ist, eine möglichst einfache Integration in den vorhandenen Arbeitsablauf. Dieser ist aber häufig unterschiedlich, weshalb hier auf nicht untersuchten Annahmen gearbeitet wird. Es wird angenommen, dass viele Entwickler selten ihr Projekt mit FindBugs analysieren, einmalig jeweils die Fehlerliste im Bug Explorer durchsehen und dann versuchen diese möglichen Fehler zu beheben oder falls sie sich für keine Lösung entscheiden können Fehler erstmal unberührt lassen. An diesem Punkt soll die Erweiterung dem Entwickler ermöglichen, den Fehler als Bug an das Projekt zu melden, um anderen Entwicklern eine Beurteilung zu ermöglichen.

Weiterhin soll die Erweiterung möglichst automatisch vorgehen um den Bug direkt melden zu können. Hierzu soll zum Beispiel die verwendete Plattform automatisch erkannt werden. Weitere Implementierungsentscheidungen werden jetzt vorgestellt.

3.2.2 Realisierung

3.2.2.1 Eclipse-Plugin

Die Erweiterung ist als Eclipse-Plugin realisiert. Sie erfordert natürlich das Vorhandensein des FindBugs Eclipse-Plugins um einerseits auf bestimmte Klassen von FindBugs wie `BugInstance` zuzugreifen und andererseits das dort eingeführte Kontextmenü zu erweitern. Um ein vorhandenes Kontextmenü zu erweitern, werden die Art des zu erweiternden Objekts sowie der vollständige Name der vorhandenen implementierenden Klasse benötigt. In diesem Fall ist das Objekt vom Typ *org.eclipse.core.resources.IMarker* und die Klasse *de.tobject.findbugs.MarkerActions*. Weiterhin muss angeben werden, welche Klasse die Aktion verarbeitet wenn geklickt wird. Dies ist in diesem Eclipse-Plugin-Projekt die Klasse *RightClickIssueShowExportAction* welche das Interface *IObjectActionDelegate* implementiert. In deren *run-Methode* kann nun die Export-Klasse aufgerufen werden.

Die Export-Klasse zeigt zunächst für Bugs, die eine zu niedrige *Confidence* haben, eine Warnung an, ob dieser Bug wirklich exportiert werden soll. Danach wird ausgehend von der vom Fehler betroffenen Quellcode-Datei das dazugehörige git-Repository gesucht. In diesem wird daraufhin nach vorhanden *remote*-Einträgen in der Konfiguration gesucht um herauszufinden,

[1] `https://sourceforge.net/p/findbugs/feature-requests/190/` (Besucht am 24.11.2014)
[2] `http://stackoverflow.com/questions/21328802/how-to-create-tickets-from-findbugs-reports` (Besucht am 24.11.2014)

auf welcher Plattform dieses Projekt verwaltet wird. Ausgehend von diesen Angaben wird die *PlatformExporterFactory.matchPlatform*-Methode verwendet um herauszufinden ob für diese Plattform eine entsprechende Klasse für den Export vorhanden ist. Falls das der Fall ist, wird eine Instanz der Klasse zurückgegeben. In dieser existiert immer, sichergestellt durch die Implementierung des *IPlatformExporter*-Interfaces, eine *exportBug*-Methode mit welcher der Bug plattformabhängig exportiert werden kann. Zur Veranschaulichung ist in Abbildung 3.1 ein UML-Klassendiagramm dargestellt.

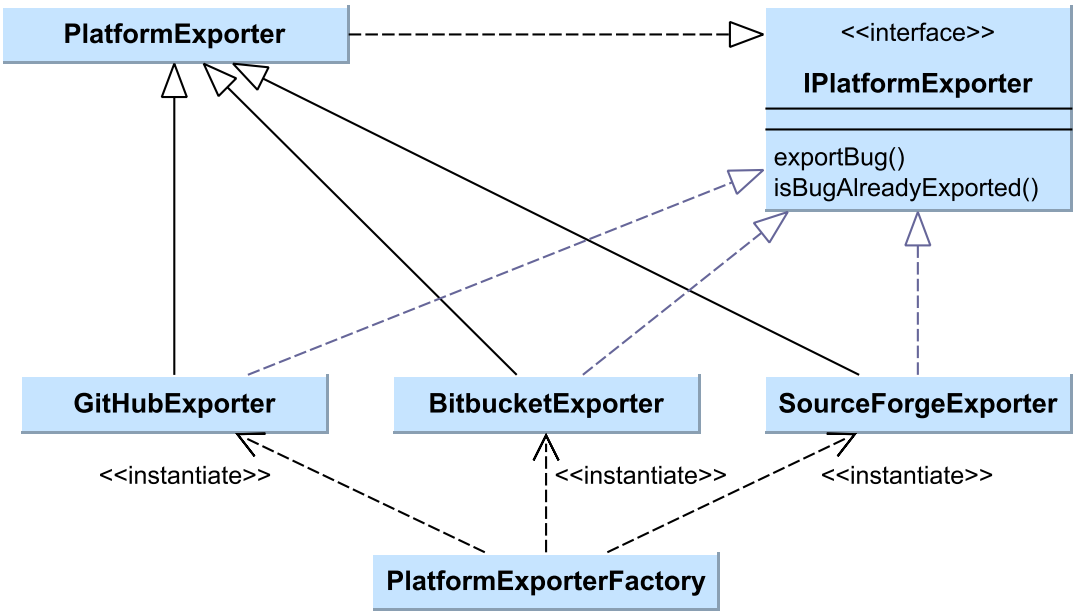


ABBILDUNG 3.1: FBIssueExport Klassendiagramm

Die Art und der Ablauf des Exports ist immer abhängig von der verwendeten Plattform. Ein Beispielablauf für GitHub wird im Folgenden beschrieben.

3.2.2.2 Beispiel Ablauf für GitHub Projekt

Für GitHub muss aus der remote-URL der Besitzer sowie der Name des Repositories herausgefunden werden. Dies geschieht mit dem regulären Ausdruck:

```
1  Pattern platformUrlPattern =
2  Pattern.compile("((git@|https://)([\\w\\.@]+)(/|:))([\\w,\\-,\\_]+)/([\\w,\\-,\\_]+)
       (.git){0,1}((/){0,1})");
```

Dieser ist sogar so allgemein gehalten, dass er auch für andere Plattformen, die das Format `<platformurl>/<owner>/<repositoryName>` verwenden, funktioniert. Nachdem der Besitzer und der Name des Repositories bekannt ist, muss überprüft werden ob das Repository ein *Fork* (Abspaltung eines anderen Projekts) ist. Falls das nämlich der Fall ist, gibt es auf GitHub für das Projekt keinen eigenen Issue-Tracker, sondern nur im ursprünglichen Projekt. Per GitHub-API kann dies mittels eines einfachen Hypertext Transfer Protocol (HTTP) GET Requests und der Verarbeitung der JavaScript Object Notation (JSON)-Antwort herausgefunden

```
1  String md = "";
2  md += "# " + bugInstance.getAbridgedMessage() + "\n";
3  md += "\n\n"
4          + bugInstance.getBugPattern().getDetailText() + "\n\n";
5
6  md += "The problem occurs in `"
7          + bugInstance.getPrimarySourceLineAnnotation().getClassName()
8          + "` on line **" + bugInstance.getPrimarySourceLineAnnotation().getStartLine
       () + "**";
9  if(bugInstance.getPrimaryMethod() != null) {
10     md += "in method `" + bugInstance.getPrimaryMethod().getMethodName() + "`";
11 } else if(bugInstance.getPrimaryField() != null) {
12     md += "in field `" + bugInstance.getPrimaryField().getFieldName() + "`";
13 } else if(bugInstance.getPrimaryLocalVariableAnnotation() != null) {
14     md += "in local variable `" + bugInstance.getPrimaryLocalVariableAnnotation().
       getName() + "`";
15 }
16 md += ":\n\n";
17
18
19 md += "```java\n";
20 md += getSourceCodeFragment(ProjectUtils.getSourceFile(project, bugInstance),
       bugInstance.getPrimarySourceLineAnnotation().getStartLine() - 5, bugInstance.
       getPrimarySourceLineAnnotation().getEndLine() + 5 );
21 md += "```\n\n";
22
23 md += "We have **" + bugInstance.getPriorityString() + "** confidence for this **" +
        BugRankCategory.getRank(bugInstance.getBugRank()) + "** bug!";
24 md +="\n\nThis bug was found by FindBugs and exported using kmindi's [FBIssueExport
       ](https://github.com/kmindi/FBIssueExport). \n"
25         + "(FindBugs Bug-ID: "+ bugInstance.getInstanceHash() + ")";
```

ABBILDUNG 3.2: Rumpf der getBugDescription-Methode

werden. Falls das Projekt ein Fork ist, kann dort der ursprüngliche Besitzer und Name des Repositories abgelesen und im Folgenden weiter verwendet werden.

Um jetzt einen neuen Issue zu erstellen, muss zunächst ein Titel und eine Beschreibung dafür erstellt werden. Hierzu wird auf die vorhandenen Daten der *BugInstance* zurückgegriffen. Der Titel wird aus *bugInstance.getMessageWithoutPrefix()* gewonnen. Die ausführliche Beschreibung inkl. Quellcode-Ausschnitt wird durch die in Abbildung 3.2 aufgelistete *getBugDescription*-Methode erzeugt.

Für GitHub gäbe es auch die Möglichkeit den Issue per API direkt zu erstellen, für den Benutzer ist es aber meiner Meinung nach intuitiver, wenn das bekannte Web-Formular geöffnet wird, mit den gerade erstellen Daten gefüllt wird und er diese sehen und auch noch bearbeiten kann, bevor er selbst auf *Submit new issue* drücken kann. Zur Umsetzung wird der Browser geöffnet und per GET-Parameter die Beschreibung und der Titel angehängt. GitHub verarbeitet die Parameter um das Formular damit zu füllen.

Um verschiedene Abläufe in den Exportern zu unterstützen sind einige Java-Bibliotheken erforderlich. Diese wird im nächsten Abschnitt vorgestellt.

3.2.2.3 Software-Abhängigkeiten

Aufgrund komplexer Konfigurationsvorgänge ist das Plugin zurzeit noch ohne eine automatische Abhängigkeitskonfiguration wie sie beispielsweise Tycho[3] bieten würde. Deshalb gibt es nun zwei Arten von Abhängigkeiten für das Eclipse-Plugin. Einerseits die Abhängigkeiten die über Eclipse aufgelöst werden können und externe Abhängigkeiten.

Wichtige Eclipse-Abhängigkeiten sind:

- **FindBugs Plugin (3.0.0)** zur Analyse mit FindBugs sowie für den Zugriff auf die Ergebnisse und einzelne Felder der gefundenen möglichen Fehler.
- **JGit (3.4.1)** zum Zugriff auf die git-Repositories und die Kapselung von verschiedenen git-Befehlen in einfach zu verwendende Java-Methoden.
- **Apache log4j (1.2.15)** für Debugging- und Logging-Zwecke.

Weitere nicht über Eclipse alleine lösbare Abhängigkeiten werden durch die folgenden Bibliotheken, welche aktuell als *jar*-Dateien mitgeliefert werden, bereitgestellt:

- **Apache HttpClient (4.3.6)** für Anfragen über HTTP.
- **Jackson(-Databind) (2.4.3)** für die Verarbeitung von JSON-Antworten.

3.3 User Stories

Für einige weitere Anforderungen werden die in Alltagssprache gehaltenen, in der agilen Softwareentwicklung eingesetzten, User-Stories verwendet. Diese werden nach einem einfachen Muster wie zum Beispiel „Als <Rolle> möchte ich <Ziel>, um <etwas> zu erreichen.“ formuliert. Dabei kann <etwas> auch weggelassen werden. Im Folgenden einige mögliche User-Stories für das FindBugs Kommentarsystem.

3.3.1 Internetzugriff

„Als Benutzer möchte ich, dass die Anwendung mich frägt bevor sie auf das Internet zugreift bzw. sensible Informationen darüber versendet.“

Diese Erweiterung greift nur auf die sowieso als *remote*-Repositories angegebenen Plattformen zu. Bis jetzt wird der Benutzer nicht explizit gefragt ob ein Zugriff stattfinden darf.

[3] `http://www.eclipse.org/tycho/` (Besucht am 24.11.2014)

3.3.2 Einfach und Intuitiv bedienbar

„Die Erweiterung soll möglichst einfach und intuitiv bedienbar sein."

Diese Anforderung wird durch den hohen Grad der Automatisierung sowie durch den einfachen Ablauf des Exportierens umgesetzt.

3.3.3 Erster Start / Automatisches erkennen der verwendeten Plattform

„Die Erweiterung soll automatisch erkennen auf welcher Plattform das Projekt verwaltet wird und sich das merken."

Das automatische Erkennen der verwendeten Plattform findet zurzeit nur für mit git verwaltete Projekte statt. Bei diesen können aus der git-Konfiguration die *remote*-Repositories ausgelesen und diese auf bekannte Plattformen hin untersucht und verarbeitet werden.

3.3.4 Doppelte Issue-Tracker Einträge

„Es soll vermieden werden Bugs einzutragen, die bereits eingetragen sind."

Vor dem Exportieren überprüfen ob der Bug vielleicht schon gemeldet wurde und noch diskutiert oder schon behoben wurde. Dafür wird die Bug-ID in die Beschreibung aufgenommen um die vorhandenen Issues danach überprüfen zu können. Das Interface *IPlatformExporter* legt fest, dass ein PlatformExporter auch die Methode *isBugAlreadyExported* implementieren muss. Diese soll eine plattformabhängige Überprüfung ermöglichen. Bis jetzt ist diese nur für GitHub implementiert.

3.3.5 Mehrere Bugs gleichzeitig melden

„Als Benutzer möchte ich Bugs einer Katgeorie gemeinsam exportieren können."

Dieses Verhalten ist bis jetzt nicht implementiert. Für die Implementierung gibt es zwei Entscheidungsvarianten. Entweder alle Bugs werden einzeln gemeldet, dies ist leicht umzusetzen indem über die enthaltenen Bugs iteriert wird. Oder es wird ein einzelner Bug Eintrag mit zusammenfassenden Informationen über die enthaltenen einzelnen möglichen Fehlern erstellt.

3.4 Benutzeroberfläche

Die Benutzeroberfläche für die Export-Funktion ist sehr klein. Zurzeit besteht sie sogar nur aus der Erweiterung des Kontextmenüs der Bug-Einträge im *Bug Explorer* von FindBugs (siehe Abbildung 3.3. Für weitere Interaktion, falls nötig, werden Abfrage-Dialoge, wie in Abbildung 3.4 dargestellt, verwendet.

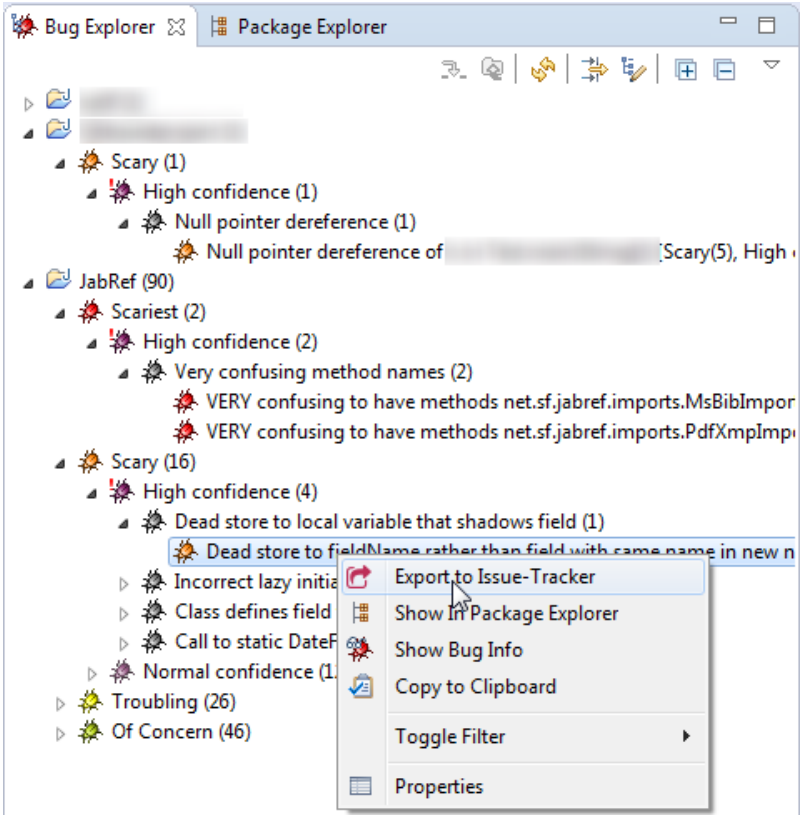


ABBILDUNG 3.3: FindBugs Issue Export Kontextmenü innerhalb des Bug Explorer Views.

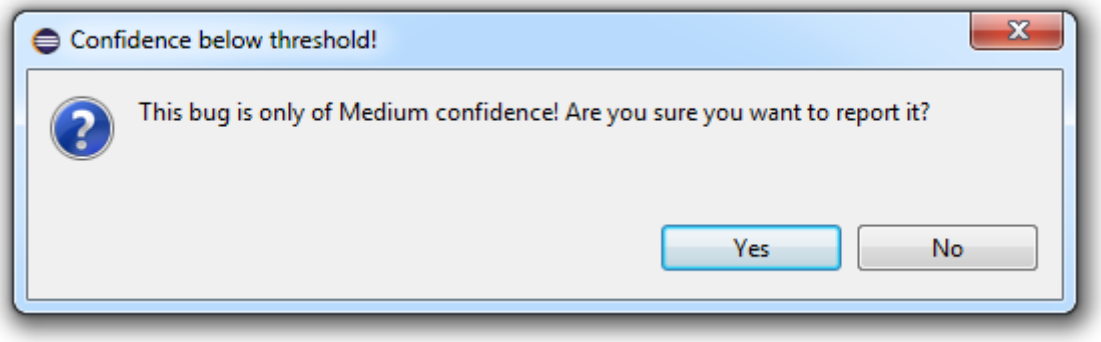


ABBILDUNG 3.4: Falls der Bug eine zu kleine *Confidence* hat, wird der Benutzer gefragt ob er diesen Bug wirklich melden will.

Die weitere Benutzeroberfläche besteht nun aus den Webseiten der jeweiligen Issue-Tracker auf denen der Bug gemeldet werden soll. Dazu wird das Formular bereits mit den nötigen Daten gefüllt, siehe Vorschau der mit Markdown formatierten Beschreibung des Bugs im Issue-Tracker auf GitHub in Abbildung 3.5.

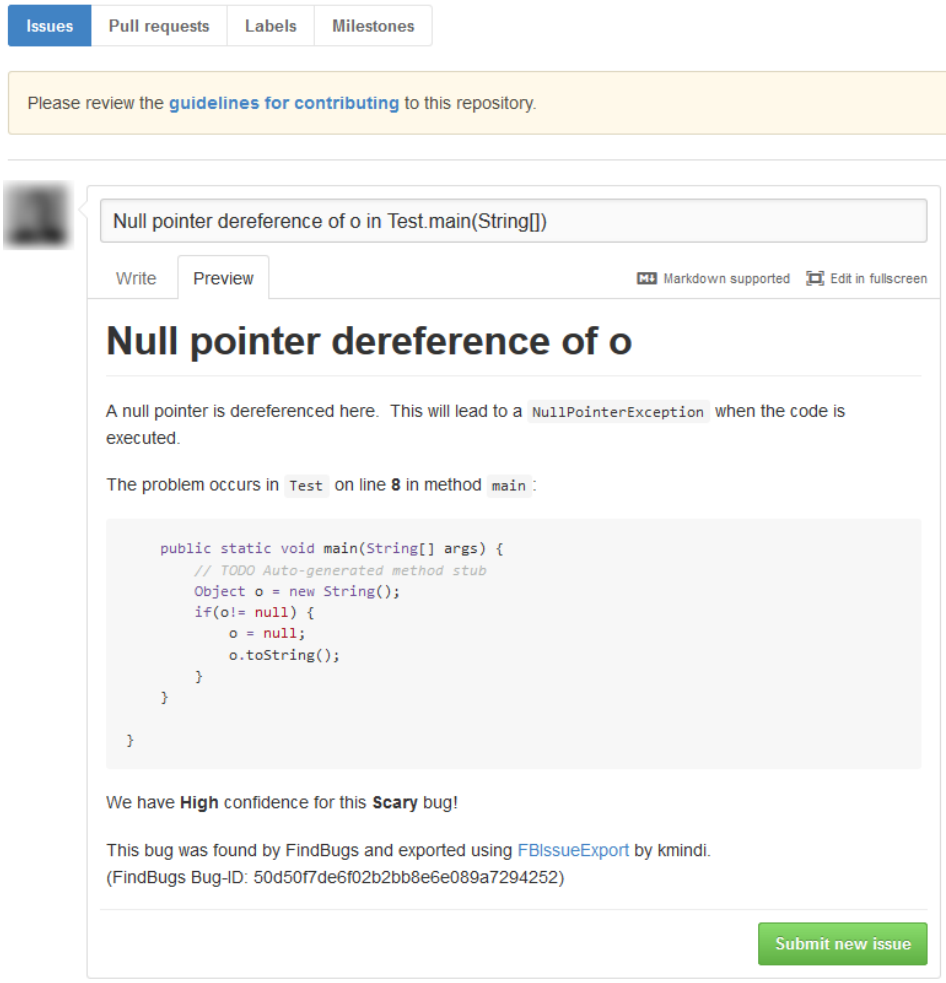


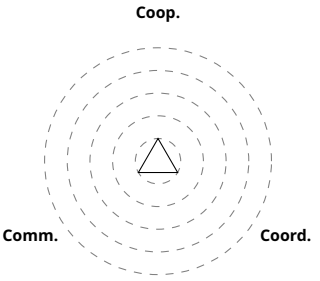
ABBILDUNG 3.5: Vorschau der mit Markdown formatierten Beschreibung des Bugs im Issue-Tracker auf GitHub.

3.5 Fazit

3.5.1 Bewertung

Es ist bereits offensichtlich, dass die Erweiterung nur ein Zwischenschritt zwischen den Fehlern und den vorhandenen Social Coding Plattformen ist. Deshalb ist es auch schwer, sie in eine der Kategorien, Kommunikation, Kooperation oder Koordination einzuordnen. Auf Basis der in der Analyse vorgestellten Fragen ergibt sich jeweils Level 1 (KommQ1:0P, KommQ2:0P, KommQ3:0P, KommQ4:0P, KommQ5:0P; KoopQ1:0P, KoopQ2:0P, KoopQ3:0P, KoopQ4:0P, KoopQ5:0P; KoorQ1:0P, KoorQ2:0P, KoorQ3:0P, KoorQ4:0P, KoorQ5:0P). Die Idee ist aber natürlich,

die Diskussion zwischen den Entwicklern zu unterstützen. Diese findet dann aber auf den entsprechenden Plattformen statt.



FindBugs Issue Export

Melden von FindBugs Bugs bei verschiedenen Issue-Trackern

	Q1	Q2	Q3	Q4	Q5	Summe
Kommunikation	0	0	0	0	0	0: **Level 1**
Kooperation	0	0	0	0	0	0: **Level 1**
Koordination	0	0	0	0	0	0: **Level 1**

3.5.2 Ausblick

Die in diesem Kapitel entwickelte Export- oder Melde-Möglichkeit für von FindBugs gefundenen möglichen Fehlern ist eine einfache Umsetzung zur Integration von Social Coding. Mit einem Rechtsklick kann ein Entwickler nun gefundene Bugs über den von seinem Projekt verwendeten Issue-Tracker melden. Unterstützt wird dabei bis jetzt nur GitHub, Bitbucket und SourceForge. Dabei funktioniert die automatische Erkennung der verwendeten Plattform zurzeit nur mit git. Neben der Erweiterung um weitere Plattformen und weitere Versionsverwaltungssysteme wird als nächster Schritt die Verwendung von Mylyn[4] gesehen. Mylyn verwaltet Aufgaben für Projekte und bietet in Eclipse eine aufgabenorientierte Ansicht an. Weiterhin bietet es Schnittstellen zu vorhandenen Plattformen an. Diese sollten von der Erweiterung vorrangig genutzt werden, falls Mylyn benutzt wird. Hierbei gilt es für die weitere Entwicklung zu beurteilen, ob eine Mylyn-Anbindung obligatorisch sein soll, sprich, dass es erforderlich ist, Mylyn zu verwenden um Bugs über die Erweiterung melden zu können. Dies hängt davon ab wie viele Entwickler Mylyn nicht einsetzen. Ein weiterer Schritt, um Social Coding in die ASA zu integrieren, ist andere ASA-Programme um so eine Möglichkeit zu ergänzen. Hierzu sollte nach der Popularität der verwendeten Programme vorgegangen werden sowie die bereits vorhandenen Möglichkeiten beachtet werden. Abschließend gilt es die Erweiterung durch eine Benutzerstudie zu evaluieren und herauszufinden, ob Entwickler durch diese tatsächlich gemeinsam besser in der Lage sind, Entscheidungen zu treffen und Lösungen für die möglichen Fehler zu finden.

Nachdem in diesem Kapitel auf vorhandene Social Coding Plattformen zurückgegriffen wurde, soll im nächsten Kapitel noch eine Erweiterung vorgestellt werden, die eine Art von Social Coding selber umsetzt.

[4] `http://www.eclipse.org/mylyn/` (Besucht am 24.11.2014)

Kapitel 4

Implementierung Kommentarsystem

4.1 Einleitung

In diesem Kapitel wird eine weitere Möglichkeit vorgestellt, wie die Automatische Statische Code-Analyse (ASA) um Social Coding ergänzt werden kann. Nachdem im vorherigen Kapitel vorhandene Ansätze nur angebunden wurden, wird nun ein autarkes System entwickelt. Die grundsätzliche Annahme ist, dass verschiedene Entwickler ein Problem zusammen durch die Diskussion, sei es über einen Chat oder ein indirektes Kommentarsystem, besser gemeinsam lösen können. Dazu trägt jeder mit seiner Erfahrung und seinem Fachwissen zur Behebung eines Problems bei. Im Weiteren soll eine Diskussion für die Bugs ermöglicht werden. Zunächst werden dazu allgemeine Anforderungen behandelt, entschieden, was für eine Art Diskussion unterstützen werden soll und welches ASA-Programm erweitert werden soll. Danach werden verschiedene Implementierungsmöglichkeiten gezeigt und es wird eine davon implementiert. Daraufhin wird näher auf die konkrete Umsetzung eingegangen und die getroffenen Architektur-Entscheidungen und Spezifikation vorgestellt. Als nächstes werden Anforderungen und ihre Umsetzung in Form von User-Stories präsentiert. Am Ende folgt noch die Beschreibung der Benutzeroberfläche sowie das bewertende Fazit.

4.2 Allgemeine Anforderungen

Egal welches Programm erweitert werden soll, es sollen auf jeden Fall folgende Dinge bei der Umsetzung beachtet werden:

- **Art der Diskussion**: Bei der Entscheidung ob direkte Kommunikation mittels Chat überhaupt gebraucht wird, wird davon ausgegangen, dass vermutlich wenige Leute gleichzeitig an den Berichten der ASA-Tools arbeiten. Deshalb lohnt es sich erst gar nicht, real-time Ansätze nicht weiter zu verfolgen. Sprich es reichen asynchrone Kommunikationsmöglichkeiten. Natürlich kann es vorkommen, dass die Entwickler sowohl zufällig als

auch geplant gleichzeitig an einem Problem arbeiten, dies soll hier aber nicht der primäre Fokus sein. Das bedeutet, dass im Folgenden also ein **Kommentarsystem** entwickelt werden wird.

- **Einfach und intuitiv**: Die Benutzeroberfläche soll einfach, intuitiv bedienbar und vergleichbar mit aktuellen Social Coding Technologien sein.
- **Gute Benutzererfahrung**: Die mit der Erweiterung verbundene Benutzererfahrung soll gut sein. Dies soll vor allem durch die gerade beschriebene Benutzeroberfläche, deren Umsetzung und Integration erreicht werden.

4.3 Auswahl des zu erweiternden ASA-Programms

Im vorhergehenden Kapitel 3 wurde bereits für FindBugs argumentiert und deswegen wird es hier auch wieder verwendet. Für FindBugs existiert schon ein Ansatz in Form der in Unterabschnitt 2.3.2 vorgestellten FindBugs Cloud. Dort wurde aber auch angemerkt, dass die Umsetzung ein *Proof-of-concept* ist und nicht weiter aktiv entwickelt wird. Aus diesem Grund wird hier nun ein ähnlicher Ansatz, der aber eine bessere Benutzererfahrung bieten soll, entwickelt.

4.4 Implementierungs-Möglichkeiten

Es gibt verschiedene Möglichkeiten die Anforderungen umzusetzen. Zunächst könnte man eine Client-Server-Architektur verwenden. Dabei wird die Authentifizierung über einen zentralen Server vorgenommen und auf diesem werden auch die Kommentare gespeichert. Eine weitere Möglichkeit wäre es, ein verteiltes Back-End zu verwenden bei dem es auch möglich ist, teilweise offline zu arbeiten. Hierfür könnte git verwendet werden. Beide Ansätze haben Vor- und Nachteile die nun gegeneinander abgewogen werden:

- **Client/Server Architektur**: Auf einem zentralen Server werden die Daten wie Bug-IDs, Bug-Beschreibungen und Kommentare sowie Daten für berechtigte Benutzer, zwecks Authentifizierung gespeichert. Ein Client baut eine Verbindung auf und lädt Bugs/Kommentare herunter und seine Kommentare, aktuelle Bugs/Änderungen hoch.

  Vor- und Nachteile einer Client/Server Implementierung:

  + Live; verbundene Clients können über Änderungen direkt informiert werden
  o Von der Quellcode-Verwaltung abgekoppelt
  - Evtl. separate Offline-Implementierung in den Clients
  - Zentraler Server
  - Extra Benutzerverwaltung/Authentifizierung erforderlich
  - Separate Datenbank erforderlich

- **Git-basiert**: Es werden die Repository-Funktionen von git dafür verwendet, die Kommentare zu speichern und unter den Teilnehmern zu verteilen. Dafür gibt es grundsätzlich zwei Möglichkeiten. Entweder die Verwaltung findet in einem separaten Repository statt. In diesem Bug-git-Repository wären dann nur die Bug-IDs, Beschreibungen und Kommentare gespeichert. Oder es wird ein Unterordner im vorhandenen Quellcode-Repository verwendet.

  Vor- und Nachteile einer git-basierten Implementierung:

  + Kein zentraler einziger Server (*single point of failure*).
  + Quellcode-nah.
  + Mit vorhandenen Tools (git u. Texteditor) verwendbar.
  + Benutzerverwaltung/Authentifizierung/Sichtbarkeit inklusive; Benutzer mit push-Rechten können, für andere sichtbar, Kommentieren.
  + Bei der Verwendung von einem Unterordner, braucht man nur Zugriff auf den Quellcode und schon hat man auch Zugriff auf alle Kommentare.
  o Nicht Live; es ist ein regelmäßiges Anfragen auf Änderungen nötig. Live-Kommentare sind aber auch nicht erforderlich, siehe Argumentation gegen IM-Funktionalität (Abschnitt 4.2).
  o Nur für Projekte die mit git verwaltet werden. Falls nicht in einem Unterordner gespeichert wird, auch unabhängig vom verwendeten Versionsverwaltungssystem.
  - Ohne Erweiterung nur E-Mail als Kontaktmöglichkeit.
  - Je nach Implementierung umständliche Umsetzung für Änderungen und Löschen von Kommentaren.

Wegen mehr Vorteilen, der möglichen direkten Integration in die Quellcodeverwaltung und die damit verbundene Nähe zum Quellcode, wird im Folgenden eine git-basierte Implementierung erstellt.

4.5 Git-basiertes Back-End

4.5.1 Einführung

Die Idee, git als Back-End für Alternative Benutzung zu verwenden, ist nicht neu. Es gibt eine ganze Reihe Programme die git als Back-End für die verschiedensten Anwendungen verwenden. Eine Liste kann unter [31] eingesehen werden. Verteilte Bug-Tracker gibt es auch. Dazu gehören beispielsweise ditz[1], git-issues[2] oder Bugs Everywhere[3]. Diese sind im großen und ganzen aber entweder nicht mehr in Entwicklung oder bieten nur ein Kommandozeilen-Interface. Alle dieser Programme legen eine eigene Dateistruktur für die Verwaltung der Bugs fest und verwenden das Versionsverwaltungsprogramm nur um diese Daten dort einzutragen und damit anderen Entwicklern verfügbar zu machen. Einerseits könnte der Einfachheit halber

[1] `https://rubygems.org/gems/ditz` (Besucht am 24.11.2014)
[2] `https://github.com/duplys/git-issues` (Besucht am 24.11.2014)
[3] `http://bugseverywhere.org/` (Besucht am 24.11.2014)

eines dieser Programme integriert werden. Für keines ist aber eine Java-Implementierung vorhanden und so müsste ein Wrapper für die jeweilige Kommandozeilenversion programmiert werden oder eine eigene Java-Implementierung.

Andererseits gibt es eine noch nicht betrachtete Möglichkeit einen Bug-Tracker mit einem Versionsverwaltungssystem abzubilden. Man könnte auch **die vorhandene Kommentarmöglichkeit der git-Commits für die Speicherung von Kommentaren verwenden**. Der Vorteil wäre, dass Metadaten zu einem Kommentar (bzw. Commit) schon vom Versionsverwaltungssystem erfasst und nur noch abgerufen werden müssen. Dies hört sich erstmal sehr einfach an, es gibt aber noch verschiedene Probleme die beachtet und behoben werden müssen. Dazu wird als nächstes die konkrete Realisierung vorgestellt.

4.5.2 Realisierung

4.5.2.1 Branch-/Tag-/Datei- und Ordnerstruktur Vereinbarungen

Nachdem ein Commit gemacht wurde und die Commit-Nachricht als Kommentar verwendet wurde, ergibt sich folgendes Problem: Im Allgemeinen weiß man nun nicht mehr, zu welchem Bug der Kommentar überhaupt verfasst wurde. Dies kann man sich aber durch die Verwendung von git-Branches merken. Dazu wird für jeden Bug und seine Kommentare ein eigener Branch verwendet. In diesem Zusammenhang lässt sich auch ein schwer wiegendes Argument für die Integration als Unterordner direkt im Quellcode-Repository finden. Wenn die Bugs in einem Unterordner und einem eigenen Branch behandelt werden, kann dieser Branch auch für Änderungen am eigentlichen Quellcode verwendet werden und die Nachrichten zu den Quellcode-Commits könnten parallel zu den Kommentaren dargestellt werden. Da für die Diskussion des Bugs aber auch eine Beschreibung und in Zukunft eventuell weitere Dateien in Ordnern gespeichert werden sollen, könnte es andererseits als störend angesehen werden, wenn diese Ordner/Dateien sowie Branches und Commits das eigentliche Quellcode-Repository aufblähen. Der Vorteil bei git ist, dass nicht alle Branches eines Repositories geladen werden müssen. So kann die Implementierung die Branches erst dann holen, wenn sie benötigt werden. Deshalb wird die Möglichkeit ein separates Repository zu verwenden verworfen. Dies kann durch die manuelle Erstellung natürlich trotzdem umgesetzt werden. Eine automatische Integration und/oder Erkennung durch die Erweiterung ist aber nicht realisiert.

Die gerade beschriebene Struktur aus Bugs, Commits und Branches wird jetzt wie folgt genauer definiert:

- **Branch für ASA-Bug Tracking**: Es muss ein Branch mit dem Namen `asa-bugs` vorhanden sein.
- **Branch für jeden Bug**: Jeder Bug erhält einen eigenen Branch mit einem eindeutigen Namen `asa-bug-ID` (die ID entspricht der FindBugs Bug-ID, siehe Unterunterabschnitt 4.5.2.2). Der Branch ermöglicht die Zuordnung der Commits zum Bug. Der Branch wird ausgehend vom `asa-bugs`-Branch erstellt.

- **Tag[4] für ersten Commit**: Um den Beginn der Commits für einen Branch erkennen zu können, ohne Commit-Inhalte verarbeiten zu müssen, soll ein weiterer Name (`asa-bug-begin-ID`) verwendet werden, der den ersten Commit für den Bug bezeichnet bzw. im Sinne von git darauf zeigt. Es könnte auch `git merge-base` verwendet werden, aber dies würde nicht mehr funktionieren wenn in Zukunft manche Branches vereint werden, da dann die *Merge-Basis* eine Neue ist.
- **Datei-/Ordnerstruktur**: Im Wurzelverzeichnis des *Working Directory*s wird ein Ordner mit dem Namen `.fbcomments` erwartet. Dieser muss im `asa-bugs`-Branch vorhanden sein. Zusätzlich gibt es dort einen Unterordner für jeden Bug. Der Name der Ordner ist die eindeutige ID eines Bugs. In jedem Bug-Ordner muss eine `DESCRIPTION.md`-Datei liegen. In dieser Textdatei befindet sich die Beschreibung des Bugs sowie weitere Informationen die von Benutzern hinzugefügt und aktualisiert werden können. Die Bug-Ordner müssen nur im Branch des jeweiligen Bugs vorhanden sein.
- **Commit-Nachrichten/Commits**: Commits können entweder leere Commits sein („git commit –allow-empty“), können eine Änderung an der `DESCRIPTION.md`-Datei enthalten oder weitere Dateien im Bug-Ordner ändern und hinzufügen. Alle Commits müssen im Branch des Bugs getätigt werden, da darüber die Zuordnung stattfindet.

Die Branch- und Tag-Repräsentation kann in Abbildung 4.1 exemplarisch betrachtet werden.

Es wird davon ausgegangen, dass die Benutzer nicht destruktiv arbeiten. Falls doch, kann über die vorhergehenden Commits der ursprüngliche Zustand wiederhergestellt werden.

4.5.2.2 Bug-IDs in FindBugs

In FindBugs können gefundenen Bugs über einen eindeutigen Wert identifiziert werden. Diese ID wird als der sogenannte `InstanceHash` bzw. die `FINDBUGS_UNIQUE_ID` bezeichnet.

Ausgehend von einer Liste von Annotationen des Bugs, werden die Hash-Werte dieser Annotationen zu einem `InstanceKey` durch Doppelpunkte getrennt aneinandergefügt. Der MD5-Hash des `InstanceKey` repräsentiert den `InstanceHash`. Zu den Annotationen gehören unter anderem die Klasse, Methode, Felder sowie ein Zeilenbereich in dem der Bug auftritt.

Falls Änderungen am Quellcode durchgeführt wurden und sich der Hash nach einer erneuten FindBugs-Analyse ändert, kann der alte `InstanceHash` aus dem Feld `oldInstanceHash` zurückgewonnen werden. Nach einer kurzen Überprüfung wird der alte InstanceHash zur Zeit nicht gespeichert.

[4] In git ist ein Tag einfach nur ein Zeiger auf einen bestimmten Commit. Also genauso wie für einen Branch nur ohne die automatische Änderung des Zeigerziels.

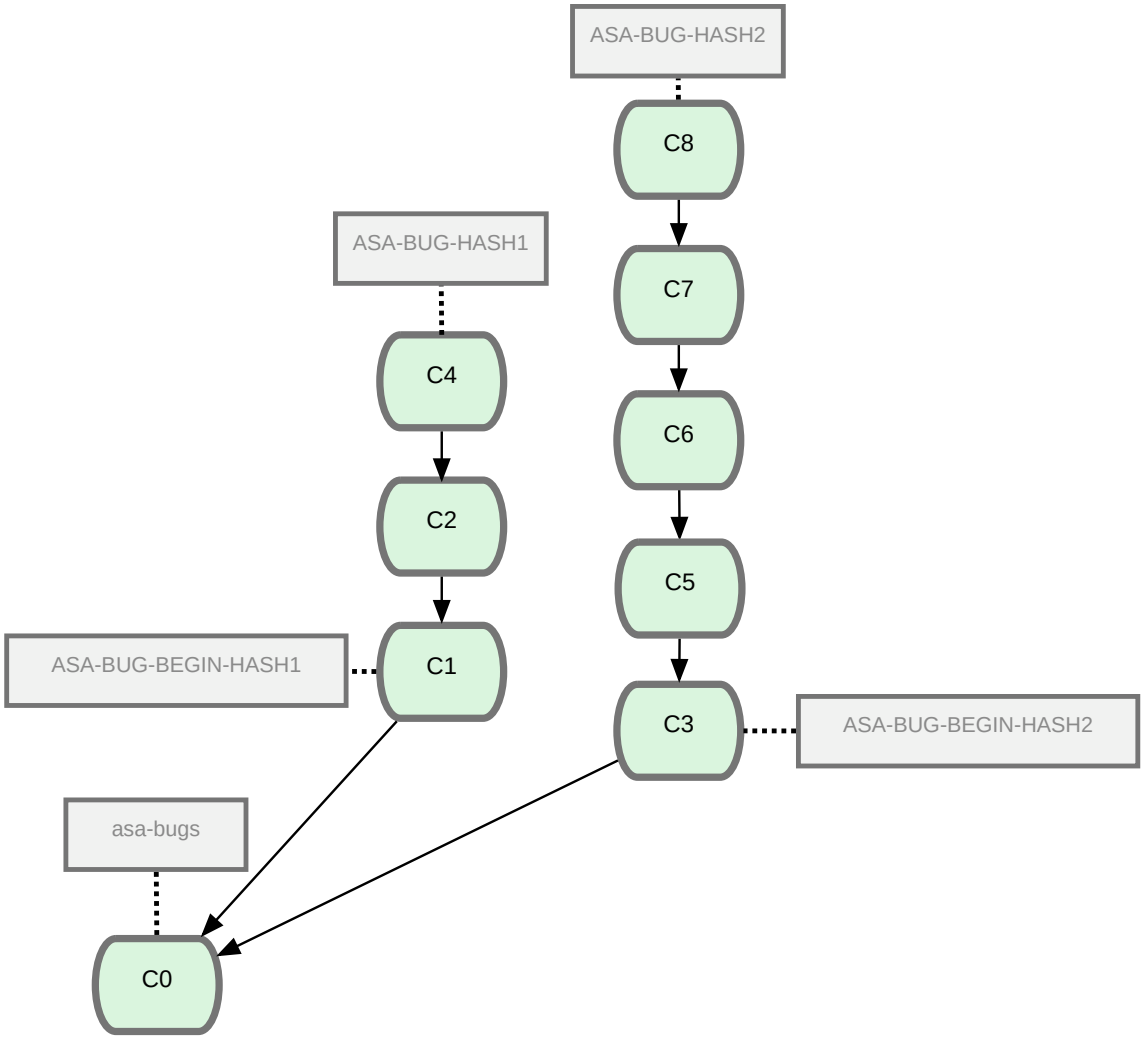


ABBILDUNG 4.1: Beispiel für die Repräsentation und Realisierung mit Branches und Tags für Bug Kommentare. Die Commits C1-C8 enthalten dabei jeweils das Kommentar zum Bug als Commit-Nachricht.

4.6 Eclipse-Plugin Implementierung

4.6.1 Aufbau

Genauso wie der FindBugs Issue Export ist diese Erweiterung auch als Eclipse Plugin realisiert. In diesem Fall verfügt sie aber über eine umfangreiche Benutzeroberfläche (Graphical User Interface (GUI)) die als *ViewPart* in Eclipse integriert ist und den Inhalt entsprechend dem aktuell markierten Bug anzeigt. Diese stellt den Bug, seine Beschreibung, bisherige Kommentare sowie ein Eingabefeld zum Verfassen eines eigenen Kommentars dar. Dafür wird auf einige Bibliotheken wie JavaFX und weitere (sieheUnterabschnitt 4.6.2) zurückgegriffen. Die ausführlichere Beschreibung folgt in Abschnitt 4.8. Im Weiteren ist die Erweiterung nach dem Model View Controller (MVC)-Prinzip aufgebaut. Dadurch werden die Daten, die Darstellung der Daten und die Programmlogik voneinander getrennt. So ist es möglich diese jeweils

unabhängig voneinander weiterentwickeln und verwenden zu können. Eine Übersicht der wichtigsten Models, Views und des dafür verwendeten Controllers findet sich in Abbildung 4.2.

Im *IssueModel* werden neben allgemeinen Informationen zum Issue, wie zum Beispiel der Bug-ID, auch eine Liste von Kommentaren sowie die Beschreibung gespeichert. Dabei wird jeweils eine Instanz des entsprechenden Models erwartet. Zur Darstellung übergibt der *IssueViewController* dem *IssueView* das Datenmodell in Form einer Instanz der Klasse *IssueModel*. So kann der *IssueView* einfach auf die Daten zugreifen, ohne über die konkrete Implementierung Bescheid wissen zu müssen. Weiterhin verwendet der *IssueView* zur weiteren Unterteilung weiter spezialisierte Views. Dazu gehören der *CommentListView*, der *CommentInputView*, der *DescriptionView* sowie der *DescriptionEditView*. Diese erweitern entsprechende JavaFX-Layout-Komponenten um sie im *IssueView* einfach verwenden zu können. Dabei greifen sie auf die entsprechenden Daten auch über die entsprechenden Models zu.

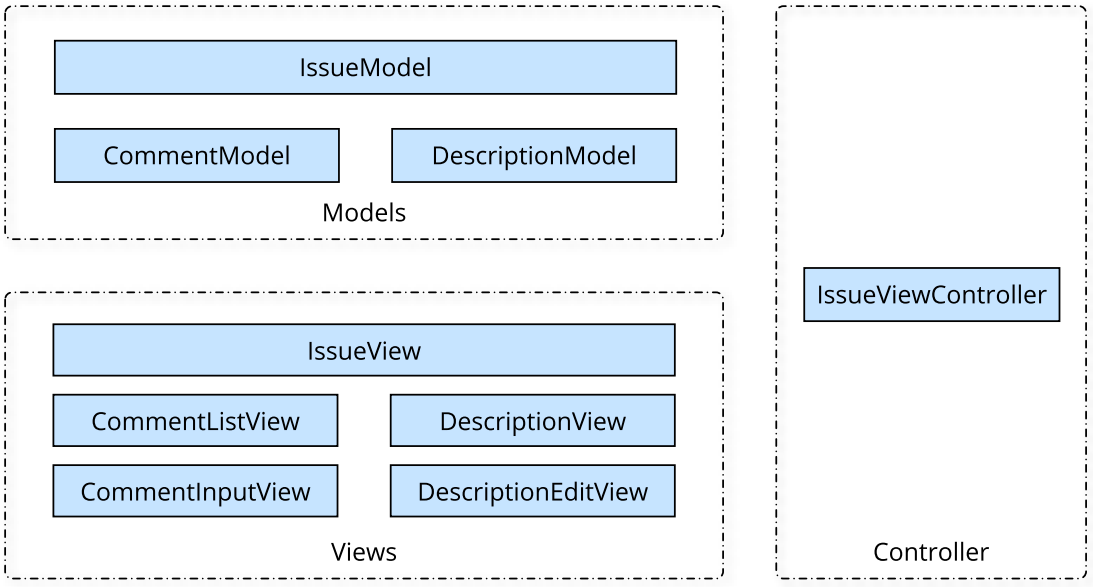


ABBILDUNG 4.2: Darstellung der vom FindBugs Kommentarsystem verwendeten MVC-Architektur.

4.6.2 Software-Abhängigkeiten

Aufgrund komplexer Konfigurationsvorgänge ist das Plugin zurzeit noch ohne eine komplett automatisierte Abhängigkeitskonfiguration wie sie beispielsweise Tycho[5] bieten würde. Deshalb gibt es nun zwei Arten von Abhängigkeiten für das Eclipse-Plugin. Einerseits die Abhängigkeiten die über Eclipse aufgelöst werden können und externe Abhängigkeiten.

Wichtige Eclipse-Abhängigkeiten sind:

- **JGit**: Zum Zugriff auf die git-Repositories und die Kapselung von verschiedenen git-Befehlen in einfach zu verwendende Java-Methoden.

[5] `http://www.eclipse.org/tycho/` (Besucht am 24.11.2014)

- **FindBugs Plugin**: Zur Analyse mit FindBugs sowie für den Zugriff auf die Ergebnisse und einzelne Felder der gefundenen möglichen Fehler.
- **JavaFX**: Für die Erweiterung von FindBugs werden JavaFX Komponenten verwendet. Diese sind für eine auf Hypertext Markup Language (HTML) und Cascading Style Sheets (CSS) basierenden, modernen und geräteunabhängigen Gestaltung von Benutzerschnittstellen konzipiert.
- **e(fx)clipse**: Um die JavaFX Komponenten bei der Entwicklung von Eclipse Plugins verwenden zu können.
- **Apache log4j**: Für Debugging- und Logging-Zwecke.

Weitere nicht über Eclipse alleine lösbare Abhängigkeiten werden durch die folgenden Bibliotheken, welche aktuell als *jar*-Dateien mitgeliefert werden, bereitgestellt:

- **txtmark**[6]: Markdown-zu-HTML-Konverter für die Aufbereitung der mit Markdown formatierten Kommentare und `DESCRIPTION.md`-Datei.
- **github-markdown-css**[7]: Zur visuellen Formatierung des erzeugten HTML. Ohne diese CSS-Angaben würden die HTML-Texte nur wenig hervorgehoben präsentiert werden.
- **PrettyTime**[8]: Zur Darstellung von Datum und Uhrzeiten als vergangene Zeit statt dem Zeitpunkt. Beispielsweise wird damit *vor zwei Minuten* statt *06.02.14 18:03:65 Uhr* ausgegeben.

4.6.3 JavaFX und Lambdas

Durch die Verwendung von JavaFX *Property* in den Eingabefeldern (*TextArea*) kann durch folgenden bequemen Code jeweils das Live-Preview des mit Markdown hervorgehobenen Texts realisiert werden:

```
1 this.textArea.get().textProperty().addListener((observable, oldValu, newValue) -> {
2     resizingWebView.get().setContent(
3         String.format(htmlFrame, cssString, String.format("<div id=\"bug-description
      \" class=\"markdown-body\">%s</div>", Processor.process(newValue))));
4 });
```

Dabei wird über die *textProperty*-Methode das entsprechende *Property* für den Textinhalt der *TextArea* geholt und im Anschluss ein *Listener* hinzugefügt. Dieser wird ausgeführt, wenn der Inhalt des Textfelds geändert wird. Das Konstrukt `() -> { }` wird als Lambda-Ausdruck bezeichnet. In diesem Fall muss keine extra benannte Funktion erstellt werden, sondern es kann mit der anonymen, namenslosen und *inline* definierten Funktion gearbeitet werden, die bei einer Änderung dann ausgeführt wird.

[6] `https://github.com/rjeschke/txtmark` (Besucht am 27.11.2014)
[7] `https://github.com/sindresorhus/github-markdown-css` (Besucht am 24.11.2014)
[8] `http://ocpsoft.org/prettytime/` (Besucht am 24.11.2014)

4.6.4 PrettyTime

Um für Benutzer schnell ablesbare Daten und Uhrzeiten darzustellen, wird die PrettyTime-Bibliothek verwendet. Mit dieser ist es möglich, als Java-*Date* gegebene Zeitpunkte in Text umzuwandeln. Benutzt wird sie wie folgt:

```
1  PrettyTime p = new PrettyTime();
2  String labelText = p.format(new Date((long) commit.getCommitTime() * 1000));
```

Nachdem bis hier einige Details der Implementierung beschrieben wurden, werden als nächstes weitere Anforderungen genannt. Danach folgt eine Beschreibung der Benutzeroberfläche.

4.7 User-Stories

4.7.1 Start

„Als Benutzer möchte ich, dass die Software bei Bedarf das Repository automatisch um die erforderlichen Strukturen ergänzt."

„Als Benutzer möchte ich, dass die Software automatisch erkennt, ob mein Projekt bereits über ein anderes Bug-Tracking-System verwaltet wird um nicht an zwei Stellen Bugs verwalten zu müssen."

Um diese beiden Anforderungen umzusetzen, soll folgender Ablauf beim Ausführen der Erweiterung stattfinden:

Beim Start der Anwendung soll überprüft werden ob die Branch-,Tag- und Ordner-Struktur bereits existiert. Wenn nicht, soll zunächst überprüft werden, ob das Projekt auf einer Plattform verwaltet wird, die einen Bug-Tracker inkl. Kommentarfunktion bereits verwendet und falls das der Fall ist, vorschlägt, dass diese Plattform auch für die Diskussion der gefundenen Bugs verwendet wird. Wenn keine solche Plattform verwendet wird, soll das Repository zum Bug-Tracken vorbereitet werden. Dafür können ähnliche Überprüfungen wie aus dem, in Kapitel 3 vorgestellten, FindBugs Issue Export Plugin verwendet werden.

Der weitere Ablauf für den Start in Form eines Business Process Model and Notation (BPMN)-Diagramms in Abbildung 4.3 dargestellt.

4.7.2 Internetzugriff

„Als Benutzer möchte ich, dass die Anwendung mich frägt bevor sie auf das Internet zugreift bzw. sensible Informationen darüber versendet."

Es wird davon ausgegangen, dass im betroffenen Projekt nur remote-Angaben für Repositories eingetragen sind, welche auch ohne Nachfrage benutzt werden dürfen. Deshalb findet keine weitere Abfrage statt.

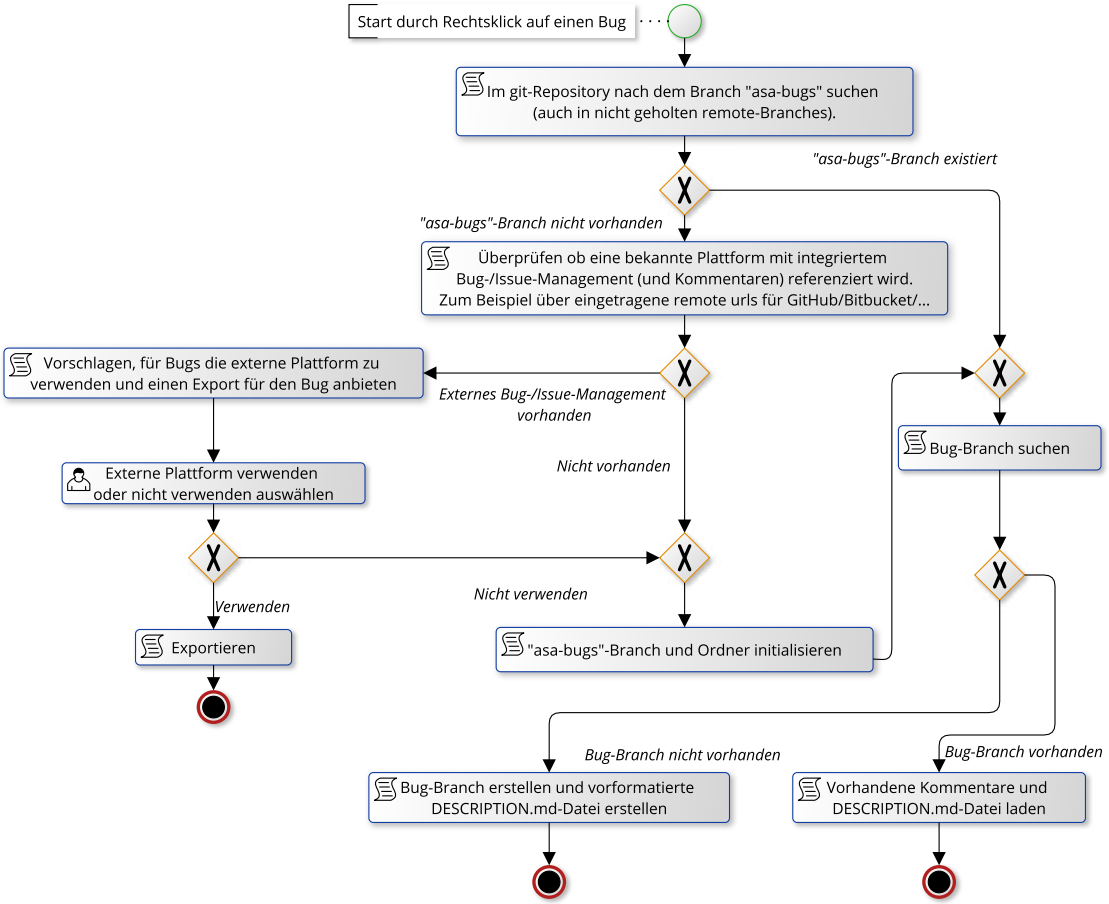


ABBILDUNG 4.3: BPMN-Diagramm für die Aktivitäten beim Start.

4.7.3 Vorformatierte Beschreibung

„Wenn ich als Benutzer einen neuen Bug kommentieren und beschreiben will, soll mir eine vorformatierte Beschreibung auf Basis der FindBugs-Erkenntnisse vorgeschlagen werden."

Eine auf den FindBugs-Erkenntnissen basierende Beschreibung im Markdown-Format wurde bereits im FindBugs Issue Export Plugin realisiert. Diese Methode kann auch hier, so wie in Abbildung 3.2 abgebildet, implementiert werden.

4.7.4 Nur kommentierte Bugs tracken

„Als Benutzer möchte ich, dass nur für Bugs die kommentiert werden/sind, Einträge und Branches im Repository erstellt werden."

Dies wird von der Erweiterung so umgesetzt, dass für noch nicht beschriebene und/oder kommentierte Bugs nur eine vorformatierte Beschreibung angezeigt wird, die der Benutzer bei Bedarf speichern kann. Erst dann wird für den Bug ein Eintrag im Repository angelegt.

4.7.5 Bearbeiten/Löschen von Kommentaren

„Als Benutzer möchte ich die Möglichkeit haben meine Beiträge bearbeiten und löschen zu können.“

Durch die Wahl von git als Back-End ist es nicht vorgesehen, beliebige Kommentare nachträglich bearbeiten zu können. Für den letzten Kommentar (Commit) ohne, dass jemand anderes einen nachfolgenden Kommentar getätigt hat, ist es möglich den Kommentar zu bearbeiten. Hier für kann zum Beispiel `git commit –ammend && git push upstream asa-bug-ID –force-with-lease` und `git rebase` verwendet werden. Falls jemand einen folgenden Kommentar getätigt hat, ist es nicht mehr möglich vorhergehende Kommentare einfach zu bearbeiten, da sich dadurch folgende Commits ändern würden. Grundsätzlich wäre es aber auch mit git möglich. Durch eine Änderung der Commits müssten alle anderen Beteiligten ihr Repository zurücksetzen um den neuen Stand zu akzeptieren. Dies wird nicht automatisiert unterstützt.

4.7.6 Bearbeiten der DESCRIPTION.md-Datei

„Als Benutzer möchte ich nicht nur Kommentare abgeben können, sondern auch Ergänzungen an der DESCRIPTION.md-Datei durchführen können.“

Durch den *DescriptionEditView* ist es nach dem Bestätigen eines Buttons für Benutzer einfach, die Beschreibung des Bugs inklusive einer Vorschau der gerenderten Beschreibung bearbeiten zu können.

4.7.7 Doppelte Kommentare

„Es soll unterbunden werden, dass man einen weiteren Kommentar veröffentlicht, wenn man selber den letzten Kommentar veröffentlich hat und diesen stattdessen bearbeiten könnte.“

Diese Anforderung wird bis jetzt noch nicht durch die Erweiterung umgesetzt.

4.7.8 Vereinigen von zwei Bugs inkl. zugehöriger Kommentare/Dateien

„Als Benutzer möchte ich, wenn ich erkenne, dass zwei verschiedene gefundene Bugs eigentlich ein und denselben Bug beschreiben, die Möglichkeit haben, diese Zusammenführen zu können.“

Diese Anforderung wird bis jetzt noch nicht durch die Erweiterung umgesetzt.

4.7.9 Export einzelner Bugs zum Bug-Tracker

„Falls mein Projekt bei GitHub oder einer anderen Plattform mit Bug-Tracker inkl. Kommentarfunktion verwaltet wird, möchte ich, dass ich einen Bug dorthin exportieren bzw. dort melden kann.“

Hierfür wurde im vorherigen Kapitel 3 bereits einen Ansatz vorgestellt, der dafür verwendet werden kann.

4.7.10 Bezeichner und (Vor-)Einstellungen

„Als Benutzer möchte ich die Möglichkeit haben, festzulegen, wie die Branches/Tags und Ordner bezeichnet werden. Diese Einstellungen sollen für ein Projekt gespeichert werden können, damit andere Benutzer auch diese Einstellungen verwenden."

Hierfür sollen in der Projekt-Konfiguration Einstellungen festgelegt werden können. Diese Einstellungen dürfen aber nach der ersten Verwendung nicht mehr geändert werden, außer es wird eine Migrationslogik eingebaut, die entsprechend anders bezeichnete Objekte umbenennt.

4.8 Benutzeroberfläche

Die Benutzeroberfläche, siehe Abbildung 4.4, basiert auf JavaFX-Komponenten und ist in einen *ViewPart* eingebettet. Die Benutzeroberfläche ist in 3 Teile aufgeteilt. Im obersten Bereich wird die Beschreibung des Bugs, hervorgehoben mit Markdown, dargestellt. Weiterhin wird der Bereich zum Bearbeiten in zwei Teile geteilt. Dann kann man im linken Bereich, so wie in der Abbildung 4.4 gerade dargestellt, die Beschreibung bearbeiten und im rechten Teil wird einem eine Vorschau der mit Markdown hervorgehobenen Beschreibung angezeigt. Das gleiche Prinzip ist im unteren Bereich der Benutzeroberfläche für das Abgeben eines Kommentars realisiert. Im mittleren Teil werden die Kommentare, auch mit Markdown hervorgehoben, dargestellt. Der dargestellte Protop ist in der Version in einem eigenen Fenster realisiert. In der Anwendung befindet sich die Darstellung komplett in einem *ViewPart* um dem *Look-and-Feel* von Eclipse zu entsprechen. Als *ViewPart* kann ein Benutzer diesen auch beliebig innerhalb von Eclipse positionieren, vergrößern und verkleinern.

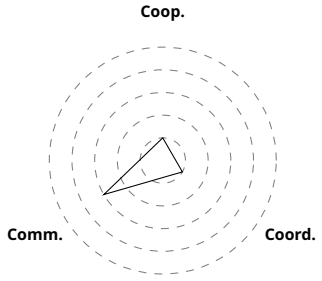
Weiterhin verhält sich die Anzeige so, dass immer die Informationen, also Beschreibung und eventuell vorhandene Kommentare, zu dem Bug angezeigt werden, der aktuell im *Bug Explorer* markiert ist. Dieses Verhalten entspricht damit dem des vorhandenen *Bug Info* Views. Für eine Darstellung der kompletten Eclipse GUI befindet sich ein Screenshot in Abbildung A.2 im Anhang.

ABBILDUNG 4.4: GUI des FindBugs Kommentarsystems. Diese Ansicht zeigt sowohl die Bearbeitungsmöglichkeit für die Beschreibung als auch die für einen Kommentar. In beiden Fällen werden *HBox*-Layouts verwendet, um rechts daneben eine Vorschau angezeigt zu bekommen, wie die Beschreibung bzw. das Kommentar nachher hervorgehoben aussehen wird.

4.9 Fazit

4.9.1 Bewertung

Es ist unmittelbar klar, dass die Erweiterung durch die Kommentare für Kommunikation sorgen kann (KommQ1:4P). Ebenso wurde durch die Verwendung von PrettyTime und Markdown dafür gesorgt, dass die Metadaten zur Kommunikation entsprechend angeordnet und formatiert sind (KommQ2:4P). Weiterhin ermöglichen sie damit auch eine schnelle Erfassung des Inhalts (KommQ3:4P). In der bis jetzt implementierten Version befindet sich noch keine Interaktionsmöglichkeit um die Entwickler kontaktieren zu können oder direkter mit ihnen kommunizieren zu können (KommQ4:0P, KommQ5:0P). Durch die Integration mit git und die inhärente Auslegung für mehrere Benutzer kann die Erweiterung auch für die Kooperation im weitesten Sinne genutzt werden (KoopQ1:2P). Dennoch wird die Kooperation nicht weiter gefördert, als es durch die indirekte Kommunikation möglich ist (KoopQ2:0P, KoopQ3:0P, KoopQ4:0P, KoopQ5:0P). Im Unterschied zur FindBugs Cloud gibt es im Moment bei dieser Erweiterung keine Möglichkeit, außer durch explizite Nennung in der Beschreibung, für eine Koordination mittels Klassifikationsfunktion oder ähnlichem (KoorQ1:0P, KoorQ2:0P, KoorQ3:0P, KoorQ4:0P, KoorQ5:0P).



FindBugs Kommentarsystem

Integrierte Kommentare für FindBugs Bugs

	Q1	Q2	Q3	Q4	Q5	Summe
Kommunikation	4	4	4	0	0	12: **Level 3**
Kooperation	2	0	0	0	0	2: **Level 1**
Koordination	0	0	0	0	0	0: **Level 1**

4.9.2 Ausblick

Die vorgestellte Implementierung sollte als nächstes bezüglich ihres Mehrwerts für Benutzer evaluiert werden. Dabei sollte auch auf die allgemeinen Anforderungen eingegangen werden und untersucht werden, ob diese zufriedenstellend umgesetzt wurden. Insbesondere wegen der schwer änderbaren und schwer löschbaren Kommentare, werden hier negative Rückmeldungen erwartet. Ausgehend von den Ergebnissen der Evaluation gilt es dann zu Beurteilen, ob eine Datei-basierte Speicherung der Kommentare sinnvoller ist. Dabei sollte dann komplett auf die Speicherung in den git-commits verzichtet werden. Andererseits ist es auch möglich, komplett auf die Speicherung von Dateien zu verzichten und alle Daten, also auch die Beschreibung, welche jetzt noch als `DESCRIPTION.md`-Datei gespeichert ist, in der Commit-Nachricht unterzubringen. Dafür könnte ein einfaches JSON-basiertes Format verwendet werden. Dadurch wäre die Implementierung sehr viel konsequenter als durch die aktuelle Kombination von beiden Speicherarten.

Allgemein sollte die Implementierung um weitere, für die Benutzer mit Interaktion belegten, Kontaktmöglichkeiten versehen werden sowie die Benutzererfahrung weiter verbessert werden. Es gibt noch viele weitere Möglichkeiten die ASA-Programme sozialer und benutzerfreundlicher zu gestalten. Dazu könnte auch die Integration einer automatischen Zuweisung von, vom aktuellen Benutzer wahrscheinlich behehbaren, Bugs zählen, wie sie in [32] beschrieben wird. Ebenfalls kann eine bessere Visualisierung der Programmabläufe, im Zusammenhang mit den Fehlerbeschreibungen, für einen effizienteren Umgang mit den gefundenen Bugs sorgen [33].

Kapitel 5

Fazit

Beginnend mit dem Kapitel 1 wurden in dieser Masterarbeit zunächst wichtige verwandte Arbeiten wie „The (R) Evolution of Social Media in Software Engineering“ [1] und „Why Don’t Software Developers Use Static Analysis Tools to Find Bugs?“ [9] vorgestellt um einen Einstieg in den Themenbereich des Social Codings sowie der in der Arbeit zu erweiternden Automatische Statische Code-Analyse (ASA) zu bieten. Danach sind die gerade genannten Themen detailliert beschrieben und insbesondere eine Definition für Social Coding hergeleitet worden. Mit der in Abbildung 1.2 dargestellten Visualisierung wurde weiter dafür gesorgt, den Begriff Social Coding schnell einordnen zu können.

Im weiteren Verlauf wurden danach verschiedene Ansätze für Social Coding analysiert und in die drei Kategorien, Kommunikation, Kooperation und Koordination des 3C-Modells sowie nach der grundlegenden Art des Ansatzes eingeteilt. Anhand der hier untersuchten Plattformen und Softwares haben sich dabei fünf grundsätzliche Ansätze für Social Coding herauskristallisiert. Dazu gehören Gleichzeitiges Editieren bzw. Live Editing als Erweiterung einer IDE, Integration von Diskussion oder Kommunikationsmöglichkeiten im Allgemeinen in eine IDE oder anderer Programme für den Umgang mit Quellcode, Online-Plattformen zur Diskussion und Online-Plattformen mit aggregierenden Funktionen und Profilen zur Vernetzung der Entwickler sowie als fünftes das sogenannte Social Tagging mit dem Inhalte durch benutzergenerierte Begriffe oder Klassen von eben diesen Benutzern selbst eingeteilt werden können.

Bei der Einteilung in die drei Kategorien des 3C-Modells fiel auf, dass dies nicht einfach ist und, trotz des in Kapitel 2 aufgelisteten Fragenkatalogs, nicht immer zufriedenstellende Ergebnisse lieferte. Dies war besonders bei der Bewertung der ersten Implementierung der Fall. Diese erweitert das Eclipse FindBugs Plugin um eine Export-Möglichkeit für Bugs um diese auf verwendeten externen Bug-Trackern bei Social Coding Plattformen wie GitHub einfach melden zu können. Der Fragenkatalog bezieht sich zwar auf die Erweiterung, aber nicht auf die davon verwendeten und damit integrierten Plattformen. Diese spielen jedoch in diesem Fall eine große Rolle und sollten bei zukünftigen Bewertungssystemen miteinbezogen werden.

Neben der Export-Erweiterung wurde im Kapitel 4 eine weitere Implementierung vorgestellt. Mit dem dort eingeführten Kommentarsystem wurde auch wieder das Eclipse FindBugs Plugin um die Möglichkeit erweitert, Bugs mit eigenen Beschreibungen im Markdown-Format zu

versehen, sowie Kommentare zu den Bugs abgeben zu können. Dabei wurde der so erstellte Bug-Tracker in das Quellcode-Repository des Projekts integriert, um von der schon vorhandenen Rechteverwaltung, der Dezentralität und der Verteilung durch git zu profitieren. Bei der Implementierung wurde weiterhin ein besonderer Ansatz gewählt: Kommentare zu Bugs werden in den Commit-Nachrichten von git gespeichert. Damit sind leider mehr Nachteile als Vorteile verbunden, was voraussichtlich auch von einer noch durchzuführenden Evaluation gezeigt werden würde. Deshalb sollen nach einer Evaluation und Beachtung weiterer Rückmeldungen die Kommentare und weitere von git bisher automatisch erfassten, mit dem Commit gespeicherten, Metadaten, in Dateien gespeichert werden.

Insgesamt kann nun auf zwei nützliche Erweiterungen für das ASA-Programm FindBugs zurückgegriffen werden. Diese sind in Zukunft, wie schon beschrieben, noch zu evaluieren und weiter auszubauen. Weiterhin gilt es in der Zukunft andere ASA-Programme für andere Programmiersprachen um ähnliche Ansätze zu erweitern, um auch dort die gefundenen Fehler gemeinsam besser verstehen, beurteilen und dann beheben zu können.

Bei der Implementierung von Social Coding muss man sich bewusst machen, dass es eine sehr große Spanne zwischen einfachen und sehr komplexen Umsetzungen gibt. Dies hängt vom zu erreichenden Ziel, dem Vorgehen und von den verwendeten und integrierten Technologien ab. Abschließend kann man festhalten, dass bei Social Coding die Benutzererfahrung eine sehr große Rolle spielt, da sie der entscheidende Faktor für die freiwillige Benutzung, das Teilen und das Weitersagen ist und damit für den Erfolg des Ansatzes sorgt.

Abbildungsverzeichnis

Abkürzungsverzeichnis

API Application Programmable Interface

ASA Automatische Statische Code-Analyse

BPMN Business Process Model and Notation

CSCW Computer Supported Cooperative Work

CSS Cascading Style Sheets

DSD Distributed Software Development

DSN Developers Social Network

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

HTML Hypertext Markup Language

IDE Integrierte Entwicklungsumgebung

JSON JavaScript Object Notation

MVC Model View Controller

SoSEA Social Software Engineering and Applications

Splint Secure Programming Lint

SSE Social Software Engineering

SSH Secure Shell

VNC Virtual Network Computing

XAML Extensible Application Markup Language

Anhang A

Anhang

Name	Typ/Art/Besonderes	Kommunikation	Kooperation	Koordination
Stack Overflow	Online-Plattform zur Diskussion	+ + + +	+ +	+
Coderwall	Entwickler-Blog	+ + + +	+	+
Masterbranch	Automatische Profilerstellung	+	+	+
CodeProject	Kombination aus Blog, Forum und Q&A-Plattform	+ + + +	+	+
Cloud9	Online-IDE mit Live-Editing	+ + + +	+ + + + +	+
GitHub Bug-Tracker	Markdown, Verlinkung und Benachrichtigungen	+ + + +	+ +	+ + + +
Visual Studio CodeLens	Inline-Informationen	+	+	+
Visual Studio Anywhere	Live-Editing und Chat	+ + + +	+ + + + +	+
GitLab Bug-Tracker	Markdown, Verlinkung und Benachrichtigungen	+ + + +	+ +	+ + + +
FindBugs Cloud	Kommentare für FindBugs Bugs	+ +	+	+ + +
FindBugs Issue Export	Melde-Möglichkeit für mögliche Bugs	+	+	+
FindBugs Kommentarsystem	Kommentare für FindBugs Bugs	+++	+	+

ABBILDUNG A.1: Zusammenfassende Tabelle der Ergebnisse aus der Analyse Kapitel 2. Die Level wurden wie folgt visualisiert: Level 1 → **+**, Level 2 → **+ +**, Level 3 → **+ + +**, Level 4 → **+ + + +**, Level 5 → **+ + + + +**

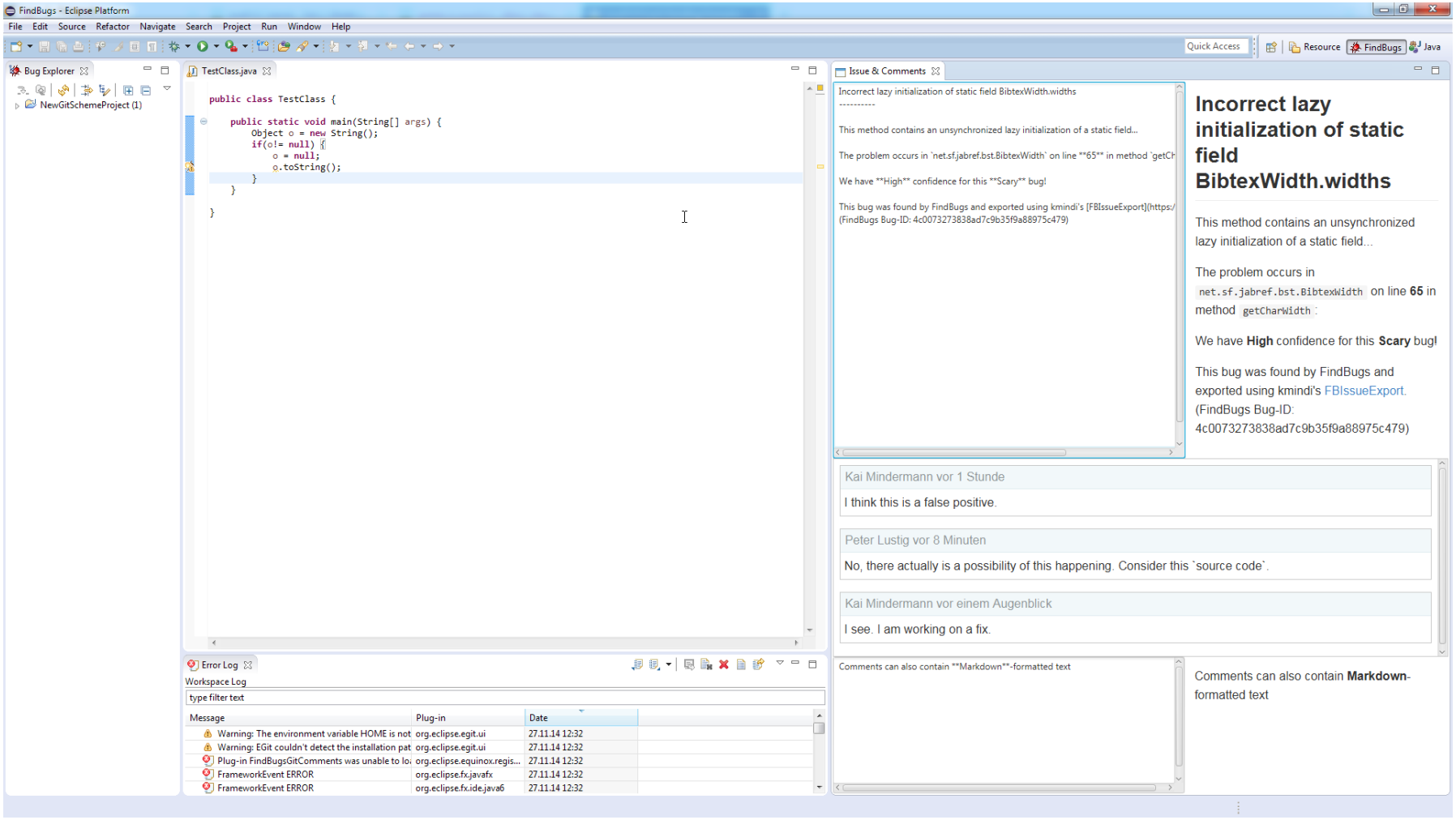


ABBILDUNG A.2: Integration des Kommentarsystems in die Eclipse GUI.

Literatur

[1] Margaret-Anne Storey et al. „The (R) Evolution of Social Media in Software Engineering“. In: *Proceedings of the on Future of Software Engineering*. FOSE 2014. Hyderabad, India: ACM, 2014, S. 100–116. ISBN: 978-1-4503-2865-4. DOI: 10.1145/2593882.2593887. URL: http://doi.acm.org/10.1145/2593882.2593887.

[2] Anja Guzzi et al. „Communication in Open Source Software Development Mailing Lists“. In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. MSR '13. San Francisco, CA, USA: IEEE Press, 2013, S. 277–286. ISBN: 978-1-4673-2936-1. URL: http://dl.acm.org/citation.cfm?id=2487085.2487139.

[3] Carl Gutwin, Reagan Penner und Kevin Schneider. „Group Awareness in Distributed Software Development“. In: *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*. CSCW '04. Chicago, Illinois, USA: ACM, 2004, S. 72–81. ISBN: 1-58113-810-5. DOI: 10.1145/1031607.1031621. URL: http://doi.acm.org/10.1145/1031607.1031621.

[4] Raphael Pham et al. „Creating a Shared Understanding of Testing Culture on a Social Coding Site“. In: *Proceedings of the 2013 International Conference on Software Engineering*. ICSE '13. San Francisco, CA, USA: IEEE Press, 2013, S. 112–121. ISBN: 978-1-4673-3076-3. URL: http://dl.acm.org/citation.cfm?id=2486788.2486804.

[5] Laura Dabbish et al. „Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository“. In: *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*. CSCW '12. Seattle, Washington, USA: ACM, 2012, S. 1277–1286. ISBN: 978-1-4503-1086-4. DOI: 10.1145/2145204.2145396. URL: http://doi.acm.org/10.1145/2145204.2145396.

[6] Michael Chui et al. *The social economy: Unlocking value and productivity through social technologies*. McKinsey Global Institute, Juli 2012. ISBN: 978-0985564711.

[7] Jason Tsay, Laura Dabbish und James Herbsleb. „Influence of Social and Technical Factors for Evaluating Contribution in GitHub“. In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. Hyderabad, India: ACM, 2014, S. 356–366. ISBN: 978-1-4503-2756-5. DOI: 10.1145/2568225.2568315. URL: http://doi.acm.org/10.1145/2568225.2568315.

[8] Amit Kumar und Avdhesh Gupta. „Evolution of Developer Social Network and Its Impact on Bug Fixing Process“. In: *Proceedings of the 6th India Software Engineering Conference*. ISEC '13. New Delhi, India: ACM, 2013, S. 63–72. ISBN: 978-1-4503-1987-4. DOI: 10.1145/2442754.2442764. URL: http://doi.acm.org/10.1145/2442754.2442764.

[9] Brittany Johnson et al. „Why Don't Software Developers Use Static Analysis Tools to Find Bugs?" In: *Proceedings of the 2013 International Conference on Software Engineering*. ICSE '13. San Francisco, CA, USA: IEEE Press, 2013, S. 672–681. ISBN: 978-1-4673-3076-3. URL: http://dl.acm.org/citation.cfm?id=2486788.2486877.

[10] Nathaniel Ayewah und William Pugh. „A Report on a Survey and Study of Static Analysis Users". In: *Proceedings of the 2008 Workshop on Defects in Large Software Systems*. DEFECTS '08. Seattle, Washington: ACM, 2008, S. 1–5. ISBN: 978-1-60558-051-7. DOI: 10.1145/1390817.1390819. URL: http://doi.acm.org/10.1145/1390817.1390819.

[11] Masao Ohira und Hayato Yoshiyuki. „A New Perspective on the Socialness in Bug Triaging: A Case Study of the Eclipse Platform Project". In: *Proceedings of the 2013 International Workshop on Social Software Engineering*. SSE 2013. Saint Petersburg, Russia: ACM, 2013, S. 29–32. ISBN: 978-1-4503-2313-0. DOI: 10.1145/2501535.2501542. URL: http://doi.acm.org/10.1145/2501535.2501542.

[12] Igor Steinmacher, Ana Paula Chaves und Marco Aurélio Gerosa. „Awareness Support in Distributed Software Development: A Systematic Review and Mapping of the Literature". In: *Comput. Supported Coop. Work* 22.2-3 (Apr. 2013), S. 113–158. ISSN: 0925-9724. DOI: 10.1007/s10606-012-9164-4. URL: http://dx.doi.org/10.1007/s10606-012-9164-4.

[13] Wikipedia. *Computer Supported Cooperative Work — Wikipedia, Die freie Enzyklopädie*. 2014. URL: http://de.wikipedia.org/w/index.php?title=Computer_Supported_Cooperative_Work&oldid=130499464 (besucht am 05. 06. 2014).

[14] Clarence A. Ellis, Simon J. Gibbs und Gail Rein. „Groupware: Some Issues and Experiences". In: *Commun. ACM* 34.1 (Jan. 1991), S. 39–58. ISSN: 0001-0782. DOI: 10.1145/99977.99987. URL: http://doi.acm.org/10.1145/99977.99987.

[15] Andreas M. Kaplan und Michael Haenlein. „Users of the world, unite! The challenges and opportunities of Social Media". In: *Business Horizons* 53.1 (2010), S. 59–68. ISSN: 0007-6813. DOI: 10.1016/j.bushor.2009.09.003. URL: http://www.sciencedirect.com/science/article/pii/S0007681309001232.

[16] Wikipedia. *Social Software — Wikipedia, Die freie Enzyklopädie*. 2014. URL: http://de.wikipedia.org/w/index.php?title=Social_Software&oldid=130614087 (besucht am 05. 06. 2014).

[17] Wikipedia. *Social software engineering — Wikipedia, The Free Encyclopedia*. 2013. URL: http://en.wikipedia.org/w/index.php?title=Social_software_engineering&oldid=540556011 (besucht am 27. 06. 2014).

[18] Jeffrey Warren. *Kogbox: social coding on the cloud*. URL: http://www.vestaldesign.com/blog/2007/12/kogbox-social-coding-on-the-cloud/ (besucht am 23. 11. 2014).

[19] Margaret Rouse. *What is social coding*. 2012. URL: http://whatis.techtarget.com/definition/social-coding (besucht am 04. 06. 2014).

[20] Flemming Nielson, Hanne R. Nielson und Chris Hankin. *Principles of Program Analysis*. Springer Publishing Company, Incorporated, 2010. ISBN: 3642084745, 9783642084744.

[21] Wikipedia. *List of tools for static code analysis — Wikipedia, Die freie Enzyklopädie*. 2014. URL: http://en.wikipedia.org/w/index.php?title=List_of_tools_for_static_code_analysis&oldid=620764872 (besucht am 12. 08. 2014).

[22] Bjarne Steensgaard. „Points-to Analysis in Almost Linear Time“. In: *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '96. St. Petersburg Beach, Florida, USA: ACM, 1996, S. 32–41. ISBN: 0-89791-769-3. DOI: 10.1145/237721.237727. URL: http://doi.acm.org/10.1145/237721.237727.

[23] Robert P. Wilson und Monica S. Lam. „Efficient Context-sensitive Pointer Analysis for C Programs“. In: *Proceedings of the ACM SIGPLAN 1995 Conference on Programming Language Design and Implementation*. PLDI '95. La Jolla, California, USA: ACM, 1995, S. 1–12. ISBN: 0-89791-697-2. DOI: 10.1145/207110.207111. URL: http://doi.acm.org/10.1145/207110.207111.

[24] Patrick Cousot und Radhia Cousot. „Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints“. In: *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL '77. Los Angeles, California: ACM, 1977, S. 238–252. DOI: 10.1145/512950.512973. URL: http://doi.acm.org/10.1145/512950.512973.

[25] Lena Mamykina et al. „Design Lessons from the Fastest Q&A Site in the West“. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, S. 2857–2866. ISBN: 978-1-4503-0228-9. DOI: 10.1145/1978942.1979366. URL: http://doi.acm.org/10.1145/1978942.1979366.

[26] John Gruber und Aaron Swartz. *Markdown*. 2012. URL: http://daringfireball.net/projects/markdown/ (besucht am 31. 07. 2014).

[27] GitHub Inc. *GitHub Flavored Markdown*. 2012. URL: https://help.github.com/articles/github-flavored-markdown (besucht am 31. 07. 2014).

[28] Ching-Yung Lin et al. „SmallBlue: Social Network Analysis for Expertise Search and Collective Intelligence“. In: *Proceedings of the 2009 IEEE International Conference on Data Engineering*. ICDE '09. Washington, DC, USA: IEEE Computer Society, 2009, S. 1483–1486. ISBN: 978-0-7695-3545-6. DOI: 10.1109/ICDE.2009.140. URL: http://dx.doi.org/10.1109/ICDE.2009.140.

[29] Li-Te Cheng et al. „Jazzing Up Eclipse with Collaborative Tools“. In: *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*. eclipse '03. Anaheim, California: ACM, 2003, S. 45–49. DOI: 10.1145/965660.965670. URL: http://doi.acm.org/10.1145/965660.965670.

[30] Luca Ponzanelli et al. „Mining StackOverflow to Turn the IDE into a Self-confident Programming Prompter“. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, S. 102–111. ISBN: 978-1-4503-2863-0. DOI: 10.1145/2597073.2597077. URL: http://doi.acm.org/10.1145/2597073.2597077.

[31] Wiki Kernel.org. *Interfaces, frontends and tools - Git SCM Wiki*. 2014. URL: https://git.wiki.kernel.org/index.php/Interfaces,_frontends,_and_tools&oldid=30071#Bug.2Fissue_trackers.2C_etc. (besucht am 19. 09. 2014).

[32] Thomas Fritz et al. „Degree-of-knowledge: Modeling a Developer's Knowledge of Code“. In: *ACM Trans. Softw. Eng. Methodol.* 23.2 (Apr. 2014), 14:1–14:42. ISSN: 1049-331X. DOI: 10.1145/2512207. URL: http://doi.acm.org/10.1145/2512207.

[33] Yit Phang Khoo et al. „Path Projection for User-centered Static Analysis Tools“. In: *Proceedings of the 8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*. PASTE '08. Atlanta, Georgia: ACM, 2008, S. 57–63. ISBN: 978-1-60558-382-2. DOI: 10.1145/1512475.1512488. URL: http://doi.acm.org/10.1145/1512475.1512488.

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