Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,
Arbeitsgruppe Software Engineering

Eine bedarfsorientierte Statuseingabe für das Eclipse Plugin Saros

Maria Formisano

formisano@zedat.fu-berlin.de

Berlin, 07. August 2014

Erstgutachter: Prof. Dr. Lutz Prechelt

Zweitgutachterin: Prof. Dr. Elfriede Fehr

Betreuerin: Julia Schenk

Zusammenfassung

Um Software in einem zeitlichen und finanziellen Rahmen erfolgreich zu entwickeln, werden verschiedene agile Methoden angewandt, wie zum Beispiel die verteilte Paarprogrammierung. Sie gibt Entwicklern in großen Teams die Möglichkeit auch bei weitreichenden Distanzen gemeinsam zu arbeiten. Saros ist ein Eclipse Plugin, das die technischen Funktionalitäten für dieses Vorgehen liefert. Paarprogrammierung ist ein durch Kommunikation stark geprägter Prozess. Bei verteilter Arbeit funktioniert diese nur sehr schwerfällig. Je umfangreicher das Programm, desto höher ist der Komplexitätsgrad. Eintretende Problemsituationen lassen sich dabei kaum vermeiden.

Ziel dieser Arbeit ist es, Saros so zu verbessern, dass Teams in einer gemeinsamen verteilten Paarprogrammierungssitzung besser und häufiger miteinander kommunizieren können, ohne sich gegenseitig aus dem Arbeitsfluss zu reißen. Insbesondere während auftretender Probleme soll sich das Nutzen des Saros-Features auf den Prozess und gleichzeitig auf die Qualität der Zusammenarbeit positiv auswirken.

Die Anwendung von Awarenessinformationen ermöglicht verteilten Paarprogrammierern zu erkennen, wo sich der Partner im Projekt befindet und woran er arbeitet. An welchen Stellen fehlende Awarenessinformationen optimiert werden können, wurden anhand der aktuellen Saros Version evaluiert und in einem Ideenpool zusammengetragen. Aus den vorhandenen Vorschlägen habe ich die Idee eines Help Request Features konzeptioniert und implementiert. Das Konzept und die Entwicklung erläutere ich dafür ausführlich. Anschließend soll im Ausblick mithilfe eines prototypischen Ansatzes beschrieben werden, wie man die Nutzer intrinsisch und extrinsisch motivieren kann, dieses Saros-Feature einzusetzen. Schließlich beendet ein Fazit unter Berücksichtigung der hier erworbenen Ergebnisse die Bachelorarbeit.

1 Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

07. August 2014

Maria Formisano

Inhaltsverzeichnis

1 Eidesstattliche Erklärung **4**

2 Einleitung **9**

2.1 Forschungsidee 11

2.2 Motivation 11

3 Theoretische Grundlagen **13**

3.1 Was ist (verteilte) Paarprogrammierung? 13

3.2 Was ist Saros? 14

3.3 Was ist Awareness? 16

3.4 Was ist Gamification? 17

3.5 Was ist das Five − Factor − Model? 19

4 Vorüberlegungen **22**

4.1 Ideenentwicklung 22

4.1.1 Verlauf 22

4.1.2 Taskbeschreibung 23

4.1.3 Status in Kontaktliste 25

4.1.4 Programmausführung 25

4.1.5 Teaching 25

4.1.6 Chat 26

4.1.7 Profil(−bild) 26

4.1.8 Farbauswahl / Selektion auch im Package Explorer 27

4.1.9 Kommentare 27

4.2 Taskbeschreibung 28

4.3 Soziales Verhalten 31

5 Szenario **33**

6 Konzept und Implementierung **35**

6.1 Wireframes und Clickdummy 35

6.2 Implementierung 37

6.3 Help Request 38

6.4 Help Reply 39
6.5 Kontaktliste 39

7 Ausblick 40
7.1 Systemerkennung 40
7.2 Gamification 41
7.2.1 Ansatz für Saros 44

8 Fazit 47

9 Anhang 48
9.1 Mindmap: Ideensammlung zur verbesserten Kommunikation mithilfe von Awarenessinformationen 48
9.2 Mindmap: intrinsische und extrinsische Motivation 49

2 Einleitung

Hochwertige Software pünktlich und im Rahmen des Budgets abzuliefern, braucht sowohl einen gut geplanten Entwicklungsprozess als auch kompetente, teamfähige Programmierer. *Agile Softwareentwicklung* ist ein Ansatz, um die Arbeitsprozesse so flexibel, transparent und dynamisch wie möglich zu halten.

Eine immer häufiger verbreitete agile Methode ist die Paarprogrammierung, die den Entwicklungsprozess von Programmcode optimiert. Das daraus folgende Resultat ist eine sehr viel geringere Fehlerrate beim Programmieren, wodurch die Programme übersichtlicher werden und die Programmierer disziplinierter arbeiten. Dies wirkt sich dann auf die Qualität des Quellcodes aus und bewirkt einen positiven Effekt auf die Teambildung.

Das Prinzip der Paarprogrammierung ist schnell erklärt: Zwei Programmierer arbeiten nebeneinander an *einem* Arbeitsplatz vor *einem* Bildschirm und verfügen dabei über *eine* Maus und *eine* Tastatur. Einer der beiden Entwickler übernimmt die Rolle des Drivers, der die alleinigen Benutzungsrechte der Eingabegeräte hat, also aktiv programmiert, während der andere die Rolle des Observers ausübt. Seine Aufgabe ist es, den Driver über Fehler, Probleme und Alternativlösungen zu informieren, gleichzeitig aber nicht die Eingabegeräte benutzen darf. Hinzu kommt ein regelmäßiger Rollen- und eventueller Partnertausch. Das dient der Problemvorbeugung von Expertenkluster: Es können sich keine einzelnen Experten von Programmteilen bilden und die anderen Programmierer verlieren nicht den Überblick über das bereits Erschaffene.

Große Projekte erfordern oft große Teams. Damit sich diese aber an einen engen Zeitplan halten können, braucht das Team qualifizierte Entwickler. Immer häufiger sind diese aufgrund ihrer individuellen Qualifikationen und Fähigkeiten an verschiedenen Standorten verteilt, daher werden immer öfter Aufgabenbereiche auch ins Ausland verlagert. Dabei muss keine Rücksicht auf geografische Grenzen und weitgreifende Distanzen [JAC+08,] genommen werden, da die Globalisierung und der technische Fortschritt diese in bestimmten Aspekten verringern. Somit müssen Teammitglieder nicht zwingend nebeneinander oder gar in einem gemeinsamen Büro sitzen, um intensiv und produktiv zusammenarbeiten zu können. Folglich wird die Umsetzung der klassischen Paarprogrammierung, also das direkte beieinander Sitzen, nahezu unmöglich. Angesichts dessen hat sich die *verteilte* Paarprogrammierung entwickelt.

Die verteilte Paarprogrammierung liefert für dieses Szenario eine modifizierte Methode.

Der Unterschied ist, dass jeder Entwickler vor seinem eigenen Monitor mit seinen eigenen Eingabegeräte sitzt. Durch kollaborative Software kann die Entwicklungsumgebung synchronisiert werden, sodass jede an dem Programm beteiligte Partei in Echtzeit über den gleichen Quellcode verfügt. Mithilfe von Awarenessannotationen kann dann verfolgt und nachvollzogen werden, womit sich der andere Entwickler gerade beschäftigt und an welchem Punkt er sich befindet. Der Nachteil dabei ist, dass durch fehlende sogenannter Awarenessinformationen[1] die natürliche Form der Kommunikation gehemmt ist, der Entwicklungsprozess schwerfällig, und mühsam wird und der Arbeitsfluss dadurch häufig unterbrochen werden muss. Durch integrierte Werkzeuge, wie (Video-)Chat, Screensharing und VoIP ist es den Entwicklern möglich während der Durchführung zu kommunizieren, sie sind aber keine äquivalente Alternative zur natürlichen Form der Kommunikation.
Am Institut für Informatik an der Freien Universität Berlin wird das Eclipse Plugin Saros entwickelt. Saros ist ein Tool, welches Funktionen für die verteilte Paarprogrammierung bereitstellt. Durch einen geteilten Editor wird die Codebasis synchronisiert, d.h. alle Teilnehmer einer Programmiersitzung haben den gleichen Stand des Programms, sehen wo sich die Entwickler befinden und woran sie gerade arbeiten. So kann verteilte Paarprogrammierung umgesetzt werden.
Ungeachtet dessen, dass Saros ein Tool für verteilte Paarprogrammierung ist, können alle Teilnehmer einer Sitzung sowohl gemeinsam, als auch individuell und unabhängig voneinander programmieren („distributed party programming“), ohne durch feste Rollenverteilungen der klassischen Paarprogrammierung eingeschränkt zu sein. Ein dabei auftretendes Problem ist, dass bei mehreren Teilnehmern einer Sitzung schnell der Überblick über die aktuellen Aufgabenverteilungen verloren gehen kann. Das trägt wenig zur Transparenz bei, sondern verkompliziert und erschwert den Arbeitsprozess. Insbesondere in Situationen, wenn Entwickler vor Schwierigkeiten stehen und kollegiale Unterstützung und Hilfe benötigen, kann die dafür notwendige Form der Kommunikation nur sehr schwerfällig ablaufen: Nach dem Erkennen von Problemen unterscheiden sich meist die Momente zwischen „sich erstmal selbst helfen“ bis zu „sich helfen lassen“. Ehe der richtige Ansprechpartner dann gefunden wird, ob der gefundene Kollege das Problem in seinem komplexen Sachverhalt direkt versteht und dafür auch Zeit hat, sind zusätzliche

[1] Informationen, die man bewusst und unbewusst von anderen Personen wahrnimmt, wie Atmung, Gestik, Mimik, Stimme und Körperhaltung.

Hindernisse die Arbeit fortzusetzen.
Eine textuelle Status Funktion, bzw. eine digitale Hilfsanfrage im Saros Interface könnte ein Ansatz sein, das Entwicklungsteam über auftretende Probleme und Fragen während des Entwicklungsprozesses zu informieren, sowie die Hemmungen zu überbrücken binnen kurzer Zeit direkt um Hilfe zu bitten, sodass Kollegen zeitnah eine Antwort liefern können.

2.1 Forschungsidee

In welchem Szenario Saros Nutzer die Status-/Hilfsanfrage Funktion anwenden können, soll Forschungsgegenstand dieser Arbeit sein.
An welchen Stellen fehlende Awarenessinformationen optimiert werden können, habe ich anhand der aktuellen Saros Version eingehend bewertet, vorhandene Forschungsergebnisse für ein gutes Nutzerverständnis mit einbezogen und existierende Funktionen aus ähnlichen Software-Tools in einem Ideenpool zusammengetragen. Aus den vorhandenen Vorschlägen habe ich ein konkretes Feature ausgewählt, welches ich implementiert habe. Das Konzept und die Entwicklung erläutere ich dafür ausführlich. Anschließend soll mithilfe eines prototypischen Ansatzes beschrieben werden, wie man die Nutzer intrinsisch und extrinsisch motivieren kann, dieses Saros-Feature einzusetzen. Im Ausblick folgen weiterführende Ideen zu diesem Feature. Schließlich beendet ein Fazit unter Berücksichtigung der hier erworbenen Ergebnisse die Bachelorarbeit.

Ziel dieser Arbeit ist es, Saros so zu verbessern, dass Teams in einer gemeinsamen Sitzung besser, also einfacher und häufiger miteinander kommunizieren können, ohne sich gegenseitig aus dem Arbeitsfluss zu reißen. Insbesondere während auftretender Probleme soll sich das Nutzen der Saros-Features auf den Prozess und gleichzeitig auf die Qualität der Arbeit positiv auswirken. Zusätzlich werde ich mich später im Ausblick dazu äußern, wie man Nutzer motivieren kann Saros intensiver zu benutzen.

2.2 Motivation

Aufgrund eigener Erfahrung weiß ich, dass das Entwickeln von Programmen ein anspruchsvoller Vorgang ist. Insbesondere bei steigender Anzahl von Teammitgliedern erhöht sich der Komplexitätsgrad. Auftretende Probleme kommen dann überraschend

schnell. Durch Internetrecherchen ist man zwar in der Lage Lösungen für sein Problem zu finden, eine umfangreiche Liste potenzieller Ergebnisse, die erst nacheinander getestet werden müssen, ist bei einer zeitliche Abgabefrist jedoch sehr hinderlich.
Durch fehlendes Wissen der Programmierfertigkeiten anderer Mitglieder weiß man selten, wer der erste Ansprechpartner für das vorhandene Problem sein kann. Hinzu kommt die Hemmung, andere zu einem vielleicht ungünstigen Zeitpunkt in ihrem Arbeitsvorgang zu unterbrechen um ihnen die Problemsituation zu erklären. Eine Mailingliste ist zwar eine Möglichkeit Anfragen und Informationen zu teilen, Unübersichtlichkeit sind aber oft eine Folge einer schnell wachsenden Anzahl von E-Mails.
Mein Ziel ist es, Saros um eine Funktion zu erweitern, die Entwicklern hilft den Kommunikationsbeginn insbesondere bei einleitenden Problemsituationen zu vereinfachen.

In dieser Arbeit wurde vorwiegend die männliche Schreibweise verwendet um den Lesefluss nicht zu erschweren. Die weibliche Form ist natürlich immer mit inbegriffen.

3 Theoretische Grundlagen

Für ein besseres Verständnis ist es notwendig, gewisse Begrifflichkeiten und Prozesse einzuführen, die im Laufe dieser Arbeit verwendet werden, um Bedeutung zu ähnlichen Begriffen abzugrenzen und diese im Zusammenhang zu klären.

3.1 Was ist (verteilte) Paarprogrammierung?

Wie bereits in der Einleitung erklärt, ist die Paarprogrammierung eine agile Methode für den Entwicklungsprozess von Programmcode. Zwei Programmierer entwickeln gemeinsam an einem Arbeitsplatz. Dabei stehen ihnen ein Bildschirm, eine Maus und eine Tastatur zur Verfügung. Einer der beiden Entwickler übernimmt die Rolle des Drivers, der die alleinigen Benutzungsrechte der Eingabegeräte hat und dadurch die Aufgabe übernimmt, aktiv zu programmieren, während der andere die Rolle des Observers ausübt. Seine Aufgabe ist es, den Driver über Fehler, Probleme und Alternativlösungen verbal zu informieren ohne dabei aktiv die Eingabegeräte zu benutzen. Die Teilnehmer tauschen regelmäßig die Rollen und den Partner. Das Ziel und damit einhergehend positive Effekte dieses Prozesses sind zum einen, dass jeder Entwickler durch den regelmäßigen Partner- und Rollentausch viel von dem Projekt überblickt und seine Kollegen kennt. Der Observer hat, während der Driver programmiert, die grundlegende Aufgabe den Code genau durchzusehen, zu kontrollieren, sowie ihm auffallende Probleme direkt anzusprechen und mit dem Driver zu besprechen. So können schon direkt beim Entwickeln des Codes Probleme erkannt und durch anschließende Diskussion korrigiert und verhindert werden. Das senkt die Fehlerrate und führt wegen der besseren Ausdruckskraft dazu, dass der Umfang des Programmes geringer [AL00,] ist. Das Programm ist also kürzer und hat weniger Fehler, wenn man als Paar entwickelt. Sie arbeiten effektiver und effizienter - auch weil sie seltener im Arbeitsprozess unterbrochen werden als allein arbeitende Programmierer. Positive Ergebnisse fördern die Motivation an der Arbeit. Hohe Motivation an der Arbeit führt zu besseren Ergebnissen [Bri02,]. Ein negativer Aspekt bei (verteilter) Paarprogrammierung ist zum Beispiel der hohe Zeitaufwand. In den Teams, die die Driver/Observer Regeln strickt befolgen, gibt es einen Entwickler, der zwar in den Entwicklungsprozess eingebunden ist, jedoch nicht aktiv selbst entwickelt. Das kann daran liegen, dass manche Aufgabentypen einfach keine zwei Entwickler benötigen, sodass sich die Paarprogrammierung an diesen Stellen weder zeitlich noch

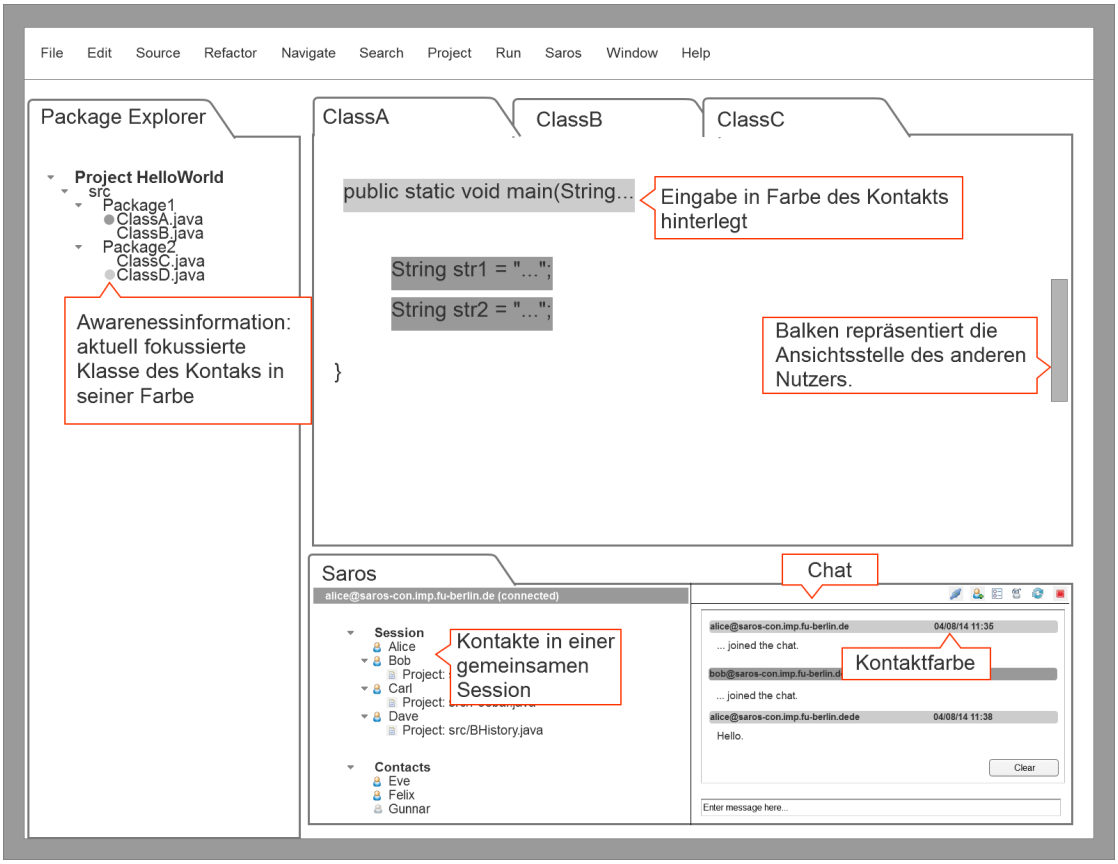
finanziell rentiert.

Die Teamfindung selbst stellt schon eine Herausforderung an sich dar. Schließlich kann es zu Konstellationen von Arbeitsgruppen kommen, bei denen sich die Entwickler unsympathisch [ST,] sind. Das kann wiederum zu untragbaren Arbeitsverhältnissen führen. Somit bleiben gute Resultate bei „schlechten“ Paaren aus. Bei „guten“ Paaren, weist der fachliche Leistungsunterschied nicht zu große Diskrepanzen auf, was sonst zu Hinderungen im Arbeitsprozess führen kann, bei denen sich einer nicht gebraucht und deshalb überflüssig fühlt. Er übernimmt meist die Rolle des „stillen“ Observers. Wichtig ist, dass sich zwei Entwickler finden, die grundharmonisch miteinander arbeiten können — das jedoch, ist bei regelmäßigem Partnertausch sehr schwierig. Paarprogrammierung ist ein sehr zeitintensives Verfahren, was primär an den Kommunikationsfertigkeiten der Entwickler zehrt. Diese müssen jede Entscheidung, jede Idee, jedes weitere Vorgehen intensiv mit mindestens einem weiteren Kollegen besprechen und rechtfertigen. Nichtsdestotrotz ist die verteilte Paarprogrammierung immer häufiger in dynamisch arbeitenden Softwareunternehmen anzufinden, da sich dieses Vorgehen äußerst positiv und langfristig auf die Produktivität und das Produkt auswirkt. Für den Einsatz von agilem Offsharing, also wenn verschiedene Unternehmen eine Kooperation für bestimmte Entwicklungsaufgaben übernehmen, bietet sich nur die verteilte Paarprogrammierung als geeignetes Vorgehen an, um gemeinsam zu entwickeln. Mithilfe von geeigneter Software ist es möglich, die Entwicklungsumgebung von zwei (und mehr) verteilten Entwicklern zu synchronisieren, um in Echtzeit gemeinsam programmieren zu können. Jeder sieht die Editierung im Programmcode, weiß welcher Entwickler am Projekt beteiligt ist und an welchem Punkt sich dieser befindet. Durch verschiedene Kanäle wie VoIP, Screensharing oder Chats sind sie in der Lage direkt miteinander zu kommunizieren.

3.2 Was ist Saros?

Saros [SARb,] ist ein Eclipse Plugin, das die Technik für verteilte Paarprogrammierung liefert. Eclipse [ECL,] ist ein Entwicklungswerkzeug zur Programmierung von Software. Es gibt verschiedene Erweiterungen für dieses Programm. Saros ist ein quelloffenes Eclipse Plugin, welches an der Freien Universität Berlin seit 2006 entwickelt wird. Nach der Installation und der Registrierung kann man Saros benutzen. Die Hauptfunktion ist

ein kollaborativer Editor, bei dem Änderungen zwischen Saros-Nutzern einer Sitzung in Echtzeit synchronisiert werden. Dort sehen sie die Editierung von den zugehörigen Autoren. Über die Kontaktliste können weitere Saros-Nutzer hinzugefügt werden. Ein eingebetteter Chat dient ihnen als Kommunikatiosmöglichkeit. Das Saros Whiteboard ist ein Werkzeug zur gemeinsamen grafischen Darstellung von Zeichnungen. Hinzukommend beinhaltet Saros Funktionen wie den Follow Mode, der die Eclipse Ansicht des Verfolgten automatisch mit der des Folgenden synchronisiert, d.h. dass der Folgende an genau dieselben Stellen im Editor springt, die der Verfolgte fokussiert. Auch Schreibrechte können individuell vergeben werden, sodass die Driver-Observer-Regeln umgesetzt werden können. Awareness Informationen wie eigene Nutzerfarben, an denen sich die Entwickler gegenseitig schnell identifizieren können, sowie das Teilen der gemeinsamen Konsole ergänzen das Programm um praktische Erweiterungen.



Eine weitere für diese Arbeit wichtige Saros Anwendungsmöglichkeit ist die Umsetzung der *erweiterten* verteilten Paarprogrammierung [JLS13,], also auch das unabhängige Programmieren innerhalb einer Saros Sitzung. Entwickler haben dann die Möglichkeit gemeinsam echte verteilte Paarprogrammierung zu betreiben, können aber bestimmte

Aufgaben einzeln entwickeln, ohne die Sitzung beenden zu müssen. Danach können die Lösungsansätze und das weitere Vorgehen in der vorhandenen Saros Sitzung miteinander geklärt werden.

3.3 Was ist Awareness?

Awareness [AWA,] (engl. *Gewahrsein Bewusstsein*, *Wahrnehmung*) liefert einer Person die Informationen, die von anderen Menschen bewusst und unbewusst wahrgenommen werden, dazu gehören Mimik, Gestik, Körperhaltung, Sprechgeschwindigkeit, Stimme und Atmung. Sie sind ein wichtiger Teil in kollaborativen Arbeitsumgebungen[CM96,]. Bei der klassischen Paarprogrammierung kann beispielsweise der Driver an der sich anspannenden Körperhaltung des neben ihm sitzenden Observers erkennen, dass dieser einen Einwand haben könnte. Hörbares Einatmen des Observers hat einen ähnlichen Effekt. Ohne mit dem Observer kommuniziert zu haben, hat der Driver durch achtsames Erkennen, bewusst Informationen erhalten, auf welche er direkt eingehen kann. Eine Folge davon könnte beispielsweise sein, dass sich der Driver die letzten zehn Zeilen Code noch einmal genauer anschaut, ohne dass der Observer seinen Einwand ausformulieren braucht. Ist der Observer sehr aufmerksam, kann er z.B. anhand der Blickrichtung des Drivers erkennen, wo sich dieser gerade befindet oder was sein nächstes Vorhaben sein könnte. Kompiliert das Programm, kann vom Observer das Verhalten des Drivers, der auf einmal inne hält und gespannt wartet, bemerkt und verstanden werden.
Diese Beispiele zeigen Problematiken [CS02,], die bei verteilter Paarprogrammierung aufkommen. Durch fehlende Awareness Informationen können Blickrichtungen, Körperanspannungen, Atmungen etc. nicht bemerkt werden. Oft führt das zu Missverständnissen. Ein Lösungsansatz fehlende Informationen zu erhalten, ist eine Voice-over-IP (VoIP) Verbindung. Der eingebettete Chat wird wenig genutzt, was Nutzertests bestätigten, [TS09,], denn Programmierer müssen dafür das Eingabegerät wechseln und zusätzlich die Blickrichtung ändern. Außerdem ist das Übermitteln von Informationen über einen Chat zu langsam um Probleme zu klären[TS09,]. Videokanäle sind eine Möglichkeit, dass sich beide Programmierer zwar sehen und hören können, allerdings reißt ein sich ständig bewegendes Kamerabild den Entwickler häufig aus der Arbeit heraus [Bri02,]. Ein Videokanal benötigt ebenfalls eine hohe Brandbreite zur Übertragung[TS09,]. Durch

Screensharing können Bildschirminhalte übertragen werden, die Saros nicht synchronisiert. Dazu gehören der Kontextklick, Menüauswahlen, Such- und Ersetzungsfunktionen. Tools und Programme, die eine Screensharing Funktion anbieten, benötigen eine hohe Bandbreite, was wenig zum sowieso schon sehr rechnerintensiven Programmieren beiträgt. Zusätzlich ist ein regelmäßiger Rollenwechsel schwerfällig. Mehrere aktive Mauszeiger können für beide zur Verwirrungen führen. Um diese zu vermeiden, muss einer der beiden Programmierer auf eine vorrübergehende Bedienung seines Rechners verzichten. Er ist dann ein passiver Gast auf der anderen Seite. Gute Awareness trägt zu einer guten Kommunikation und Koordination — die Schlüsselaktivität für Paarprogrammierung — bei[CS02,].

3.4 Was ist Gamification?

In einem späteren Abschnitt dieser Bachelorarbeit, in welchem ein Ausblick des von mir entwickelten Features beschrieben wird, gehe ich auf den Zusammenhang der Gamification ein.

Gamification ist der Gebrauch von Spielelementen und Spieltechniken in einem spielfremden Kontext. Der Begriff Gamification wurde erstmals 2008 [OPE,] benutzt und findet sich immer häufiger in Alltag und Beruf wieder. Bei Gamification handelt es sich nicht um ein Spiel sondern um Elemente, die der Software einen gewissen Spielcharakter gibt. Nutzer werden motiviert Software mit Spaß zu benutzen. Das primäre eigentliche Ziel, welches durch die Software erreicht werden soll, wird durch ein sekundäres, spaßiges Ziel erreicht.

Im Gegensatz zu „Serious Games“ geht es nicht ausschließlich um das Spielen an sich, um einem Unterhaltszweck zu dienen. Ziel ist es, eine spielerische Umgebung in einem spielfremden Kontext zu schaffen. Dafür verwendete Elemente sind z.B. das Benutzen von Highscores, Fortschrittsbalken, Ranglisten, virtuellen Gütern in Form von Belohnungen (sog. Badges = engl. *Orden*), die den Anwender motivieren sollen, ein vordergründiges Ziel zu erreichen, ein Problem zu lösen oder an das Produkt gebunden zu werden. Insbesondere in der geschäftlichen Umgebung wird vermehrt auf Gamification gesetzt um so einen spielerischen Umgang mit einem Produkt in einem Arbeitsprozess durch eine Software zu erreichen.

Bedingungen und Voraussetzungen für eine erfolgreiche Umsetzung von Gamification sind eine einfache Bedienung, Verständnis, Design und vor allen Dingen Freiwilligkeit in der Benutzung. Wenn die Gamification-Elemente die Umsetzung der Software unnötig verkomplizieren, den Arbeitsprozess verlängert oder aufzwingt, ist die Gamification Idee verwirkt.

Gamification soll Spaß in eine Anwendung bringen, der vorher gefehlt hat. Dennoch muss dieser durch anspruchsvolle Aufgaben erreicht werden, ohne zu einem zusätzlichen Arbeitsaufwand zu führen, der dem Anwender von der eigentlichen Arbeit abhält. Gamification führt nicht dazu, dass vorhandene Probleme gelöst werden, sondern den Nutzer an eine Software zu binden. Der derzeitige Software-Markt bietet ein umfangreiches Angebot. Zu jedem Produkt gibt es oft mindestens eine Alternative. Gamification-Elemente binden den Anwender an das Produkt und verbessern die Loyalität zu diesem.

Es wird von einer 80%igen [GAM,] Fehlerrate bei Produkten mit Gamification Elementen ausgegangen. Diese sind in der Umsetzung meist nicht gut gestaltet. Um also erfolgreich Gamification zu betreiben, ist es essentiell, den Nutzer der Software komplett zu verstehen. Nur so kann man die richtigen Elemente benutzen, die der Motivation des Anwenders entspricht. Man unterscheidet drei verschiedene Arten, wie man den Nutzer motivieren kann: Hard Fun, Easy Fun und Social Fun. Während das eine als primäres Ziel hat, zu gewinnen, geht es beim zweiten darum, die *Welt* zu erkunden und das dritte soll Spieler an andere Spieler knüpfen. Zusätzlich unterscheidet man vier verschiedene Spielertypen; Während der Sammler das Ziel hat, möglichst viel zu erreichen (Level, Punkte, Ranglisten), versucht der Entdecker die Spielwelt zu erkunden. Der Soziale möchte vorrangig mit anderen Spielern interagieren und der Jäger will sich im Wettbewerb mit anderen messen und nicht nur das Spiel gewinnen, sondern dieses auch gegen die Mitspieler. Damit das Spiel nicht langweilig wird, muss dem entsprechenden Erfahrungswert der Spieler der Aufgabentyp angepasst werden. Ist die erste Aufgabe am Anfang zu schwierig, so ist die Chance hoch, dass die Motivation so stark sinkt und der Spieler schnell aufgibt und das Spiel nie wieder neu aufnimmt. Ist eine Aufgabe für einen Experten zu einfach, kann es schnell zu ausgeprägten Phasen der Langeweile kommen und somit ist hier wiederum das Risiko gegeben, dass auch dieser Spieler ebenfalls nie wieder an dem Spiel teilnehmen wird. Die Punkte, welche man in der Anwendung sammeln kann, unterscheiden sich maßgeblich. Es gibt die Erfahrungspunkte (XP), welche

die eigene Leistung widerspiegeln. Alles, was der Nutzer macht, kann ihm solche Punkte einbringen. Auch die reine Anwesenheit, das heißt, selbst wenn er nichts macht, kann innerhalb der Software dazu beitragen, dass er Erfahrungspunkte verdient. Je mehr Erfahrungspunkte gesammelt werden, desto höher steigt der Einzelne im Level.
Zudem gibt es einlösbare Punkte, die man gegen virtuelle Güter eintauschen kann. Die soziale Komponente im Spiel wird durch die Reputationspunkte ausgezeichnet. Das sind die Punkte, die man von anderen Spielern erhält.
Nach dem *SAPS* Prinzip (*Status*, *Access*, *Power* und *Stuff*) werden Belohnungen vergeben. Der Status gibt dem User die Möglichkeit, sich mit anderen Spielern über ein Rangsystem zu vergleichen. Access liefert den Zugang zu der Software, die vorher nicht erreichbar und benutzbar waren. Power gibt einem Spieler die Macht über andere Nutzer, vergleichbar mit Moderatoren in einem Forum. Stuff gibt dem Spieler virtuelle Gegenstände oder Fähigkeiten als Belohnung. Der Unterschied zwischen Status und Stuff ist die Betrachtungsweise der Spielermotivation. Der Status motiviert den Spieler intrinsisch, während die Belohnungen extrinsisch sind. Zur Visualisierung von Access Gegenständen wird oft auf Badges gesetzt, deren Wert meist überschätzt wird.

3.5 Was ist das Five − Factor − Model?

Das Five-Factor-Model [NEJ09,] bzw. die Big Five ist eine Methode der Persönlichkeitspsychologie. Jeder Mensch wird dabei auf fünf Merkmale charakterisiert. Im Konzeptteil werden Annahmen zum Verständnis der Motivationsgründen für die Anwendung des implementierten Features erarbeitet. Dort begründe ich über dieses Model Konzept- und Interaktionsentwürfe.
Im Folgenden werden die Charakteristiken bei Paarprogrammierern [NEJ10,] beschrieben.

1. Neurotizismus

   Der Neurotizismus ist eine Charaktereigenschaft, welche das Verhalten der Menschen bei negativen Emotionen unterscheidet. Psychologen nennen diese Eigenschaft auch emotionale Labilität.

   Menschen mit einer hohen Ausprägung erleben häufiger Angst, Anspannung und Unsicherheit, welche bei labilen Menschen länger bestehen bleiben und leichter aus-

gelöst werden [NEJ10,]. Sie sind besorgt um ihre Gesundheit und haben Schwierigkeiten mit Stresssituationen umzugehen. Menschen mit niedrigen Werten, also die emotional stabil sind und mehr Zufriedenheit ausstrahlen, sind eher ruhig, entspannt und sicher. Sie erleben negative Gefühle weniger intensiv.

2. Extraversion

   Die Extraversion ist eine Eigenschaft, welche die Begeisterungsfähigkeit bei Probanden beschreibt. Personen, die eine höhere Bereitschaft haben, Begeisterung zu empfinden und zu zeigen, sind geselliger, aktiver, gesprächiger und optimistischer als Menschen mit niedrigen Werten. Diese hingegen verhalten sich eher zurückhaltend, haben weniger Durchsetzungsvermögen und sind eher weniger sozial aktiv.

3. Offenheit für Erfahrungen

   Mit dem Faktor der Offenheit für Erfahrungen wird das Interesse an neuen Erfahrungen, Erlebnissen und Eindrücken beschrieben. Menschen mit hohen Offenheitswerten beschreiben, dass sie sehr wissbegierig, abwechslungsreich, intellektuell, fantasievoll, experimentierfreudig und künstlerisch interessiert sind. Sie scheuen sich nicht vor unkonventionelleren Meinungen und teilen diese auch öffentlich[NEJ10,]. Personen mit niedrigen Offenheitswerten neigen zu konventionellem Verhalten und zu konservativen Einstellungen. Unbekanntes und Neues setzt sich gegen Bewährtes nicht durch. Es beschreibt intellektuelles, kulturelles oder kreatives Interesse.

4. Verträglichkeit

   Wie Extraversion ist Verträglichkeit in erster Linie ein Faktor, der intrinsisches Verhalten beschreibt.

   Ein zentrales Merkmal bei Menschen mit hohen Werten ist ihr aufopferungsvolles und selbstloses Leben. Sie sind anderen gegenüber verständnisvoll, mitfühlend, äußerst hilfsbereit, haben ein hohes Vertrauen in ihre Mitmenschen. Sie sind kooperativ, freundlich und vertrauenswürdig. Das hat zu Folge, dass sie oft sehr nachgiebig sind[NEJ10,]. Menschen mit niedrigen Werten neigen im Gegensatz dazu zu antagonistisch, egozentrisch, skeptisch und feindselig zu sein. Sie bevorzugen den Wettstreit und die Konkurrenz zu ihren Mitmenschen und sind wenig kooperativ[NEJ10,].

5. Gewissenhaftigkeit

Der Faktor der Gewissenhaftigkeit beschreibt die Fähigkeit der Selbstkontrolle, Genauigkeit sowie Zielstrebigkeit. Menschen mit hohen Werten sind sehr gut organisiert, planen ihr Leben sorgfältig und effektiv, sie stellen sich zuverlässig ihrer Verantwortung, neigen zu harter Arbeit und erfüllen Aufgaben meist sehr gründlich. Diese Eigenschaft gilt als Grundlage für wissenschaftliches und intellektuelles Arbeiten und Verständnis. Personen mit niedrigen Gewissenhaftigkeitswerten neigen zu eher spontanem und ungenauem Verhalten, sie sind unverantwortlich, impulsiv und unorganisiert[NEJ10,].

4 Vorüberlegungen

4.1 Ideenentwicklung

Mein Interesse- sowie Wunschthema dieser Forschungsarbeit war die Kommunikation von Saros Nutzern unter einem Usability-Ansatz grafisch zu optimieren. Die Zentralfrage, mit der ich mich in den vergangenen Wochen beschäftigte, war wie und vor allem welche Awareness-Informationen über das Interface besser vermittelt werden können. Dafür habe ich entsprechende Literatur zu dem Thema Awareness und verteilte Paarprogrammierung gelesen, aus welchen ich Ideenvorschläge abgeleitet und ausformuliert habe. Die Ideen bezogen sich auf die Vorstellung wie man das Interface verbessern und erweitern könnte. Diese sind in einer Mindmap [XMIa,] dargestellt und im Folgenden beschrieben.

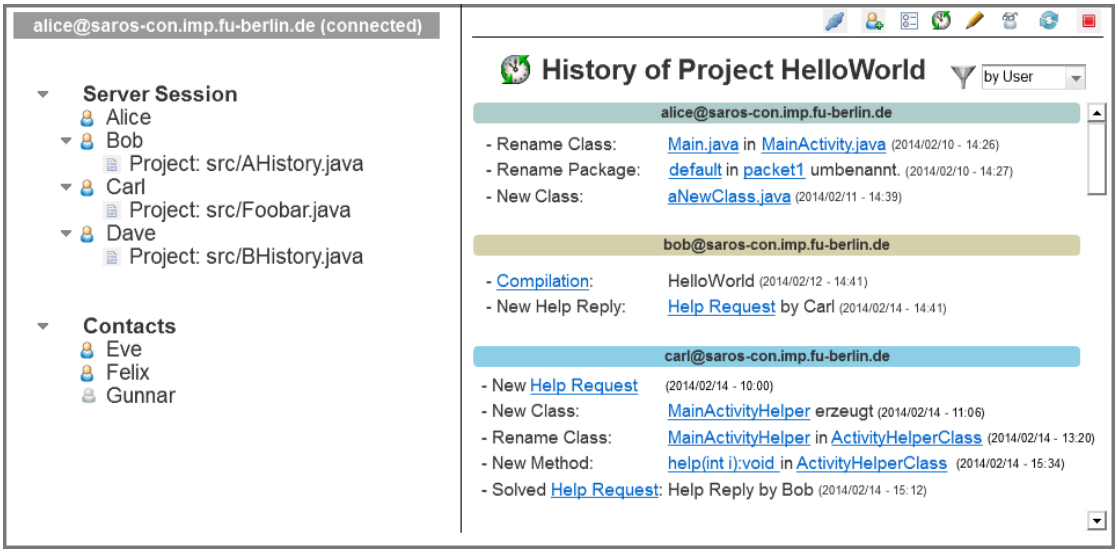
4.1.1 Verlauf

Der *Experten Finder* [ST,] ein Pattern für soziale Interaktion hat noch kein Feature in Saros implementiert, würde aber einen hohen Grad an Transparenz und die Möglichkeit liefern, wie bestimmtes Wissen gefunden werden kann. Daraus herleitend entwickelte sich die Idee eines globaler Verlauf, der alle Teilnehmer einer Sitzung, bzw. eines Projekts eine übersichtliche Auskunft darüber liefert, welche Änderungen vorgenommen wurden. Jede Änderung wird mit einem Datumsstempel und dem Autor der Änderung versehen, sodass der User die Möglichkeit hat, Verlaufseinträge nach Person oder Datum zu filtern. Bei Saros gibt es zwar einen eingebetteten, wenig genutzten Chat und damit die Möglichkeit den Verlauf innerhalb dieser Chat-View anzuzeigen, ratsam wäre dieser allerdings nicht, da es durch ständig einfliegende Nachrichten schnell zu einer grafischen Verwirrung kommen könnte.
Beispiele für Anwendungsfälle und Problemsituationen im möglichen Nutzungskontext wären:

- Welche Datei wurde zuletzt bearbeitet?
- Von wem wurde diese Datei bearbeitet?
- Zu welchem Zeitpunkt wurde das Programm zuletzt kompiliert?
- Bei wem hat es zuletzt erfolgreich kompiliert?

- Wer hat wann welches Artefakt umbenannt?
- Wie hieß das Artefakt vorher?
- Hinzufügen oder Löschen eines neuen Artefakts.

Alle Artefakte könnten in der Verlaufs-Ansicht verlinkt werden, sodass diese per Klick geöffnet und als Sprungmarke innerhalb des Editors angezeigt werden. Da der Chat ein kaum benutztes Medium ist, da das Hauptkommunikationsmedium während der Paarprogrammierung meistens auditiv ist, wäre mein Vorschlag, den Verlauf als Standardansicht anzuzeigen und den Chat auszublenden. Beide Ansichten könnten jeweils über einen eigenen Button in der Saros-Funktionsleiste ausgewählt werden. Gerade während des Entwicklungsprozesses und insbesondere bei Paarprogrammierung, bei der regelmäßig die Partner getauscht werden, ist es schwierig, den Überblick zu behalten. Auch eine längere Abwesenheit von der Projektentwicklung könnte zur (Wieder−)Einarbeitung in das Projekt, durch einen übersichtlichen Verlauf, beschleunigt werden.

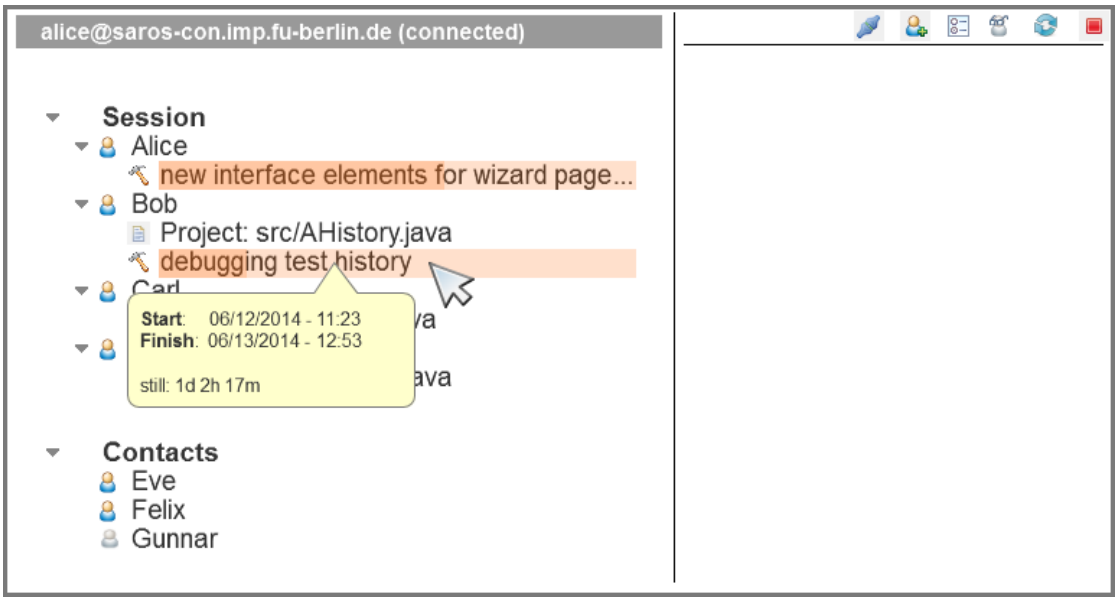


4.1.2 Taskbeschreibung

Ein von Entwicklern geäußerter Wunsch ist sich gegenseitig auf dem aktuellsten Stand bzgl. der Aufgaben- und Fehlerverwaltung zu halten. Die Idee dabei ist, dass Nutzern eine Möglichkeit gegeben wird aktuelle Informationen miteinander zu teilen, ähnlich eines Status Updates wie bei sozialen Netzwerken. Vor eine Paarformation, wählt der Entwickler laut [ST,] einen Task aus und sucht dafür einen passenden Partner. Die Idee hinter der

Saros-Taskbeschreibung entwickelte sich aus dieser Technik. In Software-Unternehmen wird häufig eine Fehlerverwaltungssoftware wie Jira [JIR,] dafür eingesetzt, so dass Nutzer Aufgaben, Ideen, Probleme und Projekte zusammenfassen, kommentieren, bewerten und verknüpfen können. Sinnvoll wäre es, diese übersichtlicher innerhalb des Interface in Erfahrung anzeigen zu können, ohne dass ein Kontextwechsel im Sinne eines Programmaufrufs forciert werden muss. Idealerweise ließe sich diese Information direkt von Jira importieren oder verlinken, sodass der Nutzer keine eigene Beschreibung mehr hinzufügen muss. So eine Beschreibung wäre mit dem entsprechenden Projektnamen verknüpft und könnte als textueller Ausschnitt angezeigt werden. Diese wären entweder ein Tooltip per Mouseover über oder direkt hinter bzw. als TreeItem unter dem eigenen Nutzernamen. Für ausführliche Informationen könnte sich ein Wizard-Fenster mit den Informationen öffnen und direkt auf die Jira -Seite verlinken. Als zusätzliche Erweiterung kann man den zeitlichen Fortschrittsanzeiger, der aktuellen Aufgabe mithilfe eines Fortschrittsbalkens innerhalb des TreeItems, anzeigen. Denn auch bei Jira ist es üblich zu jedem Task eine realistische Zeiteinschätzung mit anzugeben. Bei Saros liefere diese zeitliche Information einen grafischen Mehrwert bzgl. einer potenziellen neuen Partnersuche.

Von dieser Idee ableitend hat sich das Feature eines Help Requests entwickelt, welche ich prototypisch implementiert habe. Dabei handelt es sich um einen „Problem Status“. Im Weiteren wird diesem Feature ein eigenes, ausführliches Kapitel gewidmet.



4.1.3 Status in Kontaktliste

Aus der Taskbeschreibung weiterleitend, wie man aktuelle Aufgaben den anderen Teammitgliedern vermitteln kann, und den bereits vorhandenen verschiedenen Stati in der Saros Kontaktliste, setzte sich diese Idee fort. Zum Beispiel ändert sich das Icon einer Kontaktperson nach einer fünfminütigen Abwesenheit, was durch eine kleine Uhr an dem Kontakt-Icon erkennbar ist. Diese erinnern an die Icon"buddies" des Windows Live Messenger. Dort ändert sich die Farbe des Icons je nach Status der Person. Abstrahierend auf das Paarprogrammierungsszenario könnte man die verschiedenen Farben übernehmen und erweitern: − grün = online, grau = offline, rot = abwesend, blau = programmiert, gelb = kompiliert In Kombination mit dem oben genannten textuellen Status für die Taskbeschreibung ließe sich die Information gut kombinieren: Zum Beispiel als Tooltip über dem Kontakt-Icon, dort werden kurze Erklärungen zu dem Status geliefert.

4.1.4 Programmausführung

Aus der Idee des Verlaufs entwickelte sich die Idee den Inhalt der eigenen Konsole als Information zu in den Verlauf hinzuzufügen, damit andere diese Informationen auch erhalten können. Wie ich im Laufe dieser Arbeit feststellte, wird eine sehr ähnlichen Funktion bereits in einer anderen Abschlussarbeit entwickelt. Anders ist, dass die Inhalte nicht gespeichert werden, sondern in Echtzeit während der Ausführung von Programmcode [CON,] in der eigenen Konsole stehen. Meine Idee lautet, die gespeicherten Informationen innerhalb der Konsole für andere Saros Nutzer einlesbar zu machen. Diese Informationen könnten mit der oben beschriebenen Idee eines Verlauf erweitert werden, indem der Konsolentext in eine externe Datei geschrieben, im Projektverzeichnis gespeichert und im Verlauf verlinkt wird. Durch einen Klick auf die Verlinkung werden die Inhalte in die Konsole geladen. Das könnte auch dafür nützlich sein, um verschieden erworbene Ergebnisse vergleichen zu können.

4.1.5 Teaching

Für neue Teammitglieder eines Projekts, die sich noch nicht gut im Projektcode orientieren können, ist das Thema Teaching [ST,] ein sehr interessanter Punkt. Der Experte eines Projekts könnte für relevante Stellen einen Sammelpunkt ausrufen, sodass sich alle Nutzer in einer Klasse zusammenfinden sollen. Gerade für Anfänger bietet sich, für ein

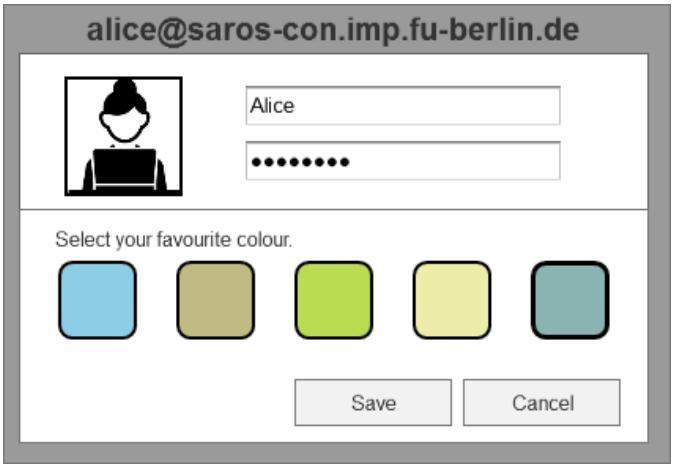
späteres Analysieren, ein Screen- Recorder mit einer Replay- Funktion an. Verschiedene Softwareanwendungen liefern dafür schon sehr gute Funktionen an, wie z.B. das Etherpad [ETH,]. Mit diesem Tool werden alle Änderungen aufgezeichnet und abspielbar zur Verfügung gestellt.

4.1.6 Chat

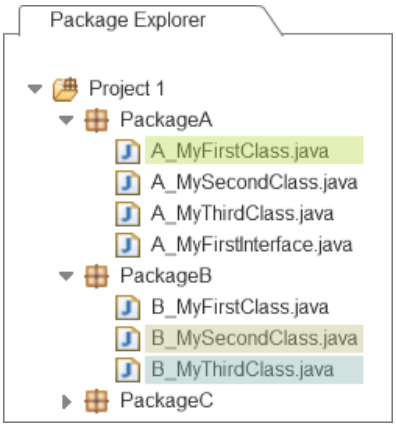
Wie bereits erwähnt, existiert für Saros schon ein eingebetteter, wenn auch wenig genutzter Chat. Dieser könnte aber unter einigen Gesichtspunkten verbessert werden. Eine gewisse Dynamik beim Schreiben, wie man sie bei dem Programm *Skype* [SKY,] finden kann, fehlt. Hierbei erscheint im eigenen Chatfenster ein Stift, welcher suggeriert, dass der andere gerade schreibt. Für den Fall, dass Alice das Schreiben unterbrochen hat, aber das Inputfeld nicht geleert ist, bleibt der Stift stehen und Bob ist immer noch der Annahme, dass Alice tippt. Deswegen liefert ein animierter, also schreibender Stift, eine bessere Awareness. Solange getippt wird, bewegt sich der Stift, wird aufgehört, bleibt dieser stehen. Wenn ein Text gelöscht wird, erscheint äquivalent ein animierter Radiergummi. Ein einfaches Muster, dass jeder Nutzer kennt.
Ein weiteres Problem, welches beim Saros- Chat auftreten könnte, wäre, dass die zur Kenntnisnahme einer eintreffenden Nachricht gehemmt ist, sobald die Soundnotifications deaktiviert sind. Das heißt, es besteht die Gefahr, dass man das Eintreffen eben dieser Nachricht nicht mitbekommt. Daher wäre es von Vorteil, die Chatnachrichten als Balloon- Notification, auf Höhe des Cursors einzublenden und diese nach kurzer Zeit auch wieder auszublenden. Diese Funktion sollte de- und aktiviert werden können.

4.1.7 Profil(−bild)

Für eine bessere Bindung zu der Software und zu seinem angelegten Account ist es sinnvoll, sich ein Profil anzulegen. Auch das Hinzufügen eines Kontobildes gehört dazu. Zum einen sehen andere Entwickler, wer sich hinter bestimmten Accountnamen verbirgt, was die Nähe zu den Mitprogrammierern und eine zuordnende Persönlichkeit schafft, zudem kann das Bild des Buddy-Icon im Roster durch das eigene ersetzt werden. Das Icon selbst hat wenig Aussagekraft über den User. Das wiederum könnte die Bindung zu einer Software erhöhen und zu dem Thema *Gamification* eine Grundbedingung liefern.



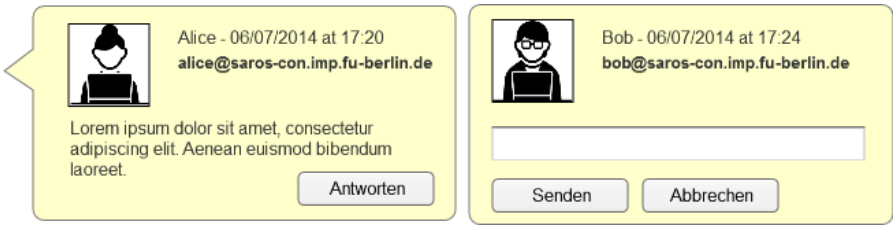
4.1.8 Farbauswahl / Selektion auch im Package Explorer



Bei der Selektion und der Textbearbeitung werden die Nutzerfarbe schon sehr prominent verwendet. Diese könnten aber noch häufiger genutzt werden, wie z.B. in der Reiterfarbe des Tabs. Im Package-Explorer wird ein kleines Icon auf der Datei sichtbar, welches vom anderen Nutzer gerade geöffnet ist. Dieses ist allerdings sehr klein und wird oft übersehen. Es größer darzustellen trägt wenig zur Ästhetik des Interface bei. Stattdessen könnte ein halbtransparenter Balken auf das Element der geöffneten Klasse hinterlegt werden. Das Eclipse Interface ist mit einer CSS Datei gestyled und damit auf diese Art anpassbar.

4.1.9 Kommentare

Microsoft Office und auch Google Docs bieten eine ästhetische und nützliche Kommentarfunktion. Solch ein Feature fehlt Saros bisher noch. Kommentare können zwar in Form von Programm-Kommentaren im Code hineingeschrieben werden, ist jedoch nur für den Fall einer aktiven verteilten Paarprogrammierungssitzung sinnvoll. Gut wäre es für eine verteilte Party- Programming, wenn man Kommentare als Objekte an den Code *anpinnen* könnte, ähnlich wie Post-it's an den entsprechenden Stellen im Code. Ein Zeitstempel, wann der Kommentar angefertigt wurde und von wem der Kommentar stammt — samt Profilbild sollte mit einer Antwort- und Zitiermöglichkeit gegeben werden.



4.2 Taskbeschreibung

Während der Entwicklung der oben beschriebenen Konzeptideen für eine bessere Kommunikation durch Awarenessinformationen, lag die Entscheidung für die Implementierung zum einen bei dem *Verlauf* und zum anderen bei der *Taskbeschreibung*, da diese konzeptionell zum Zeitpunkt des Antrittvortrages [ANT,] durchdacht waren. Ziel war es in dem verfügbaren zeitlichen Rahmen eine Konzeptidee für Saros zu implementieren. Deswegen galt es herauszufinden, welche technischen Möglichkeiten Eclipse für eine Implementierung liefert. Zusätzlich habe ich die beiden Ideen in interaktive Prototypen mit dem Programm Axure [AXU,] entworfen, um Interaktions- und Informationsdesign zu visualisieren. Parallel habe ich recherchiert, ob und wie man einzelne Schritte für Eclipse entwickeln könnte. Dabei hat sich herausgestellt, dass die Idee des Verlaufs technisch sehr umfangreich und leider nicht im Rahmen des Bearbeitungszeitraums liegt. Außerdem wurde mir während der Entscheidungsphase ein Nutzerfeedback eines Saros Nutzers weitergeleitet:

> Q: Do you have any other comments to share with us (e.g about the interface, specific features or possible improvements)?
> A: Yes. I would like to be able to quickly notify friends with some code position or line, for example, when i need their help, or urgently discussion of some moments.

Deswegen fiel die Entscheidung auf eine modifizierte Form der Taskbeschreibung. Die ursprüngliche Idee lautete nämlich, dass jeder Entwickler seine Kollegen über die eigene, aktuelle Aufgabenbeschreibung informieren muss. Darin werden die Entwickler aber wenig Mehrwert sehen, denn der Fokus liegt darin, ein sinnvolles Zusatzwissen ohne Mehraufwand bezüglich der Aufgabenbeschreibung zu liefern. Wenn der Mehrwert ausbleibt, ist ein zusätzlicher Arbeitsschritt für Programmierer wenig motivierend. Ein Feature mit scheinbar unnötigem Arbeitsaufwand ohne merkbaren Vorteil bleibt höchst-

wahrscheinlich ungenutzt. Deswegen galt es zu hinterfragen, wie man einen Mehrwert an der Information gewinnen kann und wie man Nutzer motiviert, diese Feature auch zu benutzen.
In dem Antrittsvortrag [ANT,] wurde die Taskbeschreibung mit vier relevanten Informationseingaben beschrieben:

1. eine textuelle Beschreibung des aktuellen Tasks
2. die Auswahl des aktuellen Projekts mit den entsprechenden Klassen, an denen gearbeitet wird
3. einen Zeitstempel
4. eine evt. Auswahl von mitarbeitenden Paarprogrammierern

Diese Eingaben sind zwar einfach, aber nicht schnell gefüllt, da diese Informationen manuell eingegeben werden müssten. Nun stellte sich die Frage: Warum sollte eine Taskbeschreibung in Saros überhaupt angezeigt werden, welche Informationen sind dafür tatsächlich relevant und welche Motivation brauchen Saros Nutzer um diese freiwillig anzugeben? Zu Beginn war es wichtig, zu verstehen, warum Nutzer überhaupt einen Status eingeben, deshalb habe ich mich im Rahmen dieser Arbeit, eingehend mit empirischen Forschungen [CEea09,] bzgl. Statuseingaben in sozialen Netzwerken auseinander gesetzt. Auch wenn diese Eingaben wenig mit der Paarprogrammierung und Softwareentwicklung gemeinsam haben, hat mich interessiert, was Nutzer prinzipiell motiviert, eine freiwillige Statuseingabe zu tätigen. Bei dem Antrittsvortrag [ANT,] wurde die Idee geäußert, Twitter oder Jabber Statuseingaben automatisch in Saros zu importieren. Jabber ist naheliegend, da man sich bei der Installation von Saros ohnehin einen Jabber Account anlegen muss. Hingegen sei Twitter die beliebteste Plattform von Entwicklern. Das hätte zum Vorteil, dass der Status automatisch eingelesen werden könnte und daher keine manuelle Eingabe mehr nötig wäre. Nach längerer Recherche habe ich hingegen festgestellt, dass Statuseingaben von Entwicklern selten programmatischen bzw. arbeitsrelevanten Inhalt haben, da dieser meist aus privaten Accounts stammt. Wobei sich gezeigt hat, dass insbesondere Opensource Entwickler Twitter tatsächlich als mediale Plattform nutzen. Diese scheinen aber keine Paarprogrammierung zu betreiben, sondern posten in Twitter ihre individuellen Fortschritte. Die Zielgruppe von Saros sind aber Entwickler,

die Paarprogrammierung betreiben und in kleinen Teams zusammenarbeiten. Daher habe ich die Idee verworfen, automatisch Informationen aus externen Quellen zu beziehen und weiter überlegt, wie man eine manuelle Eingabe begründen kann. Manuelle Statuseingaben in sozialen Netzwerken funktionieren deshalb, weil die Eingabe freiwillig erfolgt und weil es Informationen sind, die der individuelle Nutzer für sich und andere als interessant erachtet. Niemand *muss* eine Eingabe machen. Deswegen habe ich diese Idee für die Saros-Taskbeschreibung in den Ideenpool mit aufgenommen. Taskbeschreibungen sind keine obligatorischen Bedingungen mehr. Die Frage, welche sich nun bis zu diesem Punkt herausgebildet hat, lautet: Wann werden Nutzer eine Taskbeschreibung angeben, wenn diese nur noch optional ist? Dafür habe ich mir folgende Szenarien während einer Distributed Party Programming Session überlegt [XMIb,]:

- bei einer neuer Partnersuche (intrinsische Motivation)
- bei auftretenden Probleme (intrinsische Motivation)
- bei Belohnungen (extrinsische Motivation)
- bei Aufforderung (extrinsische Motivation)

Die ersten zwei Motivationspunkte haben im Unterschied zu den anderen beiden eins gemeinsam: Die Motivation einer Taskbeschreibung löst sich ausschließlich intrinsisch. Der Nutzer wird nur aus eigenem Selbstentschluss und nicht durch externe Argumente den Vorgang abschließen, weil er sich bei diesen beiden Szenarien einen eigenen Vorteil verspricht. Entweder sucht er einen neuen Partner für eine Paarprogrammierungssitzung, und möchte wissen, wer an welchen Artefakten gerade arbeitet um sich anschließen zu können oder er hat ein fachliches Problem, bei dem er sich nicht mehr selbst helfen kann und auf die Hilfe von Kollegen angewiesen ist. Für diese beide Szenarien braucht es nur Eingaben über inhaltliche Information der Task- bzw. Hilfsbeschreibung: In welchem Projekt man sich befindet, wer mit einem gemeinsam arbeitet oder zu welchem Zeitpunkt das stattfindet, sind irrelevant. Ich gehe davon aus, dass das zweite Szenario, also wenn Entwickler vor komplexen Problemen stehen, häufiger auftreten wird, als die Suche nach freien Paarprogrammierern. Deshalb habe ich mich entschlossen, die Taskbeschreibung zu einer Problembeschreibung zu modifizieren. Für die Implementierung wurde von mir ein Feature entwickelt, bei dem sich Nutzer Hilfs- und Antwort Anfragen schicken kön-

nen. Dieses Feature heißt ab sofort *Help Request*. Die anderen beiden Motivationspunkte (Belohnung und Aufforderung) werden im Ausblick ausführlicher beschrieben.

4.3 Soziales Verhalten

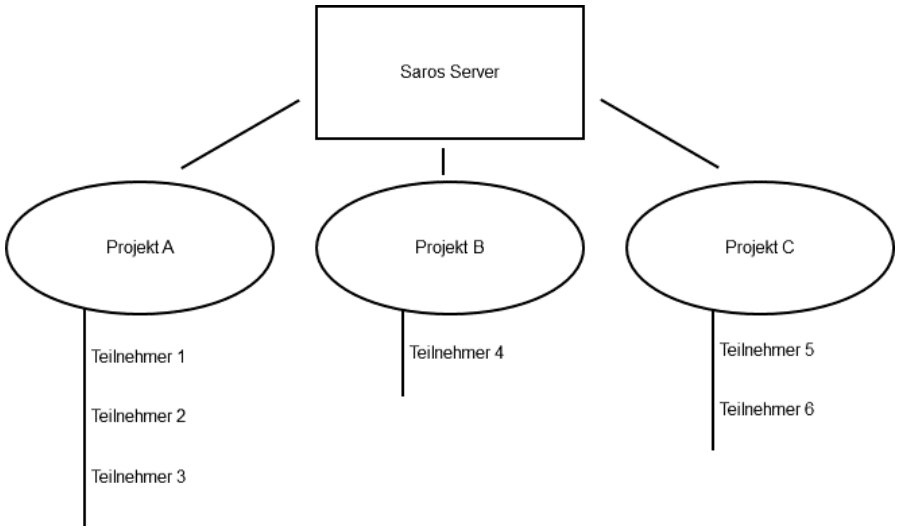
Im folgenden Kapitel wird das soziale Verhalten von Paarprogrammierern beschrieben. Dabei soll herausgefunden werden, ob eine Funktion von Usern benutzt wird. Das erfordert in der Regel einen Nutzertest. Dies war im Rahmen dieser Forschungsarbeit zeitlich nicht im Bearbeitungsrahmen, dennoch habe ich im Folgenden einige Annahmen formuliert, die Ansätze für die Anwendung des Help Requests liefern.

Im dritten Kapitel beschrieb ich das Five-Factor Model, also eine Methode jeden Menschen auf fünf Persönlichkeitsmerkmale zu charakterisieren. Bei diesen fünf Merkmalen handelt es sich um Neurotizismus (das Verhalten bei negativer Erfahrung), Introversion/Extraversion (zwischenmenschliches Verhalten und Begeisterungsfähigkeit), Offenheit für Erfahrung (Interesse bzgl. neuer Erlebnisse und Eindrücke), Verträglichkeit (zwischenmenschliches Verhalten; Bezug auf andere) und Gewissenhaftigkeit (Ehrgeiz und Genauigkeit).
In einer empirischen Studie, die die Effekte der Gewissenhaftigkeitswerte bei Paaprogrammierung mittels des Five Factor Models untersucht hat, [NEJ10,] wurde die These aufgestellt, dass die Gewissenhaftigkeit das stärkste Wirkungsmittel für effektive Paarprogrammierung sei, denn schließlich ist dieses Merkmal das wesentlichste Prädikat für akademische Leistung. Testpersonen, mit unterschiedlich stark ausgeprägten Gewissenhaftigkeitsmerkmalen haben dementsprechende auch signifikant unterschiedliche Ergebnisse in Paarprogrammiersitzungen gezeigt [NEJ09,]. Angenommen wurde, dass Menschen mit hoher Gewissenhaftigkeit zu besseren Ergebnissen bei der Arbeit neigen — also bessere Resultate bei der Paarprogrammierung erreichen. Die Ergebnisse bestätigen das allerding nicht deutlich genug. Das prägnantere Merkmal, das zu einer besseren Paarprogrammierung geführt hat, war Offenheit für Erfahrungen. Diese Testpersonen waren vorausschauend und geist- sowie einfallsreich, was sich in den besseren Ergebnissen widergespiegelt hat [NEJ10,].
Auch bei der Nutzung von Facebook als Kommunikationsmedium wurde in einer andere empirischen Studie [CEea09,] untersucht, ob Nutzer mit hohen Offenheitswerten

dazu geneigt sind, Facebook als Kommunikationstool zu nutzen. Das konnte zwar nicht bestätigt werden, allerdings haben sie ein höher ausgeprägtes Sozialverhalten auf der Plattform.
Die Annahme lautet also, dass Programmierer mit hohen Offenheitswerten wahrscheinlich häufiger ein Help Request Feature nutzen würden als mit niedrigen Werten.
Für eine Bestätigung meiner These, wäre ein qualitativer Nutzertest erforderlich, für den im Rahmen dieser Arbeit kein Raum mehr war.
Der Einsatz für den Help Request ist für jeden Nutzer die Kommunikationsaufnahme. Das kann für Programmierer schwierig sein, da ihre Arbeit sonst zu großen Teilen ein stark nach innen orientierte Prozess ist. Ich vermute, dass sich die Zeitpunkte zwischen dem ersten Stadium — das Einsehen und dem zweiten Stadium — das Erkennen eines Problems unterscheiden. Mit anderen Worten: Selbst wenn die Funktion eines Help Requests vollständig gegeben ist, wird diese auch dann nicht zeitnah verwendet, wenn eingesehen wird, dass man Hilfe bräuchte. Also die Einsicht: „Ich brauche Hilfe“ und die Erkenntnis: „Jetzt hole ich mir Hilfe“ sind zwei verschiedene Zeitpunkte. Auch diese Annahme müsste mit einer aualitativen Nutzeranalyse belegt werden. Erfolgreiche Softwareentwickler sind in der Regel sehr ehrgeizig und präferieren eine zügige, komplikationsfreie, konzentrierte Arbeitsweise [PER,]. Auch bei auftretenden Problemen sind sie in der Lage durch Recherche die Lösungen selbst zu finden. Bei Problemen während eines zeitlich begrenzten Projekts ist dafür aber wenig Spielraum. Bis zu dem Punkt, an dem sie erkennen, aufgrund zeitlicher Defizite auf weitere Recherche zu verzichten und stattdessen Kollegen um Hilfe zu bitten, vergeht meist zu viel Zeit. Ehe sie den richtigen Ansprechpartner finden, der wirklich helfen kann und auch Zeit dafür hat, geht zusätzlich viel Arbeitszeit verloren. Der nächste Schritt ist, Programmierern zu helfen, erkennen zu können, ab welchem Zeitpunkt alleine keine schnelle Lösung gefunden werden kann, um daraufhin seine Kollegen um Hilfe zu bitten um gemeinsam Lösungen zu finden. Einen theoretischen Lösungsansatz wird im Ausblick zum Thema „Extrinsische Motivation“ beschrieben.

5 Szenario

Das Szenario beschreibt in welchem Zusammenhang und zu welchen Situationen das zu implementierende Help Request Feature benutzt werden soll. Wie einleitend geschildert, haben Programmierer Schwierigkeiten sich gegenseitig zu helfen, ohne Kollegen aus der Arbeit zu reißen und vorher komplexe Sachverhalte klären zu müssen. Selbst nach diesen beiden Schritten ist nicht gewährleistet, dass der gefragte Kollege der richtige Ansprechpartner ist. Sollte dem nicht so sein, muss die Suche von Neuem beginnen. Deswegen soll der Versuch dieser Forschungsarbeit sein, Saros um eine Help Request Funktion zu erweitern, die Entwicklern die Möglichkeit gibt, schnell und einfach Probleme auszuformulieren und an andere Entwickler einer Sitzung zu senden. Auf diese Art können in kürzerer Zeit mehr potenzielle Helfer erreicht werden. Der Saros Server [SARa,] ist ein bisher noch auf Prototypbasis implementiertes Feature, das angelegte Projekte als Sitzung bereitstellt, an denen beliebig viele Entwickler teilnehmen können. Der Vorteil dabei ist, dass die Codebasis immer aktuell ist, sobald sich man sich in die Session einwählt.



Der Server liefert eine Lösung für den eintretenden Problemfall, dass die Sitzung komplett beendet wird, sobald der Host[2] die Sitzung verlässt, selbst wenn noch andere Teilnehmer innerhalb dieser Sitzung arbeiten. Die Idee ist, ein Help Request an alle Teilnehmer innerhalb einer Saros Server Sitzung zu senden. Da der Saros Server momentan noch auf Prototypebene entwickelt wird, habe ich das Feature für die aktuelle Saros Version

[2] derjenige, der die Sitzung eröffnet hat

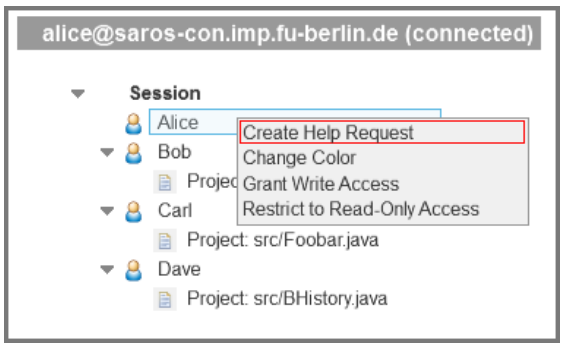
programmiert, in die der Saros Server noch nicht eingebunden ist. Help Requests werden dann über eine normale Sitzungsverbindung versendet und empfangen.
Zur Verdeutlichung des Anwendungsfalls dient das folgende Szenario:

Alice, Bob, und Carl arbeiten über den Saros Server in verschiedenen Sitzungen. Carl hat ein Problem bei seiner aktuellen Aufgabe und wählt über das Kontextmenü `Create Help Request` aus. Dabei öffnet sich ein Wizard mit einem Eingabefeld, worüber Carl eine Beschreibung seines Problems eingeben und absenden kann. Bei ihm in der Kontaktliste erscheint an seinem Namen ein Fragezeichen, damit er sieht, dass er einen unbeantworteten Help Request gesendet hat.
In den Kontaktlisten von Alice und Bob erscheint unter Carls Namen ein TreeItem mit einem Fragezeichen Icon und einem Textausschnitt der Request Nachricht. Per Doppelklick auf diesem Item sehen sie die ausführliche Request Nachricht.
Alice kann Carl nicht helfen und schließt das Fenster wieder. Bob hingegen hat einen Lösungsvorschlag und klickt auf den Button `I can help`. Bei allen Sitzungsteilnehmern weicht dem Fragezeichen Icon einem Ausrufezeichen, damit alle wissen, dass Carl jetzt Hilfe bekommt.
Carl bekommt zusätzlich eine Balloon Notification `Bob wants to help you`. Die gleiche Information steht bei allen Teilnehmern hinter Carls Kontaktnamen. Carl kann nun direkt mit Bob Kontakt aufnehmen und das Problem in einer Paarsitzung versuchen zu lösen. Ist das Problem behoben, kann Carl im Kontextmenü `Problem solved` auswählen und die Awarenessinformationen verschwinden bei allen Teilnehmern.

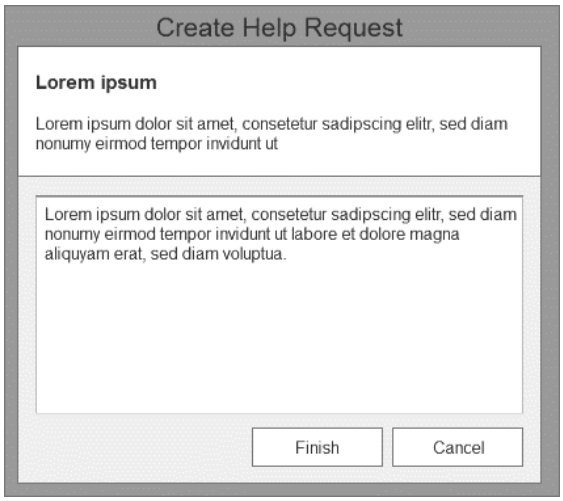
6 Konzept und Implementierung

6.1 Wireframes und Clickdummy

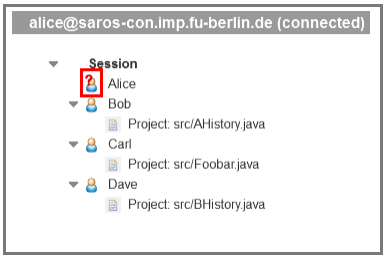
Da umfangreiche Programmierarbeiten keine Solokünstler, sondern gute Teamplayer zum erfolgreichen Abschluss führen und alle Programmabschnitte stark miteinander verkoppelt sind, brauchen Entwickler früher oder später Hilfe von ihren Kollegen. Je früher das erkannt wird, desto schneller ist das Problem behoben. Ungünstig ist falscher Ehrgeiz, wenn Programmierer glauben, sie finden die Lösung des Problems alleine. Dadurch verlieren sie oft unnötig viel Zeit. Aber wer der Experte für das jeweils auftretende Problem ist und ob diese Person dann auch Zeit für den Fragenden hat, sind die Fragen, die mithilfe eines Help Request Features eine modifizierte Form der Statuseingabe schnell Antworten liefern könnten. Darin sehe ich nicht nur einen großen Vorteil für die Implementierung dieses Features, sondern einen notwendigen Mehrwert, der den Vorgang des Hilfe-Holens verbessert und erleichtert. Der Fokus konzentriert sich auf eine bedarfsorientierte Statuseingabe in Problemsituationen. Also wer Hilfe braucht, der wird spätestens zu diesem Zeitpunkt eine Statuseingabe freiwillig eingeben wollen, denn schließlich verspricht sich der Entwickler etwas davon nämlich zeitnah von Wissenden Hilfe zu bekommen. Für eine klare Planung und Strukturierung des Implementierungsprozess habe ich vorab eine Clickdummy Version mit dem Programm Axure gebaut. Das hat mir die Möglichkeit gegeben verschiedene Ideen direkt miteinander zu vergleichen und mich gleichzeitig mit dem Aufbau, der Interaktion und der Darstellung des Features auseinander zu setzen. So konnte ich, sobald die Entwurfsideen abgeschlossen waren, diese evaluieren und direkt anpassen, ohne Zeitverluste durch aufwändiges Programmieren zu verlieren. Dieser Vorgang hat mir bei der Erstellung des Meilensteinplans geholfen, da ich so das Feature in kleine Bausteine unterteilen konnte. Auch bei Anpassungen hat mir der Clickdummy als Bauplan geholfen. Alle Konzeptentwürfe dienten mir als Orientierung. Mein Vorhaben war es, das Feature nach dem Konzeptentwurf sukzessive nachzubauen und zu implementieren. Zum Zeitpunkt der Abgabe war das Feature lokal implementiert, aber noch nicht im aktuellen Programm eingebunden. Die Idee war den lokale User (im Folgenden Alice genannt) sich in einer Session mit weiteren remote Usern (im Folgenden Bob, Carl und Dave) befinden zu lassen. Über das Kontextmenü kann Alice einen Help Request erstellen.

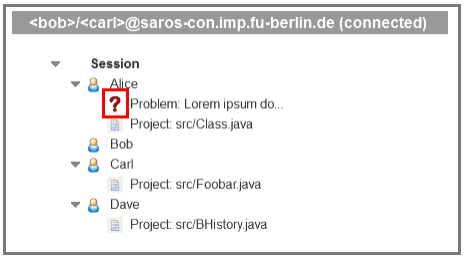


Daraufhin öffnet sich ein Wizard Fenster, das ein Texteingabefeld zur Verfügung stellt, in welches Alice ihr aktuelles Problem ausformuliert. Über den „Fertig“ Button sendet sie die Message an alle remote User in der Sitzung, also Bob, Carl und Dave. Um die Nachricht nachträglich zu editieren, klickt sie doppelt auf ihren eigenen Namen im Kontaktverzeichnis. Mit einem Rechtsklick ist sie in der Lage, die Anfrage zu löschen, sofern der Help Request nicht mehr aktuell sein sollte.

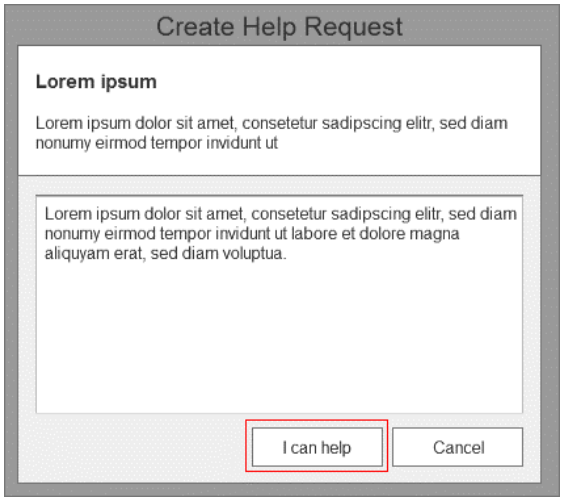


Bei einem aktuellen Help Request wird bei allen Empfängern ein TreeItem unter Alice' Kontakteintrag angezeigt, sodass alle wissen, dass Alice ein Request gesendet hat. Zusätzlich ist ein Ausschnitt der Request Nachricht zu lesen. Als Awareness Information wird ein rotes Fragezeichen-Icon bei allen Nutzern visualisiert, sodass ein Help Request von Alice sofort bei den Mitgliedern gesehen und leichter erkannt wird.

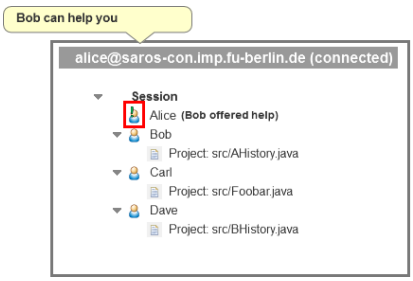


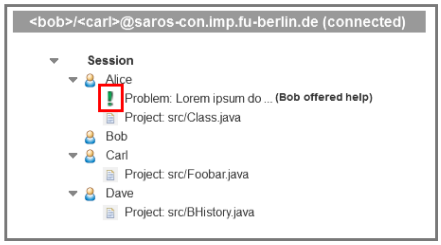


Bei weiteren Informationen sollen Bob und Carl auf das Blatt-Item mit einem Rechtsklick interagieren können, um sich das Texteingabefeld im read-only Modus anzeigen lassen zu können. Bob kann nun auf die Frage, ob er Alice helfen kann und möchte, mit dem Button I can help antworten.



Alice bekommt die Antwort in einer Balloon-Notification und als Textzusatz hinter ihrem Kontaktnamen angezeigt. Zusätzlich ändert sich bei allen Session Teilnehmern das rote Fragezeichen in ein grünes Ausrufezeichen. Alle Nutzer erfahren zusätzlich, welcher Nutzer Alice die Hilfe angeboten hat.



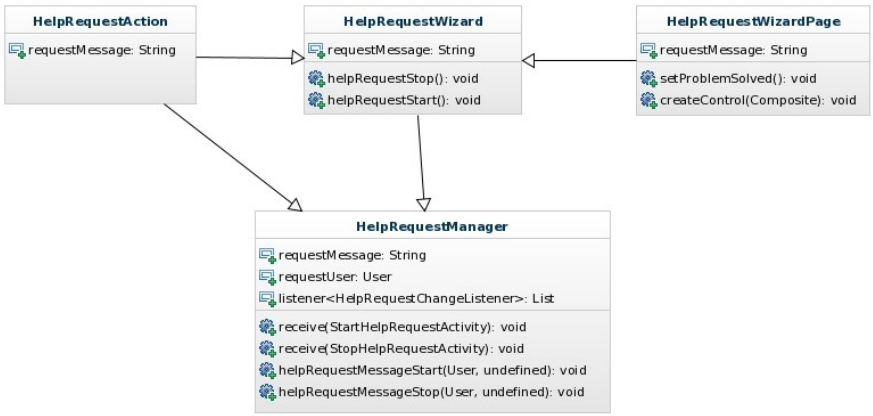


6.2 Implementierung

Für die Implementierung habe ich mich größtenteils an die Vorgabe des Konzeptentwurfs, welcher durch den Clickdummy dargestellt wurde, gehalten. Dabei habe ich mir

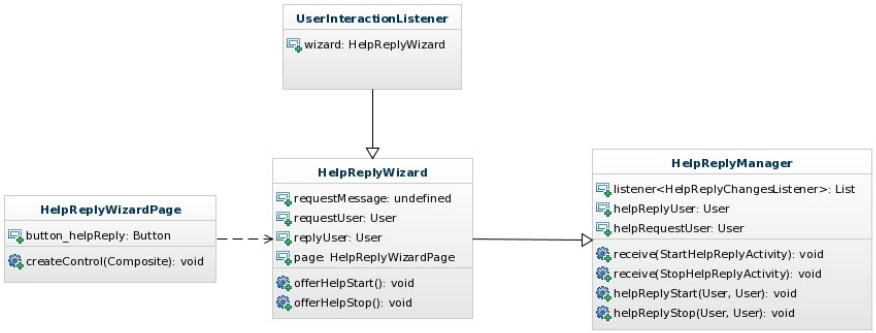
vorbehalten, dass mir dieser als Orientierung und nicht als strikte Vorlage dient. So habe ich in Ausnahmefällen gewisse Entscheidungen gefällt, die von dem Prototyp abweichen, bzw. noch nicht vollständig zum aktuellen Zeitpunkt implementiert worden sind. Im Folgenden beschreibe ich das Vorgehen der konkreten Implementierung.

6.3 Help Request



Der `HelpRequestWizard` wird dann aufgerufen, wenn der Nutzer die Call-to-Action Funktion `Create Help Request` im Interface und damit die Klasse `HelpRequestAction` (HRA) aufruft. Dann öffnet sich der `HelpRequestWizard` (HRW), der die `HelpRequestWizardPage` (HRWP) zum Erzeugen der Steuerelemente aufruft. Sollte der Nutzer einen aktuellen `Help Request` editieren wollen, so wird die `requestMessage` an den HRW und HRWP weitergegeben und im Eingabefeld angezeigt, sonst ist das Feld leer. Im Editiermodus kann der Nutzer im Wizard auf den Button `delete Help Request` den aktuellen `HelpRequest` über den Methodenaufruf `setProblemSolved()` als gelöst markieren. Der HRW beendet dann den aktuellen `Help Request` mit dem Aufruf `helpRequestStop()`. Wird auf den Call-to-Action Button `Finish` geglickt, so wird dieser gestartet bzw. aktualisiert. `helpRequestStart()`ist der Aufruf dafür. Wird ein `HelpRequest` gestartet oder beendet, speichert der aufgerufene `HelpRequestManager` (HRM), alle `Requests` in einer Liste. Wird dann je nach Methodenaufruf eine `Activity` empfangen, werden die Informationen von `requestMessage` und `requestUser` an alle anderen Nutzer innerhalb einer `Session` geschickt und über den `AwarenesCollector` Anpassungen im Interface aktualisiert.

6.4 Help Reply



Nachdem ein `HelpRequest` empfangen wurde, hat jeder `remote User` die Möglichkeit, sich zum einen die `requestMessage` genauer durchzulesen. Dafür klickt er doppelt auf das entsprechende `TreeItem`. Im Code wird in der `UserInteractionListener` Klasse ein `HelpReplyWizard` (HRW) erzeugt, der in der `HelpReplyWizardPage` (HRWP) die Interface-Elemente erstellt. Entscheidet sich der User dem `Request` zu antworten, klickt er auf dem Button `I can help`, was zur Auswirkung hat, dass im HRW die Methode `offerHelpStart()` aufgerufen wird. Der `SessionManager` erzeugt dann einen `HelpReplyManager`, der analog zum `HelpRequestManager` alle Antworten in einer Liste speichert. Die `receive()` Methoden senden die Information an den `requestUser` und die `remote User` in der `Session`, was Anpassungen des Interface zur Folge hat.

6.5 Kontaktliste

Die Klasse `HelpRequestTreeElement` wird dann aufgerufen, nachdem ein `Help Request` an die `remote User` gesendet wurde. Ein Ausschnitt der `requestMessage` wird im `Roster` angezeigt und abhängig von einem vorhandenen `Reply` ändert sich das Awareness-Icon im `Roster` von einem roten Fragezeichen zu einem grünen Ausrufezeichen.

Zum Zeitpunkt der Abgabe waren folgende Funktionalitäten noch nicht implementiert:

- Icon Anpassung im Interface des `requestUser` nach Empfang eines `HelpReply`
- Information im Interface, welcher User ein `HelpReply` gesendet hat
- zusätzliche `Balloon Notication` beim `requestUser`, nachdem ein `HelpReply` gesendet worden ist.

7 Ausblick

In diesem Abschnitt der Bachelorarbeit wird die Idee um Anwendungsfälle für den Help Request erweitert. Im dritten Kapitel wurde bereits beschrieben, welche intrinsische Motivation Nutzer haben, wenn sie einen Help Request erstellen: Wenn Entwickler einen freien Paarprogrammierer suchen oder wenn sie Probleme haben, die sie alleine nicht lösen können. Die Anwendung des Help Requests dient als Hilfsmittel für solche Szenarien. Intrinsische Motivation entspringt ausschließlich dem Selbstinteresse und aus reinem Eigenantrieb, der aus der Aufgabe selbst erfolgt. Im Gegensatz zu extrinsischer Motivation, welche durch äußere Anreize stimuliert wird. Die Entscheidungen, die aus dieser Motivationsform resultieren, sind eine weitere Möglichkeit, Tätigkeiten umzusetzen, wenn Menschen nicht durch diese allein motiviert [WPG,] sind. Aufgrund der Annahme, dass ein längeres Zögern einem Help Request vorausgeht, stellte sich die Frage, wie man Nutzer beeinflussen kann, diesen zeitlichen Rückstand zu reduzieren. Dafür haben sich zwei Ideen entwickelt: Zum einen besteht die Möglichkeit, dass der Nutzer von jemand anderem aufgefordert wird, der viel früher erkennt, dass ein Problem besteht, welches der Entwickler selbst nicht zu lösen weiß. Durch diese Aufforderung wird die Anwendung der Funktion wahrscheinlich beschleunigt. Zum anderen kann man die Interaktion nutzernah gestalten, dass durch ein Belohnungssystem die Hemmschwelle eines Hilferuf senkt.

7.1 Systemerkennung

Für den Fall der verteilten, sowie traditionellen Paarprogrammierung existiert bereits eine externe Person, die auf Problemsituationen hinweist: das ist der Observer. Für den anderen Fall der distributed party programming oder klassische Einzelprogrammierung existiert keine externe Person, die auf den problemfreien Programmierprozess achtet. Das Hinzufügen eines unbeteiligten Beobachters eines Systems, das die Aufgabe hat, solche Momente vorauszuahnen und den Programmierer Hinweise liefert, veranschaulicht die Idee eines „unsichtbaren“ Oberserver. Inspiriert durch Dostojewski [3] *Schuld und Sühne* Interpretation, erreicht man erst durch das Geständnis des eigenen Scheiterns einen unvoreingenommenen Blick auf die Wirklichkeit [Nie00,]. Abstrahiert auf die Softwareentwicklung heißt das, erst wenn der Programmierer sein Nichtwissen erkennt und akzeptiert, kann er mit Hinblick auf das Projekt seine Arbeit erfolgreich fortsetzen. Die

[3] russischer Schriftsteller des 19. Jahrhunderts im Russischen Kaiserreich

Systemerkennung beschleunigt dabei nur den Vorgang der Selbsterkenntnis. Zudem stellt sich ebenfalls die Frage, inwieweit diese Anwendung erfolgsversprechend funktionieren könnte. Eine vollständige technische, sowie User Experience Analyse im Rahmen eines Nutzertest könnten wichtige Anhaltspunkte vor, während und nach einer Umsetzung liefern. Schwierig zu definieren ist, welche Erkennungskriterien das System erfüllen müsste. Es könnte nach folgenden Problemsituationen erkannt und nachfolgend eine Aufforderung einer externen Hilfsanfrage gesendet werden, z.B. bei

- der dritten fehlgeschlagene Kompilierung
- häufigen Ausführungen von Rückgängig `CTRL+Z` und Wiederholen `CTRL+Y` im Editor
- Code Churn [4] [CODb,]

Es existiert bereits Software, die Ansätze eines Hilfssystems implementiert haben. Ein bekanntes Beispiel ist das Betriebssystem Windows. Bei Problemen mit der Netzwerkverbindung werden dem Nutzer schrittweise Lösungsmöglichkeiten angeboten. Wenn keines davon erfolgreich war, erscheint die Aufforderung, den Netzwerkadministrator um Hilfe zu bitten. In der Regel wenig zufriedenstellend, aber ein ähnliches Prinzip. Ein ähnliches Verfahren liefert Windows Live Mail bei der Passwortwiederherstellung. Wenn keine SMS oder E-Mail empfangen wird oder die Sicherheitsfragen nicht beantwortet werden konnte, wird ein Support Kontakt angeboten, an den man sich wenden kann. Auch für die Softwareentwicklung gibt es Ansätze um Problemdichten zu berechnen. Visual Studio 2013 beispielsweise erstellt einen Code Churn Report [CHU,]. Auch Microsoft hat ein Tool für eine frühe Defekterkennung statischer Analysen entworfen [NT05,]. An der Saarland Universität wurde die Frage untersucht, warum einige Programme fehleranfälliger sind als andere [TRA07,]. Das Thema der automatischen Fehler- und Defekterkennung reißt einen eigenen, umfangreichen Forschungsbereich an [BD01,].

7.2 Gamification

Das andere, bereits erwähnte Anwendungsszenario, Nutzer zu motivieren, das Help Request Feature zu benutzen, wäre die Umsetzung von Gamification Elementen. Also Nutzer auf spielerische Art die Möglichkeit geben, den vollen Funktionsumfang zu nutzen.

[4] zum wiederholten Mal wird die selbe Zeile im Code neu beschrieben

Ein Gamification Ansatz wäre dafür nicht nur für den Hilferuf interessant, sondern für Saros im Ganzen. Alltagssituationen nutzen immer häufiger, zum Teil sehr erfolgreich Gamification Elemente. So wird man beispielsweise von einem Smiley angezwinkert, wenn man sich besonders gut seiner Zahnpflege gewidmet hat [BRA,]. Streetpong [PON,]ist ein in Ampeln integrierter Computer, der an beiden Enden Fußgänger zu Spieleklassiker Pong auffordert, womit die Wartezeit für Fußgänger während der roten Ampelphase nicht nur verkürzt wird, das Warten macht zudem auch noch Spaß. Dies sind nur wenige Beispiele sehr erfolgreicher Gamification- Elemente, welche im Alltag schon ihren Platz gefunden haben. Auch in der Softwareentwicklung findet dieses Thema starken Anklang. Wie beispielsweise

- confluence Karma (Punkte, Badges) [KAR,]
- Visual Studio Achievements (Badges) [VIS,]
- Coderwall (Team Profile; über Github; Badges, Points) [CODa,]
- Stackoverflow (Status, Reputationspunkte, Level) [STA,]

Confluence hat das Problem erkannt, dass Dokumentation von Projekten gerne vernachlässigt wird. Deswegen hat Confluence ein Plugin entwickelt, dass User anspornt in einen spielerischen Wettbewerb unter Kollegen anzutreten. Je aktiver die Anwendung benutzt wird, desto höher ist der Grad der Auszeichnung. Über ein Leaderboard kann man sich direkt mit Kollegen vergleichen. Karma Punkte spiegeln die Aktivität in confluence wieder, kreative und lustige Badges gibt es für bestimmte Aktivitäten und Medals sind eine Form von Reputationspunkte, welche direkt von Kollegen untereinander vergeben werden.

Visual Studio hat die Erweiterung Achievments entwickelt, das verschiedene Kategorien von Badges für das Erreichen von bestimmten Programmierungsarbeiten freischaltet. Für jede Kategorie wird eine Seite mit eigenem Profilbild erzeugt und mit kreativen Beschreibungen die Fähigkeit des Entwicklers beschrieben. Diese kann man sogar auf sozialen Netzwerken teilen. Mit *Badges* wie „Go to hell“ für das Verwenden des Schlüsselworts `goto`, „Lonely“ für das Entwickeln am Wochenende oder „Scrollbar Wizard“ für das kodieren einer 300 Zeichen lange Zeile sind diese Badges mit Humor zu verstehen,

die einen neuen Spaßfaktor während des Programmierens liefern.

Coderwall beruht auf dem ähnlichen Prinzip wie dieses Plugin. Auch hier kann man Badges freischalten. Der Unterschied ist, dass man bei Coderwall sein Team repräsentieren kann. Der soziale Aspekt wird dabei enorm gefördert, insbesondere bei Open Source Projekten, da man sich direkt über Github damit verknüpfen kann.

Stackoverflow ist die erfolgreichste Gamification-Anwendung unter Softwareentwicklern. Die Grundidee ist, dass man über eine Plattform Fragen und Antworten zu programmatischen Themen stellt. Hier werden auf grafische und kreative Badge-Elemente wie bei Coderwall, confluence Karma oder Visual Studio Achievements verzichtet. Der Kernerfolg kommt durch die Reputationspunkte. Diese dienen der Bewertung von Fragen und Antworten, und spornen zur regen Teilnahme an. Je mehr Punkte man erhält, desto mehr Rechte erwirbt man für die Plattform. Mit 2000 Reputationspunkten kann man beispielsweise eine Frage eines anderen Users editieren. Das „Nice Question“-Abzeichen gibt es, wenn 10 weitere Nutzer die Fragen positiv bewertet haben.

Die Idee, die Programmierung mit Gamification-Elementen zu erweitern ist also nicht neu. Gamification soll die Kernaufgabe attraktiver machen. Das wäre auch in Hinblick auf Saros analog umsetzbar. Da Saros meist nur für kleinere Projektaufgaben benutzt wird, sollen Nutzer so fester an Saros gebunden werden. Gamification soll dem Zweck dienen, im Team Motivation, Integration und Kollaboration zu verbessern. Mit anderen Worten, Nutzer sollen dafür belohnt werden, gewisse Features zu benutzen, was den Programmierprozess ähnlich erheiternd bereichert wie es Visual Studio umgesetzt hat, aber auch eine fachliche Verbesserung des Projekts, wie bei Stackoverflow, wird gegeben. Auch das Wissen und Erkennen des umfangreichen Feature-Angebots könnte über einen Gamification-Ansatz optimiert werden. *Partial Sharing* z.B. ist eine Funktion, die von vielen Nutzern gewünscht wird. Dass es diese bereits gibt, wissen die meisten Nutzer anscheinend nicht. Anhand eines Badge Systems könnten Saros Entwickler so ein direktes Feedback über den Funktionsumfang des Programms erhalten. Gleichzeitig sehen Nutzer bei ihren Kollegen, welche Badges sie noch nicht freigeschaltet haben, weil von dessen Funktionen nichts bekannt war. Im Folgenden wird die Idee für eine Gamification

Umsetzung beschrieben.

7.2.1 Ansatz für Saros

Eine eigene Profilseite ist der erste Schritt um den Nutzer an Saros zu binden. Das Ausfüllen der Profilseite gibt dem Nutzer das erste Gefühl von Erfolg. Ein erstes grafisches Beispiel für ein Profil habe ich im dritten Kapitel „Vorüberlungen“ skizziert. Ein Leaderboard dient zu einem direkten Vergleich mit anderen Saros Nutzern. Da Saros ein Team-Tool ist, empfiehlt sich der Fokus, sich mit den Mitgliedern zu verknüpfen. Ein Leaderboard könnte so aufgebaut sein, dass es nach einer Anwendungsperiode den momentan besten Entwickler einen Entwickler des Monats ausgezeichnet wird. Gleichzeitig erhalten dadurch die anderen Teilnehmern Informationen zum Vergleich und die Chance zur Verbesserung für die nächste Periode. Bei Visual Studio gibt es die Möglichkeit, seinen Status aufgrund individueller Fähigkeiten zu verbessern, jedoch kann dieser auch wieder verschlechtern werden. So könnte auch bei Saros ein Status umgesetzt werden, z.B.

- Source Control Explorer — Erstes Session Teilnahme an einem Projekt
- Newbie — alle Grundfunktionalitäten entdeckt
- Explosive Hand — die meisten Bugs einer Iteration gefunden
- Eagle Eye — die meisten Code Reviews in einem Release gemacht
- Code Monkey — Mehr als 1000 eingecheckte Zeilen Code für die laufende Iteration

Bei allen Beispielen gab es die Möglichkeit sich Abzeichen zu verdienen. Wichtig dabei ist ein nicht zu stark und nicht zu schwach steigender Schwierigkeitsgrad. Beides kann das „Spiel“ ruinieren. Folgende Beispiel-Badges können einmalig vergeben für das Entdecken von gewissen Funktionen werden.

- Registrierung (Willkommen) und Profil vervollständigt
- erster Kontakt / erstes Projekt im (Saros Server) / erste Mal zu Projekt hinzugefügt
- Erstnutzung von Chat / Whiteboard / Partial Sharing / Konsole geteilt / Benutzerfarbe geändert / Sprung zu Position / Rollentausch (Follow Mode; Read Only)

- Feedback gepostet

An dem Visual Studio Achievements ist der spaßige Überraschungsfaktor in das Spiel gut integriert. Um das Spiel am Laufen zu halten und den Nutzer Easter Eggs [EAS,] aufzutischen, können folgende Badge Beispiele Inspiration für die Saros Anwendung liefern:

- Programmierung ab 23:00 („sleepwalker“)
- 10 Stunden online („workaholic“)
- 3 mal innerhalb einer Stunde inaktiver Status („ADHD victim“)
- >10 Tabs geöffnet („scatterbrain“)
- >500 Zeichen auf einmal auskommentiert („coding messy“)
- >1000 Zeilen Code („poet“)
- 5. mal hintereinander compiler fail („stubborn head“)
- Kommentar mit mehr als 200 Zeichen („beat about the bush“)
- Session verlassen / Programm beendet nach einem compiler fail („hot head“)
- >10 Interfaces in einem Projekt erstellt („architect“)
- 10 Variablen innerhalb einer Methode („memory genius“)
- Copy & Paste bei >300 Zeichenlänge („lazybones“)
- >5 mal zu Position gejumped („lost“)
- ein Jahr bei Saros („happy anniversary“)
- follow-mode seit 3 Stunden („stalker“)
- followed-mode seit 3 Stunden („vip“)
- online an Valentinstag („lonely heart“)
- online am Geburtstag („happy birthday“) + Geburtstagsmelodie
- online an Weihnachten („hohoho“)

Das Erfolgsrezept bei Stackoverflow sind die minimalistisch designten Reputationspunkte. Diese sind ausschlaggebend für die Stelle im Leaderboard. Erfahrungspunkte können sich ergeben durch individuelle Leistung:

- Anzahl der gejointen / erstellten Projekte
- Anzahl der Kontakte
- Anzahl der Stunden in einer Session
- Help Request gesendet
- Help Reply gesendet
- Anzahl der commits

Das Prinzip des unsichtbaren Observers wäre auch für den Gamification- Ansatz relevant. Denn dieser loggt mit, welches Programmierverhalten vom Nutzer umgesetzt wird. Diese beiden Techniken sprechen nicht jeden Programmierer an. Deswegen sind sowohl die Gamification- Erweiterung als auch die Systemerkennung optionale Features. Vermutlich würden nur Programmierer mit hohen Werten für Extraversion (Begeisterungsfähigkeit) und Offenheit für Erfahrungen die Umsetzung als interessant einstufen. Das sind nur Annahmen, welche erst in einer qualitativen Analyse bestätigt werden müssten.
Während des Konzeptentwurfs insbesondere nach dem Antrittsvortrag [ANT,] fehlten Szenarien, die begründeten, warum ein Help Request tatsächlich ein nutzbares Feature wären. In einer Mindmap [XMIb,] habe ich die Szenarien visualisiert und im Laufe in dieser Abschlussarbeit beschrieben. Dabei habe ich mich mit den beiden Szenarien (Systemerkennung und Gamification) zwar beschäftigt, es zeigte sich aber, dass diese so umfangreich waren, dass diese nur gedanklich skizziert, aber nicht umgesetzt habe.

8 Fazit

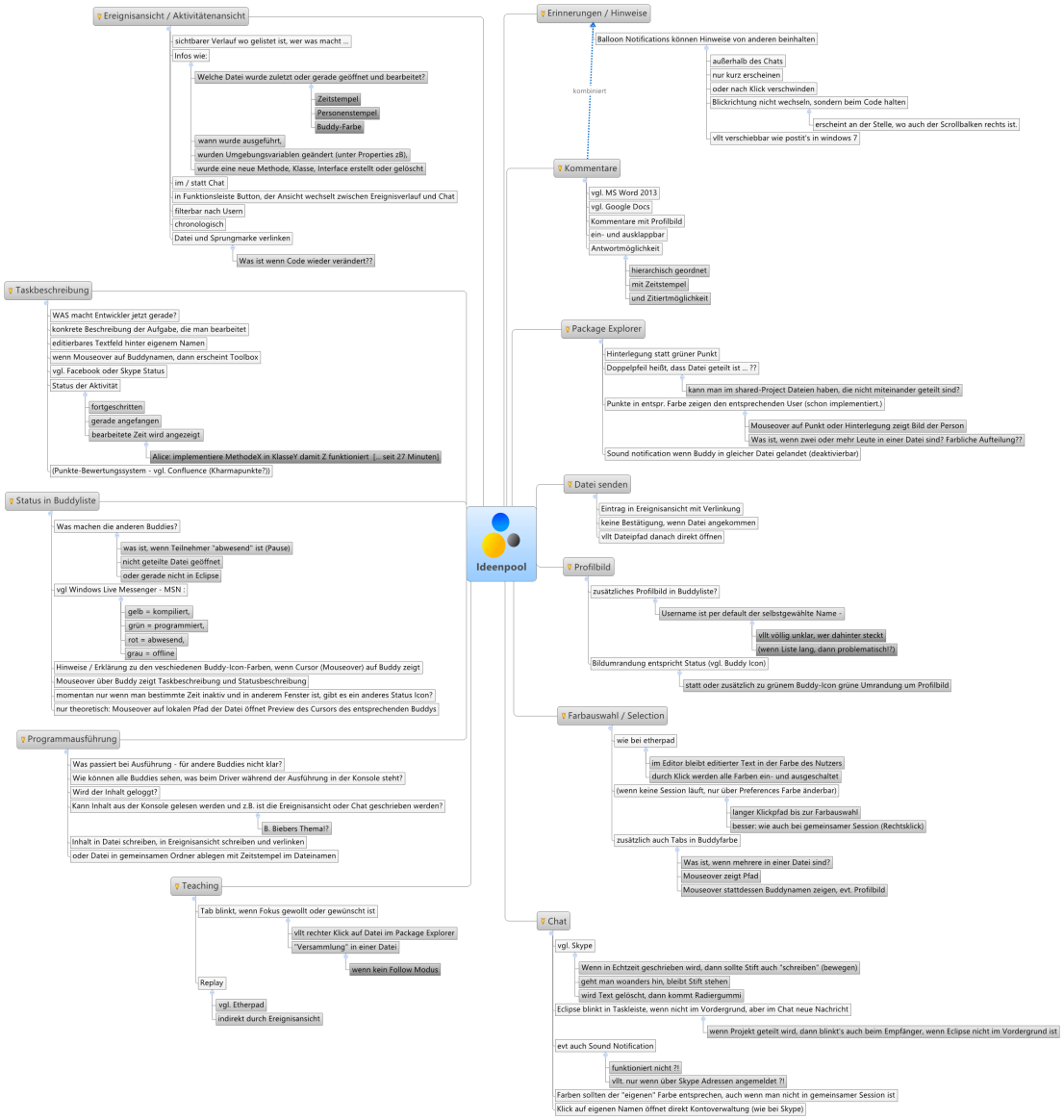
Diese Bachelorarbeit hat zusammenfassend einige Fragen thematisiert. Die Ausgangsfrage dieser Forschungsanalyse lautete: Wie kann man mithilfe von Awarenessinformationen die Kommunikation zwischen Saros Nutzern verbessern? Dafür habe ich wissenschaftliche Forschungsergebnisse studiert und anhand derer das mentale Modell von verteilten Paarprogrammierern erstellt, um deren Nutzerverhalten zu verstehen. Dieses zeigte, dass der allgemeine Wunsch besteht, Kommunikationsbremsen besser und einfacher zu lösen. In einem Ideenpool sammelte ich verschiedene Optimierungsmöglichkeiten für Kommunikationsansätze. Die Idee einer Status Funktion habe ich dabei fokussiert, da sowohl während dieser Abschlussarbeit ein echtes Nutzerfeedback genau diesen Wunsch geäußert hatte, als auch von anderen Programmierern ein grundlegendes Interesse besteht, über aktuelle Aufgaben von Kollegen Bescheid wissen zu wollen [ST,]. Resultierend habe ich Überlegungen formuliert, in welchen Fällen Saros Anwender eine Status Funktion nutzen könnten. Das wäre in den Fällen, wenn

1. Entwickler einen freien Paarprogrammierer suchen,
2. externe Hilfe benötigt wird,

Ich habe den sinnvollsten Optimierungsbedarf bei auftretenden Problemfällen priorisiert, da ich der Annahme war, dass dieser Fall eher eintritt, als die Suche nach einem Paarprogrammierer. Dafür habe ich ein Konzept entworfen, dass ich mit interaktiven Prototypen visualisiert habe. Das lieferte mir die Vorlage für die Implementierung. Während der Arbeit am Saros Code habe ich den zeitlichen Umfang der Einarbeitung unterschätzt. Auch die parallele Arbeit am Source Code der anderen Entwickler, insbesondere die Arbeit eines umfangreichen Refactorings brachte mir, durch regelmäßige Aktualisierungen des Codes, zusätzliche Arbeit. Dadurch blieben Code Fragmente lange auf der lokalen Festplatte liegen, was in einem hohen Maß Mehraufwand endetet, da die Review-Patches für die anderen Saros Entwickler übersichtlich, lesbar und gut dokumentiert werden müssen. Nach der Implementierungsarbeit habe ich in einem Ausblick theoretisch mögliche Ansätze ausgebaut, wie man dieses Feature und Saros als Ganzes optimieren könnte.

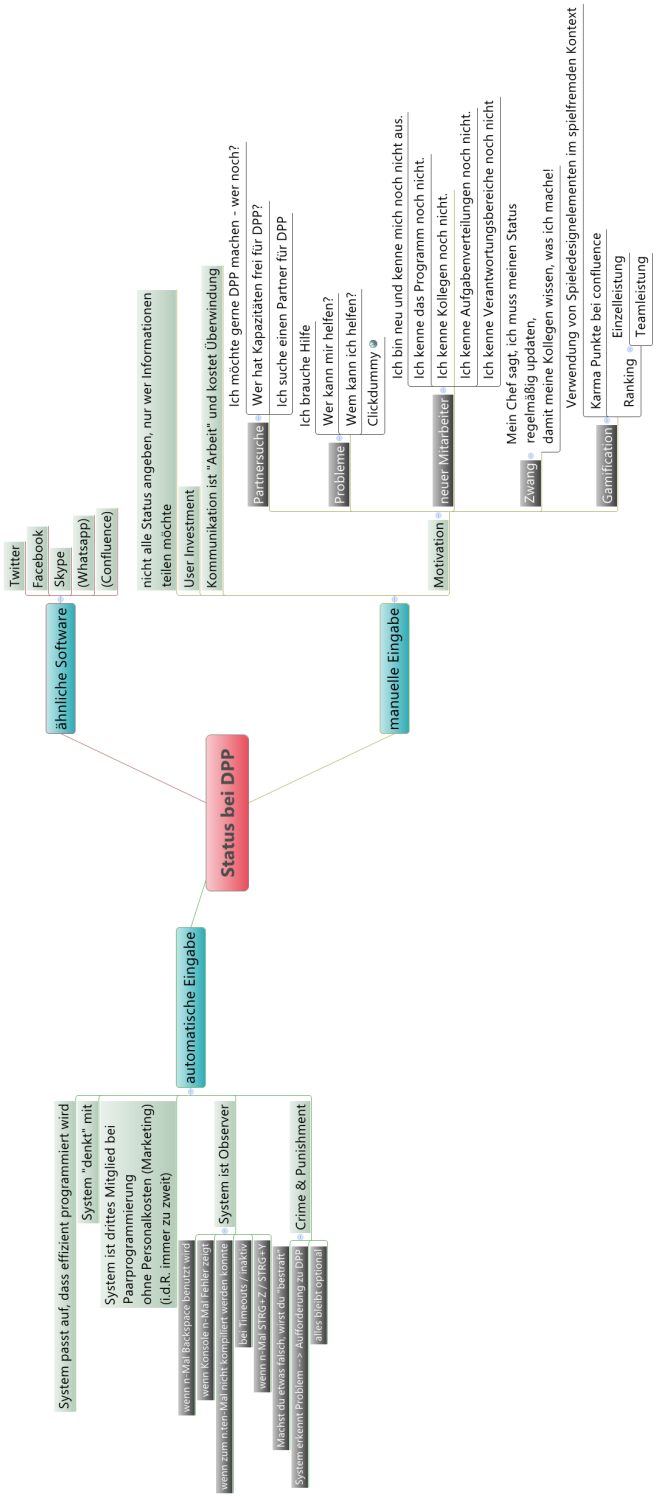
9 Anhang

9.1 Mindmap: Ideensammlung zur verbesserten Kommunikation mithilfe von Awarenessinformationen



http://www.xmind.net/m/JX6Z/

9.2 Mindmap: intrinsische und extrinsische Motivation



http://www.xmind.net/m/sJJM/

Literatur

[AL00] Alistair Cockburn and Laurie Williams. The Costs and Benefits of Pair Programming. *In eXtreme Programming and Flexible Processes in Software Engineering XP2000*, 2000.

[ANT] Antrittsvortrag Formisano, http://www.inf.fu-berlin.de/inst/ag-se/teaching/s-bse/281_formisano_verbesserte-kommunikation-concept.pdf.

[AWA] Awareness Definition, http://dictionary.cambridge.org/dictionary/british/awareness.

[AXU] Axure, http://www.axure.com.

[BD01] Brad Clark and Dave Zubrow. How Good is the Software, A Review of Defect Prediction Techniques. 2001.

[BRA] Braun Zahnbürste nutzt Gamification,http://braun.pgshop.de/mundhygiene/4210201852810-oral-b-power-elektrische-zahnbuerste-kaufen/pg_4210201852810,de_de,pd.html?cgid=braun-oral-all.

[Bri02] Brian F. Hanks. Tool Support for Distributed Pair Programming. 2002.

[CEea09] Craig Ross, Emily S. Orr, and Mia Sisic et al. Personality and motivations associated with Facebook use. *ScienceDirect*, 2009.

[CHU] Code Churn Report in Visual Studio, http://msdn.microsoft.com/en-us/library/ms244661.aspx.

[CM96] Carl Gutwin and Mark Roseman. A usability study of workspace awareness widgets. *Human Factors in Computing Systems: Conference Companion*, pages 214–215, 1996.

[CODa] Gamification bei Coderwall, https://coderwall.com/welcome.

[CODb] What is Code Churn?, http://blogs.msdn.com/b/askburton/archive/2004/09/09/227515.aspx.

[CON] Feature Implementierung konsolenteilen, http://www.inf.fu-berlin.de/w/se/thesisdppsharedconsole.

[CS02] Carl Gutwin and Saul Greenberg. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Computer Supported Cooperative Work*, pages 411–446, 2002.

[EAS] Easter Egg, http://de.wikipedia.org/wiki/easter_egg.

[ECL] Eclipse, http://www.eclipse.org/.

[ETH] Etherpad, http://etherpad.org/.

[GAM] Gartner Gamification Report: 80From Poor Design, http://www.cmswire.com/cms/social-business/gartner-gamification-report-80-of-implementations-will-fail-by-2014-from-poor-design-018424.php.

[JAC+08] John Vijay Sena Devide, Andrew Meneely, Chih-Wei Ho, Laurie Williams, and Michael Devetsikiotis. Jazz Sangam: A real-time tool for distributed pair programming on a team development platform. 2008.

[JIR] Jira, https://www.atlassian.com/software/jira.

[JLS13] Julia Schenk, Lutz Prechelt, and Stephan Salinger. Distributed-Pair Programming can work well and is not just Distributed Pair-Programming. 2013.

[KAR] Gamification bei Confluence, https://marketplace.atlassian.com/plugins/-de.communardo.confluence.plugins.karma.

[NEJ09] Norsaremah Salleh, Emilia Mendes, and John Grundy. An Empirical Study of the Effects of Personality in Pair Programming using the Five-Factor Model. 2009.

[NEJ10] Norsaremah Salleh, Emilia Mendes, and John Grundy. An empirical study of the effects of conscientiousness in pair programming using the five-factor personality model. 2010.

[Nie00] Michael Niehaus. Geständniszwang, Überlegungen zu einer Theorie des Geständnisses. *Kriminologisches Journal 32.1, S. 2-18*, 2000.

[NT05] Nachiappan Nagappan and Thomas Ball. Static Analysis Tools as Early Indicators of Pre-Release Defect Density. *Association for Computing Machinery, Inc.*, 2005.

[OPE] OpenCourseWorld Fraunhofer IESE, UX Design for Business, Modul 6 Gamification, Skript, http://www.opencourseworld.de/pages/syllabus.jsf?courseid=824440.

[PER] Persönlichkeit von Softwareentwicklern, http://msdn.microsoft.com/de-de/magazine/dd252951.aspx.

[PON] StreetPong in Berlin, https://www.youtube.com/watch?v=oe3tsxvsnpg.

[SARa] Feature Implementierung Saros Server, http://www.inf.fu-berlin.de/w/se/thesissarosserver.

[SARb] Saros, http://www.saros-project.org/.

[SKY] Skype http://www.skype.com.

[ST] Stephan Lukosch and Till Schümmer. Enabling Distributed Pair Programming in Eclipse.

[STA] Gamification bei Stackoverflow, http://stackoverflow.com/.

[TRA07] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. Predicting Defects for Eclipse. *Proceeding PROMISE 2007 Proceedings of the Third International Workshop on Predictor Models in Software Engineering, S. 9*, 2007.

[TS09] Till Schümmer and Stephan Lukosch. Understanding Tools and Practices for Distributed Pair Programming. *Journal of Universal Computer Science*, (15):3101–3125, 2009.

[VIS] Gamification bei Visual Studio 2013, http://channel9.msdn.com/achievements/visualstudio.

[WPG] Wirtschaftspsychologische Gesellschaft, intrinsische und extrinsische Motivation, http://www.wpgs.de/content/view/584/368/.

[XMIa] Mindmap zur Visualisierung der Ideengenerierung, http://www.xmind.net/m/jx6z/.

[XMIb] Mindmap zur Visualisierung der Motivation für den Help Request, http://www.xmind.net/m/sjjm/.