Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Erhebung von Benutzerfeedback aus der Nutzung eines Werkzeugs zur verteilten Paarprogrammierung

Lisa Dohrmann
Matrikelnummer: 4130066
dohrmann@inf.fu-berlin.de

Betreuer: Christopher Oezbek und Stephan Salinger
Eingereicht bei: Prof. Dr. Lutz Prechelt

Berlin, 31. Juli 2009

Zusammenfassung

Diese Bachelorarbeit hat zum Ziel, Rückmeldungen von den Benutzern des Werkzeugs *Saros* zur verteilten Paarprogrammierung in Eclipse zu erheben. Dies umfasst zum einen eine Umfrage, die in Intervallen am Ende einer Saros-Sitzung dem Nutzer angeboten wird, zum anderen die Übertragung von statistischen Informationen, die während einer Sitzung gesammelt werden. Die vorliegende Arbeit beschreibt Konzeption, Implementierung und Einsatz der für Saros hierzu entwickelten Erweiterungen.

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.

Berlin, den 31. Juli 2009

Lisa Dohrmann

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Bachelorarbeit befasst sich mit der Erhebung von Benutzerfeedback zu dem Eclipse-Plugin *Saros*, das an der Freien Universität Berlin anfänglich in der Diplomarbeit von Riad Djemili [Dje06] entwickelt und seitdem in mehreren Arbeiten [Gus07][Rie08][Jac09][Rin09] innerhalb der AG Software Engineering ausgebaut und weiterentwickelt wurde.

Saros ist ein Open Source Werkzeug zur verteilten, kollaborativen Echtzeit-Programmierung in Eclipse. Es unterstützt sowohl das Konzept der klassischen, verteilten Paarprogrammierung (engl. *Distributed Pair Programming, DPP*) [BGS02], bei der es einen Schreiber (engl. *driver*) und einen Beobachter (engl. *observer*) gibt, die räumlich getrennt zusammen dasselbe Projekt bearbeiten als auch die gleichzeitige, kollaborative Quelltextbearbeitung in Echtzeit durch mehrere Entwickler (engl. *Side-By-Side Programming, SbS*) [NJOL05].

1.1 Ziele

Saros ist ein Forschungsprojekt der AG Software Engineering. Mithilfe dieses Projekts sollen die Konzepte der Paarprogrammierung und der verteilten Paarprogrammierung untersucht und besser verstanden werden, um erklären zu können, wie diese Arbeitstechniken am effizientesten einzusetzen sind [PPG]. Um dieses Ziel zu erreichen, wurde ein prozessbasierter Ansatz gewählt, d. h. ein Ansatz, der versucht genau zu untersuchen, was während einer (verteilten) Paarprogrammierungssitzung im Detail passiert.

Saros kann auf zwei Arten zum Erreichen des Forschungsziels beitragen:

1. Durch die detaillierte Analyse einzelner Paare, die mit der Arbeitsgruppe Software Engineering kooperieren, wie dies z. B. gerade von Edna Rosen in ihrer Diplomarbeit bei der Firma Teles durchgeführt wird [Ros09].
2. Durch eine Veröffentlichung von Saros im Internet können Nutzungsstatistiken von für die Arbeitsgruppe anonymen Benutzern erhoben werden.

Beide Wege bieten neben der Erforschung von verteilter Paarprogrammierung auch die Möglichkeit, Rückmeldungen der Benutzer über ihre Erfahrungen bei der Verwendung von Saros zu erhalten, um daraus Verbesserungen für das Produkt ableiten zu können.

In dieser Arbeit habe ich mich mit dem Entwurf und der Implementierung eines Systems zum Sammeln von Benutzerfeedback in Saros beschäftigt. Das Ziel dabei ist, zum einen Daten zu erheben, die der Forschung im Bereich

der verteilten Paarprogrammierung und zum anderen der Verbesserung von Saros dienlich sind.

Unter Benutzerfeedback sind hierbei im Rahmen dieser Arbeit all diejenigen Daten zu verstehen, mit denen etwas über die *Nutzer* (Fakteninformationen), das *Nutzungsverhalten* (Verhaltensweisen im Umgang mit einer Software, Verhaltensintentionen, Ziele des Einsatzes) und über die *Bewertung* einer Software (Meinungen der Nutzer) in Erfahrung gebracht werden kann.

Die einfachste Möglichkeit vielfältige Informationen über die Benutzer und ihre Erfahrungen zu erhalten ist, sie direkt zu befragen. Dabei muss jedoch die Subjektivität vieler Antworten [BM01], insbesondere auf Fragen zum Nutzungsverhalten, beachtet werden. Um objektive Informationen über die tatsächliche Nutzung einer Software zu erhalten, gibt es hingegen nur die Möglichkeit, die Benutzer beim Einsatz der Software „zu beobachten“, d. h. Daten, die das Verhalten des Nutzers charakterisieren, automatisiert während der Benutzung zu erheben.

Aus den genannten Zielen ergeben sich die folgenden Aufgabenstellungen:

Umfragesystem Die erste Aufgabe dieser Arbeit ist es, ein Umfragesystem zu entwickeln, mithilfe dessen sich die Meinung der Nutzer zu verschiedenen Forschungsfragen im Bereich Saros und Paarprogrammierung erheben lässt. In konfigurierbaren Intervallen soll hierbei dem Benutzer am Ende einer Saros-Sitzung ein Dialog angezeigt werden, der ihn zur Teilnahme an einer Online-Umfrage auffordert. Die Umfrage sollte auf einfache Weise austauschbar sein, um verschiedene Forschungsfragen untersuchen zu können.

Statistik Framework Die zweite Aufgabe befasst sich mit dem implementatorischen Aspekt einer automatisierten Statistikerhebung. Es soll ein Framework entwickelt werden, mit dem es möglich ist, Nutzungsstatistiken während einer verteilten Programmierungssitzung mit Saros zu sammeln und bei Sitzungsende an einen Server am Informatik Institut zu übertragen. Die statistischen Daten sollen so beschaffen sein, dass sich aus ihnen Rückschlüsse über die Arbeitsweise der Nutzer ziehen lassen.
Es ist jedoch nicht Inhalt der Arbeit, die erhobenen Daten statistisch auszuwerten, sondern es geht zunächst darum, die Grundlage für dieses Vorhaben zu schaffen.

Durch beide Systeme lassen sich Daten, die die Forschung im Bereich der verteilten Paarprogrammierung voranbringen, erheben. Durch das integrierte Umfragesystem können außerdem Daten zur Verbesserung des Saros-Plugins erfragt werden, was wiederum das allgemeine Forschungsziel indirekt voranbringt.

1.2 Aufbau der Arbeit

In Kapitel 2 gehe ich zunächst auf die Grundlagen der Arbeit ein, indem ich das Konzept der (verteilten) Paarprogrammierung und das Werkzeug Saros genauer vorstelle. Außerdem beschreibe ich die Motivation hinter der Erhebung von Benutzerfeedback in Saros und welche Methoden aus welchen Gründen dafür zum Einsatz kommen.

In Kapitel 3 beschreibe ich die Anforderungen an die beiden Systeme (Umfragesystem und Statistik Framework), die als Ziel dieser Arbeit implementiert werden sollen. In den Kapiteln 4 und 5 beschreibe ich anschließend ausführlich den Entwurf und die Implementierung der beiden Systeme, wie die Daten jeweils erhoben werden und welche Ergebnisse erzielt werden konnten.

Abschließend gebe ich in Kapitel 6 eine Zusammenfassung der Arbeit und einen Ausblick auf die noch ausstehenden Tätigkeiten im Bereich der Erhebung von Benutzerfeedback in Saros.

2 Grundlagen

In diesem Kapitel möchte ich die Grundlagen, auf denen meine Arbeit aufbaut, beschreiben. Dazu gebe ich einen Überblick über das Konzept der (verteilten) Paarprogrammierung und wie es mit Saros umgesetzt wird. Des Weiteren gehe ich auf die Motivation für die Erhebung von Benutzerfeedback ein, die Methoden, die zur Datenerhebung im Saros-Projekt geeignet sind sowie auf die Probleme, die bei den gewählten Methoden zu beachten sind.

2.1 Saros und Paarprogrammierung

Das Konzept der Paarprogrammierung ist eine Variante der Programmierung, bei der zwei Entwickler Seite an Seite an einem Computer sitzend am selben Entwurf, Algorithmus, Code oder Test arbeiten. Einer der Entwickler wird *Driver* genannt und schreibt am Computer. Sein Partner, der *Navigator* oder *Observer*, hat in erster Linie die Aufgabe, die Arbeit des Drivers zu überwachen und nach taktischen (z. B. Syntax Fehlern, Aufruf der falschen Methode) oder strategischen Defekten (Implementierung erfüllt nicht die Anforderung) Ausschau zu halten. Die Rollen sollten möglichst regelmäßig gewechselt werden [WK02].

Die Vor- und Nachteile der Paarprogramierung gegenüber der traditionellen Entwicklung durch einen Programmierer werden vielfach untersucht [CW00] [MS03] und kontrovers diskutiert [ADT05].
Nach wissenschaftlichen Studien werden der Paarprogrammierung jedoch einige Vorteile gegenüber der individuellen Programmierung zugeschrieben. Zu diesen Vorteilen gehören z. B. höhere Qualität des Endproduktes, höhere Produktivität des Entwicklerpaars im Vergleich zu einem einzelnen Programmierer [MS03] sowie besserer Transfer von Wissen innerhalb des gesamten Projektteams und ein höherer Lerneffekt für die einzelnen Entwickler [CW00].

Im Unterschied zur klassischen Paarprogrammierung meint *verteilte* Paarprogrammierung, dass die beiden Entwickler eines Teams zwar immer noch gleichzeitig und gemeinsam am selben Entwurf und Code arbeiten, sich dabei aber an unterschiedlichen Orten aufhalten [BGS02]. Diese räumliche Trennung kann von verschiedenen Computern im selben Haus bis hin zu Arbeitsplätzen in unterschiedlichen Ländern reichen.

Die Schwierigkeit besteht darin, trotz der Entfernung zwischen den Partnern eine mit der klassischen Paarprogrammierung vergleichbare Situation zu schaffen. Das beinhaltet, dass beide Teilnehmer in der Lage sind mindestens denselben Bildschirminhalt zu sehen sowie diesen (im Wechsel) verändern zu können und dass ein verbaler Kommunikationsweg zwischen den

Partnern besteht. Um solch eine Situation zu schaffen, ist technische Unterstützung notwendig, z. B. durch Sprachkonferenzsoftware in Kombination mit *Desktop Sharing*[1].

Das Saros-Plugin geht nicht den Weg des *Desktop Sharing*, sondern verfolgt den Ansatz der *Collaboration Awareness*[2], indem es Mehrbenutzer-Unterstützung mit Paarprogrammierungs-Funktionalitäten direkt für die Entwicklungsumgebung Eclipse nachrüstet [DOS07].

Jeder Teilnehmer einer Programmiersitzung mit Saros benötigt ein Jabber-Konto bei einem öffentlichen oder privaten XMPP-Server, über das der Einladungsprozess zu einer Sitzung durchgeführt wird. Eine Programmierungssitzung wird mit einem bestimmten Projekt im Eclipse-Workspace von einem der Teilnehmer initiiert, der damit zum *Host* der Sitzung wird. Er kann weitere Teilnehmer zu dieser Sitzung einladen, die daraufhin eine Kopie des gewählten Projektes auf ihre Festplatte übertragen bekommen.

Der jeweilige Host verwaltet außerdem die Rolle des *Drivers*. Er kann entweder nur einem Teilnehmer Schreibrechte zuteilen (*exklusive Driver-Rolle*) oder mehreren Benutzern das gleichzeitige Schreiben ermöglichen (*Multi-Driver-Modus*). Alle Teilnehmer, die nicht Driver sind, üben die *Observer*-Rolle aus. Sie können entweder den Aktivitäten eines Drivers automatisch folgen (*Follow Mode*), wobei sie dabei stets denselben Bereich innerhalb einer Datei sehen, wie der verfolgte Benutzer oder sich selbstständig im Projekt bewegen und z. B. verschiedene Projekt-Dateien öffnen und Text selektieren.

Damit jeder Nutzer weiß in welcher Datei sich die anderen Teilnehmer gerade befinden, gibt es zahlreiche Markierungen (*Awareness Information*), die jeweils in der Farbe angezeigt werden, die dem Benutzer am Anfang der Sitzung automatisch zugeteilt wurde. So wird der momentane Sichtbereich, die aktuelle Cursorposition und der selektierte Text eines jeden Nutzers farbig dargestellt. Bei einem Teilnehmer mit Schreibrechten werden zusätzlich dessen bisherige Änderungen im Code markiert.

Saros unterstützt zusätzlich zur verteilten Paarprogrammierung Sitzungen sowohl mit mehreren Observern als auch mit mehreren Drivern und kann somit ebenfalls zur verteilten, kollaborativen Softwareentwicklung verwendet werden.

2.2 Motivation für diese Arbeit

Wie bereits in der Einleitung erwähnt, ist Saros ein Forschungsprojekt, das u. a. auf die Erhebung von statistischen Nutzungsdaten angewiesen ist, um

[1] Die Bildschirminhalte des einen Computers werden zum entfernten Computer übertragen.

[2] Die Funktionen zur Kollaboration sind direkt in das Werkzeug integriert, d. h. es ist sich über seinen verteilten, kollaborativen Einsatz „bewusst“.

das Konzept der (verteilten) Paarprogrammierung tiefer gehend untersuchen zu können. Um wiederum ausreichend verwertbare Daten zu erhalten, muss Saros auch als Open Source Projekt erfolgreich sein, denn nur erfolgreiche Projekte ziehen Aufmerksamkeit auf sich.

Es gibt eine große Anzahl von freier Software im Internet. Eine sehr bekannte und beliebte Plattform zur Verbreitung von freier Software ist der Hosting-Dienst SourceForge[3]. Auch das Saros-Plugin hat hier seine Projektseite. Nach eigenen Angaben bietet SourceForge mit über 230.000 Software Projekten[4] die größte Sammlung an Open Source Programmen im Internet [SF]. Diese Zahl stellt trotzdem immer noch nur einen Bruchteil der insgesamt im Internet verfügbaren Open Source Projekte dar, jedoch lässt sie eine ungefähre Größenordnung erahnen.

Wie Karl Franz Fogel in seinem Buch *„Producing Open Source Software: How to Run a Successful Free Software Project“* beschreibt, scheitern jedoch die meisten Open Source Projekte (über 90%) und nur wenige sind über längere Zeit erfolgreich [Fog05].
Hierbei stellt sich die Frage, was Erfolg bei Software Projekten eigentlich bedeutet und wie man ihn messen kann.

Crowston, Annabi und Howison gingen dieser Frage speziell in Bezug auf Open Source Software (OSS) nach und entwickelten eine Liste von Erfolgsindikatoren [CAH03]. Zum einen leiteten sie Maße für Erfolg aus dem von DeLone und McLean entwickelten *Model of Information System Success* [DM92] ab und zum anderen ergänzten sie weitere, die sich aus ihrer Analyse des Entwicklungsprozesses bei OSS Projekten ergaben. Einen Überblick über die entwickelten Maße gibt Tabelle 1 auf der nächsten Seite.

Für diese Arbeit sind vor allen Dingen die beiden folgenden Erfolgsmaße relevant:

Benutzerzufriedenheit Die Benutzerzufriedenheit kann verbessert werden, indem auch das Produkt immer weiter verbessert wird. Durch direkte Befragung der Nutzer lässt sich dabei am besten ermitteln, an welchen Stellen bei der Benutzung Probleme oder Unklarheiten auftraten. Dieser Aspekt motiviert die Umsetzung des Umfragesystems.

Einsatz und Verwendung Die Häufigkeit des Einsatzes einer Software kann man über eine hohe Qualität des Produkts, jedoch hauptsächlich über geeignete Werbemaßnahmen steigern. Eine hohe Qualität trägt vor allem dazu bei, dass die Nutzer der Software treu bleiben.

[3] http://sourceforge.net/
[4] Stand: Februar 2009

Erfolgsmaß	**Indikatoren**
System- und Informationsqualität	Qualität von Code (z.B. Verständlichkeit, Wartbarkeit, Konsistenz, Zuverlässigkeit, etc.) und Dokumentation
Benutzerzufriedenheit	Bewertungen durch Nutzer (z.B. auf geeigneten Webseiten), Umfragen, Meinungen auf Mailing-Listen
Einsatz und Verwendung	Häufigkeit des Einsatzes der Software, Anzahl von Nutzern, Downloads, Popularität (Anzahl Besuche der Projektseite), etc.
Erweiterungen, die speziell auf Open Source Projekte zutreffen	
Projekt Output	Entwicklung von Alpha- über Beta- hin zur stabilen Version, Zufriedenheit der Entwickler, Anzahl erreichter, selbstgesteckter Ziele
Entwicklungsprozess	Anzahl der Entwickler, Aktivitätsgrad (Beiträge von Entwicklern und Nutzern, Anzahl von Releases), Zeit zwischen Releases, Zeit zum Schließen von Bugs und Implementieren neuer Features
Ergebnis für Projektmitglieder	Wissenserwerb, Jobaussichten, Ansehen in der Community

Tabelle 1: Maße für Erfolg eines Open Source Projekts nach Crowston, Annabi und Howison [CAH03]

Wie sehr die Aufmerksamkeit auf die Software gerichtet werden konnte, lässt sich anhand der Downloadzahlen abschätzen. Wie häufig die Software allerdings tatsächlich eingesetzt wird, ist nicht ohne weiteres bestimmbar. Über das Statistik Framework, das als Ziel dieser Arbeit implementiert werden soll, lässt sich anhand der eingegangenen Statistiken zumindest abschätzen, wie häufig das Saros-Plugin wirklich verwendet wird. Hierbei muss beachtet werden, dass nicht alle Nutzer der Statistikübertragung zugestimmt haben und somit die eigentliche Nutzerzahl noch größer sein dürfte.

2.3 Datenerhebung in Saros

Bestandteil dieser Arbeit ist die Erhebung von Benutzerfeedback zweierlei Art: Direktes Feedback der Benutzer und indirektes Feedback über die Sitzung. Im Folgenden möchte ich auf die Motivation für die Wahl der Forschungsmethoden zur Erhebung des Feedbacks in Saros sowie auf Vor- und Nachteile der Methoden eingehen.

2.3.1 Wahl der Forschungsmethoden

Die Möglichkeiten bei einem Open Source Softwareprojekt Rückmeldungen der Nutzer zu erhalten, sind stark eingeschränkt. Die Benutzer, die die Software ohne Aufforderung (im Gegensatz zur Situation bei einem Experiment) und aus persönlichem Interesse einsetzen, sind unbekannt und können nicht direkt erreicht werden[5]. Eine kontrollierte Befragungssituation der Nutzer einer Software, bei der die Antworten überwacht und Unklarheiten in Fragestellungen beseitigt werden können (wie bei einer mündlichen oder schriftlichen Befragung vor Ort), ist daher nicht möglich.

Um persönliches Feedback der Saros-Benutzer zu erhalten, bleibt nur die Methode der schriftlichen Befragung, wobei eine Online-Umfrage am geeignetsten ist.

Die Methode der Befragung bietet allgemein den Vorteil, dass nicht direkt beobachtbare Dinge erfasst werden können [Att03], wie z. B.

- Fakteninformationen (Geschlecht, Alter, Herkunft, etc.)
- Häufigkeit und Dauer von Verhaltensweisen („Wie häufig haben Sie absichtlich dieselbe Datei editiert wie ein anderer Teilnehmer?“)
- konkrete Verhaltensintentionen („Was wollten Sie in der letzten Saros-Sitzung erreichen?“)
- Meinungen, Einstellungen („Wie hat Ihnen Ihre letzte Saros-Sitzung gefallen?“)

Vorteile speziell der Online-Befragung sind, dass sie schnell und kostengünstig umzusetzen ist und durch sie eine potentiell große Stichprobe untersucht werden kann [Att03].
Nachteilig ist hingegen, dass die Umfragesituation nicht kontrollierbar und die Ausschöpfungsquote eher gering ist [WM03], da es mangels persönlichen Kontakts kaum Möglichkeiten gibt, die Nutzer zur Teilnahme zu motivieren. Wenn die Umfrage zudem allen Internetbenutzern zugänglich gemacht wird, hat man keine Möglichkeit zu kontrollieren, wer sie beantwortet. Daher ist es sehr schwierig eine repräsentative Stichprobe zu erhalten [WM03].

Die Publikation der Umfrage muss daher über die Software selbst stattfinden, um sicherzustellen, dass Saros vor der Teilnahme überhaupt eingesetzt wurde, damit verfälschte Ergebnisse verhindert werden. Außerdem wird hierdurch potentiell die gesamte Population der Saros-Nutzer erreicht. Die Repräsentativität leidet allerdings unter der Selbstselektion der Teilnehmer, da

[5] Aus diesem Grund haben die meisten OSS Projekte *Mailing-Listen* für ihre Nutzer eingerichtet. Es kann jedoch nicht davon ausgegangen werden, dass sich alle Nutzer einer Software auf der zugehörigen Liste anmelden.

bei weitem nicht alle Benutzer an der Umfrage teilnehmen wollen. Die mangelnde Repräsentativität bei Online-Umfragen ist leider nicht vermeidbar [WM03].

Um Feedback über die Art des Einsatzes von Saros zu erhalten, ist eine automatisierte Beobachtung während der Verwendung des Werkzeugs die geeignetste Methode, da hierbei rein statistische, objektive Daten erhoben werden können. Würde man die Daten hingegen per Umfrage ermitteln, ist zu erwarten, dass die Antworten zum einen durch die subjektive Einschätzung und zum anderen durch beabsichtigte oder unbeabsichtigte Falschaussagen des Teilnehmers beeinflusst werden, wodurch die Ergebnisse verfälscht werden können.

Es lassen sich allerdings nicht alle Daten durch einfache Beobachtung erheben. Die Frage nach dem Ziel der letzten Programmiersitzung beispielsweise kann nur vom Benutzer selbst beantwortet werden. Will man eine möglichst vollständige Aussage über die Verwendung von Saros treffen können, bietet sich eine Kombination der beiden Methoden an (vgl. Abschnitt 6.2 auf Seite 32).

2.3.2 Problematik der Forschungsmethoden

Für ein Forschungsprojekt wie Saros, dessen Erfolg maßgeblich von der Menge an gesammelten Daten abhängt, gibt es bei den beschriebenen Methoden zur Datenerhebung neben den bereits erwähnten Nachteilen noch eine entscheidende Schwierigkeit: Wir sind auf die freiwillige Teilnahme der Nutzer angewiesen. Das Ziel muss daher sein, die Nutzer so gut wie möglich zu motivieren, uns einerseits bei der automatisierten Datenerhebung zu vertrauen und sich andererseits die Zeit zu nehmen an einer Umfrage teilzunehmen.

In Zeiten zunehmender Überwachung und Datenspeicherung ist das Thema Datenschutz und Privatssphäre im Internet bei vielen Computerbenutzern ein sensibler Punkt, wie z. B. von Dara O'Neil in den USA untersucht wurde. Laut ihrer Studie gaben etwas über die Hälfte der befragten Internetnutzer an, sehr beunruhigt über den Grad des Schutzes ihrer Daten im Internet zu sein. Etwas mehr als ein Viertel sagten aus, zumindest etwas beunruhigt zu sein [O'N01].

Culnan und Armstrong untersuchten in diesem Zusammenhang inwieweit die Vorbehalte bei Datenerhebungen im Internet abgeschwächt werden können [CA99]. Sie fanden heraus, dass sogenannte *fair information practises* Datenschutzvorbehalte vermindern können. Der Kern von *fair information practises* beinhaltet nach Culnan und Armstrong zwei Konzepte: Bekanntmachung und Einwilligung. Damit ist gemeint, dass die Personen, die Daten von und über sich preisgeben, ein Recht darauf haben zu erfahren von wem, warum, wozu und welche Daten erhoben und wie diese in Bezug auf

Vertraulichkeit und Sicherheit behandelt werden. Danach können sie frei entscheiden, ob sie der Datenerhebung zustimmen wollen oder nicht.
Diese Aufklärungsarbeit reduziere laut der Studie die Besorgnis in Bezug auf Datenschutz erheblich und schaffe Vertrauen zu den Urhebern. Dieser Aspekt sollte bei der Anforderungsspezifikation des Umfragesystems und des Statistik Frameworks berücksichtigt werden.

2.4 Datenerhebung in anderen OSS Projekten

Mechanismen zur Erhebung von Benutzerfeedback sind auch in zahlreichen anderen OSS Projekten zu finden. Im Folgenden möchte ich drei Beispiele für Datenerhebungsmechanismen in Projekten beschreiben, die alle das Ziel der Verbesserung der Software verfolgen, dafür aber ganz unterschiedliches Feedback erheben. Die Daten beziehen sich im ersten Fall auf die Popularität einzelner Funktionen, im zweiten Fall auf Defekte, die zum Versagen der Software führen und im dritten Fall auf das Nutzungsverhalten.

Debian Der *Debian Popularity Contest*[6] versucht die Verwendung von Debian Paketen abzubilden [RGBM05]. Debian Nutzer können ein Paket installieren, das jede Woche eine Liste der lokal installierten Pakete zusammen mit der letzten Zugriffszeit auf relevante Dateien an einen Server verschickt. Diese Informationen werden z. B. benutzt, um abzuschätzen, welche Pakete am häufigsten verwendet werden. Das Ergebnis kann wiederum als Kriterium dienen, um festzulegen, auf welche Pakete der Fokus bei der Qualitätssicherung gelegt werden sollte.

Mozilla Die Mozilla Produkte Firefox 3, Thunderbird 3 und SeaMonkey 2 haben ein neues, fest integriertes Open Source *Crash Reporting System* erhalten: den *Mozilla Crash Reporter* („Breakpad“) [MCR]. Er dient dazu bei Abstürzen einen Fehlerbericht mit Informationen über den Status der Anwendung zum Zeitpunkt des Absturzes an einen Webserver zu übertragen. Vor der Übermittlung wird der Nutzer um Erlaubnis gefragt und kann zusätzlich noch einen Kommentar mitsenden.

Eclipse In Eclipse ist das *Usage Data Collector* [UDC] Framework integriert. Es sammelt über Listener-Mechanismen Informationen über die Aktionen, die der Nutzer bei seiner Arbeit mit Eclipse ausführt (z. B. das Öffnen von Editoren, Views, etc.). Das Ziel ist herauszufinden, wie Eclipse in der Praxis eingesetzt wird.

[6] http://popcon.debian.org/

3 Anforderungen

Die Anforderungsspezifikation habe ich gemeinsam mit meinen Betreuern Christopher Oezbek und Stephan Salinger erarbeitet, wobei meine Betreuer die Rolle des Kunden einnahmen, d. h. grobe Anforderungen vorgaben, die ich genauer spezifizierte, schriftlich festhielt und das Ergebnis mit ihnen abstimmte.
Aus diesen Gesprächen ergaben sich die folgenden Anforderungen.

3.1 Anforderungen an das Umfragesystem

3.1.1 Funktionale Anforderungen

Der Nutzer soll in konfigurierbaren Intervallen (z. B. nach jeder Sitzung, nach jeder dritten Sitzung, nach jeder x-ten Sitzung) jeweils am Ende seiner Sitzung mittels eines Dialogfensters gebeten werden sein Feedback zu Saros in Form der Teilnahme an einer Umfrage abzugeben. Der Dialog sollte zum ersten Mal nach der ersten Sitzung erscheinen, um möglichst viele Benutzer zu erreichen; danach greift das Intervall. Der Benutzer muss die Möglichkeit haben die Aufforderung zu bejahen oder zu verneinen sowie weitere Aufforderungen gänzlich zu unterbinden. Letztere Wahl muss jedoch rückgängig zu machen sein.

Sollte der Nutzer der Umfrage durch Bejahen des Dialogfensters zustimmen, so wird er auf eine Webseite geleitet, die im Standard-Browser geöffnet wird und den Fragenkatalog enthält.

Um zu vermeiden, dass der Nutzer nach sehr kurzen Sitzungen, in denen er wenig über Saros erfahren konnte, bereits zur Umfrageteilnahme aufgefordert wird, soll der Umfragmechanismus erst nach einer Sitzungs-Mindestzeit greifen.

Eine nichtfunktionale Anforderung ist, dass der Nutzer durch das Umfragesystem so wenig wie möglich belästigt werden soll, um ihn nicht zur sofortigen Deaktivierung des Umfragesystems zu verleiten. Daraus ergibt sich wiederum die funktionale Anforderung, dass die Einstellungen zum Umfragesystem global und persistent gespeichert werden müssen, sodass sich Änderungen durch den Nutzer immer in allen Eclipse-Workspaces widerspiegeln. Des Weiteren müssen die Einstellungen ebenfalls erhalten bleiben, wenn Eclipse und das Saros-Plugin neu installiert wurden.

Konfigurationsmöglichkeiten An geeigneter Stelle muss es dem Benutzer möglich sein, das Intervall, in dem die Aufforderung zur Umfrageteilnahme erscheint, zu verändern. Außerdem sollte dort die Umfrageaufforderung komplett deaktivierbar sowie wieder aktivierbar sein. Zusätzlich muss es möglich sein, die Umfrage außerhalb des eingestellten Intervalls sofort

durchzuführen. Das Intervall zur nächsten automatischen Aufforderung wird dadurch zurückgesetzt.

3.1.2 Nichtfunktionale Anforderungen

Eine weitere Anforderung ist der Einsatz von *fair information practices* (siehe 2.3.2 auf Seite 9). Das beinhaltet in diesem Fall die Aufklärung des Nutzers vor der Teilnahme an der Umfrage von wem und zu welchem Zweck die Daten erhoben werden und wie viel Zeit das Ausfüllen der Umfrage ungefähr in Anspruch nimmt.

3.1.3 Randbedingungen

Die Umfrage soll mittels des phpESP (*php Easy Survey Package*) Umfragesystems, das bereits auf einem Server des Instituts für Informatik zur Verfügung steht, erstellt werden.

Das *php Easy Survey Package* ist ein Umfragesystem, das aus einer Sammlung von PHP Skripten besteht. Das System arbeitet mit einer Datenbank zusammen, in der die Umfragen und Umfrageergebnisse gespeichert werden und ermöglicht auch technisch wenig versierten Benutzern, Umfragen über eine Weboberfläche zu erstellen, zu verwalten und die Ergebnisse als CSV-Datei herunterzuladen oder statistisch per Kreuztabellierung auszuwerten [But].

3.2 Anforderungen an das Statistik Framework

3.2.1 Funktionale Anforderungen

Während einer Saros-Sitzung sollen verschiedene, nicht personenbezogene Daten über den Sitzungsverlauf gesammelt werden. Ich einigte mich mit meinen Betreuern auf folgenden Katalog an zu erhebenden Daten:

- Die Anzahl der bisher gestarteten Sitzungen,
- die Dauer der aktuellen Sitzung,
- die Anzahl der Teilnehmer der aktuellen Sitzung,
- die Rollen des lokalen Nutzers und deren Ausübungsdauer,
- die Anzahl der vom lokalen Nutzer geschriebenen Zeichen während der Sitzung,
- eine eindeutige Sitzungs-ID,
- ob der lokale Nutzer der Host der Sitzung war oder nicht,
- die aktuellen Feedback Einstellungen (aktiviert oder deaktiviert, eingestelltes Intervall)

- und einige allgemeine, sitzungsunabhängige Daten, wie
  - das verwendete Betriebssystem,
  - die installierte Java-Version,
  - die installierte Eclipse-Version,
  - die installierte Saros-Version,
  - und eine zufällig generierte, eindeutige User-ID.

Die gesammelten Daten müssen schließlich am Ende der Sitzung an einen unserer Server zur weiteren Auswertung übertragen werden. Hierfür muss ein Mechanismus bereitgestellt werden, mit dem Dateien auf einen Server hochgeladen werden können. Dabei sollte der Nutzer eine Fortschrittsanzeige sehen und die Übertragung gegebenenfalls abbrechen können.

Des Weiteren muss eine Möglichkeit bereitgestellt werden, über die der Nutzer um Erlaubnis zur Übertragung von während der Sitzung gesammelten, statistischen Daten gefragt werden kann. Die Antwort auf diese Frage muss vor der ersten Sitzung eingeholt werden und muss im Nachhinein geändert werden können.

3.2.2 Nichtfunktionale Anforderungen

Wie schon beim Umfragesystem besteht auch bei der Statistikerhebung die Anforderung, den Benutzer umfassend über den Verwendungszweck der erhobenen Daten aufzuklären. Es ist zusätzlich erforderlich, detaillierte Informationen über die Art der Daten bereitzustellen, da hier der Nutzer im Gegensatz zur Umfrage keinerlei Kontrolle über die preisgegebenen Informationen hat.

Eine weitere nichtfunktionale Anforderung ist, dass insbesondere keine personenbezogenen Daten oder Inhalte von Dateien gesammelt werden dürfen.

4 Umfragesystem

In diesem Kapitel beschreibe ich die Umsetzung des Umfragesystems in Saros und die zugrunde liegenden Entwurfsentscheidungen. Des Weiteren gehe ich auf die von mir entwickelten möglichen Ziele für Umfragen, die schließlich erstellte Umfrage und deren Ergebnisse ein.

4.1 Entwurf

Das innerhalb dieser Arbeit entstandene Umfragesystem wird primär über die Klasse *FeedbackManager* realisiert. Das Einzelstückexemplar dieser Klasse (engl. *Singleton*) wird am Ende einer Sitzung über einen Listener-Mechanismus aufgerufen und überprüft anschließend, ob die Umfrageaufforderung (*FeedbackDialog*) jetzt angezeigt werden muss. Abbildung 1 zeigt den *FeedbackDialog*, der eine kurze Nachricht an den Nutzer über den Zweck der Umfrage enthält, sowie ein Kontrollkästchen, über das das Umfragesystem komplett deaktivierbar ist. Um die Nachricht möglichst kurz zu halten, den Nutzer aber dennoch über die Urheber der Umfrage zu informieren, sind Links zu den Webseiten der Freien Universität und der AG Software Engineering im Dialog untergebracht.

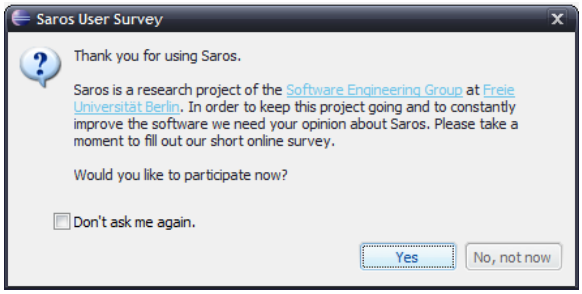

Abbildung 1: Dialog zur Umfrageaufforderung

Der Dialog soll genau dann gezeigt werden, wenn das Umfragesystem aktiviert und das festgelegte Intervall abgelaufen ist. Letzteres wird überprüft, indem ein global gespeicherter Wert (vgl. Abschnitt 4.1.1), der die Anzahl der Sitzungen bis zur nächsten Umfrageaufforderung speichert, nach jeder Sitzung durch den *FeedbackManager* dekrementiert wird. Durch das globale Speichern der Anzahl der verbleibenden Sitzungen wird erreicht, dass – egal welcher Workspace verwendet wird – der *FeedbackDialog* immer im festgelegten Intervall angezeigt wird. Andernfalls könnte ein Nutzer, der mit verschiedenen Workspaces arbeitet, den Dialog mehrmals, direkt hintereinander präsentiert bekommen, was ihn möglicherweise schnell dazu verleitet, das Umfragesystem komplett zu deaktivieren.

Die Antwort des Benutzers auf den *FeedbackDialog* wird ebenfalls vom *Feed-*

backManager verarbeitet. Hat der Nutzer den Dialog bestätigt, so wird die Webseite mit der Umfrage im Standard-Browser geöffnet. Sollte dies nicht gelingen, wird versucht, die Webseite im internen Eclipse-Browser anzuzeigen. Sollte auch diese Methode fehlschlagen, wird ein einfacher Dialog angezeigt, der auf den Fehler hinweist und den Benutzer bittet die angegebene Webseite der Umfrage selbstständig aufzurufen.
Falls der Nutzer das Kontrollkästchen „Don't ask me again" markiert hat, wird das Umfragesystem komplett deaktiviert, und zwar egal auf welche Weise der Nutzer den Dialog daraufhin geschlossen hat. Dadurch ist es möglich, einmalig an der Umfrage teilzunehmen und anschließend das Umfragesystem zu deaktivieren.

Abbildung 2 zeigt eine Übersicht über die am Umfragesystem beteiligten Klassen, ihre wichtigsten Methoden und die Verbindungen zueinander. Methodenparameter wurden aus Gründen der Übersichtlichkeit weggelassen.

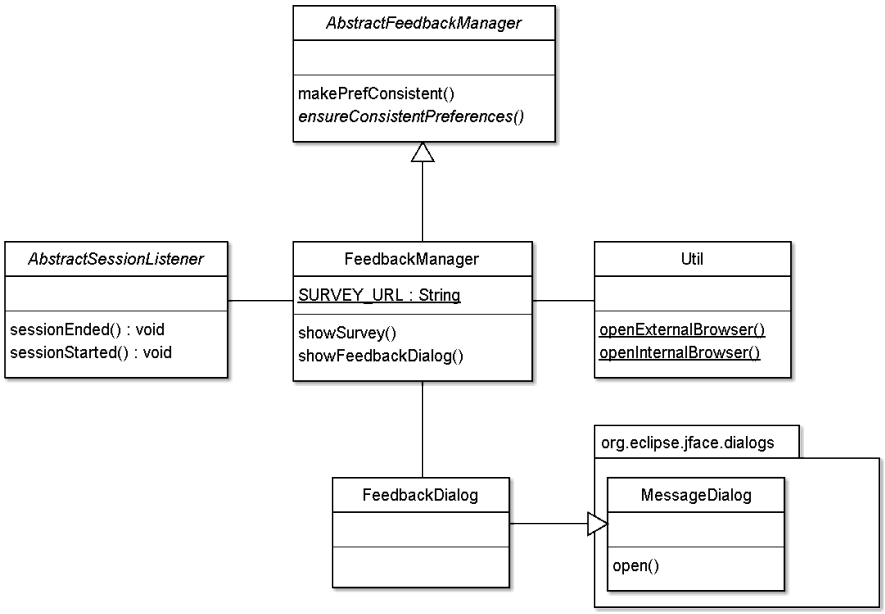

Abbildung 2: Klassendiagramm zum Umfragesystem

4.1.1 Preferences

Das von mir umgesetzte Umfragesystem benutzt drei Werte, die persistent gespeichert werden müssen, d. h. dass sie auch nach einem Neustart von Eclipse wieder verfügbar sind.

1. Status des Umfragesystems: Dieser Wert gibt an, ob das Umfragesystem aktiviert oder deaktiviert ist. Er wird für die Entscheidung verwendet, ob dem Nutzer der *FeedbackDialog* angezeigt werden darf oder nicht.

2. Umfrage-Intervall: Dieser Wert spiegelt die aktuelle Einstellung des Intervalls, in dem die Umfrage angezeigt werden soll, wider. Zum Beispiel bedeutet der Wert fünf, dass zwischen den einzelnen Umfragen fünf Sitzungen vergehen müssen.

3. Verbleibende Sitzungen: Dieser Wert gibt die Zahl der Sitzungen an, die bis zur Anzeige der nächsten Umfrageaufforderung noch vergehen müssen.

Eclipse kennt vier verschiedene Bereiche (*Scopes*), in denen Einstellungen (genauer: Schlüssel-Wert-Paare, in Eclipse-Terminologie *Preferences* genannt) durch ein Plugin gespeichert werden können [EF].

Configuration Scope In diesem Scope gespeicherte Werte sind in allen Workspaces verfügbar, die mit derselben Eclipse-Installation verwendet werden. Bei einem Einzelbenutzer bedeutet dies, dass die Werte in allen von ihm gestarteten Workspaces vorhanden sind. Für mehrere Nutzer derselben Eclipse-Installation bedeutet dies, dass die Daten gemeinsam benutzt werden.

Instance Scope Werte, die in diesem Scope gespeichert werden, sind nur in einem konkreten Workspace verfügbar.

Default Scope Daten in diesem Scope werden nicht auf der Festplatte gespeichert. Er dient hingegen dazu, Default-Werte für eigene Schlüssel zu definieren. Die Schlüssel können über eine *Initializer* Klasse beim Start des Plugins mit diesen Werten automatisch initialisiert werden. Sobald ein Wert einmal nicht in einem der anderen Scopes gefunden wurde, wird auf den Default-Wert zurückgegriffen.

Project Scope Die Werte dieses Scopes gehören zu einem einzelnen Projekt im aktuellen Workspace, wie z. B. Code-Formatierungs- und Compiler-Einstellungen. Dieser Scope wird als einziger bisher nicht in Saros verwendet.

Wenn ich im Folgenden von globalen und lokalen Werten spreche, so meine ich mit „global“ stets einen Wert, der im *Configuration Scope* und mit „lokal“ einen Wert, der im *Instance Scope* von Eclipse gespeichert ist.

Die Werte zum Status des Umfragesystems und zum Umfrage-Intervall sollen nach den Anforderungen global mit Rückfallmöglichkeit auf einen lokalen

Wert gespeichert werden, um dem Nutzer die mehrfache Konfiguration zu ersparen. Falls globale Werte vorhanden sind, werden diese benutzt, andernfalls wird auf die lokalen zurückgegriffen. Für die lokalen Preferences werden zusätzlich beim Start des Plugins Default-Werte definiert, die dann verwendet werden, wenn kein globaler oder lokaler Wert gefunden werden konnte. Es gilt somit die Regel:

> „Globaler vor lokalem vor Default-Wert."

Die globalen Preferences werden erst bei der Deinstallation von Eclipse gelöscht, die lokalen beim Entfernen des betreffenden Workspaces.

Es ist jedoch zu beachten, dass es nicht ausreicht, die Preferences nur bei ihrer Änderung global und lokal zu speichern, da dadurch Inkonsistenzen zwischen den Scopes entstehen können. Abbildung 3 zeigt eine beispielhafte Situation.

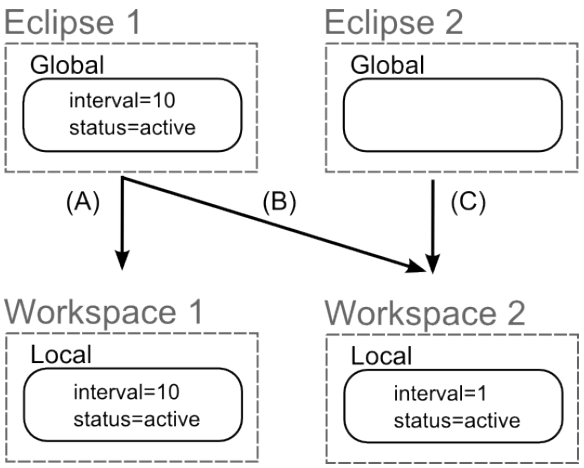

Abbildung 3: Beispiel für eine Inkonsistenzsituation zwischen den Scopes

Angenommen der Benutzer ändert das Intervall über die Konfigurationsseite (siehe Abschnitt 4.1.2 auf der nächsten Seite) auf „nach jeder zehnten Sitzung" (A). Jetzt startet er einen anderen Workspace, für den nun automatisch ebenfalls das Intervall „zehn" gilt, da diese Information vorher global gespeichert wurde. Lokal kann in diesem Workspace jedoch eine ganz andere Einstellung existieren (z. B. Intervall „eins"), womit die Scopes inkonsistent werden (B). Solch eine Inkonsistenz wird solange nicht bemerkt, wie es einen globalen Wert gibt, da dieser immer zuerst verwendet wird. Sie führt aber zu unerwartetem Verhalten, sobald der Workspace mit einer neuen Eclipse Installation gestartet wird: Plötzlich erscheint der Umfragedialog nach jeder Sitzung, da mangels eines globalen Wertes der lokale verwendet wird (C).

Inkonsistenzen müssen daher verhindert werden, und zwar bereits beim Start des Plugins. Dann wäre es bei (B) zu keiner Inkonsistenz gekommen und in der Folge zu keinem unerwarteten Verhalten bei (C).
Grundsätzlich sind vier Fälle zu unterscheiden:

- Fall 1: Die Werte sind weder global noch lokal vorhanden, es gibt nur die Default-Werte. Dies tritt auf, wenn Eclipse neu installiert und mit einem neuen Workspace gestartet wurde.
- Fall 2: Die Werte sind global aber nicht lokal vorhanden. Dies tritt auf, wenn eine bestehende Eclipse Installation mit einem neuen Workspace gestartet wurde.
- Fall 3: Die Werte sind lokal aber nicht global vorhanden. Dies tritt auf, wenn Eclipse neu installiert wurde und mit einem bestehenden Workspace verwendet wurde.
- Fall 4: Die Werte sind sowohl global als auch lokal vorhanden. Dies tritt auf, wenn eine bestehende Eclipse Installation verwendet wurde.
  - Fall 4.1: Die Scopes sind inkonsistent.
  - Fall 4.2: Die Scopes sind konsistent.

Bei Fall 4.2 muss nichts unternommen werden, denn dieser beschreibt genau die erwünschte Situation. In den restlichen Fällen müssen die Werte nach oben erwähnter Regel so kopiert werden, dass sich die Scopes am Ende gleichen. Tabelle 2 fasst das Vorgehen zusammen.

Fall	Vorhandene Werte	Maßnahme
1	Nur Defaults	Kopiere Defaults in globalen und lokalen Scope
2	Nur global	Kopiere globale Werte in lokalen Scope
3	Nur lokal	Kopiere lokale Werte in globalen Scope
4.1	Global und lokal	Kopiere globale Werte in lokalen Scope

Tabelle 2: Konsistenzwiederherstellung

Die Konsistenzherstellung wird vom *AbstractFeedbackManager* gekapselt. Eine Unterklasse muss hierbei nur angeben, für welche Preferences die Konsistenz sichergestellt werden soll.

4.1.2 Konfiguration durch den Benutzer

Die Einstellungen zum Status des Umfragesystems und des Umfrage-Intervalls müssen nach den Anforderungen durch den Benutzer konfigurierbar sein. Hierfür stellt Eclipse den Mechanismus der *Preference Pages* zur Verfügung. Sie bieten eine einfach zu implementierende Möglichkeit *Preferences* durch den Benutzer verändern zu lassen und persistent zu speichern. Das

Buch *Eclipse plug-ins* von Clayberg und Rubel [EC08] widmet der *Preference Page* ein eigenes Kapitel.

Clayberg und Rubel beschreiben darin zwei grundlegende Konzepte sowie ihre Vor- und Nachteile:

Field Editors Eine eigene Konfigurationsseite kann von der abstrakten Klasse *FieldEditorPreferencePage* erben. Dann ist sie allerdings darauf beschränkt, alle graphischen Steuerungselemente durch sogenannte *Field Editors* darstellen zu müssen. Ein *Field Editor* repräsentiert genau einen *Preference* Wert und ermöglicht es dem Endnutzer, diesen zu verändern und zu speichern. *Field Editors* sind für Entwickler besonders einfach zu benutzen, da sie nach dem Prinzip

> „Create them and forget them“

entworfen wurden, d. h. es reicht aus, einen konkreten *Field Editor* mit Zuordnung zu einem *Preference* Wert zu erzeugen und der Konfigurationsseite hinzuzufügen. Der Rest – das Anzeigen und Speichern des zugeordneten Wertes – wird automatisch erledigt. Diese einfache Methode zum Erstellen von *Preference Pages* sollte bevorzugt werden, da sie schnell und einfach umzusetzen und wenig fehleranfällig ist.

Preference Page Die zweite Möglichkeit ist eine eigene Konfigurationsseite von der abstrakten Klasse *PreferencePage* abzuleiten. Dadurch lassen sich komplexere Seiten erstellen, da beliebige GUI-Elemente[7] hinzugefügt werden können. Der Nachteil ist wiederum, dass sich der Entwickler selbst um das Laden, Validieren und Speichern der Werte kümmern muss.

Meine Entscheidung für eines der beiden Konzepte wurde hauptsächlich von zwei Anforderungen beeinflusst: Der Nutzer soll ein Intervall einstellen können, das potentiell viele verschiedene Werte annehmen kann und er soll die Umfrage auch sofort ausführen können. Für Ersteres eignet sich eine *Dropdown Listbox*, für Letzteres ein *Button*. Beides lässt sich nicht durch einen *Field Editor* abbilden, so dass ich das zweite Konzept gewählt habe.

Abbildung 4 auf der nächsten Seite zeigt die von mir erstellte *FeedbackPreferencePage*. Sie enthält zusätzlich zu den Einstellungen zum Umfragesystem auch noch die zur Statistikübertragung sowie Informationen wie man die Saros-Entwickler erreichen kann.

[7] GUI steht für *Graphical User Interface*

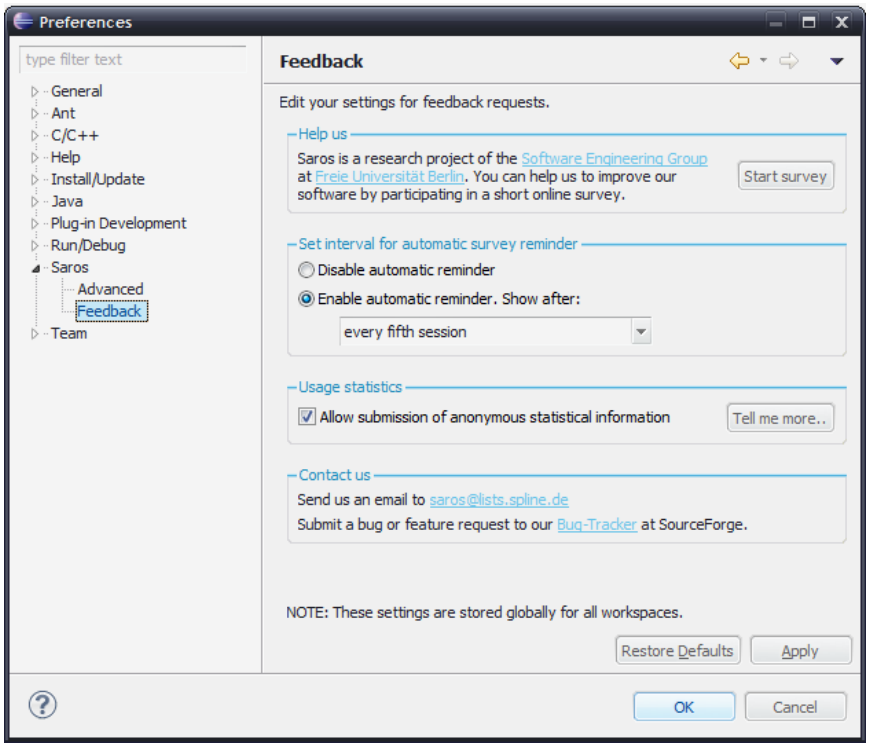

Abbildung 4: Konfigurationsseite zum Feedback

4.2 Entwicklung der Umfrage

Da eine Forschungsfrage für das innerhalb dieser Arbeit umgesetzte Umfragesystem bisher nicht konkret festgelegt war, entwickelte ich eine Reihe von möglichen Zielen, die mit einer Umfrage verfolgt werden könnten. Dafür eignete sich das *Goal Question Metric* Paradigma (kurz: GQM Paradigma) [BCR94].

Differding, Hoisel und Lott fassen diesen Ansatz treffend zusammen:

> „The GQM Paradigm [...] supports a top-down approach to defining the goals behind measuring software processes and products, and using these goals to decide precisely what to measure (choosing metrics)“. [DHL96]

GQM stellt zudem Schablonen bereit, mit denen messbare Ziele nach einem festgelegten Weg definiert werden können. Solche Schablonen strukturieren Ziele nach fünf Aspekten [BDR96]:

1. Objekt der Studie
2. Zweck der Studie
3. Qualitätsfokus

4. Blickwinkel
5. Kontext

Mittels dieser Schablone entwarf ich drei unterschiedliche Ziele, die im Anhang B.1 auf Seite 37 nachzulesen sind.

Aufgrund von Zeitdruck durch den drei-wöchentlichen Releaseplan von Saros entschieden meine Betreuer und ich, dass zunächst nur eine Kurz-Umfrage mit weniger wissenschaftlichem Anspruch umgesetzt werden sollte. Mithilfe von vier Fragen wollten wir herausfinden, wofür Saros hauptsächlich verwendet wird und welche Probleme bei der Benutzung auftreten. Die komplexeren Umfrage-Konzepte sollten vor ihrer Umsetzung noch weiter ausgearbeitet werden, welches nicht mehr Teil meiner Arbeit ist.

Der von mir entworfene Text der schließlich umgesetzten Umfrage findet sich in Anhang B.2 auf Seite 37. Die Umfrage wurde, wie durch die Anforderungen vorgegeben, mithilfe des *phpESP* Umfragesystems umgesetzt und mit Release `9.5.29r.1374` von Saros online gestellt.

4.3 Umfrageergebnisse

Die Umfrage war ab dem 29.05.09 bis zum heutigen Tage (29.07.09) genau zwei Monate, das entspricht 61 Tagen, online. Nach der Downloadstatistik auf SourceForge wurde das Saros-Plugin in diesem Zeitraum 1589 Mal heruntergeladen [SFs]. Nur sieben Benutzer haben in dieser Zeit an der Umfrage teilgenommen, welches verschwindend geringen 0,44% entspricht. Es ist allerdings davon auszugehen, dass nicht alle, die das Plugin heruntergeladen haben, tatsächlich eine Sitzung gestartet haben. Die Umfrage wird zudem nur nach einer Sitzung, die länger als fünf Minuten dauerte, angezeigt. Wer das Plugin nur einmal kurz testen wollte, hat diese Grenze womöglich nicht überschritten. Selbst wenn man davon ausgeht, dass nur jeder zehnte Benutzer die Umfrageaufforderung überhaupt gesehen hat, sind es dennoch nur 4,4%, die auch an der Umfrage teilgenommen haben.
Damit bestätigte sich leider die Befürchtung, dass die Teilnehmerzahl gering sein würde. Die Umfrage war jedoch absichtlich so ausgelegt, dass auch schon wenige Ergebnisse hilfreiche Informationen liefern können.

Die Ergebnisse zeigen, dass bei vier von sieben Teilnehmern Probleme bei der Benutzung von Saros auftraten. Nur ein Teilnehmer gab jedoch an, dass er aufgrund der Probleme sein gesetztes Ziel nicht erreichen konnte.
Die Antworten zeigen außerdem, dass Saros in der Mehrheit von einem Entwickler-Paar eingesetzt wurde, das verteilte Programmierung mit Saros testen und durchführen wollte (57%). Als weitere Ziele wurden genannt, dass einem Kollegen neuer Code gezeigt oder dass Saros einfach nur ausprobiert werden sollte.

Aufgrund der geringen Teilnehmerzahl lassen sich aus diesen Ergebnissen nur unter Vorbehalt Schlüsse ziehen. Die Antworten vermitteln aber den Eindruck, dass Saros hauptsächlich zur verteilten Programmierung von zwei Entwicklern verwendet wird und nicht in größeren Gruppen. Unklar ist, ob dabei auch das Konzept der Paarprogrammierung umgesetzt wird, d. h. ob die klassische Rollenaufteilung in Driver und Observer eingehalten wird und sich die Nutzer entsprechend dieser Rollen verhalten.

Direkten Nutzen bringen die Antworten zu Fragen nach den aufgetretenen Problemen und insbesondere die Beschreibungen des beobachteten Fehlverhaltens. So erfuhren wir z. B. von zwei Benutzern, dass Saros schlecht mit Eclipse-Plugins zur Versionsverwaltung zusammenarbeitet, denn es wurde den Benutzern angezeigt, dass Dateien verändert wurden, obwohl sie selbst keine Änderungen durchgeführt hatten. Des Weiteren erfuhren wir, dass das Hinzufügen von Kontakten, die ein Konto bei *Jabber.org*[8] besitzen, nicht funktionierte.
Zusätzlich zu Problemschilderungen bekamen wir außerdem Vorschläge für neue Funktionen. Es wurde z. B. gewünscht, dass auch mehrere Projekte in einer Sitzung bearbeitet werden können.

In zwei Fällen bekamen wir zudem ein ausgesprochen positives Feedback zu Saros. Beide Male wurde hervorgehoben, dass bereits andere Software zur verteilten Programmierung eingesetzt wurde, diese aber nicht das Gewünschte leistete. Das Konzept von Saros sei hingegen „fantastic“.

> „The tool is fantastic: keep up the good work. Our time was crying out for something like this as we all live hundreds of miles apart: the follow feature is top-notch, and in general the tool allows us to be far more productive than previous attempts to pair-program on VNC.“

Die Umfrage bot den Nutzern zusätzlich die Möglichkeit ihre E-Mail Adresse zu hinterlassen, um sich auf unserer *Announcement-Mailing-Liste* einzutragen, über die Ankündigungen zu neuen Saros Releases versendet werden. Immerhin drei Teilnehmer nutzten diese Möglichkeit, was ein tiefer gehendes Interesse als nur einmaliges Ausprobieren vermuten lässt.

[8] *Jabber.org* ist der öffentliche Server der Jabber Software Foundation

5 Statistik Framework

In diesem Kapitel werde ich auf die von mir entwickelte Architektur für das Statistik Framework sowie die zum Einsatz gekommenen Technologien eingehen und das Kapitel mit einem kurzen Überblick über die bisherigen Ergebnisse der Statistikerhebung abschließen.

5.1 Entwurf

Bei der Entwicklung einer Architektur für das Statistik Framework kam es mir vor allen Dingen auf die leichte Erweiterbarkeit an. Es sollte auf einfache Weise möglich sein, das Sammeln der Statistik um neue Werte zu erweitern, ohne den vorhandenen Code stark zu verändern. Das lässt sich durch eine *Implicit Invocation* Architektur erreichen. Sie wird von Garland und Shaw wie folgt beschrieben:

> „The idea behind implicit invocation is that instead of invoking a procedure directly, a component can announce (or broadcast) one or more events. Other components in the system can register an interest in an event by associating a procedure with the event. When the event is announced the system itself invokes all of the procedures that have been registered for the event.“ [GS93]

Durch *Implicit Invocation* wird eine lose Kopplung der einzelnen Module erreicht, da es für die Komponente, die die Ereignisse ankündigt, nicht von Belang ist, welche oder wie viele Objekte sich bei ihr registrieren und welche Aufgabe diese haben.

Ein weiteres Ziel meinerseits war, eine hohe Kohäsion innerhalb eines Moduls zu erreichen, d. h. dass jedes genau eine wohldefinierte Aufgabe erfüllt. Hieraus entstand die Idee, eine Schnittstelle *Collector* zu definieren, deren konkrete Implementierungen jeweils für einen spezifischen Aufgabenbereich in Saros statistische Werte sammeln (wie z. B. Daten zu Rollenwechseln einer Sitzung). Jeder konkrete Collector kann für sich als Modul angesehen werden, da er für eine wohldefinierte Aufgabe verantwortlich ist.
Um die gesammelten Daten der Collectoren entgegenzunehmen und gebündelt an unseren Server zu verschicken, wurde als zentrales Steuerungsmodul der *StatisticManager* entworfen. Eine Übersicht über die Architektur und die beteiligten Klassen zeigt Abbildung 5 auf der nächsten Seite. Aus Gründen der Übersichtlichkeit habe ich einige Methoden sowie Parameter nicht aufgeführt.

Die *Implicit Invocation* Architektur wird im Framework an zwei Stellen deutlich.

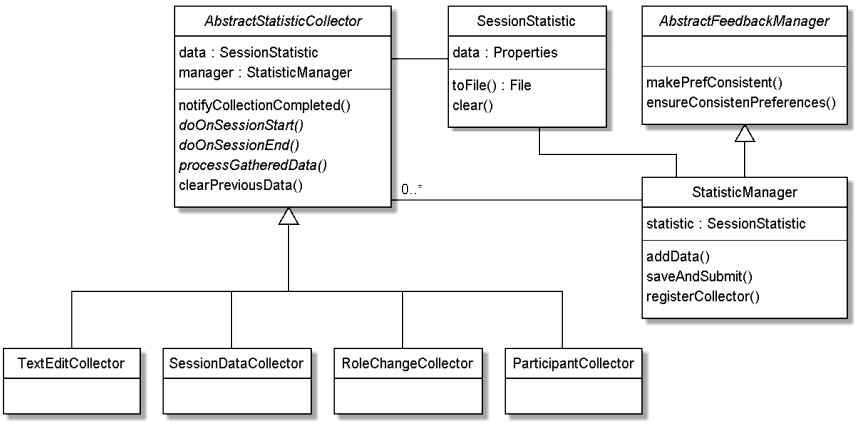

Abbildung 5: Klassendiagramm zum Statistik Framework

1. Die Collectoren sammeln die statistischen Daten über Listener-Mechanismen, die nach dem *Observer Entwurfsmuster* aufgebaut sind (siehe Abschnitt 5.2 auf der nächsten Seite) und werden nicht direkt, sondern implizit bei Auftreten eines Ereignisses aufgerufen.
2. Der StatisticManager erhält die Statistik-Daten jedes Collectors, sobald dieser die Daten verarbeitet hat, d. h. der StatisticManager fragt nicht wiederholt nach dem Ergebnis, sondern wird implizit aufgerufen, sobald das Ergebnis da ist.

Typischer Ablauf *PicoContainer* (siehe Abschnitt 5.3 auf Seite 29) ist für die Erzeugung der Collector-Exemplare beim Start des Plugins zuständig, welche sich anschließend beim StatisticManager registrieren. Am Ende einer Sitzung speichert jeder Collector die von ihm gesammelten Daten direkt oder nach weiterer Verarbeitung als Schlüssel-Wert-Paare in einer Datenstruktur (*SessionStatistic*) zwischen. Sobald dieser Prozess abgeschlossen ist, benachrichtigt der Collector den StatisticManager über die Fertigstellung seiner Arbeit und übergibt ihm die gesammelten Daten in der Datenstruktur.

Der StatisticManager ist wiederum dafür verantwortlich, die Daten der einzelnen Collectoren aufzusammeln und sie, nachdem alle eingetroffen sind, am Ende der Sitzung zusammengefasst in einer Datei an einen festgelegten Server zu übermitteln. Der StatisticManager braucht hierbei einen Mechanismus, um feststellen zu können, wann alle Daten bei ihm eingetroffen sind.

Dafür verwaltet er eine Liste von aktiven Collectoren, aus der er einen Collector entfernt, sobald sich dieser mit seinen Ergebnissen gemeldet hat. Gibt es keine aktiven Collectoren mehr, so sind alle Daten angekommen und können nun an den Server übertragen werden.

Erweiterbarkeit Um einen neuen Collector zu implementieren, reicht es aus, eine neue Klasse, die von *AbstractStatisticCollector* erbt, zu erstellen, die abstrakten Methoden zu implementieren und die Collectorklasse zur Verwaltung via PicoContainer an Saros zu übergeben. Der AbstractStatisticCollector kapselt dabei die Kommunikation mit dem StatisticManager, d. h. die Registrierung sowie die Übergabe der Daten am Sitzungsende erfolgt automatisch für jede Unterklasse.

Zustimmung zur Statistikübertragung Durch die Anforderungen ist festgelegt, dass die Erlaubnis des Benutzers zur Statistikübertragung einmalig vor der Sitzung eingeholt werden muss. Dadurch soll verhindert werden, dass das Plugin am Ende der Sitzung in einen undefinierten Zustand gelangt, wenn es nicht weiß, ob die Statistik übertragen werden darf oder nicht. Das führte zu der Entwurfsentscheidung, dass das Verbinden mit dem Jabber-Benutzerkonto nicht zugelassen werden soll, wenn der Nutzer keine Aussage zur Statistikübertragung abgegeben hat. Stattdessen wird ihm bei einem Verbindungsversuch ein Wizard angezeigt (siehe Abbildung 6 auf der nächsten Seite), der um Erlaubnis zur Übertragung fragt und Auskunft über die Art und den Zweck der Daten gibt. Dabei wird explizit auf die Anonymität der Daten hingewiesen. Er enthält außerdem einen Link zu einer von mir erstellten Wiki-Seite[9], auf der detailliert über die erhobenen Daten Auskunft gegeben wird.

5.2 Statistikerhebung

Je nach Aufgabengebiet registrieren die Collectoren entsprechende *Listener-Objekte* für die sie interessierenden Ereignisse. Die Listener-Mechanismen waren größtenteils bereits in der Saros-Software implementiert, nur die Benachrichtigung über lokale und entfernte Text-Ereignisse (Hinzufügen und Löschen von Text durch einen der Sitzungsteilnehmer) musste ich noch hinzufügen.

Der Listener-Mechanismus ist nach dem Observer-Entwurfsmuster aufgebaut, das von der *Gang of Four*[10] in ihrem Buch *Design Patterns - Elements of Reusable Object-Oriented Software* beschrieben wird [GHJV95].
Wie in Abbildung 7 auf Seite 27 dargestellt, verwaltet ein beobachtbares Objekt (*Observable*) eine Liste von Beobachtern (auch *Listener* oder *Observer*

[9] https://www.inf.fu-berlin.de/w/SE/DPPFeedback
[10] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides

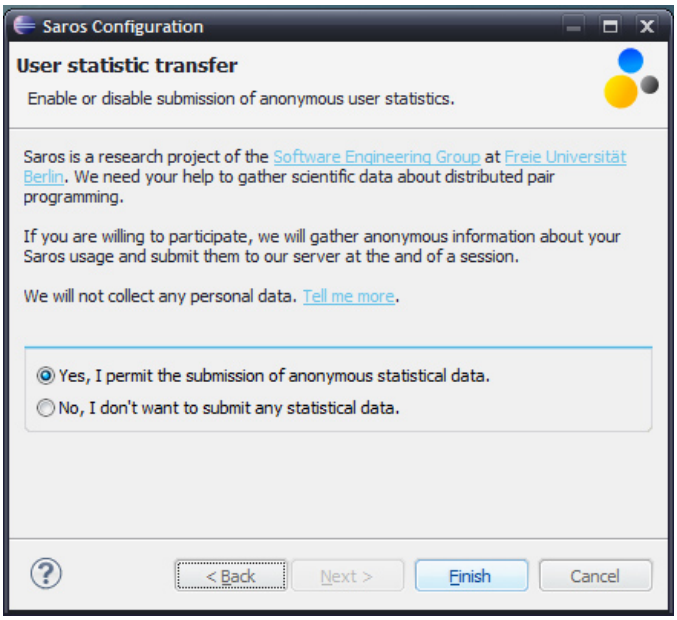

Abbildung 6: Frage nach der Erlaubnis zur Statistikübertragung

genannt), die sich bei ihm an- und abmelden können. Tritt eine Zustandsänderung des beobachtbaren Objektes auf, so werden alle Listener darüber benachrichtigt, indem eine Methode der gemeinsamen Schnittstelle aufgerufen wird. Bei Bedarf können der Methode noch Parameter übergeben werden, die das Ereignis spezifizieren.

Um z. B. über Rollenwechsel informiert zu werden, registriert der *RoleChangeCollector* eine Implementierung der *ISharedProjectListener* Schnittstelle beim *ISharedProject*. Ein *ISharedProject* entspricht dem momentan zwischen den Teilnehmern geteilten Projekt oder anders ausgedrückt, der aktuellen Sitzung. Über die *roleChanged*-Methode des Listeners wird der Collector über jeden Rollenwechsel benachrichtigt, indem als Parameter der Nutzer, dessen Rolle sich geändert hat und die neue Rolle übergeben werden.

Bei der Verarbeitung der Ereignisse durch die Collectoren wurde darauf geachtet, dass der Laufzeitoverhead während der Sitzung minimal gehalten wird, indem Berechnungen erst am Ende der Sitzung stattfinden.

Ein Beispiel, wie die von den einzelnen Collectoren gesammelten statistischen Werte im Detail aussehen, befindet sich im Anhang (siehe Abschnitt C auf Seite 39). Ich möchte an dieser Stelle nur überblickshaft beschreiben, welche Daten die von mir implementierten Collector-Klassen sammeln und welches Ziel jeweils dahinter steht.

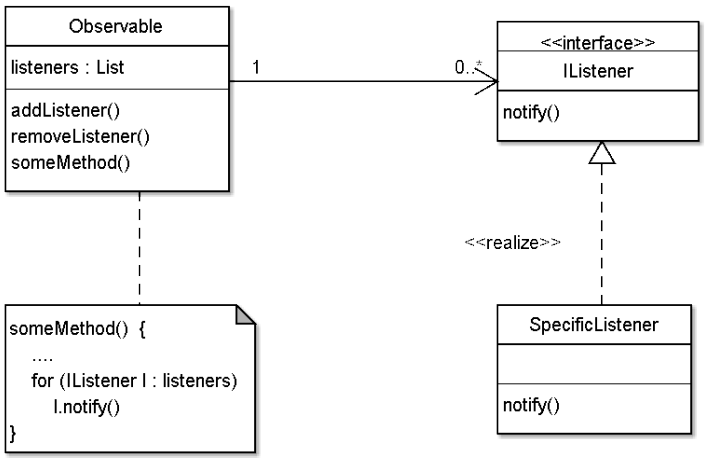

Abbildung 7: Observer-Entwurfsmuster, wie es in Saros benutzt wird

5.2.1 ParticipantCollector

Mittels dieser Klasse wird die Gesamtanzahl der Teilnehmer an der Sitzung gesammelt und für jede auftretende Anzahl gleichzeitig anwesender Teilnehmer, wie lange sie zusammen gearbeitet haben. Wenn z. B. drei Nutzer an einer Sitzung teilnehmen, so wird die Zeit, die sie gemeinsam in der Sitzung sind, gemessen. Verlässt ein Nutzer die Sitzung, so wird die Zeit, die zwei Nutzer zusammen anwesend waren, gemessen. Tritt nun wieder ein dritter Teilnehmer der Sitzung bei, so wird die Zeit von drei anwesenden Nutzern zu der bereits existierenden Zeit von drei Teilnehmern addiert. Dabei ist es irrelevant, ob die drei Personen den drei anfänglichen Teilnehmern entsprechen oder ob ein völlig neuer Teilnehmer hinzukam. Dies spiegelt sich nur in der Gesamtanzahl der Teilnehmer wider, die im ersten Fall drei und im zweiten Fall vier betragen würde.

Die Daten werden mit dem Ziel erhoben festzustellen, welches die durchschnittliche Teilnehmerzahl bei kollaborativen Programmierungssitzungen ist und ob die Gruppierung zwischendurch wechselt.

5.2.2 RoleChangeCollector

Er sammelt die Anzahl der Rollenwechsel des lokalen Nutzers und welche Rolle (*Driver* oder *Observer*) wie lange ausgeübt wurde, wobei jeweils sowohl die Zeit gemessen als auch die Prozentzahl im Vergleich zur gesamten Sitzungszeit berechnet wird.

Durch diese Informationen soll festgestellt werden, wie stark das Konzept der klassischen Paarprogrammierung verfolgt wird: Wird die Trennung zwi-

schen Driver und Observer eingehalten und werden die Rollen regelmäßig getauscht?

5.2.3 SessionDataCollector

Er sammelt einige generelle Informationen zur Sitzung, die sich nicht unbedingt von Sitzung zu Sitzung verändern. Dazu gehören Informationen über die Umgebung, in der das Saros-Plugin läuft (Eclipse-Version, Java-Version, Betriebssystem) sowie die aktuellen Einstellungen zum Umfragesystem (aktiviert/deaktiviert, Intervall) und die User-ID. Zusätzlich werden veränderliche Informationen wie die Sitzungszeit, die Session-ID, die Anzahl der bisher gestarteten Sitzungen gespeichert und ob der lokale Nutzer der Host der Sitzung war oder nicht.

Über diese Informationen wollen wir herausfinden, in welchem (Software-) Umfeld Saros eingesetzt und wie das integrierte Umfragesystem angenommen wird: Wird es sofort deaktiviert? Des Weiteren lässt sich feststellen, in welchem Umfang Saros benutzt wird: Gibt es überhaupt Nutzer, die Saros mehr als einmal einsetzen?

5.2.4 TextEditCollector

Er sammelt Informationen über die lokal getätigten Text-Ereignisse, d. h. wie viele Zeichen vom lokalen Benutzer während der Sitzung getippt wurden und wie häufig Text-Ereignisse auftraten. Diese beiden Werte können voneinander abweichen. Zwar wird für jedes vom Benutzer getippte Zeichen genau ein Text-Ereignis ausgelöst, beim Einsatz von *Copy&Paste* sowie bei der automatischen Generierung von Methoden über die Eclipse-Funktionen oder bei *Refactorings* kann ein Ereignis jedoch viele Zeichen umfassen. Eine Abweichung der beiden Werte gibt demnach Aufschluss über die Häufigkeit des Einsatzes dieser Mechanismen.

Des Weiteren werden die lokalen Text-Ereignisse mit den entfernt ausgelösten Ereignissen in Bezug auf Parallelität verglichen. Für verschiedene Zeitintervalle (z. B. 1s, 2s, 5s) wird hierbei überprüft, ob während dieses Zeitschlitzes sowohl ein lokales als auch ein entferntes Text-Ereignis eingetroffen sind. Falls ja, wird die Anzahl der in diesem Intervall vom lokalen Nutzer getippten Zeichen aufaddiert. Alle lokalen Text-Ereignisse, die sich auf diese Weise keinem Intervall zuordnen lassen, werden als *nicht parallel* aufsummiert. Zusätzlich zu den gezählten Zeichen, die parallel geschrieben wurden, wird auch die Anzahl der Text-Ereignisse erfasst, denn andernfalls könnte eine falsche Schlussfolgerung entstehen. Wenn der lokale Benutzer z. B. eine größere Menge Text innerhalb von zwei Sekunden mit dem Auftreten einer Text-Aktivität eines anderen Teilnehmers einfügt, werden diese Zeichen als parallel bezüglich des Intervalls „1s“ gezählt. Das würde bei der

Auswertung der Statistik die Vermutung nahe legen, dass die Teilnehmer sehr stark parallel gearbeitet hätten, wobei jedoch in Wahrheit nur eine einzige Copy&Paste-Aktivität parallel stattfand.

Von dem zeitlichen Vergleich der Text-Ereignisse erhoffen wir uns, nach einer ausreichenden Menge gesammelter Daten, eine Aussage über den Einsatz der Mehrschreiberfunktionaliät von Saros treffen zu können: Wird überhaupt (annähernd) gleichzeitig geschrieben? Und falls ja, wie häufig?

5.3 Technologien

Bei der Implementierung des Statistik Frameworks kamen einige Bibliotheken zum Einsatz, deren Verwendungszweck ich im Folgenden kurz skizzieren möchte. Gerade in Bezug auf die Übertragung der Statistik zum Server bot sich der Rückgriff auf bewährte Lösungen an.

PicoContainer PicoContainer ist ein Open-Source *Dependency Injection Framework* [Fow04]. Bei *Dependency Injection* handelt es sich um ein Entwurfsmuster, das die Abhängigkeiten zwischen Komponenten oder Objekten minimiert, indem die Verantwortung für das Erzeugen und Verknüpfen von Objekten an einen Container übertragen wird.
In Saros kommt PicoContainer vielfältig zum Einsatz. Er verwaltet alle Objekte, die während des gesamten Plugin-Lebenszyklus als Singletons existieren müssen.

Jakarta Commons HttpClient Der HttpClient ist eine Bibliothek, die Methoden für den komfortablen Zugriff auf Ressourcen via HTTP bereitstellt [JCH].
Die Bibliothek kommt bei der Statistik-Übertragung auf der Client-Seite zum Einsatz. Mithilfe der bereitgestellten Methoden wird eine Datei auf einen Server mittels `HTTP-POST` und Content-Type `multipart/form-data` hochgeladen.

Apache Commons FileUpload Commons FileUpload ist eine Bibliothek, die serverseitig (z. B. in Servlets) das Hochladen von Dateien ermöglicht, indem sie Methoden bereitstellt, um eine `HTTP-POST` Anfrage mit Content-Type `multipart/form-data` zu parsen und das Ergebnis zur Weiterverarbeitung zur Verfügung zu stellen [ACF]. Die Bibliothek kommt im Servlet, an das die Statistik-Dateien gesendet werden, zum Einsatz.

5.4 Statistikergebnisse

Wie in der Einleitung bereits erwähnt, ist es nicht Ziel dieser Arbeit, eine Auswertung der erhobenen Daten durchzuführen. Im Folgenden möchte ich daher nur einen kurzen Überblick über die bisherigen Ergebnisse geben.

Das Statistik Framework wurde in die Release-Version 9.6.23 des Saros-Plugins eingebaut, es war allerdings erst ab Release 9.7.10 vollständig einsatzfähig, d. h. alle oben beschriebenen Collectoren waren implementiert.
In der Zeit vom 10.07.09 bis heute (29.07.09), das entspricht 20 Tagen, sind insgesamt 71 Statistikdateien auf unserem Server angekommen, wovon 66 von der neueren Version 9.7.10 stammen. Statistiken, die vom Entwicklerteam erzeugt wurden, sind hierin nicht enthalten, denn diese werden serverseitig ausgefiltert. Das Saros-Plugin wurde in diesem Zeitraum laut SourceForge Statistik 583 Mal heruntergeladen [SFs].

Die 71 Statistikdateien gehören zu 51 verschiedenen Sitzungen, in einigen Fällen haben also mehrere Teilnehmer derselben Sitzung die Übertragung erlaubt. Die Statistiken wurden des Weiteren von 19 verschiedenen Benutzern erzeugt, wobei mehrere Sitzungen eines Nutzers zumeist am selben Tag stattfanden. Durch den Vergleich mit der Downloadzahl lässt sich eine Teilnehmerquote von 3,3% errechnen, die damit ungefähr achtmal größer als die Rücklaufquote der Umfrage ist. Wieder ist zu beachten, dass nicht alle Nutzer, die das Plugin herunterladen, dieses auch einsetzten. Eine zurückhaltende Schätzung, bei der nur jeder fünfte Nutzer tatsächlich eine Sitzung gestartet hat, ergibt eine Teilnehmerquote von ungefähr 16%.

Auffällig ist außerdem, dass beinahe die Hälfte der 71 Statistikdateien (44%) an einem Tag von zwei zusammenarbeitenden Benutzern übertragen wurden. 61% der Sitzungen fanden mit zwei Teilnehmer statt, 35% mit nur einem Teilnehmer und die restlichen 4% mit mehr als zwei Teilnehmern. Die Sitzungen mit nur einem Teilnehmer sind eventuell auf Probleme bei der Einladung anderer Nutzer zurückzuführen oder Saros sollte einfach nur kurz getestet werden.

Zusammenfassend lässt sich sagen, dass auch die Statistikerhebung mangelnde Repräsentativität aufweist, da fast die Hälfte der Daten von nur zwei Nutzern stammen. Der Zeitraum ist zudem zu kurz für eine effektive Auswertung. Als Ergebnis kann zumindest festgehalten werden, dass mehr Benutzer der Statistikübertragung zustimmen, als an der Umfrage teilnehmen und dass Saros hauptsächlich von Zweier-Gruppen und selten von größeren Gruppen eingesetzt wird.

6 Fazit

6.1 Zusammenfassung

In dieser Arbeit wurden zwei Systeme zur Erhebung von Benutzerfeedback in Saros entworfen, implementiert und in der Praxis eingesetzt.

Zum einen handelt es sich um ein Umfragesystem, das Saros-Benutzer in konfigurierbaren Intervallen am Ende einer Sitzung zur Teilnahme an einer Online-Umfrage auffordert. Hierbei wurde besonderes Augenmerk darauf gelegt, dass die Benutzer so wenig wie möglich mit der Konfiguration des Umfragesystems (Festlegung des Intervalls, Aktivierung/Deaktivierung) behelligt werden sollten, um sie nicht von dem weiteren Einsatz von Saros abzuschrecken.

Die Integration des Umfragesystems in Saros hat den Vorteil, dass potentiell alle Nutzer des Werkzeugs erreicht werden können. Wie der Produktiveinsatz jedoch zeigte, ist mit einer sehr geringen Rücklaufquote von weniger als 3% zu rechnen. Von dem Ziel, über diesen Weg repräsentative Umfragen durchführen zu können, sollte daher Abstand genommen werden. Der Fokus sollte hingegen auf die Meinungsforschung mit offenem Antwortformat gelegt werden, denn hierbei können auch schon wenige Antworten einen hohen Nutzen bringen, wie die ersten Ergebnisse der Kurz-Umfrage zeigten. Durch sie konnten z. B. neue Anforderungen identifiziert und Probleme aufgedeckt werden. Nicht zu unterschätzen ist zudem die Wirkung von positivem Feedback auf die Zufriedenheit und Motivation der Entwickler. Der ehemals gänzlich unbekannte Nutzer bekommt auf einmal „ein Gesicht“ und es wird deutlich für wen die Entwicklungsarbeit geleistet wird.

Zum anderen wurde ein Statistik Framework entwickelt, mit dessen Hilfe während einer Programmiersitzung mit Saros statistische Nutzungsdaten gesammelt und an das Entwicklerteam übermittelt werden können. Das Framework wurde unter dem Gesichtspunkt der einfachen Erweiterbarkeit entwickelt, um je nach Bedarf weitere Daten erheben zu können. Bisher werden Daten gesammelt, die eine Sitzung mit Saros grundlegend beschreiben (Sitzungszeit, Teilnehmerzahl, Rollenwechsel, Umgebung etc.).

Die bisherigen Ergebnisse der Statistikerhebung haben gezeigt, dass auch hier Probleme mit mangelnder Repräsentativität auftreten. Überdurchschnittlich aktive Nutzer können die Ergebnisse mit ihren Daten dominieren und die Auswertung verfälschen. Durch eine zufällige Auswahl einer Untermenge von Sitzungen dieser Nutzer könnte dieses Problem jedoch umgangen werden. Die Teilnehmerquote ist auch hier gering (10 - 15%), sie liegt jedoch im Vergleich zur Rücklaufquote der Umfrage wesentlich höher. Dies dürfte vor allen Dingen daran liegen, dass der Nutzer bei der Statistikerhebung nicht

aktiv gefordert wird, sondern dieser nur einmalig zustimmen muss. Auch wenn die Ergebnisse derzeit nicht repräsentativ sind, lassen sich aus ihnen gewisse Trends im Nutzungsverhalten ablesen. Bei einer ersten, kurzen Auswertung hat sich z. B. abgezeichnet, dass Saros bisher sehr selten von Teams mit mehr als zwei Mitgliedern verwendet wird.

Das Statistik Framework kann außerdem im Bereich der detaillierten Analyse einzelner Paare (*PP-Coaching*)[Ros09][Plo] hilfreiche Daten liefern, die ohne automatisierte Methoden schwer zu erheben sind (z. B. die Anzahl geschriebener Zeichen in einer Sitzung und der Grad der Parallelität von Schreibaktivitäten).

6.2 Ausblick

Nachdem mit meiner Arbeit die Infrastruktur für eine umfangreiche Datenerhebung mit Saros geschaffen wurde, geht es jetzt darum, diese effektiv einzusetzen.

Im Bereich des Umfragesystems sollte der Ansatz des Austauschs der Umfragen weiter verfolgt und ein umfangreicherer Fragenkatalog für ein wohldefiniertes Ziel entwickelt werden. Man muss hierbei beachten, wie viel Vorwissen die Umfrage über Saros und Paarprogrammierung voraussetzt. Denkbar wäre es, z. B. erst ab einer bestimmten Anzahl absolvierter Sitzungen eine andere Umfrage anzubieten. Das hat den Vorteil, dass man bei einem Nutzer, der bereits zehn Sitzungen absolviert hat, von einem anderen Wissenstand bezüglich Saros ausgehen und die Fragen entsprechend detaillierter formulieren kann.

Bei der Ausarbeitung weiterer Umfragen ist vor allen Dingen die als sehr gering erwartete Teilnehmerquote und die mangelnde Repräsentativität zu beachten. Es sollten daher Mittel und Wege diskutiert werden, mit denen die Motivation der Benutzer zur Teilnahme verstärkt werden kann. Eines der Probleme bei der Motivation ist, dass die Nutzer keinen konkreten Gegenwert für ihren Zeitaufwand sehen. Eine erste Idee hierzu ist die Beantwortung der Umfrage mit der Teilnahme an einer Verlosung zu verbinden. Der Wert der möglichen Preise muss selbstverständlich in kleinem Rahmen gehalten werden (Tasse mit FU-Logo, T-Shirt mit Saros Logo, etc.). Außerdem müssen Kosten für das Versenden der Preise (eventuell ins Ausland) miteinkalkuliert und daraufhin eine Kosten-Nutzen-Analyse durchgeführt werden.

Im Bereich der Statistikerhebung wird bisher eine gewisse Grundmenge an Daten gesammelt, die nun durch Hinzufügen weiterer Collectoren auf eine noch zu definierende Forschungsfrage ausgerichtet werden kann. Beispielsweise könnte die Frage untersucht werden, ob Saros zur verteilten *Paar*programmierung oder zur gleichzeitigen, kollaborativen Programmierung eingesetzt wird. Hierbei sollte gemessen werden, inwieweit die Benutzer

sich an das Konzept der klassischen Paarprogrammierung halten, bzw. wie sehr sie von Funktionen, die von diesem Konzept abweichen, wie dem *Multi-Driver-Modus* oder der Möglichkeit des unabhängigen Explorierens des Codes, Gebrauch machen. Das Augenmerk der Statistikerhebung könnte dabei mehr in Richtung des Aufmerksamkeitsfokusses des Benutzers verschoben werden, um zu messen, wie häufig die Aufmerksamkeit der Teilnehmer auf dieselbe Sache gerichtet ist bzw. wie häufig nicht. Dafür müssen geeignete Maße gefunden und implementiert werden.

Ein erster Ansatz in dieser Richtung ergibt sich aus den bereits implementierten *Viewports* für jeden Benutzer, die den aktuellen Sichtbereich jedes Teilnehmers farbig markieren. Der Sichtbereich gibt an, von welcher Zeile bis zu welcher Zeile ein Nutzer den Inhalt der gerade geöffneten Datei sehen kann. Für die Teilnehmer einer Sitzung könnten somit paarweise die Überlappungen der Sichtbereiche berechnet werden. Hierbei wären Maße wie „vollständige Überdeckung“, „teilweise Überlappung“ und „keine Überlappung“ denkbar.

Ein sehr wichtiger Punkt, der zudem in der nahen Zukunft verfolgt werden sollte, ist die Auswertung der Daten. Hierfür muss ein weitgehend automatisiertes Verfahren entwickelt werden, dass die statistischen Daten nutzbringend aufbereitet.

Des Weiteren wurde durch das Statistik Framework die Grundlage für das Übertragen beliebiger Dateien an unseren Server geschaffen. Saros kann somit auf einfache Weise um einen *Crash-Reporting-Mechanismus* erweitert werden, über den wir Fehlerberichte zu den Sitzungen, die bisher nur lokal gespeichert werden, an unseren Server übertragen können.

Ebenfalls sollte der Ansatz der Zusammenführung von Umfrageergebnissen mit den statistischen Daten weiter verfolgt werden. Wie im Grundlagenkapitel angedeutet, können nicht alle Daten einer Sitzung durch reine Beobachtung erhoben werden. Wenn Umfrageergebnisse über die Sitzungs- und Benutzer-ID einer Statistikdatei zugeordnet werden können, eröffnet das neue Auswertungsmöglichkeiten. Zum Beispiel lässt sich mehr über die Motivationen der einzelnen Nutzer in der Sitzung erfahren, wodurch die rein objektiven, statistischen Daten in einen Kontext gesetzt werden können.

A Anwendungsfälle

A.1 Umfragesystem

Im Folgenden ist der Hauptakteur der Anwendungsfälle stets der lokale Nutzer, weswegen ich ihn nicht jedes Mal explizit erwähne. Das betrachtete System (*System under Discussion*) ist Saros.

Synchronisieren der Feedback-Einstellungen (US01)

Vorbedingung: keine

Nachbedingung: Die Feedback-Einstellungen im Workspace und in der globalen Konfiguration sind konsistent (siehe Abschnitt 4.1.1 auf Seite 16).

Ablauf:

1. Der Benutzer installiert Eclipse und das Saros-Plugin neu.
2. Der Benutzer startet Eclipse mit einem bestehenden Workspace (d. h. einem Workspace, der vorher schon einmal mit Saros verwendet wurde).

Alternativen:

1a. Der Benutzer installiert Eclipse nicht neu.

2a. Der Benutzer verwendet einen neuen Workspace.

Umfrageaufforderung (US02)

Vorbedingung: Der Benutzer befindet sich in einer Sitzung. Das Umfragesystem ist aktiviert und es sind bereits so viele Sitzungen durchgeführt worden, dass das eingestellte Intervall nach dieser Sitzung abläuft.

Nachbedingung: Das Intervall für die Umfrageaufforderung wurde zurückgesetzt.

Ablauf:

1. Der Benutzer verlässt die Sitzung.
2. Der Benutzer sieht den Dialog, der ihn zur Teilnahme an einer Umfrage zu Saros auffordert.
3. Der Benutzer setzt keinen Haken bei *„Don't ask me again“*.
4. Der Benutzer schließt den Dialog mit einem Klick auf *„No, not now“*.

Alternativen:

3a. Der Benutzer setzt einen Haken bei *„Don't ask me again“*.

3a1. Das System deaktiviert das Umfragesystem, nach dem Schließen des Dialogs.

4a. Der Benutzer schließt den Dialog mit einem Klick auf *„Yes“*.

4a1. Die Webseite mit der Umfrage wird im Standard-Browser geöffnet.

4a2. Das Öffnen der Webseite im Standard-Browser schlug fehl. Es wird versucht die Webseite im internen Eclipse-Browser zu öffnen. Wenn auch dies fehlschlägt, wird eine entsprechende Meldung sowie die Umfrage-URL angezeigt.

4b. Der Benutzer schließt den Dialog über das Kreuz in der rechten oberen Ecke.

Ändern der Feedback-Einstellugen (US03)

Vorbedingung: keine

Nachbedingung: Die getätigten Einstellungen wurden global und lokal gespeichert.

Ablauf:

1. Der Benutzer öffnet die Feedback Konfigurationsseite.
2. Der Benutzer wählt ein anderes Intervall aus dem Auswahlmenü.
3. Der Benutzer verlässt mit einem Klick auf *OK* die Einstellungsseite.
4. Das System setzt das Intervall zurück.

Alternativen:

2a. Der Benutzer aktiviert das Feedback.

2a1. Das System setzt das Intervall zurück.

2b. Der Benutzer deaktiviert das Intervall.

2c. Der Benutzer startet die Umfrage direkt über den Button *Start now.*

2c1. Das System setzt das Intervall zurück.

A.2 Statistikerhebung

Da der Nutzer an der Statistikerhebung nicht aktiv beteiligt ist, sondern nur die Ereignisse generiert, die daraufhin vom Statistik Framework gesammelt und verarbeitet werden, existieren hierfür keine Anwendungsfälle. Es existiert jedoch ein Anwendungsfall zur Erfragung der Zustimmung zur Statistikübertragung.

Zustimmung zur Statistikübertragung (SE01)

Vorbedingung: Der Benutzer hat die Statistikübertragung bisher weder erlaubt noch abgelehnt.

Nachbedingung: Die Antwort des Benutzers wurde global und lokal gespeichert. Der Benutzer ist mit seinem Jabber-Konto verbunden.

Ablauf:

1. Der Benutzer klickt auf das Symbol zur Verbindung im Roster.
2. Der Benutzer sieht einen Dialog, der ihn um Erlaubnis zur Übermittlung von statistischen Daten bittet.
3. Der Benutzer klickt auf *Finish*.

Alternativen:

2a. Der Benutzer wählt *„No, I don't want to submit any statistical data."*

3a. Der Benutzer bricht den Wizard ab.

3b1. Der Benutzer wird nicht verbunden.

B Umfrage

B.1 Mögliche Ziele einer Umfrage

1.

Objekt	Saros
Zweck	Verstehen der Benutzbarkeitseinschätzung, Verbessern von Saros
Fokus	Qualität der Benutzbarkeit
Blickwinkel	Benutzer, Entwickler
Kontext	Bisherige Saros Sitzungen des Befragten

2.

Objekt	Anforderungen an Saros
Zweck	Verstehen ("Wofür wird Saros benutzt und von wem?")
Fokus	Effektivität (im Sinne von Verhältnis der gesetzten Ziele des Nutzers zu tatsächlich erreichtem Ziel), Zuverlässigkeit
Blickwinkel	Saros Team
Kontext	Letzte Saros Sitzung des Benutzers

3.

Objekt	DPP
Zweck	Charakterisieren von DPP, Evaluieren gegenüber anderen Methoden
Fokus	Effektivität von DPP. Wie gut werden Ziele erreicht?
Blickwinkel	DPP-Forscher
Kontext	Nutzung von DPP-Werkzeugen, insbesondere Saros

B.2 Kurzumfrage

Dies ist der Text der ersten, kurzen Umfrage, wie er dem Benutzer bei seiner Teilnahme präsentiert wird. Bei der ersten Antwort kann eine der Antwortmöglichkeiten ausgewählt werden, bei den restlichen Fragen sind Freitextantworten gefordert.

Short Saros User Survey

Tell us about your latest Saros session.

Saros is a research project of the Software Engineering Group at Freie Universität Berlin. In order to keep this project going and to constantly improve the software we need your opinion about Saros.

Our goal for this survey is to determine for what Saros is mostly used and what kind of problems still occur. The survey contains only four questions that shouldn't take you longer than five minutes to answer.

1. How did your latest Saros session work out?
   - Everything went great.
   - Everything went as expected.
   - There were some minor problems, but we were able to complete our task(s).
   - There were some major problems that prevented us from completing our task(s).
   - Other:
2. Which goals did you want to accomplish in your latest Saros session?
3. What didn't go well (if anything) and what happened?
4. Do you have any other comments to share with us (e.g about the interface, specific features or possible improvements)?
5. If you want to subscribe to our low-volume announce list on SourceForge to receive the latest news about Saros, you can enter your email address here.

   Your address is only used for this purpose and will not be passed on to third parties.

C Beispiel für eine Statistikdatei

Das folgende Listing zeigt ein Beispiel für den Inhalt einer Statistikdatei. Aus den Informationen kann man ablesen, dass es sich um eine Sitzung mit zwei Teilnehmern handelte, die siebeneinhalb Minuten dauerte. Der lokale Nutzer war Host der Sitzung und hat zwei Mal seine Rolle gewechselt, wobei er die meiste Zeit ein Driver war. Beide Teilnehmer haben einige Zeichen gleichzeitig geschrieben, d. h. es wurde der *Multi-Driver-Modus* eingesetzt.

Listing 1: Inhalt einer Statistikdatei

```
#Saros session data
#Wed Jul 29 15:38:55 CEST 2009
data_transfer.Jingle/TCP.average_throughput_kbs=0.9
data_transfer.Jingle/TCP.number_of_events=4
data_transfer.Jingle/TCP.total_size_kb=2
data_transfer.Jingle/TCP.total_time_ms=3177
eclipse.version=3.5.0.v20090525
feedback.disabled=false
feedback.survey.interval=5
java.version=1.6.0_14
os.name=Windows XP
random.user.id=2009-07-29_15-38-55_575021536
role.1.duration=2.91
role.1=driver
role.2.duration=1.75
role.2=observer
role.3.duration=2.86
role.3=driver
role.changes=2
role.driver.duration=5.77
role.driver.percent=77
role.observer.duration=1.75
role.observer.percent=23
saros.version=9.7.10.r1509
session.count=1
session.id=819750571
session.time.users.1.percent=31
session.time.users.1=2.32
session.time.users.2.percent=69
session.time.users.2=5.19
session.time=7.51
session.users.total=2
textedits.chars=109
textedits.count=104
textedits.nonparallel.chars=83
textedits.nonparallel.percent=79
textedits.parallel.interval.1.chars=8
textedits.parallel.interval.1.count=8
textedits.parallel.interval.1.percent=8
textedits.parallel.interval.10.chars=11
textedits.parallel.interval.10.count=11
textedits.parallel.interval.10.percent=10
textedits.parallel.interval.15.chars=6
textedits.parallel.interval.15.count=6
textedits.parallel.interval.15.percent=6
user.is.host=true
```

Literatur

[ACF] Apache commons FileUpload. http://commons.apache.org/fileupload/.

[ADT05] Mustafa Ally, Fiona Darroch, and Mark Toleman. A Framework for Understanding the Factors Influencing Pair Programming success. In *In Proceedings of the XP 2005 Conference*, pages 82–91, 2005.

[Att03] Jürgen Atteslander, Peter und Cromm. *Methoden der empirischen Sozialforschung*. Walter de Gruyter, 10. edition, 2003.

[BCR94] Victor Basili, Gianluigi Caldiera, and Dieter H. Rombach. The goal question metric approach. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley, 1994.

[BDR96] Lionel C. Briand, Christiane M. Differding, and H. Dieter Rombach. Practical Guidelines for Measurement-Based Process Improvement. http://wwwagse.informatik.uni-kl.de/pubs/repository/briand96c/guidelines-96.pdf, 1996.

[BGS02] Prashant Baheti, Edward F. Gehringer, and P. David Stotts. Exploring the Efficacy of Distributed Pair Programming. In *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2002*, pages 208–220, London, UK, 2002. Springer-Verlag.

[BM01] Marianne Bertrand and Sendhil Mullainathan. Do People Mean What They Say? Implications for Subjective Survey Data. *American Economic Review*, 91(2):67–72, May 2001.

[But] Butterfat LLC. phpESP (php Easy Survey Package). http://www.butterfat.net/wiki/Projects/phpESP/. online, accessed 2009-07-30.

[CA99] Mary J. Culnan and Pamela K. Armstrong. Information Privacy Concerns, Procedural Fairness, and Impersonal Trust: An Empirical Investigation. *Organization Science*, 10(1):104–115, 1999.

[CAH03] Kevin Crowston, Hala Annabi, and James Howison. Defining Open Source Software Project Success. In *ICIS 2003. Proceedings of International Conference on Information Systems 2003*, pages 327–340, 2003.

[CW00] Alistair Cockburn and Laurie Williams. The Costs and Benefits of Pair Programming. In *eXtreme Programming and Flexible Processes in Software Engineering XP2000*, pages 223–247. Addison-Wesley, 2000.

[DHL96] Christiane Differding, Barbara Hoisl, and Christopher M. Lott. Technology Package for the Goal Question Metric. Technical report, University of Kaiserslautern and Fraunhofer Institute for Experimental Software Engineering, 1996.

[Dje06] R. Djemili. Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierungverteilter Paarprogrammierung. Master's thesis, Freie Universität Berlin, Inst. für Informatik, 2006.

[DM92] William H. DeLone and Ephraim R. McLean. Information Systems Success: The Quest for the Dependent Variable. *Information Systems Research*, 3:60–95, 1992.

[DOS07] Riad Djemili, Christopher Oezbek, and Stephan Salinger. Saros: Eine Eclipse-Erweiterung zur verteilten Paarprogrammierung. In Wolf-Gideon Bleek, Henning Schwentner, and Heinz Züllighoven, editors, *Software Engineering (Workshops)*, volume 106 of *LNI*, pages 317–320. GI, 2007.

[EC08] Dan Rubel Eric Clayberg. *Eclipse plug-ins*. Addison-Wesley, third edition, 2008.

[EF] What is a preference scope? http://wiki.eclipse.org/FAQ_What_is_a_preference_scope%3F.

[Fog05] Karl Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media, Inc., 2005.

[Fow04] Martin Fowler. Inversion of Control Containers and the Dependency Injection pattern. http://www.martinfowler.com/articles/injection.html, January 2004.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[GS93] David Garlan and Mary Shaw. An Introduction to Software Architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, pages 1–39. World Scientific Publishing Company, 1993.

[Gus07] Björn Gustavs. Weiterentwicklung des Eclipse-Plug-Ins Saros zur Verteilten Paarprogrammierung. http:

//www.inf.fu-berlin.de/inst/ag-se/theses/Gustavs_Saros_DPPII.pdf, October 2007.

[Jac09] Christoph Jacob. Weiterentwicklung eines Werkzeuges zur verteilten, kollaborativen Softwareentwicklung. Master's thesis, Freie Universität Berlin, 2009.

[JCH] Jakarta commons HttpClient. http://hc.apache.org/httpclient-3.x/.

[MCR] Mozilla Crash Reporter: Breakpad. http://kb.mozillazine.org/Breakpad. http://kb.mozillazine.org/Breakpad.

[MS03] Michele Marchesi and Giancarlo Succi, editors. *Extreme Programming and Agile Processes in Software Engineering, 4th International Conference, XP 2003, Genova, Italy, May 25-29, 2003 Proceedings*, volume 2675 of *Lecture Notes in Computer Science*. Springer, 2003.

[NJOL05] Jerzy R. Nawrocki, Michal Jasiñski, Lukasz Olek, and Barbara Lange. Pair Programming vs. Side-by-Side Programming. In Ita Richardson, Pekka Abrahamsson, and Richard Messnarz, editors, *EuroSPI*, volume 3792/2005 of *Lecture Notes in Computer Science*, pages 28–38. Springer, 2005.

[O'N01] Dara O'Neil. Analysis of Internet Users' Level of Online Privacy Concerns. *Social Science Computer Review*, 19(1):17–31, 2001.

[Plo] Laura Plonka. Analysis of Professional Pair Programmers. https://www.inf.fu-berlin.de/w/SE/AnalysisOfProfessionalPairProgrammers.

[PPG] Pair Programming Group. https://www.inf.fu-berlin.de/w/SE/PairProgrammingHome.

[RGBM05] Gregorio Robles, Jesus M. Gonzales-Barahona, and Martin Michlmayr. Evolution of Volunteer Participation in Libre Software Projects: Evidence from Debian. In Marco Scotto and Giancarlo Succi, editors, *Proceedings of the First Conference on Open Source Systems Genova*, pages 100–1007, 2005.

[Rie08] Oliver Rieger. Weiterentwicklung einer Eclipse-Erweiterung für verteilte Paar-Programmierung im Hinblick auf Kollaboration und Kommunikation. Master's thesis, Freie Universität Berlin, 2008.

[Rin09] Marc Rintsch. Agile Weiterentwicklung eines Software-Werkzeuges zur verteilten Paarprogrammierung. https:

//www.mi.fu-berlin.de/wiki/pub/Main/MarcRintsch/studienarbeit.pdf, June 2009.

[Ros09] Edna Rosen. Bewertung verteilter Paarprogrammierung im betrieblichen Umfeld. https://www.inf.fu-berlin.de/w/SE/ThesisDPPManagement, 2009.

[SF] What is SourceForge.net? http://sourceforge.net/apps/trac/sourceforge/wiki/What%20is%20SourceForge.net?

[SFs] Usage Statistics For Saros - Distributed Pair Programming. http://sourceforge.net/project/stats/?group_id=167540&ugn=dpp&type=&mode=60day.

[UDC] Eclipse: Usage Data Collector. http://www.eclipse.org/epp/usagedata/. http://www.eclipse.org/epp/usagedata/.

[WK02] Laurie Williams and Robert Kessler. *Pair Programming Illuminated*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[WM03] Daniela Schlütz Wiebke Möhring. *Die Befragung in der Medien- und Kommunikationswissenschaft: Eine praxisorientierte Einführung*. VS Verlag, 2003.