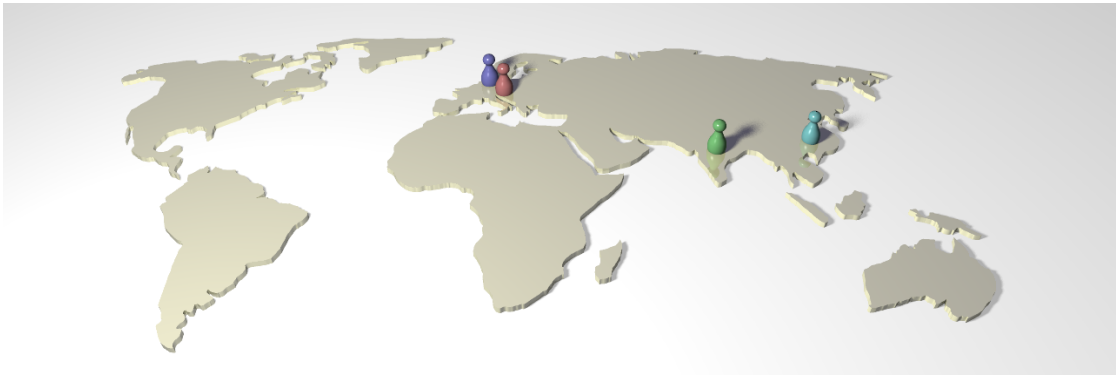
Freie Universität Berlin

Diplomarbeit zur Erlangung des akademischen Grades Diplominformatiker

Technisches Projektmanagement im OpenSource-Projekt Saros



Marc Rintsch

Matrikelnummer: 3429607

marc@rintsch.de

Eingereicht bei: Prof. Dr. Lutz Prechelt

Betreuer: Christopher Oezbek und Stephan Salinger

Berlin, 5. März 2010

Danksagung

Dank geht an Christoph für's Überreden zum *Saros*-Team zu kommen, Christopher und Stephan für's Betreuen, Edna für die Infrastruktur zur Aufzeichnung von *Teles*-Sitzungen und Unterstützung bei der Einarbeitung, Inés und Christoph für die zahlreichen Korrekturvorschläge, dem *Saros*-Team für die freundliche und angenehme Arbeitsatmosphäre, Andreas für moralische Unterstützung und aufmunternde Worte, und – last but not least – meiner Mutter für ihre Geduld und Liebe.

Ich versichere, dass ich die vorliegende Arbeit mit dem Titel "Technisches Projektmanagement im OpenSource-Projekt Saros" selbstständig verfasst habe. Alle Stellen der Arbeit, die anderen Werken wörtlich oder sinngemäss entnommen sind, sind unter Angabe der Quelle als Entlehnung kenntlich gemacht.

Berlin, den 5. März 2010
Marc Rintsch

Inhaltsverzeichnis

1 Einleitung **5**
- 1.1 Ziele . . . 5
- 1.2 Forschungsansatz . . . 6
- 1.3 Aufbau der Arbeit . . . 6

2 Saros **9**
- 2.1 Geschichte . . . 9
- 2.2 Benutzeroberfläche . . . 10
  - 2.2.1 Awareness . . . 10
  - 2.2.2 Roster-View . . . 13
  - 2.2.3 Shared Project Session-View . . . 14
  - 2.2.4 Dekorationen und Annotationen . . . 16
- 2.3 Technologien . . . 17
  - 2.3.1 Apache Commons . . . 17
  - 2.3.2 PicoContainer . . . 18
  - 2.3.3 XMPP . . . 20

3 *Teles*-Sitzungen **23**
- 3.1 Das Unternehmen . . . 23
- 3.2 Ziele . . . 23
- 3.3 Ablauf einer *Teles*-Sitzung . . . 24
- 3.4 Einarbeitungsphase . . . 25
- 3.5 Sitzungsdaten . . . 26
  - 3.5.1 Videomitschnitt . . . 26
  - 3.5.2 Notizen . . . 27
- 3.6 Beobachtungen bei den Sitzungen . . . 29
  - 3.6.1 Installationsprobleme . . . 29
  - 3.6.2 Dokumente als Chat zweckentfremdet . . . 30
  - 3.6.3 Inkonsistenzen . . . 31
  - 3.6.4 Rollenwechsel über den *Shared Project Session*-View . . . 32
  - 3.6.5 Unerwünschte Umwandlung von Zeilenumbrüchen . . . 33
  - 3.6.6 Anzeige des Einladungsfortschritts . . . 33
  - 3.6.7 Awareness . . . 37
  - 3.6.8 Verbindungsprobleme . . . 37
  - 3.6.9 Umgang mit *Eclipse* . . . 38
  - 3.6.10 Sonstiges . . . 40

3.7 Das Ende der Sitzungen 41

4 Entwicklungsprozess **43**
4.1 Das Saros-Team 43
4.2 Ablauf eines Release-Zyklus 45
4.3 Rollen 46
4.4 Die Release-Woche 48
4.5 Tests 49
4.5.1 Testdurchführung 51
4.5.2 Kritik am Testverfahren 52
4.5.3 Testlink 52
4.5.4 Testdurchführung II 54
4.6 Review-Prozess 54
4.6.1 Überwachung des Review-Prozesses 56
4.6.2 Probleme und Kritik am Review-Prozess 57

5 Implementierung **59**
5.1 Umwandlung der Kodierung von Zeilenumbrüchen 61
5.2 Jabber-Präsenzinformation 71

6 Outreach **75**

7 Nachgedanken **77**
7.1 Rückblick 77
7.2 Ausblick 77

Literaturverzeichnis **79**

A Vorlage für Notizen zu Teles-Sitzungen **83**

B E-Mail Konversation zu `Ctrl+Esc` **85**

1 Einleitung

1.1 Ziele

Ursprünglich war als Schwerpunkt der Arbeit die technische Betreuung der verteilten Programmiersitzungen mit *Saros* bei der Firma *Teles* geplant; als Ergänzung zu der Einführung und Betreuung dieses Prozesses durch Edna Rosen [Ros09]. Bei diesen, in dieser Arbeit als *Teles*-Sitzungen bezeichneten, Programmiersitzungen zwischen Software-Entwicklern in Österreich (Wien) und Bangalore in Indien, war mein Ziel durch Beobachten und Nachfragen, technische Probleme zu identifizieren und mich um die Beseitigung, Abmilderung, oder zumindest die Erfassung zu kümmern. Lösungen wollte ich entweder selbst erarbeiten oder Aufgaben – soweit möglich – an andere Mitwirkende am *Saros*-Projekt delegieren. Dieser Teil ist in Abschnitt 3 auf Seite 23 beschrieben.

Bei den Anfangs wöchentlich angesetzten *Teles*-Sitzungen gab es allerdings immer mehr verschobene Termine, bis die Sitzungen schliesslich gänzlich auf unbestimmte Zeit ausgesetzt wurden, da die Entwicklerteams in Österreich und Indien keine gemeinsamen Aufgaben mehr hatten. Schon in der Phase, in der Sitzungstermine bei *Teles* ausfielen, kam daher von meinen Betreuern Christopher Oezbek und Stephan Salinger der Vorschlag auch die OpenSource-Projekte, die Eike Starkmann im Rahmen seiner Arbeit ansprechen und von *Saros* überzeugen würde, technisch zu betreuen. Eine kurze Beschreibung zu Eike's Arbeit findet sich in [Sta09].

Christopher Oezbek, der bis dahin die technische Entwicklung von Saros massgeblich gestaltete und steuerte, verliess die Arbeitsgruppe. Somit entstand temporär eine personelle Lücke von mehreren Wochen, bis mit Karl Beecher ein neuer Projektleiter diese Aufgabe übernahm. Meine Betreuer fragten mich, ob ich für die Übergangszeit einen Teil von Christopher's Aufgaben übernehmen und dem neuen Projektleiter in der Einarbeitungsphase zur Seite stehen möchte. Zu der bisher rein technischen Seite der Arbeit kam damit also auch das Gebiet der Steuerung und Gestaltung von Software-Entwicklungsprozessen hinzu, was im Abschnitt über den *Saros*-Entwicklungsprozess – Abschnitt 4 auf Seite 43 – ausgearbeitet ist.

Zusätzlich zu den Tätigkeiten, die mit der Weiterentwicklung von *Saros* selbst und dem Thema der eigenen Arbeit zu tun haben, ist jedes Mitglied im *Saros*-Team gehalten, sich stetig um die Vergrösserung des Bekanntheitsgrades von *Saros* zu bemühen. Diese Thematik ist unter der Überschrift "Outreach" zu finden und ist in Abschnitt 6 auf Seite 75 näher beschrieben.

1.2 Forschungsansatz

Der Forschungsansatz der vorliegenden Arbeit hat primär den Charakter einer Feldstudie, bei der *Saros* im Einsatz bei Benutzern unter technischen Gesichtspunkten, und der Entwicklungsprozess von *Saros* im Entwicklerteam beobachtet wurde. Der Ansatz enthält weiterhin Komponenten der kanonischen Aktions- oder Handlungsforschung (*Canonical Action Research (CAR)*), da ich nicht nur Beobachter war, sondern auch aktiv das Forschungsobjekt beeinflusst habe. Dies bedeutet, ich habe bei den Teles-Sitzungen nicht nur Defekte oder Probleme erfasst, sondern auch behoben, und den *Saros*-Entwicklungsprozess aktiv mitgestaltet.

Bei der Handlungsforschung, welche auf Kurt Lewin im Bereich der Sozialpsychologie zurück geht [Lew48], wird der Forscher aktiver Teil des Forschungsobjekts, statt nur ein externer Beobachter zu sein, der bemüht ist, den Einfluss auf die beobachteten Grössen soweit als möglich zu minimieren, wie es in der empirischen Forschung üblich ist. Ziel ist es auch nicht unbedingt generalisierbare Ergebnisse zu erlangen, sondern eine konkrete Situation praktisch zu verbessern. Bei *Saros* wäre das zum Beispiel das relativ aufwändige Testverfahren zu vereinfachen. Die entsprechenden Bemühungen sind in Abschnitt 4.5 auf Seite 49 dokumentiert.

Da die Ergebnisse aus der Handlungsforschung nicht verallgemeinerbar sein müssen und in den Sozialwissenschaften die Einflussnahme auf das Forschungsobjekt moralisch oder politisch motiviert sein kann, und damit der Unterschied zwischen Wissenschaft und Ideologie schwindet, wird diese Forschungsmethode gelegentlich als unwissenschaftlich kritisiert [SHE99]. Die Gefahr der Ideologie ist auch in dem techniklastigen Gebiet der Software-Entwicklung nicht auszuschliessen. Jemand mit Einfluss auf den Entwicklungsprozess könnte eine eigene Agenda verfolgen, um zum Beispiel eine bestimmte Technik, Verfahrensweise, oder ein Produkt im Projekt zu etablieren, auch wenn das nach objektiven Massstäben nicht die vernünftigste Wahl ist. Das muss noch nicht einmal absichtlich oder bewusst geschehen. Viele Entwickler haben teilweise starke, individuelle Zu- oder Abneigungen gegenüber Betriebssystemen, Programmiersprachen, Texteditoren, IDEs, Software-Entwicklungsmodellen, oder auch gegen Firmen und damit deren Produkten, die ihre Handlungen und Entscheidungen mitbeeinflussen.

1.3 Aufbau der Arbeit

Im Kapitel 2 auf Seite 9 wird *Saros* beschrieben. Die Geschichte der Entwicklung, die Benutzeroberfläche, und verwendete Technologien zur Implementierung des Plugins.

Nach diesen Grundlagen folgen in Kapitel 3 auf Seite 23 Informationen über die Kooperation mit der Firma *Teles* und die in diesem Rahmen durchgeführten *Saros*-Sitzungen zwischen zwei Standorten des Unternehmens. Es werden dort meine Ziele formuliert

und der technische Ablauf einer Sitzung wird beschrieben – inklusive der Daten die dabei anfallen. Ferner werden Beobachtungen, die ich machte dokumentiert. Aus einigen davon liessen sich Aufgaben für mich oder andere *Saros*-Team-Mitglieder ableiten.

Das Kapitel 4 auf Seite 43 handelt vom *Saros*-Entwicklungsprozess und meiner Rolle als technischer Projektmanager. Das *Saros*-Team wird vorgestellt. Der Release-Zyklus und der Ablauf eines einzelnen Releases werden erläutert. Vertiefend wird auf die beiden Qualitätssicherungsmassnahmen der manuellen Tests und des Review-Prozesses eingegangen.

In beiden eben beschriebenen Kapiteln sind Implementierungsaufgaben für mich angefallen, die in Kapitel 5 auf Seite 59 kurz erörtert werden. Zwei der Aufgaben werden detaillierter beschrieben.

Im letzten Kapitel – Kapitel 7 auf Seite 77 – wird ein Rückblick auf die Arbeit und ein Ausblick auf die mögliche Zukunft gegeben.

2 Saros

Saros[1] ist ein Werkzeug zur kollaborativen, verteilten Software-Entwicklung in Echtzeit, das als Plugin für *Eclipse*[2] implementiert wurde. Das Programm wurde und wird über mehrere Studien-, Bachelor-, und Diplomarbeiten hinweg entwickelt.

Bei einer *Saros*-Sitzung bearbeiten mehrere Teilnehmer gemeinsam die Artefakte eines Projekts in *Eclipse*. Dabei gibt es zwei verschiedene Kategorien von Rollen mit jeweils zwei Möglichkeiten, die jeder Teilnehmer zu einem gegebenen Zeitpunkt besitzen kann. Beim Start der Sitzung wird einmal für die gesamte Dauer der Sitzung entschieden, wer Host und wer Client ist, und während der Sitzung kann der Host jederzeit für alle Teilnehmer – inklusive sich selbst – entscheiden, wer Driver und wer Observer ist.

Drivern ist es während der Sitzung erlaubt Artefakte zu verändern, wohingegen Observer nur beobachten und kommunizieren dürfen. Als flüchtige Veränderung in der Anzeige der Artefakte können sie Text in ge“share”ten Dokumenten selektieren und diese Markierung wird bei allen Teilnehmern im lokalen Editor hervorgehoben.

Die Kommunikation zwischen den Teilnehmern einer *Saros*-Sitzung wird über das *Jabber*- beziehungsweise *XMPP*-Protokoll abgewickelt.

2.1 Geschichte

Die erste Version von *Saros*, welche Riad Djemili im Rahmen einer Diplomarbeit entwarf und implementierte [Dje06], ermöglichte die verteilte Paarprogrammierung. Also eine Abwandlung der klassischen Paarprogrammierung, die unter anderem ein wichtiger Bestandteil beim *eXtreme Programming* [BA04] ist. Bei der Paarprogrammierung sitzen für die Implementierung zwei Entwickler physisch vor dem selben Rechner. Dabei programmiert der eine Entwickler und erklärt fortwährend was er gerade tut, während der Zweite ihm dabei zusieht und -hört, Fragen stellt, und auf Fehler oder unklare Stellen im Quelltext hinweist. Die beiden Rollen werden *Driver* und *Observer* genannt, weshalb die Begriffe für die Rollen in *Saros* übernommen wurden.

Dieses Vorgehen soll die Qualität des Quelltextes steigern, was einige Studien untermauern [Nos98, Wil00]. Eine Erklärung dafür ist, dass auf der einen Seite der Driver

[1] Saros: `http://dpp.sourceforge.net/`
[2] Eclipse: `http://www.eclipse.org/`

sein Vorgehen erklärt, also gezwungen ist ständig selber zu reflektieren, was er gerade tut, und auf der anderen Seite schon während des Implementierungsvorgangs der Observer eine Code-Durchsicht vornimmt. Das Eine hilft Fehler zu vermeiden, das Andere sie frühzeitig zu entdecken.

In einer anschliessenden Studienarbeit [Gus07], hob Björn Gustavs die Einschränkung auf, dass nur zwei Teilnehmer an einer Programmiersitzung teilnehmen können, und ermöglichte die Weitergabe der Driver-Rolle während einer laufenden Sitzung.

Um mehreren Teilnehmern gleichzeitig die Driver-Rolle geben zu können, integrierte Oliver Rieger in seiner Diplomarbeit [Rie08] mit dem Jupiter-Algorithmus [NCDL95] eine Nebenläufigkeitskontrolle, die schon im kollaborativen Texteditor *ACE*[3] Anwendung fand, und Ausgangspunkt für die Online-Kommunikations- und Kollaborationsplattform *Google Wave*[4] ist [WM].

Die Einbindung der Nebenläufigkeitskontrolle und die zusätzliche Konsistenzüberwachung, sowie die direkte Dateiübertragung wurden in der Diplomarbeit von Christoph Jacob [Jac09] getestet und entscheidend verbessert.

Meine Studienarbeit [Rin09] vor dieser Diplomarbeit befasste sich mit der Erweiterung der Präsenzinformation an die Erfordernisse bei mehr als zwei Teilnehmern, zum Beispiel verschiedene Benutzerfarben bei Annotationen, die noch nicht unterschiedlich eingefärbt wurden. Desweiteren verbesserte ich die Sequenzialisierung von nebenläufig verschickten Aktivitäten und die Serialisierung selbiger als XML.

Weitere Arbeiten folgten und werden bei der Vorstellung der aktuellen Mitglieder des *Saros*-Teams in Abschnitt 4.1 auf Seite 43 kurz angerissen.

2.2 Benutzeroberfläche

Die grafische Benutzeroberfläche von *Eclipse* mit *Saros*-Plugin bei einer laufenden Saros-Sitzung mit zwei Teilnehmern ist in Abbildung 2.1 auf der nächsten Seite zu sehen. Neben den beiden Views, die *Saros* bereitstellt, und die im unteren Teil des Bildschirmschnappschusses zu sehen sind, bietet das Plugin Dekorationen und Annotationen in bereits vorhandenen Views zur Förderung der *Awareness*. In den folgenden Abschnitten wird dieser Begriff erklärt, und dann die einzelnen Bereiche in der Benutzeroberfläche kurz vorgestellt, in denen *Saros* Informationen darstellt.

2.2.1 Awareness

Als *Awareness*, zu Deutsch Bewusstheit oder Gewärtigkeit, wird im Forschungsbereich der rechnergestützten Zusammenarbeit das Wissen und das Verständnis um die Tä-

[3] ACE: `http://sourceforge.net/projects/ace/`
[4] Google Wave: `http://wave.google.com/`

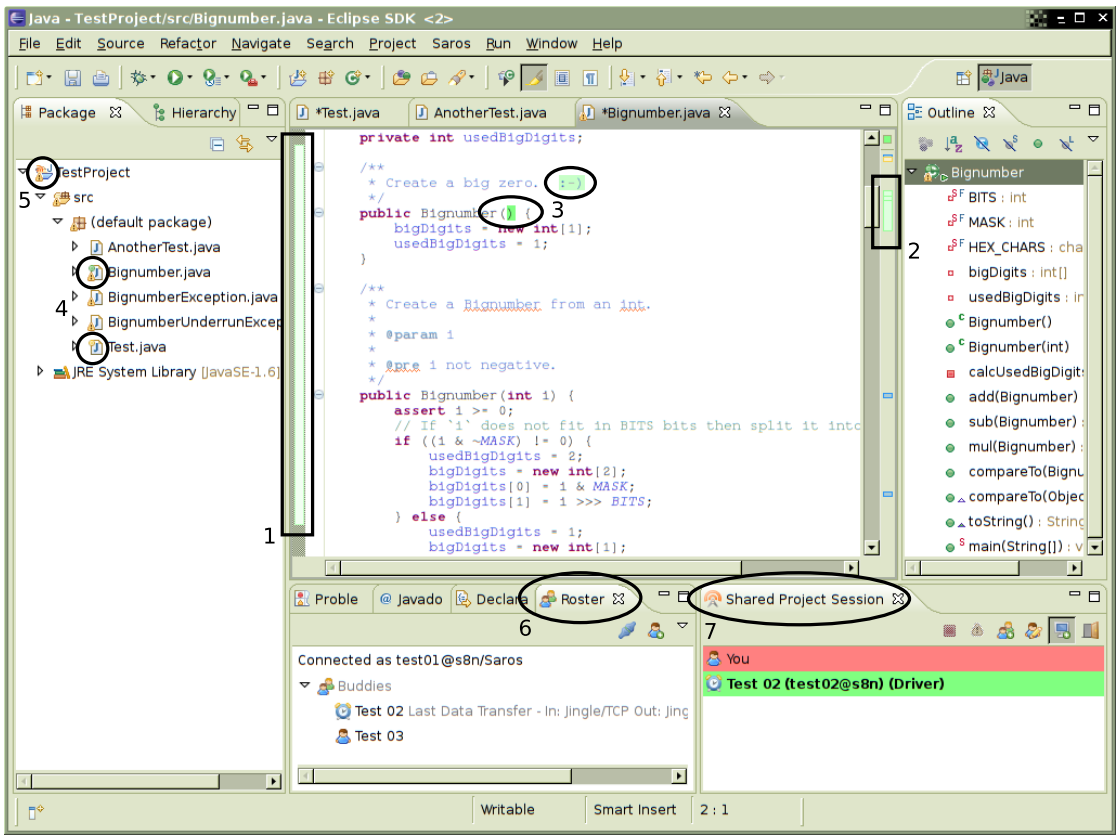


Abbildung 2.1: *Eclipse* (Version 3.5, unter Linux) mit laufender *Saros*-Sitzung (Saros 9.12.4). **(1)** und **(2)** Sichtbarkeitsbereich von Benutzer *Test 02* absolut und relativ zur Dateilänge in Zeilen. **(3)** Von Benutzer *Test 02* kürzlich hinzugefügter Quelltext und die Position seiner Schreibmarke. **(4)** Beim Benutzer *Test 02* offene Dateien und die aktuell sichtbare. **(5)** Dekoration des ge"share"ten Projekts (oben links im Icon) **(6)** *Roster*-View zeigt bekannte und verfügbare Kontakte an. **(7)** *Shared Project Session*-View zeigt die Teilnehmer einer *Saros*-Sitzung.

tigkeiten der anderen Mitarbeiter verstanden. Diese Awareness wird benötigt, um den eigenen Teil der Aufgabe zu erfüllen [DB92]. Bei Werkzeugen für die verteilte Zusammenarbeit muss man also versuchen, möglichst viele relevante Informationen, die man bei direkter Zusammenarbeit hätte, die aber durch die räumliche Distanz verloren gehen, über das Werkzeug zu vermitteln.

Awareness kann in folgende vier Kategorien eingeteilt werden [GGR96]:

Informelle Awareness Die Wahrnehmung wer in der Nähe oder verfügbar ist und was er oder sie gerade tut. Das erleichtert die informelle, beiläufige Zusammenarbeit.

Soziale Awareness Die Wahrnehmung ob jemand bei der Zusammenarbeit aufmerksam oder abgelenkt ist, in welchem Gemütszustand er sich befindet, oder wie interessiert er dabei ist. Neben der Sprache wird diese Information auch durch Augenkontakt, Gesichtsausdruck, und Körpersprache vermittelt.

Gruppenstruktur Awareness Die Wahrnehmung wer in der Gruppe welche Rolle, Verantwortlichkeit, und Position einnimmt.

Arbeitsbereich Awareness Die Wahrnehmung über die Verwendung des gemeinsam genutzten Arbeitsbereichs. Wer interagiert mit welchen Artefakten und was genau stellt er gerade oder in naher Zukunft mit ihnen an.

Die Unterstützung dieser Arten von Awareness ist in *Saros* unterschiedlich stark ausgeprägt. Die informelle Awareness wird durch den *Roster*-View geleistet, wo man sehen kann, wer potentiell für eine Zusammenarbeit in Frage kommt. Eine rudimentäre Chat-Funktionalität ist in *Saros* vorhanden, war aber zeitweise deaktiviert. Der Chat wird durch die Arbeit von Olaf Loga [Log10] wiederbelebt und weiterentwickelt, so dass man nicht nur sieht wer verfügbar ist, sondern die Mitarbeiter auch informell ansprechen, Fragen stellen, oder zu einer engeren Mitarbeit mittels *Saros* einladen kann. Was im Gegensatz zu einem gemeinsamen Büro fehlt, ist eine Möglichkeit einzuschätzen zu können, wie beschäftigt jemand ist, also wie sehr auch nur eine kleine Zwischenfrage von einer konzentrierten Arbeit ablenken würde.

So gut wie gar nicht unterstützt wird die soziale Awareness, zumindest nicht von *Saros* selbst. Da es bei der Paarprogrammierung wichtig ist, dass der Driver möglichst durchgehend mitteilt, was er sich bei dem was er gerade tut denkt, empfiehlt das *Saros*-Team einen Audiokanal über eine externe Software bereitzustellen. Das kann die freie Software *Mumble*[5] sein, aber auch eine Alternative wie *TeamSpeak*[6] oder *Skype*[7]. Über so einen Audiokanal kann man schon mehr Informationen über den Gemütszustand und das Interesse eines Mitarbeiters erfahren, als über den Quelltext, den er bearbeitet – wenn man einmal von ganz offensichtlichen Fällen bei der Wahl von Bezeichnern oder in Kommentaren im Quelltext absieht. Bei einem Audiokanal werden neben dem Gesprächsinhalt noch Emotionen über die Intonation, Lautstärke, oder Sprechgeschwin-

[5] Mumble: `http://mumble.sourceforge.net`
[6] TeamSpeak: `http://www.teamspeak.com/`
[7] Skype: `http://www.skype.com/`

digkeit vermittelt, die bei einem textbasierten Chat fehlen. Auch hier soll *Saros* von Olaf erweitert werden, indem eine Sprachübertragungsmöglichkeit (*Voice over IP* (*VoIP*)) direkt in das Plugin integriert wird.

Die Awareness der Gruppenstruktur wird zum Einen durch den besonderen Status des Hosts in einer *Saros*-Sitzung unterstützt, und zum Anderen über die beiden unterschiedlichen Rollen Driver und Observer. Nur der Host kann diese Rollen vergeben und entziehen, und er hat ausserdem die Möglichkeit die Sitzung für alle Teilnehmer zu beenden. Kenntnisse über diese Rechte und Rollen können die Sitzungsteilnehmer über den *Shared Project Session*-View erhalten.

Welche Artefakte gerade von anderen Mitarbeitern in *Eclipse* geöffnet sind, und wo sich die Leute "befinden", also die Position der Schreibmarke, der gerade sichtbare Bereich in einem Dokument, aber auch welche Textstellen in der näheren Vergangenheit geändert wurden, fällt in die Kategorie Arbeitsbereich Awareness. Solche Informationen werden als Dekorationen beziehungsweise Annotationen in Views angezeigt, die bereits von den IDE-Plugins zur Verfügung gestellt werden. Die Anzeige der Driver-Rolle im *Shared Project Session* könnte man auch noch zur Arbeitsbereich Awareness zählen, da diese Rolle die Absicht nahelegt, Dokumente nicht nur zu betrachten, sondern auch zu manipulieren.

2.2.2 Roster-View

Im *Roster*-View[8], Abbildung 2.1 auf Seite 11 unten mittig im Bild, werden die potentiell verfügbaren Teilnehmer für eine *Saros*-Sitzung angezeigt. Genau genommen kann die angezeigte Gruppe noch grösser sein, denn hier werden nicht nur Benutzer vom *Saros*-Plugin angezeigt, sondern alle Kontakte, die über das verwendete Jabber-Benutzerkonto bekannt sind. Die Anzeige entspricht in dieser Hinsicht der Liste von Kontakten in einem *Instant Messaging- (IM-)* oder Chat-Client. Es kann also durchaus vorkommen, dass in dieser Liste Kontakte aufgeführt und auch als "online" gekennzeichnet werden, die man trotzdem nicht zu einer *Saros*-Sitzung einladen kann, weil sie einen Chat-Client wie zum Beispiel *Pidgin*[9] verwenden, der die Erweiterungen des XMPP-Protokolls von *Saros* nicht versteht.

Für die einzelnen Kontakte wird neben dem Namen und ob sie beim Server angemeldet sind (online/offline), die "Anwesenheit" angezeigt, die ihr Client, entweder *Saros* oder ein anderer XMPP-Client, an den Server übermittelt. Das "Uhr"-Icon steht für Abwesenheit. Die Implementierung dieser Anzeige in *Saros* ist in Abschnitt 5.2 auf Seite 71 beschrieben.

Wenn Olaf Loga im Rahmen seiner Arbeit einen Chat in *Saros* integriert hat, erfüllt der *Roster*-View über die gleiche Aufgabe wie die Kontaktliste in Jabber-IM-Clients und

[8] Das englische "roster" kann man mit *Anschriftenliste* oder *Mitgliedsverzeichnis* ins Deutsche übersetzen.
[9] `http://www.pidgin.im/`

man kann Kontakte, die nicht mit *Saros* "online sind", auch von dort aus ansprechen, beispielsweise um sie zu fragen, ob sie *Eclipse* mit dem Plugin starten wollen, damit man eine *Saros*-Sitzung mit ihnen durchführen kann.

Die beiden Icons oben rechts im *Roster*-View erlauben das Herstellen oder Trennen der Verbindung zum Jabber-Server, dessen Zugangsdaten in den Einstellungen des Plugins hinterlegt wurden, und das Hinzufügen eines Kontakts zur Liste. Ganz rechts befindet sich noch ein Menü mit einem einzelnen Eintrag, mit dem man einen im Roster ausgewählten Kontakt zu einer bestehenden *Saros*-Sitzung einladen kann. Dieser Eintrag wurde in dem Menü "versteckt", weil das Icon zu viel Ähnlichkeit mit dem zum Hinzufügen von neuen Kontakten zum Roster hat, und es zu Verwechslungen kam.

2.2.3 Shared Project Session-View

Unten rechts im Bild in Abbildung 2.1 auf Seite 11 befindet sich der *Shared Project Session*-View. Im Sprachgebrauch des *Saros*-Teams wird er auch oft verkürzt als *Session*-View bezeichnet, wobei dann aus dem Kontext klar werden muss, ob der View in der Benutzeroberfläche, oder die ihn implementierende Klasse *SessionView* gemeint ist.

Während einer Sitzung werden hier die Teilnehmer, mit den ihnen zugewiesenen Benutzerfarben hinterlegt, angezeigt. Es gibt für maximal fünf Benutzer unterschiedliche Farben, alle weiteren Benutzer werden in der gleichen Farbe wie der Erste angezeigt. Auf die Farben wird am Ende des nächsten Abschnitts noch einmal eingegangen.

Pro Benutzer wird ein Icon am Anfang der farblich hinterlegten Zeilen angezeigt, welches die Rolle wiederspiegelt. Bei Observern ist das ein Piktogramm einer Person und bei Drivern wird bei der Person zusätzlich ein grosser Stift dargestellt. Im Schnappschuss vom Bildschirm ist dieses Icon allerdings gerade durch das Uhr-Piktogramm überdeckt, welches die "Abwesenheit" vom Benutzer mit der Driver-Rolle symbolisiert. Dass man ihn trotzdem als Driver erkennen kann, ist durch den Zusatz "(Driver)" im Textteil der Zeile sichergestellt. Das Driver-Icon entspricht vom Aussehen dem auf der Schaltfläche zum Entziehen aller Driver-Rollen, welches in einem der folgenden Absätze beschrieben, und auch noch einmal gezeigt wird.

Die Icons oben im *Shared Project Session*-View haben, von links nach rechts, folgende Funktion:

Das "Stopp"-Icon ist ein reiner Indikator für die Benutzer. Es "leuchtet auf", wenn die Klasse *StopManager* das Projekt gesperrt hat. "Aufleuchten" meint hier, dass es aus dem normalerweise inaktiven, ausgegrauten Zustand in den aktiven Zustand wechselt. Die der Anzeige zugrundeliegende Sperrung verhindert, dass Teilnehmer Aktivitäten generieren können. Das wird bei Aktionen verwendet, bei denen sich mehrere *Saros*-Exemplare in einem definierten Zustand befinden müssen, um Nebenläufigkeitsprobleme wie Wettlaufsituationen zu vermeiden. Der *StopManager* wird zum Beispiel beim Abschliessen des Einladungsprozesses,

für Rollenwechsel, oder bei der Konsistenzwiederherstellung aktiviert. Da es in der Vergangenheit zu Verklemmungen durch unsachgemässe Verwendung des *StopManager*s kam, wurde das Icon als Indikator von mir implementiert, damit man auch ohne einen Blick in die Logdateien zu werfen, erkennen kann wenn *Saros* "hängt", weil die Sperrung durch den *StopManager* nicht wieder aufgehoben wird. Insbesondere kann man Testpersonen oder externe Benutzer leicht nach dem Zustand dieses Indikators befragen. Alternativ müsste man ihnen erklären wo sie auf ihrer Plattform die *Saros*-Logdateien finden, und nach welchen Einträgen sie suchen müssen.

Das Warndreieck "leuchtet auf", wenn die Konsistenzüberwachung eine Inkonsistenz entdeckt hat. Das ist dann der Fall, wenn die vom Host verschickten Prüfsummen für Dateien, von denen bei einem Client abweichen. Ein Klick auf das aktivierte Warndreieck startet eine Konsistenzwiederherstellung, das heisst, die Datei wird vom Host zum Client übertragen, bei dem diese Schaltfläche aktiviert wurde. Da die Dateiinhalte beim Host die Referenz darstellen, kann folglich das Icon beim Host selbst niemals aktiv werden.

Über dieses Icon können weitere Kontakte aus dem *Roster*-View zur laufenden *Saros*-Sitzung eingeladen werden. Diese Möglichkeit steht nur dem Host offen.

Auch dieses Icon wird ein Client nur im deaktivierten, ausgegrauten Zustand sehen. Es ermöglicht dem Host allen Drivern diese Rolle auf einen Schlag zu entziehen.[10]

Bei diesem Icon handelt es sich um einen "toggle button", das heisst die Schaltfläche kann "einrasten", wie in Abbildung 2.1 auf Seite 11 zu sehen ist. Ist das der Fall, befindet sich der Benutzer im Verfolgermodus. Sein lokales *Eclipse* vollzieht mittels *Saros* in diesem Modus nicht nur Änderungen von entfernten Drivern an Dokumenten nach, sondern auch das Öffnen, Schliessen, und Wechseln von Editoren eines ausgewählten verfolgten Drivers. Auch der Sichtbarkeitsbereich im aktiven Editor dieses Drivers wird lokal nachvollzogen. Im Verfolgermodus kann man den verfolgten Driver, denn es kann ja mehrere Driver geben, daran erkennen, dass sein Eintrag in der Liste unter den Icons in fetter Schrift gesetzt ist.

Die offene Tür symbolisiert den Ausgang aus einer *Saros*-Sitzung. Hier muss wieder Host und Client unterschieden werden. Während ein Client über diese Schaltfläche nur selbst die Sitzung verlässt, beendet der Host durch das Verlassen der Sitzung, selbige für alle Teilnehmer.

[10] Das wird meiner Meinung nach aus dem Piktogramm selbst ("kleine Person mit grossem Stift") nicht annähernd deutlich. Es fehlt ein "negatives" optisches Signal, zum Beispiel ein kleines rotes "X" oder das gesamte Icon in rot durchgestrichen.

2.2.4 Dekorationen und Annotationen

Dekorationen sind im Sprachgebrauch von *Eclipse* Auszeichnungen, die vorhandene Icons überlappen und damit zusätzliche Informationen vermitteln.[11] Diese Technik wird zum Beispiel von Plugins zur Unterstützung von Programmiersprachen verwendet, um Projektressourcen mit einem kleinen roten "X" zu markieren, wenn das dazugehörige Quelltextfragment syntaktische Fehler aufweist, oder Projekte mit einer Kennzeichnung des Projekttyps zu versehen.

In Abbildung 2.1 auf Seite 11 sieht man im *Package Explorer*-View oben links bei dem Icon für das Projekt *TestProject* zum Beispiel drei Dekorationen über der halboffenen Dokumentenmappe:

- oben links einen Sendeturm mit orangefarbenen "Funkwellen" als Kennzeichen eines ge"share"ten Projekts,
- oben rechts ein "J" als Kennzeichen für ein Java-Projekt,
- und unten links ein kleines gelbes Warndreieck, als Zeichen das innerhalb des Projektes Ressourcen bestehen, für die es Warnungen gibt.

Die letzten beiden Dekorationen stammen von den Plugins, welche die *Java Development Tools (JDT)* ausmachen. Für den kleinen Sendeturm ist *Saros* verantwortlich.

Innerhalb des Projektbaums im *Project Explorer*-View werden von *Saros* die Ressourcen durch einen farbigen Punkt dekoriert. Für die sowohl lokal, als auch bei entfernten Teilnehmern ein Editor geöffnet ist gelb, und solche, wo der geöffnete Editor zusätzlich gerade aktiv ist grün. Diese Dekorationen sind an die Ressourcen und nicht an bestimmte Views gebunden. Das macht sich dadurch bemerkbar, dass sie in allen Views sichtbar sind, die ein Icon für die jeweilige Ressource darstellen. Die Icons für Dateien erscheinen zum Beispiel genau so dekoriert im Dateibaum im *Navigator*-View (nicht im Bild zu sehen), aber auch rechts im Bild beim Icon für die Klasse im *Outline*-View.[12]

Als Annotationen werden bei *Eclipse* die Markierungen von Textstellen und -bereichen innerhalb des Editorfensters und an dessen seitlichen Rändern bezeichnet. Saros stellt über diesen Mechanismus in den Rändern den Sichtbarkeitsbereich (englisch: *viewport*) von entfernten Drivern dar, und im Textbereich die letzten Textänderungen von Drivern und die Position von entfernten Schreibmarken und Textselektionen.

Im linken Rand wird der absolute Sichtbarkeitsbereich eines entfernten Drivers, und im Rechten der Sichtbarkeitsbereich proportional zur Länge des Dokuments in Zeilen angezeigt. In Abbildung 2.1 auf Seite 11 hat der Driver *Test 02* also einen Sichtbarkeitsbereich der circa drei Zeilen kleiner ist, als der vom Host, dessen *Eclipse* auf dem Bild

[11] Die Namensähnlichkeit zum Dekorator-Entwurfsmuster [GHJV95] ist höchstwahrscheinlich nicht zufällig, denn es kommt bei der Implementierung zur Anwendung.

[12] Der kleine grüne Punkt setzt sich optisch leider nicht besonders stark vom dahinter liegenden, grossen grünen Klassen-Icon ab.

zu sehen ist. Oben sieht der Host eine Zeile mehr als der entfernte Benutzer und unten sind es zwei. Die relative Anzeige des entfernten Sichtbarkeitsbereich im linken Rand ist vor allem nützlich, wenn der Verfolgermodus nicht aktiviert ist. Dann kann man trotzdem wahrnehmen, welchen Teil des Dokuments ein Driver sieht, und auch in welche Richtung er die Anzeige bewegt, wenn sich sein Sichtbarkeitsbereich ändert.

Im Textbereich gibt es drei verschiedene Annotationen, die durch farbige Hinterlegung der betroffenen Zeichen dargestellt werden. Die Cursor-Annotation zeigt die Position der Schreibmarke von entfernten Benutzern an. Die Selection-Annotation den Bereich, von deren ausgewählten Texten. Beide Annotation werden mit der gleichen Farbe dargestellt, so ist also eine Cursor-Annotation und die Selektion eines einzelnen Zeichens nur über den "Tooltip" unterscheidbar, den der Benutzer angezeigt bekommt, wenn er mit dem Mauszeiger über der Annotation verweilt. Im Bild ist eine Cursor-Annotation zu sehen. Die dritte Art der Annotation markiert die letzten Änderungen eines entfernten Benutzers. Im Bildschirmschnappschuss ist der "Smiley" im Kommentar von Benutzer *Test 02* kürzlich hinzugefügt worden. Das Plugin verwendet für diese sogenannte Contribution-Annotation einen Verlauf der letzten eingegangenen Textänderungen der Sitzungsteilnehmer. Die Länge dieses Verlaufs umfasst momentan die letzten 20 *TextActivity*-Nachrichten pro Teilnehmer.

Die Farben für die Annotationen leiten sich von den Benutzerfarben ab, die auch im *Shared Project Session*-View verwendet werden. Es sind aufgehelltere Varianten davon, damit der farbig hinterlegte Text keinen zu geringen Kontrast bekommt und weiterhin gut lesbar bleibt. Bei den Annotationen in den Rändern machen die helleren Farbtöne Sinn, weil die meisten Annotationen im Rand, von den Plugins zur Java- oder C- und C++-Entwicklung, kräftige Farben verwenden. Beispiele für solche Annotationen sind Quelltextstellen mit syntaktischen Fehlern, Warnungen, oder speziellen Kommentaren. Insgesamt ist eine gute Farbwahl nicht so leicht, da die Farben sowohl eindeutig unterscheidbar für die Teilnehmer sein müssen, andererseits nicht mit Farben anderer Annotationen kollidieren dürfen, oder in Kombination mit den Farben vom Syntax-Highlighting in Quelltexteditoren schlecht lesbar sein dürfen.

2.3 Technologien

2.3.1 Apache Commons

Die *Apache Commons*[13] bestehen aus einer Sammlung von Bibliotheken aus dem Umfeld des *Apache*-Projekts, die wiederverwendbare Komponenten zur Verfügung stellt. Es werden Bibliotheken für verschiedene Themenbereiche angeboten. Bei *Saros* werden Folgende eingebunden:

[13] Apache Commons: `http://commons.apache.org/`

- *Codec* zum En- und Dekodieren – zum Beispiel von Binärdaten mittels *Base64* [FB96] oder Zeichenketten nach dem URL-Schema [BLMM94],
- *IO* zum Lesen und Schreiben auf Ein- und Ausgabeströmen und von Dateien,
- *Lang* enthält viele Hilfsfunktionen und Klassen für die Grundklassen im Paket *java.lang* aus der Java-Standardbibliothek,
- und *Logging* als Schicht zwischen der Anwendung und verschiedenen Logging-Implementierungen.

Da die *Apache Commons* auch für ältere Projekte noch verwendbar sein sollen, ist die Unterstützung für neuere Spracheigenschaften von Java, wie zum Beispiel *Generics*, nicht gegeben.

2.3.2 PicoContainer

PicoContainer[14] bietet einen Container, welcher Komponenten mittels *dependency injection* (*DI*) zusammensetzen kann [Fow04]. Einem solchen Container-Objekt werden Klassen und Beziehungen zwischen Exemplaren dieser Klassen bekannt gemacht. Der Container kann Anhand der Typen in den Konstruktoren oder Annotation an Feldern die Abhängigkeiten zwischen den Klassen erkennen. Bei Anforderung eines Exemplars einer der Klassen vom Container, werden diese Informationen verwendet, um alle nötigen Exemplare in der richtigen Reihenfolge zu erzeugen und zusammenzusetzen.

In *Saros* können auf diese Weise Abhängigkeiten zwischen vielen Singleton-Objekten aus dem Quelltext in einen *PicoContainer* verlagert werden. Der Container spart damit an vielen Stellen Quelltext zum Anfordern von Singletons, der sich auf konkrete, statische Klassen bezieht. Der Austausch so eines Singletons durch eine alternative Implementierung hätte also eine Anzahl von über den Quelltext verstreuten Änderungen nach sich gezogen. Zudem wäre der Austausch statisch gewesen. Mit *PicoContainer* braucht dem Container nur die alternative Klasse bekannt gegeben werden. Die Auswahl von alternativen Implementierungen kann auch an Vorbedingungen geknüpft zur Laufzeit geschehen.

Es ist auch möglich in Objekte, welche ausserhalb eines *PicoContainer* erstellt wurden, Abhängigkeiten aus einem Container einzubringen. Das ist beispielsweise sinnvoll, wenn die Exemplare nicht vom Container erzeugt werden können. Bei *Saros* betrifft dies Klassen, die zwar zum *Saros*-Code gehören, aber deren Exemplare von *Eclipse* erstellt werden. *View*s wären hier ein Beispiel.

Wenn *Saros* im Debug-Modus gestartet wird, erstellt die Klasse *DotGraphMonitor* beim Beenden von *Eclipse* eine Datei, die einen gerichteten Graphen beschreibt, mit den Klassen, von denen der Container Exemplare verbunden hat, als Knoten und den Injektionen als gerichtete Kanten. Die Datei liegt in einem Format vor, welches mit Hilfe von

[14] PicoContainer: `http://www.picocontainer.org/`

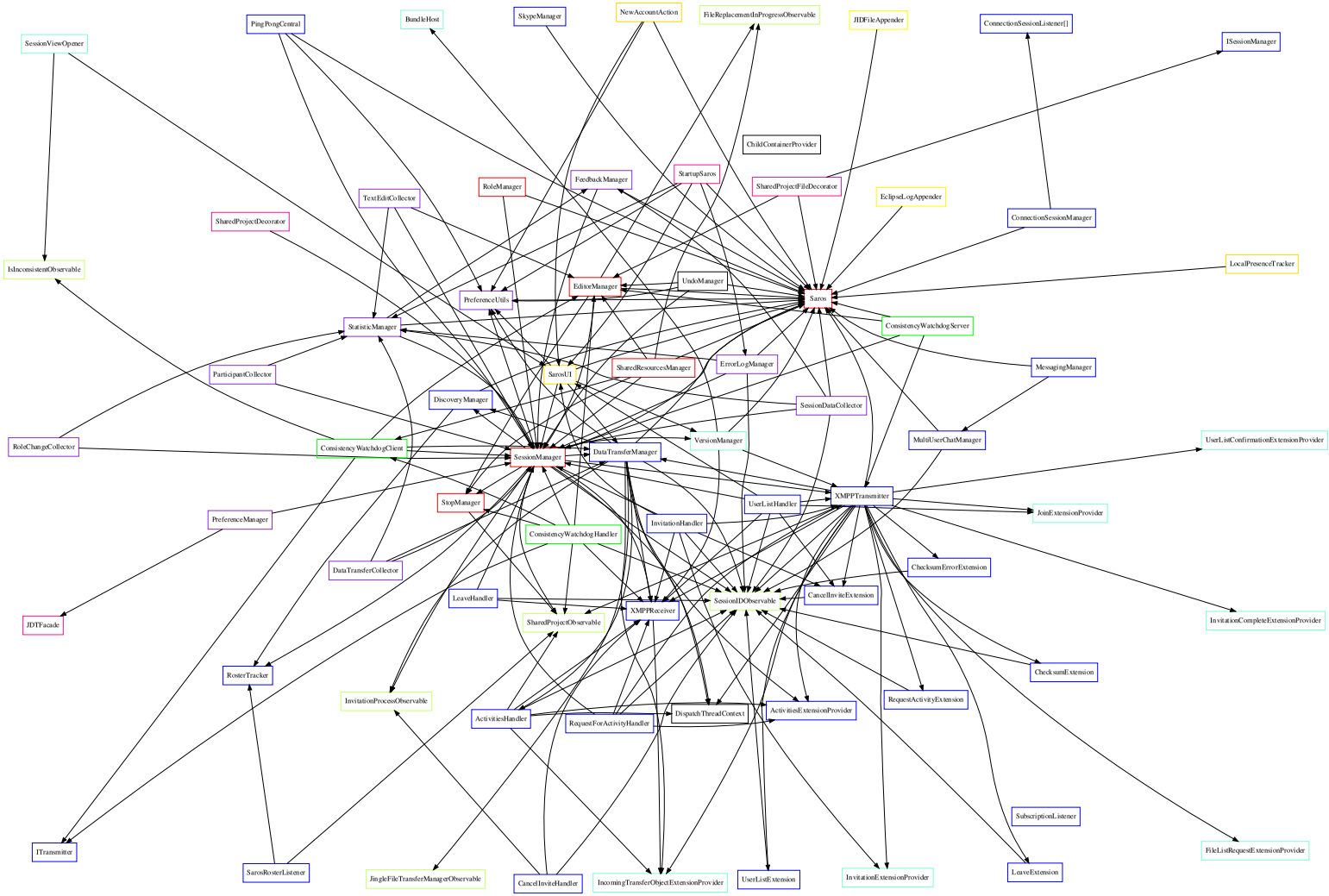


Abbildung 2.2: *Saros*-Klassen, die mit *PicoContainer* verwaltet und verbunden werden. Knoten entsprechen Klassen und eine gerichtete Kante $A \rightarrow B$ bedeutet, dass Klasse A eine Referenz auf Klasse B hält. Der farbige Rahmen kennzeichnet die Kategorie, in welche die Klasse gehört.

Graphviz[15] [GN99] als Grafik dargestellt werden kann. In Abbildung 2.2 auf der vorherigen Seite ist so eine Grafik zu sehen. Die Anzahl der Klassen und Beziehungen ist zu gross, um bei dem auf der Seite zur Verfügung stehenden Platz die Klassennamen lesen, oder einzelne Beziehungen gut verfolgen zu können. Man kann aber aus dieser "Vogelperspektive" ein paar Klassen ausmachen, die offensichtlich eine zentralere Rolle spielen, da sie viele Kanten besitzen. Das dieses Beziehungsgeflecht nicht mehr durch Quelltext in den *Saros*-Quellen erzeugt wird, sondern von *PicoContainer* dynamisch zur Laufzeit, hat den Nachteil, dass sich die Beziehungen nicht mehr durch statische Quelltextanalyse erfassen lassen. Und damit lassen sie sich auch nicht mehr so gut über die Suchfunktionen in *Eclipse* nachverfolgen.

2.3.3 XMPP

Das *Extensible Messaging and Presence Protocol* (*XMPP*) [SA04], welches auch unter dem älteren Namen *Jabber*-Protokoll bekannt ist, bildet die Grundlage der Kommunikation zwischen *Saros*-Exemplaren. Das XML-Format zur Beschreibung von Nachrichten wird von *Saros* erweitert, um Aktivitäten wie beispielsweise Textänderungen und das Öffnen und Schliessen von Editoren oder Awareness-Informationen zu übertragen. Die Nachrichten gehen dabei über einen oder auch mehrere Jabber-Server. Nicht jeder Nutzer muss mit dem selben Jabber-Server verbunden sein, solange die Server untereinander Nachrichten austauschen können.

Bei grösseren Datenmengen, zum Beispiel die Dateien des Projekts, die bei einer Einladung zu einer *Saros*-Sitzung vom Host zu Clients übertragen werden, wird versucht eine direkte Verbindung zwischen den beteiligten *Saros*-Benutzern herzustellen. Diese Peer-to-Peer-Verbindung ohne den Umweg über den oder die Jabber-Server wird mit der *Jingle*-Erweiterung [LBSA+09] des XMPP-Protokolls aufgebaut.

Sollte keine direkte Verbindung hergestellt werden können, fällt *Saros* auf die Übertragung mittels der *In-Band Bytestreams*-Erweiterung [KSA09], kurz *IBB*, zurück. Dabei werden die Daten in kleinere Pakete aufgeteilt und als XMPP-Nachrichten via Jabber-Server übermittelt.

In Abbildung 2.3 auf der nächsten Seite ist eine mögliche Verbindungskonstellation während einer *Saros*-Sitzung dargestellt. Die vier Teilnehmer sind nicht alle mit dem selben Jabber-Server verbunden. *Client C* verwendet einen anderen Jabber-Server als die anderen Sitzungsteilnehmer. Folglich gehen alle XMPP-Nachrichten zwischen *Client C* und den anderen Teilnehmern über zwei Jabber-Server. Die gestrichelte Linie zwischen dem *Host* und *Client C* soll eine mit *Jingle* initiierte Direktverbindung darstellen, über die gerade Daten vom *Host* an *Client C* übertragen werden.

[15] Graphviz – Graph Visualization Software: `http://www.graphviz.org/`

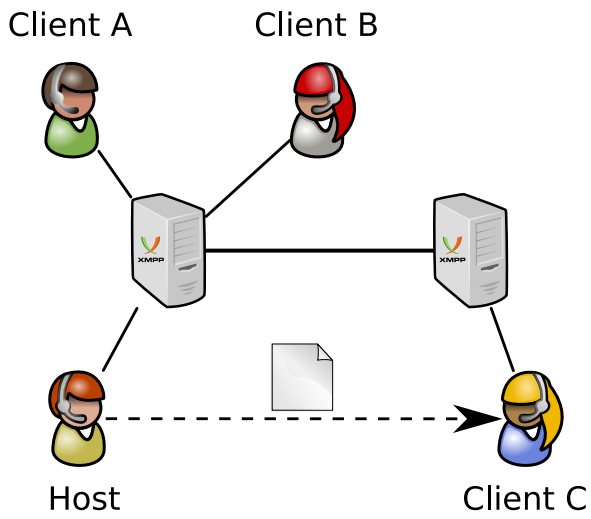


Abbildung 2.3: Beispielkonstellation von Netzverbindungen während einer Sitzung.

Die Netzwerkschicht von *Saros* verwendet die *Smack*-Bibliothek[16] zur Kommunikation via XMPP.

[16] Smack API: `http://www.igniterealtime.org/projects/smack/index.jsp`

3 *Teles*-Sitzungen

Im Folgenden werden meine Tätigkeiten und Beobachtungen bei den verteilten Paarprogrammierungssitzungen von Entwicklern der Firma *Teles* beschrieben. Nach einer kurzen Vorstellung des Unternehmens (Abschnitt 3.1), folgt eine Zusammenfassung der von mir verfolgten Ziele (Abschnitt 3.2), eine Beschreibung einer *Teles*-Sitzung (Abschnitt 3.3 auf der nächsten Seite) und der Einarbeitungsphase (Abschnitt 3.4 auf Seite 25). Vor den Beobachtungen (Abschnitt 3.6 auf Seite 29) werden die zu einer Sitzung erhobenen Daten beschrieben (Abschnitt 3.5 auf Seite 26).

3.1 Das Unternehmen

In der ersten Jahreshälfte 2009 sind sind die *AG Software Engineering* (kurz *AG SE*) des Informatikinstituts der Freien Universität Berlin und die *Teles AG Informationstechnologien* (kurz *Teles*) eine Kooperation eingegangen. *Teles*[1] ist ein mittelständisches, international aufgestelltes, und börsennotiertes deutsches Unternehmen mit Hauptsitz in Berlin, das 2008 einen Umsatz von 23,8 Mio. € machte und weltweit Standorte unterhält [SS09]. Das Unternehmen entwickelt und vertreibt Lösungen im Bereich Telekommunikation – unter anderem Software für IP-Telefonie.

Die *AG SE* stellte *Saros* und mit Edna Rosen eine Diplomandin zur Begleitung der Einführung der verteilten Paarprogrammierung mittels *Saros* zur Verfügung. Die Firma *Teles* steuerte Entwickler an zwei Standorten – Österreich und Indien – mit einer gemeinsamen Aufgabe zur Kooperation bei. In Aussicht gestellt wurden ausserdem noch Entwickler eines anderen Projekts an Standorten in Deutschland und China.

3.2 Ziele

Geplant war, dass ich die Einführung von *Saros* bei *Teles* technisch unterstütze. Das bedeutet, die wöchentlichen *Teles*-Sitzungen beobachten und technische Probleme dabei identifizieren. Als technische Probleme gelten dabei sowohl Defekte in *Saros*, als auch fehlende Funktionalität, die eine *Teles*-Sitzung effizienter oder effektiver machen kann. Ferner sollte ich den Entwicklern bei *Teles* für technische Fragen rund um *Saros* zur Verfügung stehen und sie bei *Saros*-Releases über die Neuerungen informieren, die sie im Besonderen betreffen.

[1] Teles AG Informationstechnologien: `http://www.teles.de/`

Die identifizierten Probleme waren im Bug- und Feature-Tracker bei *SourceForge* zu dokumentieren, nach Wichtigkeit für *Teles* zu priorisieren, und nach Möglichkeit zu beheben oder an andere Mitglieder des *Saros*-Teams zu delegieren, sofern dies in den Aufgabenbereich des jeweiligen Mitglieds und dessen Zeitplanung passte. Die Priorisierung sollte sowohl in Absprache mit der Projektleitung von *Saros*, als auch mit den Entwicklern bei *Teles* geschehen, um sicherzustellen, dass die Defekte, welche die Anwendung von *Saros* bei *Teles* am meisten behindern, und neue Features, welche die Arbeit dort am besten unterstützen würden, bevorzugt bearbeitet werden.

Um Aufgaben gegebenenfalls an andere Mitglieder im *Saros*-Team delegieren und den Fortschritt der Umsetzung verfolgen zu können, brauchte ich einen ständigen Überblick darüber, was die anderen gerade taten (siehe Abschnitt 4.1 auf Seite 43). Zusätzlich musste ich auch Rücksprache mit ihnen halten, ob eine Aufgabe auch ihrer Meinung nach in ihr Aufgabengebiet passte, und ob die Zeitplanung die Übernahme einer weiteren Aufgabe zuliess.

Desweiteren war geplant die technischen Probleme über die Zeit zu beobachten und zu bewerten. Fragen über die Zu- oder Abnahme der Häufigkeit, oder Änderungen der Art der auftretenden Defekte im Laufe der Sitzungen, sollten beantwortet werden.

Aus diesen Zielen ergaben sich für mich folgende Aufgaben pro Sitzung:

- E-Mail vor der Sitzung verschicken (“Neues von *Saros*”),
- Sitzung begleiten und aufzeichnen,
- Sitzungsaufzeichnung auswerten,
- Log-Dateien zusenden lassen und auswerten,
- aus den Daten Defekte und mögliche nützliche Features ableiten.

Zu der E-Mail vor der Sitzung kamen nach Abschluss von Edna Rosen’s Arbeit noch das Versenden, Einsammeln und Sichern der von ihr entworfenen Fragebögen an die Entwickler bei *Teles*, bis sich ein Nachfolger für ihre Arbeit gefunden hat, um diese Datenquelle weiterhin kontinuierlich aufzuzeichnen.

3.3 Ablauf einer *Teles*-Sitzung

Die Diplomarbeit von Edna Rosen konzentrierte sich auf die Etablierung des Softwareprozesses der verteilten Paarprogrammierung, welche in wöchentlich angesetzten Sitzungen durchgeführt wurde. Der Etablierungsprozess umfasste für Edna 16 Sitzungen. Der Inhalt einer Sitzung und die zeitliche Aufteilung davon wurden über diese Zeit von ihr angepasst, um den Prozess zu verbessern. Meine Aufgabe begann erst relativ spät im Etablierungsprozess, und zu dem Zeitpunkt war seit Sitzung #12 ein stabiler Ablauf erreicht, der für eine Sitzung in der Regel folgende Inhalte und Zeiten vorsah [Ros09]:

- Start um 10^{30} Uhr, cirka 90 Minuten verteilte Entwicklung
- Start der Reflektion gegen 11^{45} Uhr, cirka 15 Minuten Reflektion
- Ende um 12^{00} Uhr

Der Zeitrahmen wurde so gelegt, dass er für den Standort Wien (Österreich) unmittelbar vor der Mittagspause endete und für den Standort Bangalore (Indien) direkt nach der Mittagspause begann.

In den $1\frac{1}{2}$ Stunden verteilter Entwicklungszeit war die Verwendung von Saros vorgesehen. Die Inhalte konnten dabei verschiedener Natur sein. Es kamen sowohl die Paarprogrammierung, das Vorstellen von ausserhalb der Sitzung erstelltem Quelltext, als auch Sitzungen vor, die zu einem grossen Teil aus Fehlersuche und Tests bestanden.

In der Reflektion sollten die Entwickler über den Entwicklungsprozess reflektieren um sich darüber klar zu werden, was gut und was schlecht lief und wie man den Prozess verbessern könnte. Begleitet und moderiert wurde dieser Teil der Sitzung von Edna.

Der Begriff *Teles*-Sitzung umfasst all die oben aufgeführten Punkte und soll als Abgrenzung gegen die anderen Arten von Sitzungen in dieser Arbeit dienen. Also zum Beispiel eine *Saros*-Sitzung, die ein Bestandteil von *Teles*-Sitzungen war, oder eine verteilte Paarprogrammierungssitzung, die wiederum Inhalt einer *Saros*-Sitzung sein kann, aber nicht muss, da in einer *Saros*-Sitzung wie weiter oben erwähnt auch andere Tätigkeiten als Programmieren in Frage kamen.

Die *Teles*-Sitzungen wurden als Film mitgeschnitten. Das Bild zeigt den Desktop des Entwicklers aus Wien und der Ton ist ein Mitschnitt der Audio-Kommunikation aller Sitzungsteilnehmer.

Teilgenommen haben an diesen Sitzungen je ein Entwickler in Wien und einer in Bangalore. In Bangalore waren ursprünglich zwei Entwickler vorgesehen – während der Zeit, in der ich mit diesen Sitzungen zu tun hatte, nahm allerdings immer nur ein und derselbe Entwickler aus Indien teil. Im Folgenden wird der Entwickler aus Österreich Entwickler #1 und der aus Indien Entwickler #2 genannt, wobei das die gleiche Zuordnung von Bezeichnung zu realer Person ist, die auch Edna in ihrer Arbeit verwendet hat. Desweiteren haben Edna als Begleiterin des Paarprogrammierungsprozesses oder ich als Ansprechpartner für technische Fragen zu *Saros* von Berlin aus teilgenommen. In den ersten drei der insgesamt fünf Sitzungen an denen ich teilnahm waren Edna *und* ich beteiligt.

3.4 Einarbeitungsphase

Um eine Idee von den Inhalten und dem Ablauf einer *Teles*-Sitzung zu bekommen, schaute ich mir als erstes die Aufzeichnung der *Teles*-Sitzung #13 von 2009-07-15 an.

An den drei folgenden Sitzungen nahm ich zusammen mit Edna teil. Erst nur als stiller Beobachter in Sitzung #14, dann mit Vorstellung meiner Person bei den anderen Teilnehmer in Sitzung #15 für diese und die folgenden Sitzungen als aktiver Ansprechpartner.

In den ersten beiden Sitzungen hatte ich noch mit technischen Problemen zu kämpfen. Anfangs benutzte ich einen eigenen Rechner für meine Audioverbindung zur Sitzung, der per WLAN mit dem VPN des Fachbereichs verbunden war. Diese VPN-Verbindung brach bei beiden Sitzungen mindestens einmal ab. Ausserdem hatte ich nicht die von Edna erarbeiteten Empfehlungen zu den Einstellungen der verwendeten VoIP-Software *Mumble* bis ins Letzte befolgt – insbesondere hatte ich keine durchgängige Audio-Übertragung gewählt, sondern eine Sprachaktivierte um Bandbreite über die Funkverbindung zu sparen. Das hat zu Aussetzern bei ausgehenden Sprachdaten geführt, so dass mich die anderen Teilnehmer nicht immer verstehen konnten. Da Edna und ich nebeneinander im gleichen Büro sassen, und sie sowohl direkt hören konnte was ich sagte, als auch was über die Audioverbindung bei allen ankam, war sie so freundlich und geistesgegenwärtig in solchen Fällen meine Antworten auf Fragen von Entwickler #2 zu wiederholen.

3.5 Sitzungsdaten

An Daten sind pro Sitzung mindestens ein Videomittschnitt, Notizen zur Sitzung und Nachbereitung, die *Saros*-Logdateien der beteiligten Entwickler, und von den Entwicklern ausgefüllte Fragebögen von und für Edna Rosen beziehungsweise ihren Nachfolger angefallen und auf einem Dateiserver der Arbeitsgruppe abgelegt worden. Der Videomittschnitt enthält die Sitzung selbst, sowie eine Reflektion über die Sitzung, sofern eine Reflektion stattfand. Die beiden Teile existieren für manche Sitzungen auch in getrennten Videodateien.

3.5.1 Videomitschnitt

Die Infrastruktur für die Videoaufzeichnung wurde in Zuge von Edna's Arbeit entworfen und aufgebaut [Ros09]. Ausführlicher ist eine von ihr verfasste Beschreibung und detaillierte, bebilderte Anleitung für den Aufbau, die Konfiguration der Software, und des technischen Ablaufs der Aufzeichnung selbst, inklusive einer Checkliste um das Aufzeichnungssystem als Ganzes in den richtigen Zustand zu versetzen. Diese Anleitung findet man auf dem Dateiserver auf dem auch die Videomittschnitte später abgelegt wurden.

Der Videomittschnitt ist wichtig, bei einer Mitschrift während der Sitzung nicht viel notiert werden kann, ohne Gefahr zu laufen, etwas während des Notierens zu verpassen.

Ausserdem bekommt man beim ersten Eindruck nicht immer alles mit. Ich hatte insbesondere Probleme mit dem Akzent beim indischen Entwickler #2, aber auch sonst gab es manche Situation, die ich mehrfach ansehen und -hören musste, um sie komplett zu erfassen.

Die Videoaufzeichnungen wurden mit der Software *Camtasia Studio*[2] in einem eigenen, proprietärer *camrec*-Format gespeichert und auf dem Dateiserver der Arbeitsgruppe abgelegt. Da *Camtasia Studio* nur für die Betriebssysteme Windows und MacOS verfügbar ist, wollte ich wissen welche Möglichkeiten es gibt die Videos plattformunabhängig und wenn möglichst mit freier Software zugänglich zu machen. *Camtasia Studio* bietet mit der Exportmöglichkeit als AVI-Datei ein verbreitetes Containerformat für Audio- und Video-Daten. Der Video-Codec wird in den Datei-Eigenschaften unter Windows als *TSCC (TechSmith Camtasia Codec)* angegeben und das Audio ist als MP3 kodiert. Die resultierende Dateigrösse und die Dauer des Exportierens einer *camrec*- in eine AVI-Datei deuten auf ein einfaches Umkopieren von einem Containerformat in ein Anderes hin, dadurch sollte also kein Qualitätsverlust entstehen. Das Audioformat MP3 ist weit verbreitet und der *TSCC*-Codec lässt sich mit der *libavcodec* von *FFmpeg*-Projekt[3] dekodieren und damit auch mit vielen freien Abspielprogrammen (*MPlayer*, *VLC media player*, *Xine*, ...), welche diese Bibliothek verwenden. Ich habe das Abspielen unter Windows und Linux mit dem *VLC media player*[4] erfolgreich getestet.

Da beim Export Markierungen und Anmerkungen verloren gehen, die man in *Camtasia Studio* zu frei wählbaren Zeitpunkten im Film setzen kann, wurde weiterhin das *camrec*-Format zur Archivierung des Materials eingesetzt. Diese Anmerkungen wurden zum Beispiel verwendet, um bei einem durchgehenden Mitschnitt einer kompletten *Teles*-Sitzung aus verteilter Programmierung und Reflektion, den Anfang der Reflektion zu kennzeichnen.

3.5.2 Notizen

Um meine Beobachtungen zu einer Sitzung zu erfassen, habe ich eine sehr einfache Vorlage zur Niederschrift entworfen, die in Anhang A auf Seite 83 abgedruckt ist. Als Format wählte ich eine einfache Textdatei mit *reStructuredText*[5] zur Textauszeichnung. *reStructuredText* wurde mit dem Gedanken entwickelt, möglichst "unaufdringlich" zu sein, so dass man den Quelltext lesen kann, ohne durch Textauszeichnungen zu stark vom Inhalt abgelenkt zu werden. Man kann solche Texte also im Regelfall sowohl als normalen Text, aber auch umgewandelt in HTML oder PDF gut lesen.

Die Vorlage gliedert sich in die Abschnitte Sitzungsdaten, Videoauswertung, Generelle Beobachtungen und Anmerkungen, Logs, und Zusammenfassung.

[2] Camtasia Studio: `http://www.techsmith.com/camtasia.asp`
[3] FFmpeg: `http://ffmpeg.org/`
[4] VLC media player: `http://www.videolan.org/vlc/`
[5] reStructuredText: `http://docutils.sourceforge.net/`

Bei den Sitzungsdaten wird die fortlaufende Nummer der *Teles*-Sitzung, das Datum an dem die Sitzung stattfand, ihre Länge in Minuten, die teilnehmenden Entwickler bei *Teles*, und der Sitzungstyp erfasst. Letzterer gibt an, was der Inhalt der Sitzung war, also zum Beispiel Wissenstransfer, Fehlersuche, Quelltextdurchsicht, oder Paarprogrammierung.

Im Abschnitt für die Videoauswertung stehen mit Zeitpunkten versehene Notizen von Gegebenheiten während der *Teles*-Sitzung, die mir erwähnenswert erschienen. Hauptaugenmerk lag auf der Art und Weise wie *Saros* verwendet wurde, und natürlich Auffälligkeiten und Fehlverhalten der Software. Bei der Verwendung war insbesondere interessant zu beobachten, ob die Awareness-Informationen wahrgenommen und genutzt wurden, und wo Informationen offenbar gefehlt haben. Bei den Zeitpunkten habe ich mich für die Zeitangaben relativ zum Aufzeichnungsbeginn entschieden, statt zum Beispiel der lokalen Uhrzeit, damit man die Notizen zusammen mit dem Videomaterial und einer gängigen Abspielsoftware möglichst einfach nachvollziehen kann. Diese relative Zeit wurde von der Aufzeichnungssoftware auch als ständige Einblendung im Videobild integriert, so dass die Zuordnung von Notizen zu Videosequenzen auch bei geschnittenem Videomaterial noch möglich ist.

Vor dem Abschnitt für die Notizen zur Videoauswertung war im ersten Entwurf der Vorlage noch ein Abschnitt für Notizen während der Sitzung vorgesehen. Diese Notizen wurden während der Sitzung formlos mit Zeitpunkt versehen und meist nur stichpunktartigen Beschreibungen von mir per Stift zu Papier gebracht und dann direkt nach Sitzungsende in die Textdatei übertragen und etwas ausführlicher ausformuliert. Bei der Nachbearbeitung der ersten Sitzung, zu der ich sitzungsbegleitende Notizen niederschrieb, bemerkte ich, dass dort am Ende zwei strukturell identische Abschnitte standen, die man beide gedanklich zu einem chronologischen Gesamtbild zusammenfassen musste, so dass es mir sinnvoller erschien den Zeitablauf in nur einem einzigen Abschnitt aufzubereiten. Damit ging die Information verloren, welche Punkte schon während der Sitzung aufgefallen sind, und welche erst in der Nachbereitung der Aufzeichnung, was mir aber nicht wichtig erscheint. Am Ende des Abschnitts zur Videoauswertung sollte die Zeit notiert werden, die für die Auswertung benötigt wurde, um Aussagen darüber treffen zu können ob und wie sich das Verhältnis von Sitzungszeit zu Auswertungszeit über die Sitzungen hinweg verhält. Diese Angabe habe ich eingeführt, als mir bei Sitzung #16 auffiel, dass so eine Nachbearbeitung mehr Zeit benötigte, als ich vermutet hätte.

Nach den Notizen, die sich auf bestimmte Zeitpunkte in der Aufzeichnung der Sitzung beziehen, ist der Abschnitt Beobachtungen und Anmerkungen für übergreifendere Erkenntnisse oder auch Fragen gedacht. Darauf folgt ein Abschnitt für auffällige Einträge in den *Saros*-Logs, die bei der Sitzung angefallen sind. Es gibt dort jeweils einen Unterabschnitt für jeden einzelnen Entwickler. Abgeschlossen wird die Vorlage mit einer Zusammenfassung.

		Dauer in Minuten	
Nummer	**Datum**	**Sitzung**	**Auswertung**
13	2009-07-15	56	
14	2009-08-05		
15	2009-08-11	93	
16	2009-08-12	76	
17	2009-09-10	51	≈150
18	2009-09-06	71	≈210

Tabelle 3.1: Übersicht über die *Teles*-Sitzungen. Sitzung #14 wurde wegen technischer Probleme abgebrochen.

3.6 Beobachtungen bei den Sitzungen

Eine Übersicht der beobachteten *Teles*-Sitzungen ist in Tabelle 3.1 zu sehen. Bei der ersten Sitzung in der Tabelle war ich nicht "live" dabei, da sie zeitlich vor der Einarbeitungsphase lag. Die Informationen aus dieser Sitzung stammen also aus der Aufzeichnung, die Edna Rosen erstellt hat.

Bei *Teles*-Sitzung #14 gibt es keine Sitzungsdauer, da es Probleme beim Installieren von *Saros* gab, die zu einem vorzeitigen Abbruch der Sitzung führten. Die Umstände werden in Abschnitt 3.6.1 näher beleuchtet. Vor Ende des Paarprogrammierungsteils in *Teles*-Sitzung #17 kündigte Entwickler #2 an, dass am Standort in Indien für ungefähr 10 Minuten viel Bandbreite für eine Datenübertragung benötigt wird und die dortige Netzwerkkapazität voraussichtlich komplett ausschöpft. Daraufhin entschied Entwickler #1 diesen Teil der Sitzung abzubrechen, und nach den 10 Minuten die Reflektion durchzuführen, die allerdings wegen einer sehr schlechten Audioverbindung ebenfalls abgebrochen wurde. Darum ist das Video nur 51 Minuten lang.

Aus den beiden Angaben der Dauer der Auswertung des Videomittschnitts lässt sich, ausser dass es bei beiden dreimal länger gedauert hat als die Sitzungsdauer, nicht viel sagen. Interessant wäre es gewesen, dieses Verhältnis über mehr Sitzungen zu beobachten – ich rechnete mit einer Sitzungsanzahl in der gleichen Grössenordnung die Edna's Arbeit zugrundeliegt. *Teles*-Sitzung #16 war die Letzte, an der Edna teilnahm.

Als Typ der Sitzungen, also welche Tätigkeiten sie zum Inhalt hatten, wurde bei den in der Tabelle aufgeführten Sitzungen ausnahmslos "Paarprogrammierung" von den Entwicklern angegeben.

3.6.1 Installationsprobleme

Da das auf *XMPP* [SA04] aufbauende Übertragungsprotokoll von *Saros* nicht separat spezifiziert ist, sondern sich aus den Klassen und logischen Abläufen im Programm ergibt, die sich bei jeder grösseren Modifikationen am Programm ändern, ist es nicht

stabil. In der Regel sind auch direkt aufeinanderfolgende Versionen des Plugins zueinander inkompatibel. Darum ist es wichtig, dass alle Teilnehmer die gleiche Version verwenden und bei *Saros*-Releases bekamen die Entwickler bei *Teles* eine E-Mail mit der Bitte das Plugin bis zur jeweils nächsten Sitzung zu aktualisieren.

Bei *Teles*-Sitzung #14 hatte Entwickler #2 dieses versäumt. Das hätte normalerweise nur zu einer kurzen Verzögerung führen müssen. Die Ursache der Probleme war allerdings nicht sofort offensichtlich, da Entwickler#2 am Anfang auf Nachfrage angab die Aktualisierung des Plugins bereits durchgeführt zu haben. Als der Versionsunterschied als Problemquelle identifiziert werden konnte, hätte die Aktualisierung über die Plugin-Verwaltung von *Eclipse* durchgeführt werden können. Man muss dafür nur im Menü von *Eclipse* über *Help → Check for Updates* in den entsprechenden Dialog gehen und die Plugins aus den Vorschlägen auswählen, die man aktualisieren möchte. Der Aktualisierungsmechanismus von *Eclipse* lädt dann das aktuellere Plugin aus dem Netz und installiert es. Das funktionierte in diesem konkreten Fall nicht, da die Firewall im Netz von Entwickler #2 den Zugriff auf *SourceForge.net* blockierte, wo das *Saros*-Projekt gehostet wird und die veröffentlichten Plugins ablegt.

Diese Sitzung brachte für mich den Bugtracker-Eintrag #*2832593 Provide Drop-In Zip* und für Tas Sóti, der den Einladungsdialog von *Saros* komplett überarbeitete, die Anforderung, dass im Fall von unterschiedlichen *Saros*-Versionen beide Parteien darüber benachrichtigt werden müssen [Só09]. Aktuelle *Saros*-Versionen leisten das, so dass man nach dieser Fehlerquelle nicht mehr lange suchen muss.

3.6.2 Dokumente als Chat zweckentfremdet

In fast jeder *Teles*-Sitzung wurden Informationen in den Quelltext geschrieben oder kopiert, die dort nicht hingehörten, sondern nur an den anderen Entwickler übermittelt werden sollten. Das waren IP-Adressen und Zugangsdaten von Servern, die von Entwickler #1 in eine beliebige, gerade geöffnete Quelltextdatei geschrieben wurden, damit Entwickler #2 sie per Kopieren und Einfügen in eine andere Anwendung übernehmen konnte, Fragmente von Konfigurationsdateien, oder Auszüge von Fehlermeldungen und Stapelabzügen aus Log-Dateien, um darüber reden zu können. *Saros* wurde also als Chat-Anwendung zweckentfremdet.

An der Stelle wäre eine echte Chat-Anwendung, intern oder als externes Programm, oder zumindest eine extra “Scratch”-Datei innerhalb des Projektes in Eclipse, speziell zum Informationsaustausch eine bessere Lösung. Damit bestünde nicht die Gefahr, dass Teile einer solchen Kommunikation aus versehen in den Quelltexten verbleiben und im besten Fall spätere Leser einfach nur verwirren, in schlecheren Fällen wichtige Rechnernamen und Passwörter Unbefugten zugänglich machen.

Nur in einer einzigen von mir beobachteten Sitzung (#18) wurde *Google Talk*[6] verwen-

[6] Google Talk: `http://www.google.com/talk/`

det, um eine Fehlermeldung aus einer Logdatei zu übermitteln. Dieser Dienst wurde von den Entwicklern ausserhalb der *Teles*-Sitzungen zur Kommunikation genutzt.

Diese Zweckentfremdung von beliebigen Dokumenten als Chat, wird vermutlich unterbunden oder zumindest deutlich vermindert, wenn Olaf Loga im Rahmen seiner Arbeit [Log10] den in *Saros* integrierten Chat wiederbelebt.

3.6.3 Inkonsistenzen

Eigentlich müssten immer alle Teilnehmer einer *Saros*-Sitzung identische Dokumente im ge“share”ten Projekt haben. Durch Fehler in *Saros*, aber auch durch Aktionen der Benutzer, die von *Saros* nicht verhindert werden und teilweise auch nicht verhindert werden können, kann es zu Inkonsistenzen zwischen den Daten beim Host und denen bei Clients kommen. Beispielsweise wenn Dateien aus dem Projekt mit Programmen ausserhalb von *Eclipse* bearbeitet werden. Um solche Situationen erkennen zu können, gibt es in *Saros* eine Konsistenzprüfung, den sogenannten *Consistency-Watchdog*, der regelmässig Prüfsummen über geöffnete Dateien vom Host an die Clients verschickt, und dort darüber informiert, wenn die Prüfsummen nicht übereinstimmen. Dann hat der Client die Möglichkeit über die Konsistenzwiederherstellung die Datei vom Host anzufordern.

Während *Teles*-Sitzung #13 gab es zwei Inkonsistenzen auf der Seite von Entwickler #2, bei denen er ohne Ankündigung die Inkonsistenzbehebung aktivierte. Das kam mindestens einmal sehr überraschend für Entwickler #1, und leider “hing” die Inkonsistenzbehebung auch in beiden Fällen und der Vorgang musste ergebnislos abgebrochen werden. Deshalb wurde die *Saros*-Sitzung auch einmal neu gestartet. Der zweite “Hänger” passierte am Ende der für das Programmieren vorgesehenen Zeit und beendete diesen Teil der *Teles*-Sitzung.

In *Teles*-Sitzung #15 gab es relativ kurz nach der abgeschlossenen Einladung eine Inkonsistenzmeldung, die es nicht hätte geben dürfen. Entwickler #2 war als Client im Verfolgermodus und hatte schon vor der *Saros*-Sitzung eine Datei geöffnet, deren projektrelativer Pfad identisch zu dem einer beim Host geöffneten Datei aus dem ge“share”ten Projekt war, die aber zu einem anderen Projekt gehörte. *Saros* enthielt auch zu diesem Zeitpunkt intern schon eine Abfrage, die sicherstellen sollte, dass die Konsistenzprüfung nur Dateien aus dem aktuell ge“share”ten Projekt berücksichtigt! Nachdem die Datei auf beiden Seiten geschlossen wurde, verschwand die Warnung, und es konnte weitergearbeitet werden. Auch ein erneutes Öffnen der Datei beim Host liess den *ConsistencyWatchdog* nicht wieder anschlagen und das Problem konnte von mir weder im Quelltext noch bei Tests nachvollzogen werden. Ein Eintrag dafür im Bugtracker – *#2837566 CW checks files not belonging to shared project* – ist noch offen, allerdings auf die niedrigste Priorität gesetzt. Das Verhalten ist seit dieser Sitzung auch noch nicht wieder beobachtet oder von Benutzern gemeldet worden.

Im Laufe dieser *Saros*-Sitzung gab es noch eine zweite Inkonsistenz, die aber zu erwar-

ten war, da sich Entwickler #1 explizit als Observer über eine Warnung in Form eines Dialogfensters hinwegsetzte, und den Schreibschutz bei einer Datei aufhob. In dem Moment war er sich nicht im Klaren darüber, dass er noch Observer war, weil er davon ausging sich kurz vorher die Driver-Rolle gegeben zu haben. Warum das nicht der Fall war, wird in Abschnitt 3.6.4 erklärt. Diese Inkonsistenz konnte durch die Konsistenzwiederherstellung erfogreich beseitigt werden.

Auch in der letzten *Teles*-Sitzung, Nummer 18, gab es eine Inkonsistenz. Entwickler #2 war exklusiver Driver, aber Entwickler #1 verwendete bei einer gemeinsamen Fehlersuche die Autokorrektur-Funktion vom Java-Editor um sich die vorgeschlagenen Korrekturmöglichkeiten anzeigen zu lassen, und wählte eine davon zur Anwendung aus. Er bemerkte selbst sofort, dass er etwas getan hatte, was er nicht hätte tun dürfen und fragte *"Shall we restart the session because there's an inconsistency now?"* Die Inkonsistenz wurde aber bei Entwickler #2 angezeigt und konnte problemlos aufgelöst werden ohne die *Saros*-Sitzung neustarten zu müssen.

Auch bei den wenigen *Teles*-Sitzungen sieht man, dass Inkonsistenzen in der Praxis doch recht häufig vorkommen. Fast in jeder der Sitzungen waren die Entwickler damit konfrontiert. Die "Hänger" in *Teles*-Sitzungen #13 und die Frage von Entwickler #1 in der letzten Sitzung, ob ein Neustart der *Saros*-Sitzung durchgeführt werden soll, deuten auf Probleme mit der Konsistenzwiederherstellung in der Vergangenheit und daraus resultierender Unsicherheit bei den Entwicklern hin.

3.6.4 Rollenwechsel über den *Shared Project Session*-View

Als Entwickler #1 sich als Host und Observer in *Teles*-Sitzung #15 selbst die exklusive Driver-Rolle geben wollte, klappte das nicht auf Anhieb, weil man unter Windows das Kontextmenü zu einem Eintrag im *Shared ProjectSession*-View nur bekam, wenn der Rechtsklick mit der Maus über dem *Text* des Eintrags, aber nicht über dem Rest der Zeile rechts davon, ausgeführt wurde. In dem Fall bekam man das Kontextmenü für den zuletzt ausgewählten Eintrag. Wenn man also übersah, dass der Eintrag in der Zeile nach dem ausklappen des Kontextmenüs nicht ausgewählt dargestellt wurde, bestand die Gefahr, dass man nicht realisierte, dass die Menüpunkte nicht zu dem Eintrag gehörten, den man eigentlich haben wollte und konnte den Eindruck gewinnen, die Anwahl von Menüpunkten hat gar keine, oder nicht die gewünschte Wirkung. Dieses Verhalten zeigte sich, bei identischem Code, nicht unter Linux und MacOS. Der Umstand, dass der eigene Eintrag im *Shared ProjectSession*-View nicht mit dem Namen oder der Jabber-ID beschriftet ist, sondern mit einem kurzen *"You"*, liess in dem Fall also nur eine kleine Zielfläche für den Rechtsklick, wie man in Abbildung 2.1 auf Seite 11 sehen kann.

Statt sich selbst die Driver-Rolle zu geben, gab er sie Entwickler #2, der sie schon hatte, und versuchte dann eine Datei zu verändern, was zu einer der in Abschnitt 3.6.3 auf der vorherigen Seite beschriebenen Inkonsistenzen führte.

Für diese unglückliche Handhabung in der Benutzeroberfläche wurde von mir im Bugtracker #*2847756 Selecting a user in session view doesn't work in whole line* angelegt und erfolgreich bearbeitet.

3.6.5 Unerwünschte Umwandlung von Zeilenumbrüchen

In *Teles*-Sitzung #13 hat Entwickler #1 angemerkt, dass *Saros* bei jeder während einer *Saros*-Sitzung geöffneten Textdatei die Kodierung der Zeilenumbrüche in die Unix-Konvention umgewandelt hat. Das Teles-Projekt verwendet für die Kodierung der Zeilenumbrüche die Windows-Konvention, da die beteiligten Entwickler mit Windows-Rechnern arbeiten. Dass heisst aus Sicht von *Eclipse*, zum Beispiel bei der eingebauten Versionsverwaltung und beim Anzeigen von Dateiunterschieden, und der "externen" Versionsverwaltung, hat sich durch diese Konvertierung *jede* Zeile jeder während der Sitzung geöffneten Datei verändert und somit die echten Änderungen schwerer nachvollziehbar gemacht. Sofern es überhaupt Änderungen gab. Die Umgehungslösung hierfür war das manuelle Um- also Zurückwandeln der Zeilenumbruchkodierung nach einer *Saros*-Sitzung und vor dem Commit in die Versionsverwaltung. Wenn dieser Schritt vergessen würde, fallen die Folgen möglicherweise erst viel später auf.

In *Teles*-Sitzung #15, der Ersten in der ich nicht nur stiller Beobachter war, wurde mir von Entwickler #1 bestätigt, dass das ein wichtiger Punkt ist. Auf eine Nachfrage, ob noch weitere Punkte von der *Teles*-Seite beanstandet werden, kam erfreulicherweise die Antwort *"No, at the moment there is nothing more. So, it works already really stable."*

Die unerwünschte Zeilenumbruchkodierung ist als #*2828917 Unwanted automatic conversion of line delimiters* in den Bugtracker eingegangen und wurde von mir, wie in Abschnitt 5.1 auf Seite 61 beschrieben, gelöst.

3.6.6 Anzeige des Einladungsfortschritts

Bei der Synchronisation der Projektdaten am Anfang der *Saros*-Sitzungen ist mir aufgefallen, dass man die voraussichtliche Dauer des Dateiabgleichs nicht besonders gut anhand des angezeigten Fortschrittsbalkens ablesen beziehungsweise abschätzen kann. Eine Bemerkung von Entwickler #1 in *Teles*-Sitzung #15 – *"Okay, we have already almost 50%, so it's fast today hopefully"* – bestätigte das, denn diese 50% Fortschritt entsprachen nicht der Hälfte der gesamten Dauer der Einladung. Der in der GUI dargestellte Fortschritt läuft zeitlich nicht gleichmässig ab. Als erstes wurden die zu übertragenen Dateien beim Host in ein ZIP-Archiv verpackt, was verhältnismässig schnell geht und in diesem Fall ungefähr die ersten 45% des Fortschrittsbalkens einnahm. Dann lief die vergleichsweise langsame Übertragung der Daten zum Client bis cirka 68% des Balkens, gefolgt vom Entpacken auf der Client-Seite. Die meiste Zeit wurde also in nur 23% der Fortschrittsanzeige verbraucht. Abbildung 3.1 auf der nächsten Seite veranschaulicht dies.

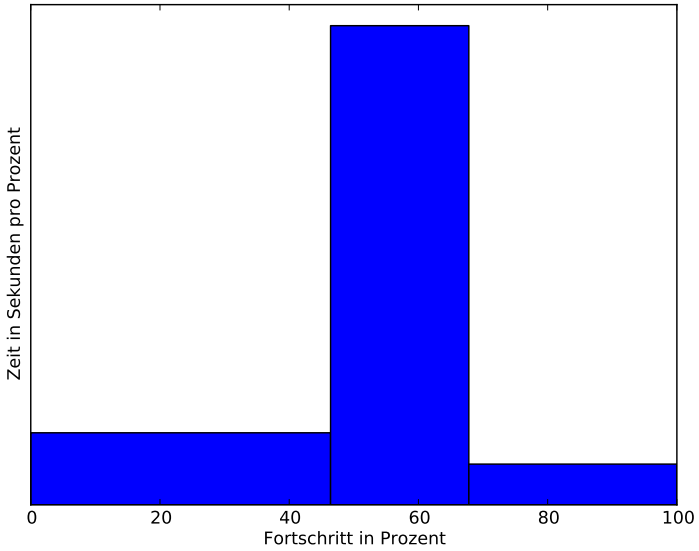


Abbildung 3.1: Zeitverbrauch im Verhältnis zur Fortschrittsanzeige beim Host bei der Einladung bei *Teles*-Sitzung #15. Die Höhe der Rechtecke ist so skaliert, dass die Fläche proportional zur verbrauchten Zeit ist. Das erste Rechteck beinhaltet die Berechnung der Prüfsummen über die Dateien, die Übertragung der Dateilisten inklusive der berechneten Prüfsummen, und die Erstellung des Archivs. Das zweite Rechteck beeinhaltet die Übertragung des Archivs. Und das letzte Rechteck das Entpacken des Archivs beim Empfänger und den Abschluss der Einladung.

	Anteil in %		
	alt	**neu**	
Beschreibung	***(r1561)***	***(r1831)***	***(r1832)***
Verfügbarkeit prüfen		2	1
Version prüfen		3	1
Einladung versenden	5	5	1
Dateilisten abgleichen	15	10	1
Archiv erzeugen	21	5	3
Archiv versenden	49	70	90
Einladung abschliessen	5	5	3
Summe	95	100	100

Tabelle 3.2: Anteile der einzelnen Schritte der Einladung an der Fortschrittsanzeige beim Host vor und nach der Überarbeitung durch Tas Sóti [Só09]. Vor der Überarbeitung der Einladung war der Anteil des Archiv versendens an der Fortschrittsanzeige deutlich kleiner. Ausserdem war die Handhabung im Quelltext nicht besonders sauber, wie man an der Summenzeile sehen kann: 5% der Fortschrittsanzeige waren keiner Tätigkeit zugeordnet.

Diese Beobachtung habe ich nach der Sitzung Tas Sóti mitgeteilt, der in seiner Arbeit [Só09] den Einladungsdialog überarbeitete, und für ihn *#2861406 Progress display of project transfer in invitation dialog* im Bugtracker eröffnet. Tas hat daraufhin die Anteile der einzelnen Schritte des Einladungsprozesses am Fortschrittsbalken neu gewichtet, wie man in Tabelle 3.2 auf der vorherigen Seite sehen kann.

Die in der Tabelle angegebenen 49% für das Versenden des Archivs weichen stark von den im Videomittschnitt beobachteten 23% für diesen Schritt ab. Der Grund liegt in der Art und Weise, wie die Fortschrittsanzeige auf einzelne Aufgaben heruntergebrochen wurde. *Eclipse* bietet für Fortschrittsanzeigen die Schnittstelle *IProgressMonitor*. Vom Konzept her ist so ein Monitor ein Objekt, bei dem man eine Anzahl von Arbeitseinheiten festlegen kann, die zusammen 100% ausmachen und dann kann man im Laufe der Zeit über die *IProgressMonitor.worked()*-Methode eine bestimmte Anzahl von erledigten Einheiten angeben, solange bis alle Arbeitseinheiten erledigt sind. Man kann sich auch einen Kind-Monitor geben lassen, der eine bestimmte Anzahl von Arbeitseinheiten des Eltern-Monitors repräsentiert, und auch diesen wieder in beliebige Arbeitseinheiten unterteilen.

Ein Beispiel: Sei p ein Exemplar, das *IProgressMonitor* implementiert und von der GUI zur Anzeige eines Fortschrittsbalken verwendet wird. p wurde in vier Arbeitseinheiten unterteilt. Jetzt wird ein Kind-Monitor p_2 angefordert, der zwei dieser Arbeitseinheiten repräsentiert, also 50% des Gesamtfortschritts. Der Kind-Monitor wird in fünf Arbeitseinheiten unterteilt und zwei davon werden mit der *worked()*-Methode als erledigt gemeldet. Das wird in der GUI als 20% Fortschritt angezeigt, wie Abbildung 3.2 verdeutlicht.

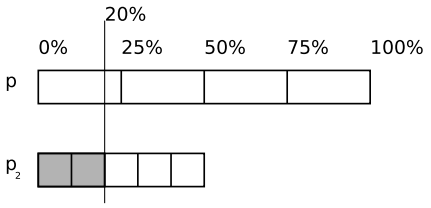


Abbildung 3.2: Fortschrittsmonitor-Beispiel

In Abbildung 3.3 auf der nächsten Seite ist die Aufteilung des Fortschrittsmonitors für den Einladungsprozess zu sehen, wie sie in dem bei *Teles*-Sitzung #15 verwendeten *Saros*-Release praktiziert wurde. Wie man sieht gingen von den 49% noch einmal knapp 5% für das Einlesen des Archivs ab, das versendet werden soll. Der verbleibende Anteil wird auf die verschiedenen zur Verfügung stehenden Übertragungsarten verteilt. Im günstigen Fall hatte man zwei Möglichkeiten – die *Peer-to-Peer*-Verbindung über das *Jingle*-Protokoll, oder der langsamere Weg als *In-Band Bytestream* (*IBB*) mit Umweg über einen oder mehrere Jabber-Server. Beide Möglichkeiten werden nacheinander ausprobiert, das heisst wenn die Übertragung mit *Jingle* funktioniert, macht die Zeit für die Übertragung cirka 22% des Fortschritts aus, und die anderen 22% in der Anzeige für das nicht verwendete *IBB* werden ganz einfach ohne Zeit zu verbrauchen, übersprungen. Diese 22% lassen die im Mitschnitt beobachteten cirka 23% plausibel erscheinen.

In der Praxis konnte ich die Auswirkungen des überarbeiteten Einladungsdialogs, und damit der neuen Gewichtung, bei *Teles* nicht mehr beobachten, da die entsprechende *Saros*-Version nach *Teles*-Sitzung #18 veröffentlicht wurde.

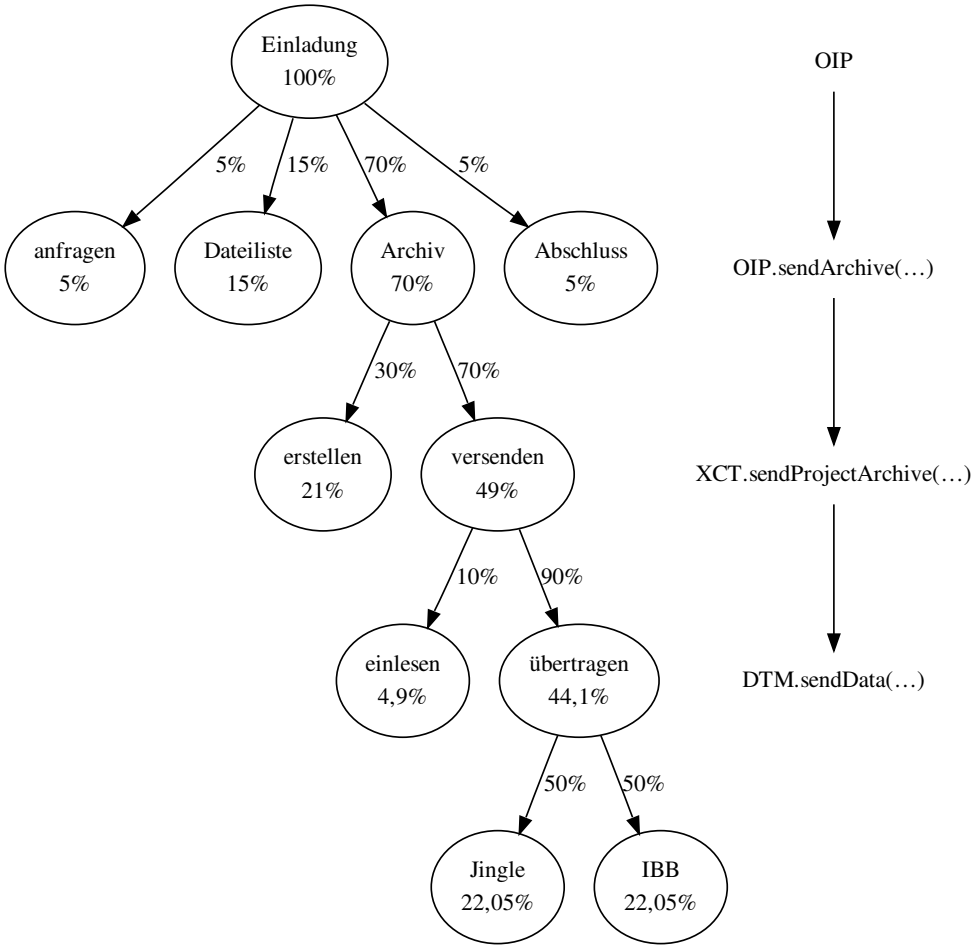


Abbildung 3.3: Aufteilung der Fortschrittsanzeige bei der Einladung für einen Teilnehmer *vor* der Überarbeitung des Einladungsprozesses durch Tas Sóti [Só09]. Die Knoten des Baums repräsentieren Fortschrittsmonitore und enthalten eine Beschreibung und den Anteil an der Fortschrittsanzeige. Die Kanten sind mit den Anteilen am übergeordneten Knoten beschriftet. Rechts steht die Klasse beziehungsweise die jeweilige Methode in welcher der jeweilige Fortschrittsmonitor in *(r1561)* weiter unterteilt wird. Die Klassennamen sind abgekürzt: *OIP* steht für *OutgoingInvitationProcess*, *XCT* für *XMPPChatTransmitter*, und *DTM* für *DataTransferManager*.

3.6.7 Awareness

Bei den meisten *Teles*-Sitzungen konnte man über den Videomittschnitt des Desktops von Entwickler #1 die Verwendung von externen Anwendungen beobachten. Das waren zum Beispiel Textverarbeitungen und PDF-Betrachter mit Dokumentation oder den Anforderungsdokumenten für das gerade entwickelte Feature, oder auch eine Anwendung zur Verwaltung der Datenbank, die von der entwickelten Software verwendet wird. Für beide Fälle gab es in der *Teles*-Sitzung Situationen, in denen bei nicht-verteilter Paarprogrammierung der Partner schneller hätte eingreifen, beziehungsweise erkennen können, dass etwas nicht ganz korrekt läuft. Zum Beispiel bei kleinen Schreibfehlern in Namen, die aus dem Anforderungsdokument oder den Spaltennamen einer Datenbanktabelle in den Quelltext übernommen wurden. Das hatte keine gravierenden Folgen, aber auch grössere Probleme hätten so übersehen werden können, die eher aufgefallen wären, wenn beide Partner physisch vor dem selben Rechner nebeneinander sässen.

Mir fiel auf, dass Informationen über den Audiokanal kommuniziert wurden, die eigentlich auch über die Annotationen hätten klar sein müssen, wie zum Beispiel ob sich Entwickler #2 in einer bestimmten Datei befindet und dort in einer bestimmten Zeile. Diese Fragen wurden von Entwickler #1 auch gestellt, wenn die Cursor-Annotation von Entwickler #2 an der entsprechenden Stelle sichtbar war. Es entstand bei mir der Eindruck, dass diesen visuellen Informationen nicht vertraut wird. Es müsste noch genauer untersucht werden, ob und falls ja, wie man das durch technische Massnahmen verbessern könnte – ausser natürlich, dass sicherstellt gestellt wird, dass die angezeigten Informationen korrekt sind.

Zu Beginn der *Saros*-Sitzung in der letzten *Teles*-Sitzung gab es keine Warnung von der erstmals eingesetzten Versionsüberprüfung bei der Einladung, die Tas Sóti aufgrund der in Abschnitt 3.6.1 auf Seite 29 beschriebenen Probleme mit inkompatiblen *Saros*-Versionen, implementierte. Das die von mir implementierte Anzeige des “Away”-Icon bei inaktivem *Eclipse*-Fenster für Entwickler #2 bei Entwickler #1 angezeigt wurde, war eine weitere Bestätigung, dass beide Seiten die selbe und aktuelle *Saros*-Version verwendeten. Die Implementierung der Icon-Anzeige in *Roster*- und *Shared Project Session*-View ist in Abschnitt 5.2 auf Seite 71 beschrieben.

3.6.8 Verbindungsprobleme

In den Log-Dateien von beiden *Teles*-Entwicklern gab es wiederholt Hinweise auf Verbindungsabbrüche und Probleme beim Wiederverbinden, deren Ausnahmen und Stapelabzüge in Richtung SSL-Sockets wiesen. In den Sitzungen selbst ist das nicht aufgefallen, da diese Vorfälle immer in Zeiträume fielen, in denen *Saros* vorübergehend nicht verwendet wurde, oder nach Sitzungsende, wenn die *Saros*-Sitzung nicht gleich nach

Abschluss der *Teles*-Sitzung beendet wurde.[7]

Die Ausnahmen hatten ihren Ursprung in der *Smack*-Bibliothek, welche auf der *Saros*' Netzwerkschicht aufsetzt. Die Netzwerkschicht von *Saros* fiel in den Aufgabenbereich von Sandor Szücs [Sz"09], weshalb ich mich mit ihm zusammensetze um seine Meinung über die Log-Einträge einzuholen. Das Ergebnis war, dass man oberhalb von *Smack* keine Aussagen über die genauen Ursachen treffen kann, und unsere Kenntnisse über die Interna dieser Bibliothek nicht ausreichten. Speziell um die Verbesserung *Smack*-Bibliothek kümmert sich mittlerweile Henning Staib [Sta10], der seine Arbeit allerdings erst nach dem Ende der *Teles*-Sitzungen aufnahm.

3.6.9 Umgang mit *Eclipse*

In *Teles*-Sitzung #18 bekommt Entwickler #2 die Driver-Rolle um eine neue Klasse zu implementieren. Während er das tat merkte er an, dass die Tastenkombination Ctrl + Esc um die Autovervollständigung von *Eclipse* zu aktivieren nicht funktioniert. Da ich immer Ctrl + [ ] zum Aktivieren dieser Funktionalität verwende, dachte ich mir zunächst nichts dabei und schlug diesen Weg vorerst als Alternative vor.

Als ich nach der Sitzung überprüfen wollte, warum *Saros* diese Tastenkombination blockiert, musste ich feststellen, dass es sich um ein globales Tastenkürzel von Windows handelt, womit man jederzeit das Startmenü aufklappen kann. Technisch wird diese Tastenkombination im Normalfall unter Windows also kein Anwendungsprogramm erreichen. In den voreingestellten Tastatureinstellungen von *Eclipse* (*Window → Preferences → General → Keys*) war auch keine Funktionalität für die Tastenkombination hinterlegt und auch keine Alternative für die Autovervollständigung, die in *Eclipse* als *"Content Assist"* bezeichnet wird. *Eclipse* bietet neben der eigenen Tastaturbelegung in der Grundausstattung auch Tastenkürzel-Schemata an, die denen der *Microsoft Visual Studio*-IDE[8] oder dem *Emacs*-Texteditor[9] ähnlich sind. Bei beiden ist die Tastenkombination Ctrl + Esc unbelegt, und die Autovervollständigung liegt auch auf Ctrl + [ ].

Daraufhin fragte ich per E-Mail nach und es stellte sich heraus, dass Entwickler #2 die Tastenkombinationen verwechselt hatte. Die folgende Konversation ist aus den insgesamt vier E-Mails extrahiert und von redundanten "TOFU"[10]-Zitaten befreit, die in Anhang B auf Seite 85 in voller Länge abgedruckt sind:

> **Ich:** I have tried to reproduce your troubles with the Ctrl+Esc keyboard shortcut but was not able to do so. At least under Windows XP this key

[7] Entwickler #1 hat die *Saros*-Sitzung sogar einmal über Nacht weiterlaufen lassen.

[8] Microsoft Visual Studio: `http://msdn.microsoft.com/vstudio/`

[9] GNU Emacs: `http://www.gnu.org/software/emacs/`

[10] TOFU ist eine im deutschsprachigen Usenet gängige Abkürzung für "Text oben, Fullquote unten", und bezeichnet die Angewohnheit bei Antworten den neuen Text über ein Vollzitat der beantworteten Nachricht zu schreiben.

combination is the same as pressing the "Windows" key, i.e. it opens the start menu in the task bar and never reaches the active application. How did you configure your Windows and Eclipse to get this keyboard shortcut working?

An easy work around is the combination Ctrl+Space that opens the auto completion drop down menu at the cursor position.

Entwickler #2: I need this thing

But it was not working for me when I was in saros session.

Ich: Do you have made any changes to the key mapping in the preferences or do you use any plug-in to simulate the behaviour of a specific editor? Because I cannot reproduce the problem with Ctrl+Space and I can't get Ctrl+Esc working in the first place, because Windows grabs that key combination.

Hard for me to fix a problem without being able to reproduce it. :-(

Entwickler #2: It has been a confusion for ctlr+Esc. Instead I wanted cltr+space. Cltr+ Esc doesnot really exist for Eclipse.

Bei dem *"it"* in der ersten Antwort von Entwickler #2 bin ich mir nicht sicher, ob es sich auf die von ihm gewünschte Tastenkombination, die Funktionalität der Autovervollständigung, oder die vorgeschlagene Umgehungslösung, also eigentlich die Standardlösung, bezieht.

Den nach der Sitzung angelegten Eintrag im Bugtracker – *#2861409 Ctrl + Esc does not work for auto completion* – habe ich nach der letzten E-Mail als gegenstandslos (*"invalid"*) geschlossen.

In der selben *Teles*-Sitzung bekommt Entwickler #2 die Driver-Rolle um eine Klasse zu implementieren. Dabei vergisst er zwischen dem Klassennamen und einem Interface-Namen das Java-Schlüsselwort **implements**. Die visuellen Hinweise in *Eclipse*, dass der Quelltext nicht korrekt war, wurden von Entwickler #2 offenbar nicht wahrgenommen. Wenn man das Schlüsselwort **implements** zwischen Klassennamen und Interface-Namen vergisst, wird der Interface-Name rot unterstrichen und im linken Rand des Editors erscheint ein Fehler-Icon für die Zeile. Am rechten Rand wird oben ein kleines rotes Quadrat angezeigt, wenn irgendwo innerhalb der Datei ein syntaktischer Fehler existiert, und im Rand gibt es für jeden Fehler eine Annotation auf die man mit der Maus klicken kann, um zu der fehlerhaften Zeile zu gelangen.

Eine *Teles*-Sitzung davor fragte Entwickler #1 ob an einer bestimmten Stelle im Quelltext bei einer Bedingung ein logisches "Und" oder ein "Oder" steht. Im Quelltext stand ein "`&&`" und kein "`||`".

Diese Beobachtungen zusammengenommen machen auf mich im Nachhinein den Eindruck, als wenn Entwickler #2 normalerweise nicht mit *Eclipse* arbeitet und sich im

Laufe der 18 *Teles*-Sitzungen auch nicht gut eingearbeitet hat, wenn er erst in der letzten Sitzung die Autovervollständigung vermisst. Vielleicht gab es auch technische Probleme mit der Anzeige oder physiologische mit der Wahrnehmung, denn insbesondere die Frage nach der logischen Verknüpfung interpretiere ich so, dass er sie entweder optisch nicht erkennen konnte, oder dass ihm die Bedeutung von "&&" in Java nicht bekannt war. Letzteres halte ich für sehr unwahrscheinlich.

3.6.10 Sonstiges

Meine Rolle für die Entwickler bei *Teles* war die eines technischen Ansprechpartners für *Saros*. Darunter fiel das Beantworten von Fragen, sowie das Stellen von Fragen zu technischen Aspekten von *Saros*. Zum Beispiel die allgemeine Frage, ob es aus Sicht der Entwickler besondere Punkte gibt, die verbesserungswürdig sind, oder ob sie mit einer implementierten Problemlösung zufrieden sind. Auch Anmerkungen, das und wie man Features von *Saros* verwenden kann. Wobei Letzteres eventuell auch in den Ablauf des Paarprogrammierungsprozesses eingreifen könnte und deshalb mit Edna Rosen als Betreuerin dieser Prozesseinführung abgesprochen werden sollte.

Was mir nicht zustand, war das Einmischen in die Programmiertätigkeit der Entwickler oder Anmerkungen zum Inhalt ihrer Arbeit. Was nicht immer ganz leicht fiel – als Programmierer stumm an einer Paarprogrammierungssitzung teilzunehmen ohne teilweise Offensichtliches anzusprechen und damit Teil des "Paars" zu werden. Als beispielsweise in *Teles*-Sitzung #18 ein Tippfehler gesucht wurde, den ich schon in dem Augenblick wo er passierte wahrnahm, waren das für mich "gefühlt" sehr lange 2 Minuten und 47 Sekunden, die Entwickler #1 benötige das fehlende 'e' in *EditorListner* zu finden und nachzutragen.

Bei allgemeinen Hinweisen zur Bedienung von *Eclipse* war ich mir unsicher. Bei Entwickler #1 ist mir zum Beispiel in mehreren *Teles*-Sitzungen aufgefallen, dass er zum Suchen nach Text in einem Editor vor dem Aufrufen des Suchdialogs immer erst manuell die Schreibmarke an den Anfang der allerersten Zeile im Dokument setzt. Das ist etwas was ich am Anfang in *Eclipse* auch eine Zeit lang tat, weil der Suchdialog die Eigenschaft hat, in der Voreinstellung immer nur ab der Position der Schreibmarke bis zum Ende eines Dokuments zu suchen, so also keine passenden Textstellen *vor* der Position der Schreibmarke findet. Dann habe ich die Option *wrap search* in dem Dialog entdeckt, welche bewirkt, dass am Ende des Dokuments die Suche an den Anfang "umgebrochen" wird, man also auf diese Weise das gesamte Dokument durchsuchen kann, unabhängig davon, wo die Schreibmarke gerade steht. Ich habe mich letztlich dagegen entschieden das zur Sprache zu bringen, da ich mich nicht in der Position sah, als externer Beobachter, einem langjährigen Entwickler sein Werkzeug zu erklären.

3.7 Das Ende der Sitzungen

In der Reflektion in *Teles*-Sitzung #18 wusste Entwickler #1 nicht genau wie es längerfristig weitergeht, da die konkreten, künftigen Aufgaben für die beiden Standorte Wien und Bangalore noch nicht bekannt waren. Damit war auch unbekannt, ob diese sich für verteilte Entwicklung eignen würden.

Die 19. *Teles*-Sitzung wurde dann viermal in Folge wöchentlich per E-Mail von Entwickler #1 abgesagt. Das erste Mal weil seine Stimme "weg" war, zweimal ohne Begründung, und das letzte Mal weil es zwischen Wien und Bangalore keine gemeinsame Aufgabe zu bearbeiten gab. In dieser vierten Absage kündigte Entwickler #1 seinen Urlaub an, stellte aber in Aussicht, das nach vier Wochen die *Teles*-Sitzungen wieder aufgenommen werden könnten: *"I think we can restart saros sessions when I am back (Week 47)."* Dazu kam es allerdings nicht mehr, weil sich die Situation bezüglich gemeinsamer Aufgaben an den beiden Standorten nicht veränderte.

Eine Zusammenarbeit zwischen *Teles*-Entwicklern in Berlin und China kam nicht, wie am Anfang in Aussicht gestellt, zustande. Mündlich habe ich erfahren, dass das Projekt, welches an diesen beiden Standorten entwickelt wird, in der Programmiersprache *C++* verfasst ist, und dass es für die Entwickler nicht in Frage kam von ihren gewohnten Entwicklungswerkzeugen auf *Eclipse* mit dem *C/C++ Development Tools (CDT)*[11] umzusteigen.

Zu einer Datenerhebung für die ursprünglich geplante Bewertung der Häufigkeit und Art von Defekten und deren Entwicklung über die Zeit kam es nicht, weil mir die Anzahl der Sitzungen dafür zu klein erschien. Statt der erwarteten circa 16 *Teles*-Sitzungen gab es abzüglich der Einarbeitungsphase letztlich nur zwei.

Trotzdem denke ich, dass das Beobachten von Entwicklern, die *Saros* für eine produktive Aufgabe einsetzen, eine wertvolle Quelle war und in Zukunft vielleicht auch wieder sein könnte. Man sieht wie Menschen ohne Vorkenntnisse mit *Saros* in der Praxis zurechtkommen, und erhält qualitativ gute Fehlerberichte, inklusive Videomittschnitt und eindeutig zugeordneten Log-Dateien, sowie eine Möglichkeit auch Rückfragen zu stellen. Etwas was zum Beispiel bei den elektronisch ausgefüllten Umfragebögen, die durch Lisa Dohrmann's Arbeit [Doh09] anfallen, in der Regel fehlt.

[11] CDT: `http://www.eclipse.org/projects/project_summary.php?projectid=tools.cdt`

4 Entwicklungsprozess

Als Christopher Oezbek das *Saros*-Team verliess, übernahm ich einen Teil der Aufgaben, die er als technischer Projektleiter hatte. Dazu gehörte die Vergabe von Rollen im Entwicklungsprozess an die *Saros*-Team-Mitglieder, einen Überblick über die Aufgaben und Pläne der *Saros*-Entwickler das jeweils nächste Release betreffend zu haben, und die Qualitätssicherungsmassnahme der Patch-Durchsichten zu überwachen und falls nötig zu steuern.

Es war ein wöchentlicher Termin mit der Projektleitung angesetzt, bei dem ich eine Einschätzung des Stands der Umsetzung der einzelnen Aufgaben der *Saros*-Entwickler abgeben konnte. Ausserdem wurden dort Bugtracker-Einträge gemeinsam priorisiert und vor Releases über sinnvolle Arbeitspakete für das nächste Release diskutiert.

In den folgenden Abschnitten wird das *Saros*-Team vorgestellt (Abschnitt 4.1); der Entwicklungszyklus, die beteiligten Rollen, und der Ablauf eines einzelnen Releases beschrieben (Abschnitte 4.2 auf Seite 45, 4.3 auf Seite 46, und 4.4 auf Seite 48); sowie die beiden Qualitätssicherungsmassnahmen der manuellen Tests und des Patch-Review-Prozesses näher beleuchtet (Abschnitte 4.5 auf Seite 49 und 4.6 auf Seite 54).

4.1 Das Saros-Team

Um Aufgaben an andere Mitwirkende am *Saros*-Projekt delegieren zu können, und auch für Aufgaben als technischer Projektmanager, wie zum Beispiel die Releaseplanung, benötigt man einen Überblick über die Rollen und Tätigkeiten der einzelnen Beteiligten. Es folgt eine Übersicht über alle *Saros*-Team-Mitglieder, deren Wirken sich zeitlich mit meiner Arbeit überschnitt, mit einer kurzen Beschreibung ihrer Tätigkeiten. Die Aufstellung ist alphabetisch nach Nachnamen sortiert:

Karl Beecher ist der neue technische Projektleiter von *Saros*. Er hat Christopher Oezbek abgelöst. Als englischer Muttersprachler war sein Beitritt zum *Saros*-Team der ausschlaggebende Grund auf Englisch als Verkehrssprache auf der *Saros*-Mailingliste umzusteigen.

Lisa Dohrmann erstellte während ihrer Bachelor-Arbeit eine Online-Umfrage[1], auf die *Saros* in regelmässigen Abständen am Ende eine *Saros*-Sitzung hinweisen kann,

[1] Saros Online-Umfrage über die jeweils letzte *Saros*-Sitzung: `http://survey.mi.fu-berlin.de/public/survey.php?name=SarosFastUserFeedback_1`

und implementierte ein Rahmenwerk, das während einer Sitzung statistische Daten sammelt und an einen Server zur späteren möglichen Auswertung übermittelt [Doh09]. Beiden Funktionen muss der Benutzer zustimmen ("Opt-In") und kann sie auch jederzeit in den Einstellungen von Saros komplett deaktivieren. Bei den anonym erhobenen Statistikdaten können Benutzer ein Pseudonym angeben, falls sie es den *Saros*-Entwicklern ermöglichen möchten mehrere Sitzungen einer Person beziehungsweise einem Pseudonym zuzuordnen. Desweiteren kann der Benutzer erlauben, dass am Ende einer *Saros*-Sitzung neben den Statistikdaten auch eine Log-Datei von *Saros* an den Server übermittelt wird. Entweder nur mit Fehlern, oder mit allem was im Log aufgezeichnet wurde.

Alena Kiwitt [Kiw10] stellte in ihrer Bachelor-Arbeit Anforderungen auf, die ein *Eclipse*-Plugin erfüllen muss, um als *Saros*-kompatibel gelten zu können. Sie definierte Tests um diese Anforderungen überprüfen zu können, und wendete sie auf eine Reihe von Plugins an. Wenn ein Plugin nicht kompatibel zu *Saros* war, und das an *Saros* lag, war es ihre Aufgabe das Problem zu identifizieren, zu dokumentieren, und falls möglich zu beheben.

Stephan Lau [Lau10] implementierte im Rahmen seiner Bachelor-Arbeit eine Screensharing-Funktionalität. Dafür war als Grundlage ein Rahmenwerk zur Übertragung von Datenströmen notwendig, mit dessen Hilfe die Videodaten des Screensharing übermittelt werden können, und das später auch für andere Datenübertragungen von *Saros* verwendet werden kann. Zum Beispiel für das Archiv mit den Projektdateien, welches der Host bei einer Einladung zu einer *Saros*-Sitzung an die Clients verschickt.

Olaf Loga [Log10] reaktivierte und überarbeitete im Rahmen seiner Bachelor-Arbeit die Chat-Funktionalität in *Saros* und arbeitete an einer Sprachübermittlung (*VoIP*) direkt über das Plugin, statt über eine unabhängige, externe Software, wie es bisher vom *Saros*-Team empfohlen und praktiziert wurde. Diese Kommunikationsmöglichkeiten sollten so umgesetzt werden, dass man sie später statistisch auswerten kann.

Christopher Oezbek war der alte technische Projektleiter von *Saros*, sowie Betreuer von *X-Arbeiten*[2] und wurde von Karl Beecher abgelöst. Desweiteren war er mit Stephan Salinger zusammen Betreuer dieser Arbeit.

Stephan Salinger war gemeinsam mit Christopher Oezbek Betreuer dieser Arbeit und war und ist *Saros*-Projektleiter und Betreuer anderer *X-Arbeiten* im Saros-Projekt.

Tas Sóti überarbeitete in seiner Bachelor-Arbeit [Só09] den Einladungsprozess von *Saros*. Er fügte eine Versionsüberprüfung hinzu, die vor inkompatiblen *Saros*-Versionen zwischen Host und Client warnt, und ermöglichte die Anzeige des Einladungsfortschritts bei allen Beteiligten. Es wurde auch sichergestellt, dass der Ein-

[2] *X-Arbeiten* ist die zusammenfassende Bezeichnung von Diplom-, Studien-, Bachelor- und Masterarbeiten innerhalb der Arbeitsgruppe *Softwaretechnik*.

ladungsvorgang jederzeit von beiden Seiten aus abgebrochen werden kann, und dass der Einladungsdialog nicht über die gesamte Dauer der Einladung modal angezeigt wird, und damit die Bedienung von *Eclipse* vollständig blockiert.

Henning Staib [Sta10] beschäftigte sich im Rahmen seiner Diplomarbeit mit der *Smack*-Bibliothek, die von *Saros* zur Kommunikation über das *XMPP*-Protokoll verwendet wird. Seine Aufgaben lagen im Beheben von bekannten Fehlern in der Bibliothek, sowie der Überprüfung, ob *Saros* die API korrekt verwendet. Ein weiterer Aspekt war die Reaktivierung der Entwicklergemeinde von *Smack*.

Eike Starkmann [Sta09] strebte im Rahmen seiner Diplomarbeit danach, die Verwendung von *Saros* in OpenSource-Projekten zu etablieren. Damit könnte man, ähnlich wie bei den *Teles*-Sitzungen (siehe Abschnitt 3 auf Seite 23) Erkenntnisse aus dem praktischen Einsatz von *Saros* gewinnen, den Bekanntheitsgrad steigern, und eventuell sogar externe Entwickler für die Mitarbeit an *Saros* interessieren.

Sandor Szücs [Sz"09] verbesserte in seiner Diplomarbeit die Netzwerkschicht von *Saros*. Informationen über Art und Zustand von Verbindungen sollten dem Benutzer zugänglich gemacht werden. Die Robustheit und Handhabung von *Peer-to-Peer*-Datenübertragungen verbessert werden. Und er arbeitete an einem Test-Rahmenwerk zur automatisierten, verteilten Überprüfung von Belastbarkeit und Zuverlässigkeit der Netzwerkschicht.

Sebastian Ziller behandelte in im Rahmen seiner Diplomarbeit [Zil09] Nebenläufigkeitsaspekte in *Saros*. Dazu gehörte die Fehlersuche und -beseitigung in der Implementierung der vorhandenen Operationen, ein globaler Sperrmechanismus, um Operationen über mehrere *Saros*-Exemplare hinweg zu synchronisieren, und die Entwicklung einer nebenläufigen Undo-Funktionalität.

4.2 Ablauf eines Release-Zyklus

Saros wird nach einem evolutionärem/inkrementellen Prozess-Modell entwickelt, wie man man es häufig in OpenSource-Projekten antrifft. Die Releases finden in der Regel alle drei Wochen am Freitag statt, gemäss des *OpenSource*-Mottos *"Release early, release often"* aus [Ray99]. Die dritte Woche, ist dabei als Release-Woche bezeichnet, in der ein fester Ablauf für einen Test und die Veröffentlichung selbst festgeschrieben ist [ST10], der sich grob in die Schritte

- Testvorbereitungen,
- Testdurchführung,
- Fehlerbeseitigung und Entscheidung ob ein Release gemacht werden kann,
- und Planung für das nächste Release aufteilt.

Anfangs waren für diese Punkte die Tage Mittwoch bis Freitag vorgesehen, was sich zum Einen als zeitlich etwas knapp herausstellte, weil der Test einen ganzen Tag in Beschlag nahm – den Donnerstag – und damit nur Freitag Vormittag zum Beheben von Fehlern, die ein Release verhindern, blieb, und zum Anderen weil der Donnerstag der Wochentag von Mittwoch bis Freitag war, an dem die die grösste Zahl von *Saros*-Entwicklern einer Erwerbstätigkeit nachging, also nicht am Institut war. Ab Release 9.10.2 wurde der Plan deshalb etwas gestreckt, so dass die Testvorbereitungen am Dienstag, der Test selbst am Mittwoch, und das Release am Freitag war. Der ganze Donnerstag und Freitag Vormittag sind für die Fehlerbeseitigung vorgesehen, wobei wie schon erwähnt am Donnerstag viele Entwickler nicht, oder zumindest nicht voll zur Verfügung standen.

4.3 Rollen

Für ein Release gab es vier Rollen zu besetzen, für die jedes an der Programmierung beteiligte Team-Mitglied in Frage kam. Das waren der *Test-* und der *Releasemanager* (*TM* und *RM*) sowie jeweils ein Assistent zu diesen beiden Rollen (*ATM* und *ARM*). Die zwei Gruppen hatten zwar ihren jeweiligen eigenen Aufgabenbereich, mussten allerdings auch zusammenarbeiten, damit ein Release reibungslos funktioniert. Die Fehlerbeschreibungen im Bugtracker und in den Tests setzten in der Regel zumindest eine grobe Ahnung von der Implementierung von *Saros* voraus. Deshalb war die Vergabe der Rollen auf Projektbeteiligte beschränkt, die auch Umgang mit dem Quelltext von *Saros* hatten.

Die Assistenten sollten die “Haupt”rollen zum Einen natürlich unterstützen, aber auch “ausbilden”. Denn es war vorgesehen, dass falls möglich, in der Rolle Unerfahrene von jemandem unterstützt werden sollten, der die “Hauptrolle” schon mindestens einmal ausgefüllt hatte. Auf diese Weise sollte das Wissen und die praktische Erfahrung besser an die jeweils nächste Generation von Team-Mitgliedern weitergegeben werden. Würde man die Rollen vom Wissenstand her umgekehrt besetzen, bestünde die Gefahr, dass der Erfahrenere zu schnell die der Rolle zugeordneten Aufgaben abarbeitet und der Unerfahrenere nicht alles erfasst, und auch keine praktische Erfahrung mit den erforderlichen Arbeitsabläufen erlangt.

Beim Test wurde zusätzlich zu den beiden Testmanagern noch mindestens ein “einfacher” Tester benötigt, weil einige Tests voraussetzten, dass mindestens drei Leute etwas gleichzeitig ausführen. Diese Rollen wurden normalerweise nicht explizit im Voraus festgelegt. Es wurde davon ausgegangen, dass die am Test-Tag anwesenden Team-Mitglieder als Tester zur Verfügung stehen, es sei denn sie haben etwas Dringenderes zu erledigen.

Ab Release 9.10.30 fiel die Vergabe der Rollen als vorübergehender technischer Projektmanager in meinen Aufgabenbereich. Dazu nahm ich mir bei jedem Release die

	TM	ATM	RM	ARM
Eike Starkmann	2	1	1	0
Henning Staib	0	0	0	0
Marc Rintsch	1	3	0	1
Robert Kunze	0	1	-	-
Sandor Szücs	2	1	1	1
Stephan Lau	0	0	0	0

Tabelle 4.1: Die Tabelle zeigt, wie oft jeder die jeweilige Rolle bis zur Planung von Release 9.10.30 inne hatte. Es sind nur zu diesem Zeitpunkt aktive "Sarosianer" aufgeführt und die Zählung geht bis zum Release 2009-05-29 zurück.

Aufzeichnungen, wer in der Vergangenheit welche Rollen besetzte, als Entscheidungsgrundlage, um für die beiden Hauptrollen Personen mit möglichst wenig Erfahrung in der jeweiligen Rolle auszuwählen, und als Assistenten diejenigen von den Erfahrenen, die schon länger keine Rolle mehr ausgefüllt hatten. Auf diese Weise sollten die Aufgaben möglichst gerecht verteilt werden. Neben den beiden Bedingungen, die eher als Richtlinien, denn feste Regeln zu sehen sind, musste ich noch andere Dinge berücksichtigen. Zum Beispiel wenn wichtige Ereignisse ein Ausfüllen einer Rolle verhinderten. Das war bei Eike Starkmann beispielsweise einmal der Fall, als er an einem Test-Tag, für den er für die *ATM*-Rolle in Frage kam, eine Einladung zu einem Entwickler-Chat von einem OpenSource-Projekt wahrnehmen musste.

Die Aufzeichnungen der vergangenen Release-Planungen im internen, nicht-öffentlichen Wiki-Bereich des *Saros*-Teams reichten sechs Releases bis zur Planung von Release 9.5.29 zurück. Danach ergab sich die Tabelle 4.1. Man sieht dort deutlich, dass Henning Staib und Stephan Lau neu im *Saros*-Team waren, weshalb ich ihnen für Release 9.10.30 die Hauptrollen gab. Henning als Testmanager und Stephan Lau als Releasemanager. Unterstützt wurden die beiden von Sandor Szücs (*ATM*) und Eike Starkmann (*ARM*).

Etwas irreführend ist der Wert 0 für meine Person in der *RM*-Spalte, der vermuten lässt, dass ich keine Erfahrung mit dieser Rolle hatte. Das liegt an dem begrenzten Zeitraum für den Aufzeichnungen über die Release-Planungen im Wiki vorlagen. Ich hatte die Rolle zu dem Zeitpunkt schon mehrfach während meiner Studienarbeit ausgefüllt.

Robert Kunze ist in der Tabelle mit aufgeführt, mit der Einschränkung, dass er nicht für eine der Releasemanager-Rollen in Frage kommt, weil seine Arbeit nichts mit der Implementierung und Entwicklung zu tun hatte. Das galt im Grunde auch für Eike Starkmann, aber Eike hatte in seiner Arbeitsbeschreibung auch Implementierungstätigkeiten aufgeführt.

Da man sich für die Rollen doch etwas näher mit der Implementierung oder zumindest der Entwicklungsumgebung und der Versionsverwaltung auseinandersetzen musste, beschloss ich Robert bei der zukünftigen Rollenvergabe nicht mehr zu berücksichtigen. Was ihn nicht als einfachen Tester ausschloss, sofern er verfügbar war und teilnehmen

wollte.

4.4 Die Release-Woche

Im "Entwicklerhandbuch" [ST10], einer Wiki-Seite mit Anleitungen für verschiedene Tätigkeiten im *Saros*-Entwicklungsprozess, wird im Abschnitt *Our Release Schedule* der Inhalt der Release-Woche stichpunktartig zusammengefasst.

Dienstag Für die Vorbereitungen des Test-Tags erstellt der Releasemanager eine Aufstellung der behobenen Defekte und neuen Features seit dem letzten Release – das sogenannte "Changelog". Diese Aufstellung sollte nach Wichtigkeit für den Benutzer von *Saros* geordnet, und in für ihn verständlichen Formulierungen abgefasst sein. Das Ergebnis wird als E-Mail an die *Saros*-Mailingliste geschickt und dient dem Testmanager und seinem Assistenten als Grundlage zur Aktualisierung der Testfälle für den folgenden Test-Tag.

Mittwoch Am Test-Tag erstellt der Releasemanager einen Entwicklungszweig für den Test in der Versionsverwaltung und lässt die vorhandenen Unittests ablaufen. Über den angelegten Entwicklungszweig und die Ergebnisse der Unittests informiert er das Team mittels einer E-Mail an die *Saros*-Mailingliste. Den Hauptteil des Tages wird dann das in Abschnitt 4.5 auf der nächsten Seite beschriebene Testverfahren vom Testmanager, seinem Assistenten, und mindestens einem weiteren Tester, durchgeführt. Nach dem Abschluss des Testverfahrens fasst der Testmanager die Ergebnisse in einer E-Mail an die *Saros*-Mailingliste zusammen. Das beinhaltet insbesondere kritische Defekte und Regressionen und falls möglich die Bestätigung, dass laut "Changelog" behobene Defekte tatsächlich nicht mehr auftreten.

Donnerstag Dieser Tag ist komplett für das Beheben der gefundenen kritischen Defekte im Entwicklungszweig vorgesehen.

Freitag Vom Release-Tag ist der Vormittag für das Beheben von kritischen Defekten im Entwicklungszweig vorgesehen. Am frühen Nachmittag wird von Release- und Testmanager noch ein eher informeller Test durchgeführt, mit besonderem Augenmerk auf die kritischen Defekte, die seit dem Test-Tag behoben wurden, und die Unittests werden noch einmal ausgeführt. Nach diesen Tests hat der Releasemanager die Aufgabe die Änderungen im Entwicklungszweig wieder in den Hauptzweig fliessen zu lassen, und das Release zu erstellen und anzukündigen. Die Ankündigung geht dieses mal nicht nur über die *Saros*-Mailingliste, sondern auch über die öffentliche Ankündigungs-Mailingliste des Projekts auf *SourceForge*. Zum Abschluss des Tages ist ein Treffen des gesamten *Saros*-Teams für die Planung des nächsten Releases angesetzt.

Bei der Beschreibung der Testvorbereitungen kam Anfangs offenbar nicht deutlich genug zum Ausdruck, dass die beiden Aufgaben für Release- und Testmanager *zusammen*

bis 16 Uhr zu erledigen sind, und die zweite Aufgabe von der Ersten abhängig ist. Zumindest mir ist es als Releasemanager einmal passiert, dass ich erst gegen 15^{30} Uhr meine Ergebnisse an den Testmanager weiterreicht, und in der Rolle des Testmanager musste ich auch einmal bis zu dieser Uhrzeit auf den Releasemanager warten. Das Missverständnis kam wahrscheinlich dadurch zustande, dass in den Stichpunkten in der Anleitung auf oberster Ebene der Wochentag und die Angabe "bis 16:00" stand und darunter je ein Unterpunkt für den *RM* und einer für den *TM* ohne dass diese sich aufeinander bezogen. Dadurch konnte es passierten, dass sich der jeweilige Rolleninhaber nur seinen Teil der Aufgabe durchlas. Den übergeordneten Punkt versah ich 2009-09-22 mit einem entsprechenden Hinweis.

Als der Zeitplan ab Release 9.10.2 um den Donnerstag als vollständigen Tag zum Beheben von kritischen Defekten erweitert wurde, war nur einer von vier Entwicklern an diesem Tag nicht mit einer Erwerbstätigkeit beschäftigt. Für den grösseren Anteil der Entwickler blieb zur Fehlerbeseitigung also weiterhin nur der Freitag Vormittag beziehungsweise der Donnerstag zusätzlich zur Erwerbstätigkeit. Bis Anfang 2010 hat sich das Anwesenheitsverhältnis mit zwei von fünf Entwicklern an diesem Tag nur wenig verbessert.

Jeden ersten Mittwoch im Monat war Wartungstag des Rechnerbetriebs am Institut. An diesem Tag bestand also die Gefahr, dass der Test-Tag durch Wartungsarbeiten an der IT-Infrastruktur des Instituts beeinträchtigt wird. Bisher entstanden dadurch noch keine konkreten Probleme, aber den Mittwoch als Test-Tag sollte man eventuell trotzdem überdenken, oder zumindest sicherstellen, dass keine Tests am ersten Mittwoch im Monat angesetzt werden.

Am Freitag musste nach dem informellen "Sanity Test" spätestens entschieden werden, ob eine Release wegen zu schwerwiegenden Defekten verschoben werden muss. In dem Fall wurde die Release-Planung vorgezogen und ein neuer Release-Tag in der folgenden Woche angesetzt, der je nach Einschätzung der Betroffenen wie lange die Eingrenzung und Beseitigung dauern wird, ausgewählt wurde.

4.5 Tests

Die Tests, welche am Test-Tag durchgeführt wurden, haben den Charakter von *Systemtests* [Bal98, S. 537 ff.]. Dabei wird die Software als Ganzes, aus Sicht des Endbenutzers, und mit Augenmerk auf die spezifizierten Anforderungen getestet. Die Testumgebung sollte so weit als möglich dem späteren Einsatzszenario entsprechen.

Am Vortag des Tests wurde ein Entwicklungszweig in der Versionsverwaltung angelegt und das Testverfahren vorbereitet, so dass im Idealfall der gesamte Test-Tag ausschliesslich zum Durchführen der Tests genutzt werden konnte. Das Testverfahren wurde im Laufe der Zeit mehrfach überarbeitet. Ursprünglich basierte es aus einem Paket

von mehreren *Word*-Dokumenten. Die Tests waren nach Themengebieten auf 11 Dokumente aufgeteilt:

0. (Deckblatt)
1. Konfiguration
2. Einladung
3. Sitzung
4. Speichern
5. Singledriver
6. Multidriver
7. Awareness
8. Verfolgermodus
9. Konsistenzwiederherstellung
10. Feedback

Die Dokumente waren als Formulare gestaltet, die während des Testverfahrens durchgearbeitet und ausgefüllt wurden. Das Deckblatt erfasste dabei Informationen wie den Namen des Testers, den verwendeten Jabber-Server und die Jabber-ID, das Datum, den Rechnernamen, die verwendete Software und deren Versionen (Java, Eclipse, Saros, Betriebssystem, ...), oder die Art der Netzverbindung (LAN, WLAN, welches Netz).

Ein einzelner Testfall war auf einem Blatt im Querformat gesetzt. Unter der Überschrift folgte eine Beschreibung der Schritte für den Testfall. Am unteren Ende der Querseite stand eine Tabelle mit bekannten Problemen aus dem Bugtracker, die als Bug-ID und Titel des Bugtracker-Eintrags in den ersten beiden Spalten angegeben wurden. In den restlichen vier Spalten stand die Nummer des Schritts, in dem sich das Problem zeigte, die Revision in der Versionsverwaltung, in welcher das Problem behandelt wurde, und zwei Spalten mit Feldern zum ankreuzen. Das Erste falls das Problem im Test beobachtet wurde, und das Zweite falls das Problem nicht beobachtet werden konnte *und* der Tester davon ausging, dass der Grund dafür war, dass das Problem behoben wurde.

Zwischen Ablaufbeschreibung und der Auflistung von bekannten Problemen war Platz für Notizen und Beobachtungen des jeweiligen Testers vorgesehen.

Da die Tabellen mit den bekannten Problemen auch die Behobenen enthielten, also manchmal um neue Probleme ergänzt, aber nie verkürzt wurden, gab es auf der Rückseite jedes Testfalls eine identische Tabelle, in welche alte, gelöste Probleme verschoben wurden.

4.5.1 Testdurchführung

Von den Dokumenten wurde für jeden Testteilnehmer ein Exemplar ausgedruckt. Anfangs alle Dokumente mit allen Seiten. Da alle Tests ab einem bestimmten Umfang nicht mehr an einem Test-Tag durchgeführt werden konnten, gehörte es zur Aufgabe des Testmanagers aus den vorhandenen Tests diejenigen auszusuchen, welche die Fehlerbeseitigungen und neuen Features im geplanten Release am besten abdeckten und im Zeitrahmen des Test-Tags durchführbar waren.

In den Testbeschreibungen waren für die verschiedenen Teilnehmer Pseudonyme vorgesehen, die ich im Folgenden als "ABC-Namen" bezeichnen möchte, da die Anfangsbuchstaben dem Alphabet folgen: Alice, Bob, Carl, Dave, und Edna. Das Deckblatt sah vor, dass ein Testteilnehmer einen dieser Namen auf dem Deckblatt für alle Testfälle auswählte.

Am Ende des Test-Tags sammelte der Testmanager alle ausgedruckten und ausgefüllten Formulare ein, und erstellte eine Zusammenfassung der Ergebnisse und gefundener Fehler für die *Saros*-Mailingliste. Seine Aufgabe war es ausserdem die Bugtracker-Einträge zu aktualisieren.

In der Praxis wurden die Namen teilweise je Testfall neu zugeordnet, da einige Testfälle es auch erlaubten bei genügend anwesenden Testern parallel von Untergruppen ausgeführt zu werden. Dann musste im Notizfeld angegeben werden, welche Rolle aus der Beschreibung der Tester besetzte. Der Testmanager musste also bei der Zusammenfassung darauf achten, dass für bestimmte Testfälle beispielsweise jemand der laut Deckblatt und Log-Dateien Dave hiess, tatsächlich die Rolle von Alice innehatte.

Beim Test für Saros 9.10.2 entschieden Eike (*TM*) und ich (*ATM*) die Testfälle nicht mehr auszudrucken. Die Tester sollten die ausgewählten Dokumente am Bildschirm lesen und die Notizen zu den einzelnen Testfällen unter Angabe von Testfall (Abschnittsnummer und Überschrift), Rolle beziehungsweise Pseudonym, und Uhrzeit in einer einfachen Textdatei erfassen. Die Vergangenheit zeigte, dass die Ausdrucke eine Papierverschwendung darstellten. Im Büro des *Saros*-Teams sammelten sich die Ausdrucke der vergangenen Tests im Regal, und da die Testfälle doppelseitig gedruckt waren, konnte man die Rückseiten auch nicht als Schmierpapier verwenden. Zusätzlich gab es einen Anteil an Seiten mit Testfällen ohne handschriftliche Notizen, denn diese wären dort nur im Falle von Regressionen erforderlich gewesen. Es zeigte sich auch, dass handschriftliche, eilig niedergeschriebene Kurznotizen von verschiedenen Personen für den Testmanager beim Schreiben der Zusammenfassung manchmal schwer zu entziffern und interpretieren waren. Neben der Vermeidung dieser beiden Nachteile gab es bei der Niederschrift am Rechner in eine Textdatei noch den Vorteil, dass die Tester bei auftretenden Defekten einen kurzen Blick in die Log-Datei von *Saros* werfen konnten und auffällige Passagen oder gar Fehlermeldungen inklusive Stapelabzügen direkt zu ihren Beobachtungen in die Textdatei kopieren konnten.

4.5.2 Kritik am Testverfahren

Bei den Testfällen in Word-Dokumenten gab es sowohl technische als auch inhaltliche Probleme. Technisch ist das Dateiformat, und dort insbesondere die Wahl von Tabellen als Gestaltungsmittel nicht robust. Man kann diese Dateien nur mit der Anwendung bearbeiten, mit der sie auch erstellt wurden, wenn man sicher sein will, dass das Layout intakt bleibt. Damit waren diese Dokumente also an die Anwendung *Word* gebunden und zum Beispiel nicht mit *OpenOffice.org*[3] oder einer anderen freien Textverarbeitung nutzbar. Obwohl die Dateien aus diesem Grund nur mit *Word* bearbeitet wurden, gab es trotzdem Probleme. Bei der Aktualisierung der Testfälle für *Saros* 9.10.2 waren zum Beispiel alle Überschriften nach dem Öffnen der Dateien sehr stark nach rechts gerückt und wegen des geringeren Platzes auf mehrere Zeilen umgebrochen. Das wiederum führte dazu, dass sich der Rest des Seiteninhalts nach unten verschob und damit die Tabelle mit den bekannten Problemen bei dem jeweiligen Testfall nicht mehr auf die Seite passte und auf die Folgeseite gedrängt wurde. Die Ursache war ein Tabulator mit 15 Zentimeter Einzug in der Formatvorlage für Überschriften, den niemand dort explizit gesetzt hatte. Der war einfach und unerwartet da! Ich vermute, dass es daran lag, dass unterschiedliche Versionen von *Word* zum Einsatz kamen.

Inhaltlich war es unvorteilhaft, dass nur die Schritte für einen Testfall beschrieben waren, aber nicht wie das erwartete Ergebnis auszusehen hatte. Es lag bei den einzelnen Testern, dieses aus der Beschreibung der Schritte und ihrem Kenntnisstand über die gewünschte Funktionalität von *Saros* abzuleiten.

Die Tabelle mit den bekannten Problemen pro Testfall war oft unverständlich wenn der Tester den Defekt, der sich hinter der angegebenen ID und der Überschrift verbarg, nicht kannte. Für einen Systemtest haben viele Bugtracker-Einträge auch einen zu hohen Detailgrad. Wenn ein Eintrag das Problem auf der Ebene der Implementierung beschreibt, ist es bei einem Test aus Anwendersicht nicht immer möglich zu entscheiden, ob das Problem aufgetreten ist, oder nicht vielleicht ein anderes, welches zu vergleichbaren Symptomen führt. Umgekehrt lässt sich bei Abwesenheit des Symptoms nicht immer entscheiden ob das Problem behoben wurde, wenn man aus Anwendersicht nicht den genauen Programmfluss kennt und damit weiss, dass alle Ausführungspfade die zu dem Problem führen können, auch wirklich vom Test abgedeckt wurden. Mit der “fixed?”-Ankreuzspalte hat der Testfallbogen also etwas vom Tester abgefragt, was Softwaretests nicht leisten können und auch nicht sollen: Den “Beweis” das Fehler *nicht* vorhanden sind.

4.5.3 Testlink

Viele *Saros*-Team-Mitglieder haben das Testverfahren kritisiert, aber der Leidensdruck war nicht gross genug, dass etwas daran verändert wurde. Es hätte eine Alternative

[3] OpenOffice.org: `http://www.openoffice.org/`

evaluiert werden müssen, und alle Testfälle hätten in ein neues Verfahren übertragen werden müssen. Zu dieser Aufgabe hatte niemand Zeit. Die Situation änderte sich mit Karl Beecher als neuem technischen Projektleiter. Die Testfälle mussten alle von Deutsch nach Englisch übersetzt werden, dass heisst es mussten auf jeden Fall alle Testfälle überarbeitet werden. Da bot es sich an, sie auch gleich in ein neues Verfahren zu übernehmen. Karl evaluierte die webbasierte Software *TestLink*[4] und teilte jedem *Saros*-Mitglied ein Kontingent an Testfällen aus den *Word*-Dokumenten zur Übertragung und Übersetzung in das von ihm aufgesetzte *TestLink*-System zu.

In *TestLink* werden Testfälle in Test-Suiten erfasst, die hierarchisch verschachtelt werden können. Zu einer Test-Suite gehört ein Name und eine Beschreibung. Zusätzlich lassen sich Dateien an eine Test-Suite anhängen, zum Beispiel mit Testdaten, die für die enthaltenden Testfälle benötigt werden.

Ein einzelner Testfall wird über eine eindeutige ID identifiziert, die aus einem Kürzel besteht, welches in den Projekteinstellungen hinterlegt ist, und einer fortlaufenden Nummer, die von *TestLink* generiert wird. Genau wie eine Test-Suite hat ein einzelner Testfall auch eine Überschrift und eine Beschreibung und es lassen sich ebenfalls Dateien anhängen. Bei den *Saros*-Testfällen wäre da zum Beispiel eine Textdatei mit dem Inhalt einer englischsprachigen Ausgabe von Leo Tolstoy's *Krieg und Frieden* [Tol01] von *Project Gutenberg*[5], die für einige Belastungstest als grosse ASCII-kodierte Textdatei verwendet wurde.[6] Für die spätere Durchführung eines Testfalls sind zwei weitere Angaben wichtig: Die Beschreibung der Schritte, die durchgeführt werden müssen, und eine Beschreibung des erwarteten Ergebnisses.

Aus den Testfällen einer Test-Suite lassen sich Testpläne zusammenstellen, welche die Grundlage für konkrete Testdurchführungen bilden. Ein Tester führt einen Testplan für eine Version der Software durch. Dabei werden die Testfälle mit einem zusätzlichen Freitextfeld für Notizen und einem Optionsfeld für ein formales Ergebnis des Tests. Die Auswahl besteht aus *Not Run*, *Pass*, *Fail*, und *Blocked*. Die ersten drei Wahlmöglichkeiten sollten selbsterklärend sein. *Blocked* ist für Tests gedacht, die nicht ausgeführt werden konnten, beispielsweise ein Test, der statt eines lokalen Jabber-Servers einen Öffentlichen vorschreibt, es aber während der Testdurchführung keine Netzverbindung gibt oder der öffentliche Jabber-Server gerade nicht läuft.

Die alten Testfälle wurden bei der Übertragung in *TestLink* nach Einzeldokumenten in Test-Suiten zusammengefasst und die Zusammenstellung eines Testplans entsprach dem Aussuchen der Testfälle im alten Verfahren. Die Struktur des alten Verfahrens liess sich gut auf *TestLink* abbilden. Ein Mehrwert ist die Niederschrift des erwarteten Ergebnisses pro Testfall, etwas was sich vorher jeder einzelne Tester bei jeder Testdurchführung erneut erarbeiten musste.

Verlorengegangen sind bei der Übernahme der Testfälle die Listen mit den bekannten

[4] TestLink: `http://www.teamst.org/`

[5] Project Gutenberg: `http://www.gutenberg.org/`

[6] "Gross" bedeutet hier in Zahlen 3,1 MiB aufgeteilt auf 65.336 Zeilen.

Problemen je Testfall. Da diese Problembeschreibungen von Testern, die mit den konkreten Defekten nicht vertraut waren, nicht gut eingeschätzt oder interpretiert werden konnten, ist das meiner Meinung nach kein grosser Verlust. *TestLink* eröffnet hier sogar die Möglichkeit auf Basis der Bugtracker-Einträge eigenständige Testfälle zu erstellen und so neben den Systemtests auch an der Implementierung orientierte Tests mit *TestLink* zu erfassen und abzudecken.

4.5.4 Testdurchführung II

Beim ersten Einsatz von *TestLink* am Test-Tag für *Saros* 9.12.4 führten alle Tester parallel den gleichen Testplan aus, das heisst jeder einzelne Tester hatte einen Webbrowser mit dem Testplan geöffnet und jeder hat individuell zu jedem Testfall das Ergebnis in *TestLink* eingetragen, welches er lokal bei sich beobachten konnte.

In den Ergebnissen von *TestLink*-Testplänen werden die einzelnen ausgeführten Testfälle farblich nach dem Ergebnis des letzten Eintrags zu einem Testfall im Testplan markiert – Grün für bestanden (“pass”), Rot für durchgefallen (“fail”) und Blau für blockiert (“blocked”). Bei dem Verfahren des *Saros*-Teams arbeiten mehrere Personen den gleichen Testfall ab und können dabei zu *unterschiedlichen* Ergebnissen kommen. Es kann also zum Beispiel bei vier Testern und einem Einladungsvorgang, bei dem einer die anderen Drei einlädt, passieren, dass bei einem der Vier nach der Einladung ein Fehler auftritt. Bei dem Testfall hätte *Saros* demnach dreimal bestanden und wäre einmal durchgefallen. Wie das auf der zusammenfassenden Ergebnisseite von *TestLink* dargestellt wird, hängt von der Reihenfolge ab, in der die Tester ihre Beobachtungen in das System speichern. Wenn also sinnvollerweise dieser Test Rot dargestellt werden soll, denn es besteht Handlungsbedarf dort etwas an *Saros* zu verbessern, muss sichergestellt werden, dass derjenige, bei dem die Einladung nicht reibungslos funktionierte, sein Votum zu dem Testfall gegenüber *TestLink* als Letzter abgibt. Dazu müsste man die Einträge während des Tests entsprechend koordinieren, oder wie von mir nach dem Test vorgeschlagen wurde, die Ergebnisse von jedem Tester wie bisher in einer Textdatei protokollieren lassen, und den Testmanager nach den Tests einen konsolidierten Eintrag pro Testfall in *TestLink* eintragen lassen.

4.6 Review-Prozess

Der Patch-Review-Prozess, also eine Quelltextdurchsicht der Änderungen, ist eine Qualitätssicherungsmassnahme, welche die Entwicklung ständig begleitete. Änderungen der *Saros*-Entwickler durften nicht unbegutachtet in den Hauptzweig im *Subversion*-Repository des Projekts einfliessen, sondern mussten vorher von anderen *Saros*-Entwicklern durchgesehen und für gut befunden werden. Vor dem Weggang von Christopher Oezbek gab es schon die Regelung, dass Änderungen in Form von Patches an die

Saros-Mailingliste geschickt werden mussten, und dort jeder gegen einen Patch ein Veto einlegen konnte oder auch ein positives Urteil abgeben konnte. In den Hauptzweig übernommen werden durften Patches, die kein Veto bekamen, oder innerhalb der ersten 24 Stunden nach Veröffentlichung mindestens einmal positiv bewertet wurden. Nach Ablauf der 24 Stunden durfte die Quelltextänderung auch ohne Durchsicht "committed" werden. Da Christopher ständig einen wachsamen Blick auf die "Commits" hatte, ging fast nichts unbeobachtet in das *Subversion*-Repository ein. Nach seinem Weggang musste sein Augenpaar ersetzt werden, und da niemand die gesamte Quelltextbasis so gut überblickte wie er, insbesondere niemand der nicht nur für den Zeitraum einer X-Arbeit zur Verfügung stand, wurde diese Aufgabe allen *Saros*-Entwicklern aufgetragen. Die Regeln für die Patch-Durchsichten wurden verschärft. Es gab drei offizielle numerische Bewertungen:

-1 stellt ein Veto dar und muss begründet werden. Der Patch muss überarbeitet werden, indem die Gründe für den Einspruch beseitigt werden. Das Veto kann nicht durch gute Bewertungen aufgewogen werden.

+1 bedeutet, dass der Patch sowohl in den Details für gut befunden wurde, als sich nach Ansicht des Gutachters gut in das Gesamtbild einfügt, das heisst auf Ebene der Architektur und der Geschäftslogik sinnvoll ist.

+0 ist eine positive Bewertung, allerdings ist sich der Gutachter nicht sicher was die Geschäftslogik angeht, zum Beispiel weil er sich in dem veränderten Teil des Programms nicht auskennt – mit den grösseren Zusammenhängen und Abläufen dort nicht vertraut ist.

In der Praxis kamen auch "+1"-Bewertungen vor, die an Bedingungen geknüpft waren. Beispielsweise so etwas wie *"+1 if you correct this potential NPE: ..."*.[7], was streng genommen ein Veto hätte sein müssen. Die Bedingungen waren in solchen Fällen in der Regel kleine Änderungen, von denen der Gutachter ausging, dass seine Vorschläge unstrittig sind. Also zum Beispiel hinweise auf Tippfehler in Kommentaren, oder eindeutige und leicht zu behebende Verletzungen der Code-Richtlinien.

Wenn kein Veto eingelegt wurde, brauchte ein Patch innerhalb der ersten 48 Stunden mindestens zwei Bewertungen mit "+1" um in den Hauptentwicklungszweig eingebracht werden zu dürfen. Danach reichte ein "+1" *und* mindestens ein "+0" aus.

Es war erlaubt die Durchsicht mit dem Autoren des Patches gemeinsam durchzuführen. Dabei sollte der Autor zwar seine Änderungen erklären, aber nicht selbst durch den Quelltext navigieren. Damit der Autor nicht zu schnell durch seinen Quelltext führt und den Gutachter dabei eventuell "abhängt", beziehungsweise dem Gutachter keine Details entgehen, sollte der Gutachter die Geschwindigkeit vorgeben. So ein *pair review* machte vor allem Sinn, wenn sich niemand fand, der mehr als ein "+0" vergeben konnte oder wollte, weil nur der Autor der Änderung sich in dem betroffenen Teil des Quelltextes wirklich gut auskannte. Diese *pair reviews* sind also nicht nur eine Qua-

[7] Aus einer Antwort von Tas Sóti auf einen Patch von Sandor Szücs auf der *dpp-devel*-Mailingliste.

litätssicherungsmassnahme, sondern können auch den Wissenstransfer zwischen den Entwicklern fördern.

4.6.1 Überwachung des Review-Prozesses

Es fiel in meinen Aufgabenbereich den Review-Prozess zu überwachen und falls nötig steuernd einzugreifen, nachdem Christopher Oezbek die Arbeitsgruppe verlassen hatte.

Um einen besseren Überblick über die offenen, noch nicht begutachteten Patches zu haben, entschied ich als erstes, dass die Patches nicht mehr an die *Saros*-Mailingliste geschickt werden, sondern über die bisher ungenutzte Mailingliste *dpp-devel*[8] auf *SourceForge* laufen. Da diese Liste öffentlich ist, und auch in Erwartung von Karl Beecher als künftigem *Saros*-Team-Mitglied, änderte sich die Sprache der Beschreibung und die Bewertungen der Patches von Deutsch auf Englisch.

Für die E-Mails mit den Patches und Bewertungen erstellte ich mir auf dem Mailserver einen eigenen Ordner und einen Filter in den E-Mail-Programmen, die ich verwendete, welcher die E-Mails automatisch beim Abrufen der eingehenden Mails in diesen Ordner verschob und als "wichtig" kennzeichnete. Diese Kennzeichnung gehört zum *IMAP4*-Protokoll [Cri03] und wird dort als *"flagged"* bezeichnet und durch *"Message is 'flagged' for urgent/special attention"* beschrieben. Die Kennzeichnung ist, da auf dem Server gespeichert, unabhängig vom verwendeten E-Mail-Programm. E-Mail-Nachrichten in dem Ordner liess ich solange als "wichtig" gekennzeichnet, bis der dazugehörige Patch vollständig durch den Review-Prozess begleitet wurde. Die "Threading"-Ansicht der E-Mail-Programme half dabei die E-Mails, die sich auf einen Patch bezogen, zu gruppieren. Als nützlich hat sich auch eine Erweiterung für das E-Mail-Programm *Thunderbird*[9] erwiesen, die 'Patches in E-Mails für die Anzeige mit einem Syntax-Highlighting versieht.[10]

Nachdem in den ersten vier Wochen nach dem Umzug der Patches auf die *dpp-devel*-Mailingliste insgesamt drei E-Mails nur dazu dienten auf je einen Patch aufmerksam zu machen, der noch nicht begutachtet wurde, entschloss ich mich aktiver in den Review-Prozess einzugreifen. Dazu suchte ich mir regelmässig den ältesten, noch offenen Patch aus der Liste, und sprach die im Büro anwesenden *Saros*-Entwickler darauf an. Falls der Autor des Patches anwesend war und die Änderungen durch den Patch nicht trivial waren, regte ich ein *pair review* an. Als Zeitpunkt um einen Patch anzusprechen eignen sich gemeinsame Mittagspausen – damit ist sichergestellt, dass durch die Anfrage keine wichtige Arbeit unterbrochen wird, oder ein Entwickler aus dem "hack mode" [Ray96] gerissen wird.

[8] dpp-devel-Mailingliste: `https://lists.sourceforge.net/lists/listinfo/dpp-devel`
[9] Thunderbird: `http://www.mozillamessaging.com/`
[10] *Colored Diffs* Erweiterung für *Thunderbird*: `https://addons.mozilla.org/en-US/thunderbird/addon/4268`

4.6.2 Probleme und Kritik am Review-Prozess

Es gab kleinere technische Probleme beim Review-Prozess. Ein Umbenennen von Klassen zieht eine Umbenennung der betreffenden Java-Quelltextdatei nach sich. Das wird von der Versionsverwaltung *Subversion* speziell unterstützt, damit die aufgezeichnete Änderungsgeschichte der alten Datei auch unter dem neuen Namen erhalten bleibt. Wenn man sich allerdings ein Diff von so einer Umbenennung erzeugen lässt, dann wird die Datei unter dem alten Namen als gelöscht angeführt, der Inhalt aber nicht unter dem neuen Namen wieder hinzugefügt. Wenn die Datei also nicht nur umbenannt wurde, sondern auch Änderungen am Quelltext der Klasse durchgeführt wurden, sind diese Änderungen nicht im Diff ersichtlich. Dieses Verhalten zeigte sich nicht nur über die GUI, welche von *Eclipse* für *Subversion*-Operationen zur Verfügung gestellt wird, sondern auch ausserhalb von *Eclipse*, wenn *Subversion* direkt auf der Kommandozeile mit der Anweisung `svn diff` verwendet wird.

Unterschiedliche E-Mail-Programme bei den Beteiligten, können auch zu kleineren Hürden führen. Die beiden Programme, die ich verwendete – *KMail* und *Thunderbird* – können zu einer E-Mail hinzugefügte Textdateien "inline" im Textkörper der E-Mail anzeigen, und gleichzeitig bei den Anlagen aufführen, so dass man sie dort leicht auswählen und unter dem vorgeschlagenen Namen speichern kann. E-Mails werden intern üblicherweise nach dem MIME-Format strukturiert, und können dort aus mehreren Teilen bestehen [FB96]. Zu jedem Teil kann zu den enthaltenden Daten vermerkt werden, ob sie "inline", also im Textkörper der E-Mail angezeigt werden sollen, oder ob es sich um eine Anlage handelt [TDM97]. In beiden Fällen kann auch ein Dateinamen vorgeschlagen werden, falls der Anwender des E-Mail-Programms den entsprechenden Teil der E-Mail separat speichern möchte. Wenn für einen Teil "inline" *und* ein Dateiname vermerkt sind, dann zeigen einige E-Mail-Programme den Inhalt im Textkörper *und* unter dem entsprechenden Namen in der Liste der Anlagen. Andere führen den Teil nicht in den Anlagen auf. Dort muss man sich den Textteil aus dem Textkörper der E-Mail kopieren. Im *Saros*-Team betraf das beispielsweise die Benutzer von *Apple Mail*.

Tas Sóti fand den Review-Prozess über eine Mailingliste zu unübersichtlich und äusserte Bedenken, dass sich der Lebenszyklus von Patches auf der Liste schlecht verfolgen lässt.[11] Er schlug als Alternative vor, einen zusätzlichen Tracker für Patch-Durchsichten einzurichten, neben den bereits vorhandenen für Bug-Meldungen und Feature-Wünsche. Sandor Szücs argumentierte dagegen, dass sich Tracker nicht so gut zum Diskutieren eignen wie Mailinglisten. Insbesondere wenn die Diskussionen umfangreicher sind, und in mehreren Argumentationssträngen verlaufen. Aus seiner Sicht ist das Hauptproblem – das die Entwickler die Patches möglichst zügig begutachten sollten – nicht durch einen Patch-Tracker lösbar.

[11] Die Diskussion lässt sich ab der E-Mail mit dem Betreff *"[Dpp-devel] Review process in general + Patch: XMPPChatTransmitter&Cancellation"* von 2009-11-10 auf der *dpp-devel*-Mailingliste nachlesen.

Eine Liste von Werkzeugen speziell zur technischen Unterstützung von Durchsichten lieferte Tas bereits in seiner ersten E-Mail zu dem Thema. Das waren namentlich *Jupiter*[12], *ReviewClipse*[13], und *SmartBear*[14]. Letzteres ist, im Gegensatz zu den anderen beiden, kommerzielle Software und auch kein *Eclipse*-Plugin, sondern ein webbasiertes Werkzeug. Seine Zeitplanung liess keine Evaluierung dieser Werkzeuge zu.

SourceForge bietet *Codestriker*[15] als "hosted app" zur Unterstützung von Patch-Durchsichten für Projekte. Diese Software habe ich evaluiert und bin zu dem Schluss gekommen, dass sie bei den Anforderungen vom *Saros*-Team an den Review-Prozess eher Nachteile gegenüber der Patch-Durchsicht über die Mailingliste aufweist.

Einzelne Patches zur Durchsicht, werden von *Codestriker* als "Topic" verwaltet [Sit08]. Ein Topic besteht aus einem Titel, Autor, und dem Patch der begutachtet werden soll. Zusätzlich hat ein Topic einen Status aus der Menge *Open*, *Closed*, *Commited*, und *Obsoleted*. In der Übersicht der Topics werden diese Informationen angezeigt. Nicht sichbar ist dort, wieviele Durchsichten für die jeweiligen Topics es schon gibt und wie diese ausgefallen sind. Im Umkehrschluss lässt sich also aus der Übersicht nicht erkennen, welche Patches noch begutachtet werden müssen. Dafür müsste man in jedes Topic einzeln "hineinschauen". Das ist meiner Meinung nach ein Ausschlusskriterium für dieses Werkzeug für den Einsatz im Entwicklungsprozess beim *Saros*-Projekt.

Bei einer Durchsicht kann der Gutachter Kommentare zu einzelnen Zeilen des Patches in einem Topic abgeben und andere Gutachter oder der Autor des Patches können diesen Kommentaren weitere hinzufügen. Diese Handhabung über eine Webschnittstelle erschien mir deutlich umständlicher als in einer E-Mail einen Teil eines Patches zu zitieren und darunter einen Kommentar zu schreiben, auf den dann andere Gutachter oder der Autor antworten können.

Da mir *Codestriker* nicht als geeignet erschien und sich sonst kein *Saros*-Entwickler zu dem Thema auf der Mailingliste zu Wort meldete, beschloss ich vorerst die Patch-Durchsichten weiter über die *dpp-devel*-Mailingliste abzuwickeln.

Anstatt mit regelmässigen, kleineren Patches gegen den Hauptzweig im *Subversion*-Repository zu arbeiten, setzte sich im Laufe der Zeit bei den *Saros*-Entwicklern immer mehr das Anlegen von Entwicklungszweigen für grössere Features durch. Für den Umgang mit Durchsichten vor der Übernahme solcher umfangreicheren Änderungen gibt es noch wenig Erfahrungswerte im *Saros*-Team.

[12] Jupiter: `http://code.google.com/p/jupiter-eclipse-plugin/`
[13] ReviewClipse: `http://www.inso.tuwien.ac.at/projects/reviewclipse/`
[14] SmartBear: `http://smartbear.com/`
[15] Codestriker: `http://codestriker.sourceforge.net/`

5 Implementierung

In den Folgenden Abschnitten werden zwei ausgewählte Implementierungen detaillierter beschrieben – der Umgang von *Saros* mit Zeilenumbrüchen in Abschnitt 5.1 auf Seite 61 und die Anzeige von Jabber-Präsenzinformation in Abschnitt 5.2 auf Seite 71.

Die wichtigsten Implementierungsarbeiten ausser den beiden eben genannten, werden in der folgenden Aufzählung zusammengefasst. Jeweils mit ID und Titel aus dem Bugtracker und einer kurzen Beschreibung von Problem und Lösung.

- *#2832593 Provide Drop-In Zip*

  In *Teles*-Sitzung #14 konnte Entwickler #2 am Standort in Indien das *Saros*-Plugin nicht aktualisieren, da das lokale Netzwerk keinen Zugriff auf *SourceForge* erlaubte. Daraus ergab sich die Anforderung eine Alternative Form der Aktualisierung beziehungsweise Installation zu ermöglichen. *Eclipse*-Plugins lassen sich auch als sogenannte "Drop-Ins" ausliefern. Das sind ZIP-Archive, die in einen bestimmten Ordner innerhalb der *Eclipse*-Verzeichnisstruktur abgelegt werden. So ein Archiv habe ich erstellt und den Vorgang im Entwicklerhandbuch [ST10] dokumentiert. Die Erstellung eines "Drop-In"-Archivs gehört seitdem zu den Aufgaben des Releasemanagers bei jedem Release.

- *#2256164 Problems with different character encodings*

  Verwandt mit der in Abschnitt 5.1 auf Seite 61 beschriebenen Problematik mit der Kodierung von Zeilenumbrüchen ist die Kodierung des Zeichensatzes. Innerhalb von Java-Programmen werden Zeichenketten als Unicode repräsentiert. Auf dem Hintergrundspeicher werden allerdings keine Zeichen, sondern Bytes gespeichert, das heisst die Unicode-Zeichenketten müssen mittels einer Kodierung in Folgen von Bytewerten überführt werden. Wenn dafür bei den Beteiligten nicht die selbe Kodierung verwendet wird, wertet die Konsistenzüberwachung die unterschiedlichen Bytewerte als Inkonsistenz, da die Prüfsummen über die Bytewerte auf dem Hintergrundspeicher gebildet werden.

  Die Lösung entspricht auch der bei der Kodierung der Zeilenumbrüche – die Einstellung der Zeichensatzkodierung des Host wird bei der Einladung an die Clients übertragen.

- *#2849925 Contribution annotation doesn't work with PDT*

  Die Contribution-Annotation funktionierte nicht bei Editoren, die vom *PHP De-*

velopment Tools-Plugin[1] zur Verfügung gestellt wurden. Alena Kiwitt war zu dem Zeitpunkt, an dem dieser Defekt entdeckt wurde noch nicht im *Saros*-Team. Da sich ihre Arbeit [Kiw10] mit der Kompatibilität von anderen *Eclipse*-Plugins und *Saros* befasst, hätte ich diese Aufgabe sonst delegiert.

Die Ursache war die Art und Weise wie *Saros* das zuständige Objekt für die Annotationen für Contribution-Annotationen ermittelte. Das geht über ein Dateiobjekt für die Datei oder über einen offenen Editor in *Eclipse*. Bei den Editoren, welche die Plugins für Java- und C/C++-Unterstützung zur Verfügung stellen, führen beide Wege zum selben *IAnnotationModel*-Exemplar. Bei PDT-Editoren bekommt man jeweils ein anderes, und nur das über den Editor kann für Annotationen für diesen Editor verwendet werden.

- *#2847756 Selecting a user in session view doesn't work in whole line*

  Die Auswahl einer Zeile im *Shared Project Session*-View mittels Mausklick funktionierte nicht im gesamten Bereich der Zeile, sondern nur über dem Bereich wo Text dargestellt wurde. Zumindest war das unter Windows der Fall – unter Linux und Mac-OS reagierte die GUI an dieser Stelle wie von Entwickler #1 bei *Teles* erwartet. Aufgefallen ist dieser Defekt in *Teles*-Sitzung #15. Siehe Abschnitt 3.6.4 auf Seite 32.

  Die Lösung war sehr einfach. Beim "Style"-Argument der Tabelle musste nur die Konstante *SWT.FULL_SELECTION* verwendet werden.

- *#2869752 Viewport follows refactoring*

  Eclipse beziehungsweise das JDT-Plugin stellt umfangreiche Refaktorisierungsmöglichkeiten zur Verfügung. Es lassen sich mit wenigen Mausklicks oder Tastenanschlägen weitreichende Veränderungen an Quelltexten vornehmen. Zum Beispiel das Umbenennen einer Methode oder eines Feldes, bei dem alle betroffenen Quelltextstellen im Projekt geändert werden. Diese Änderungen werden als *TextEditActivity*s von *Saros* übertragen, die nicht von manuell ausgelösten Aktivitäten unterschieden werden. Bei einer *TextEditActivity* nimmt *Saros* immer an, dass die Schreibmarke des Verursachers nach der Anwendung hinter dem betroffenen Text stehen muss. So wird eine unruhige Cursor-Annotation vermieden, die durch die unterschiedlichen Übertragungswege entstehen kann. Textänderungen in ge"share"ten Dokumenten gehen den Umweg über den Host, während die Position der Schreibmarke direkt an andere Teilnehmer verschickt wird. Wenn sich ein Teilnehmer im Verfolgermodus befindet, folgt sein Sichtbarkeitsbereich auch den Textänderungen, die durch die automatisierte Refaktorisierungen erzeugt wurden.

  Damit sich der Sichtbarkeitsbereich von Verfolgern nicht durch solche "synthetischen" Textänderungen verändert, werden bei der Aktualisierung des Sichtbar-

[1] PDT: `http://www.eclipse.org/pdt/`

Bytewerte	System
13	Commodore 8-Bit-Rechner, Apple II, Mac OS bis Version 9
10	Unix, Linux, Mac OS X, AmigaOS, BSD
13, 10	CP/M, DOS, Windows, Atari TOS
155	Atari 8-Bit-Rechner [Cha85]

Tabelle 5.1: Verschiedene Kodierungen von Zeilenumbrüchen für unterschiedliche Computersysteme. Die Werte der Bytes sind in dezimaler Schreibweise angegeben.

keitsbereichs durch *TextEditActivity*s nur noch solche berücksichtigt, die sich innerhalb des Sichtbarkeitsbereichs des Verfolgers befinden.

- *#2871841 Renaming contact in roster does not always work*

  Das Umbenennen von Kontakten im *Roster*-View funktionierte nicht immer beim ersten Versuch. Nachdem die in Abschnitt 5.2 auf Seite 71 beschriebene Anzeige der An- beziehungsweise Abwesenheit implementiert wurde, wirkte sich ein Umbenennen manchmal nur für eine der beiden Zustände aus. Für den Benutzer äusserte sich das zum Beispiel in der Anzeige des alten Namens, wenn der Kontakt als Anwesend geführt wurde und bei Abwesenheit wurde der neue Name angezeigt.

  Ursache war die direkte Verwendung von *RosterEntry*-Objekten aus der *Smack*-Bibliothek als Daten für die Anzeige im *TreeView*. Die Klasse *RosterEntry* implementiert zwar die Methode *equals()* aber nicht *hashCode()*. Sie bricht den Vertrag, der in der Java API Dokumentation für *java.lang.Object* beschrieben wird, dass Objekte, die laut *equals()*-Methode gleich sind, auch den gleichen Hash-Wert haben müssen. Sonst kann man Exemplare der Klasse zum Beispiel nicht in Datenstrukturen verwenden, die auf Hash-Tabellen basieren.

  Die *RosterEntry*-Objekte wurden von mir in eine Klasse “verpackt”, die den oben angesprochenen Vertrag erfüllt. Ausserdem habe ich den Quelltext für den *Roster*-View “objektorientierter” gestaltet, in dem es für jede Art von Element in der angezeigten Baumstruktur eine Klasse gibt. Vorher gab es mehrere Methoden, die mittels **instanceof**-Vergleichen Entscheidungen getroffen haben. Nun liegt die jeweilige Teilverantwortung in den einzelnen Klassen.

5.1 Umwandlung der Kodierung von Zeilenumbrüchen

Zeilenumbrüche in ge“share”ten Textdateien wurden von *Saros* grundsätzlich beim Öffnen der Datei in die Kodierung nach Unix-Konvention umgewandelt. Diese Manipulation geschah noch vor dem Laden des Dateiinhalts in den Editor und veränderte die Datei auf dem Hintergrundspeicher. Durch die Umkodierung wurden auf einfache Weise Probleme mit der Konsistenzüberwachung vermieden. Die Konsistenzüber-

wachung würde sonst bei unterschiedlich kodierten Zeilenumbrüchen, aber ansonsten gleichem Dateiinhalt, eine Inkonsistenz melden. Wenn die verwendeten Kodierungen auf beiden Seiten nicht die gleiche Länge in Zeichen hat, gäbe es ausserdem Probleme mit jenen Aktivitäten, die sich auf eine Position in einem Dokument beziehen. Dieses Problem ist darauf zurückzuführen, dass diese Information als Versatz in Zeichen vom Dokumentanfang in den Nachrichten beschrieben wird. Durch die Wahl einer Kodierung, die nur aus einem Byte/Zeichen besteht, wurde desweiteren sichergestellt, dass das Einfügen oder das Löschen von Zeilenumbrüchen, auch bei nebenläufiger Bearbeitung, eine atomare Operation ist; also keine anderen Zeichen die Markierung des Zeilenumbruchs spalten können. Verschiedene betriebssystemspezifische Konventionen für die Kodierung von Zeilenumbrüchen sind in Tabelle 5.1 auf der vorherigen Seite aufgeführt.

Problematisch ist die Umwandlung für Benutzer, die eine andere Art von Kodierung für Zeilenumbrüche und eine Versionsverwaltung, wie zum Beispiel *Subversion* verwenden.[2] Bei den *Teles*-Sitzungen zwischen Wien und Bangalore ist dies der Fall, da alle Beteiligten Windows verwenden.[3] Die Problematik wurde von Entwickler #1 in der Teles-Sitzung #13 (siehe Abschnitt 3.6.5 auf Seite 33) angesprochen und ging als *#2828917 Unwanted automatic conversion of line delimiters* in den Bugtracker ein.

Aus Sicht der Versionsverwaltung hat sich bei *Teles* nämlich jede während der Sitzung geöffnete Datei verändert, und dass in *jeder* Zeile, auch wenn dort von den Benutzern gar keine Veränderungen durchgeführt worden sind. Als manuelle Umgehungslösung kann der Benutzer vor dem Commit der tatsächlich, willentlich durchgeführten Änderungen, bei allen, beziehungsweise mindestens den während der *Saros*-Sitzung geöffneten Dateien, die Zeilenumbrüche von *Eclipse* wieder in die Voreinstellung seiner Plattform beziehungsweise des Projekts umwandeln lassen.[4] Das ist einerseits sehr lästig und wird dieser Schritt vergessen, dann ergeben sich negative Folgen. Die veränderten Zeilenumbrüche "verdecken" dann die relevanten Änderungen im Quelltext bei der Anzeige von Dateiunterschieden in Eclipse[5] und die ganzen inhaltlich unveränderten Zeilen werden in der Versionsverwaltung als Änderungen aufgezeichnet.

Lösungsansatz

Behoben werden kann das Problem dadurch, dass überall die Einstellung der Zeilenumbruchkodierung des Hosts verwendet wird. Das passierte bereits während des Einladungsprozesses beim Abgleich der Dateien, da dort Dateien, welche bis auf die Kodierung der Zeilenumbrüche identisch sind, durch die Prüfsummen über die Byte-

[2] Das gilt auch für die in *Eclipse* integrierte, lokale Versionierung und ebenfalls für alle Werkzeuge und Anwendungen, welche eine andere Kodierung von Zeilenumbrüchen als die Unix-Konvention erwarten.

[3] Ersichtlich aus den *Saros*-Logdateien von den *Teles*-Sitzungen.

[4] *Eclipse* Menüleiste → *File* → *Convert line delimiters to* → ...

[5] Das "Diff" von Eclipse 3.4 kennt keine Einstellung, um "whitespace" Zeichen zu ignorieren und hebt auch keine Unterschiede *innerhalb* von geänderten Zeilen visuell hervor.

werte auf dem Hintergrundspeicher als unterschiedlich erkannt werden, und deshalb die Dateien vom Host zum Client übertragen werden.

Neben den bereits vorhandenen Dateien muss die vom Host verwendete Kodierung von Zeilenumbrüchen aber auch bei allen Clients, welche die *Driver*-Rolle besitzen, in von ihnen neu angelegten Dateien und in Dateien, die vor dem Hinzufügen eines Zeilenumbruchs nicht bereits mindestens Einen enthalten, verwendet werden. Dazu muss vom Host bei der Einladung seine bevorzugte Kodierung von Zeilenumbrüchen an die anderen Teilnehmer übertragen und dort in den Projekteingenschaften von *Eclipse* eingetragen werden. Bei den eingeladenen Clients muss man sich die ursprünglich eingestellte, lokale Kodierung merken, um sie am Ende der *Saros*-Sitzung wiederherstellen zu können.

Dass ein Zeilenumbruch bei Verwendung der Windows-Konvention mit zweien, statt wie vorher nur einem Zeichen kodiert wird, hatte sowohl vorhersehbare als auch potentielle Auswirkungen auf den vorhandenen Code. Zum Einen gab es eine Stelle im Quelltext zum Überprüfen ob die Schreibmarke sich am Ende einer Zeile befindet, welche für die Anzeige der Cursor-Annotation[6] zuständig war und dabei fest von einem Zeichen für die Markierung eines Zeilenumbruchs ausging. Zum Anderen war nicht ganz klar, ob der Algorithmus zur Nebenläufigkeitskontrolle und die Weise, wie er in *Saros* eingebunden ist, mit der Veränderung zurecht kommt, und es nicht passieren kann, dass durch nebenläufige Aktionen die beiden Zeichen derart getrennt werden, dass ein weiteres Zeichen dazwischen eingefügt werden könnte.

Alternativen

Die oben beschriebene und umgesetzte Lösung ist immer noch ungünstig, wenn Teilnehmer Plattformen mit unterschiedlichen Konventionen bei der Kodierung von Zeilenumbrüchen verwenden und gleichzeitig Versionsverwaltungssoftware wie zum Beispiel *CVS*, welche bei Textdateien zwischen Repository und lokalen Daten automatisch die Zeilenumbrüche konvertiert. Dort wird dann in den Dateien die Konvention des Hosts verwendet und die oben genannten Probleme bestehen bei allen Teilnehmern, deren lokale Einstellungen auf Systemebene von denen des Hosts abweichen, und die veränderte Dateien committen wollen. In diesen Fällen muss auch weiterhin die bereits beschriebene Umgehungslösung angewendet werden. Für *Teles* stellt das kein Problem dar, weil dort keine Versionsverwaltungssoftware verwendet wird, die Zeilenumbrüche von sich aus konvertiert. Ausserdem arbeiten alle Teilnehmer auf der gleichen Plattform und deshalb auch mit der gleichen Konvention bei der Kodierung der Zeilenumbrüche.

Man könnte natürlich die Zeilenumbrüche von Saros bei der Übertragung transparent umkodieren lassen, wie das zum Beispiel *CVS* macht. Dann müsste allerdings ein viel

[6] Die Cursor-Annotation ist eine Markierung, welche die Position der Schreibmarke von entfernten Benutzern darstellt. Siehe auch Abschnitt 2.2.4 auf Seite 16 und Abbildung 2.1 auf Seite 11.

höherer Aufwand betrieben werden. Beispielsweise kann man beim Abgleich bei der Einladung dann nicht mehr einfach die Prüfsumme über den Dateiinhalt erstellen, sondern müsste bei jeder Datei entscheiden, ob es sich um eine Textdatei handelt, deren Zeilenumbrüche in die Prüfsumme in einer kanonischen Form eingehen muss. Diese Umwandlung der Zeilenumbrüche müsste dann auch immer beim Einlesen und Schreiben von Textdateien während einer Sitzung passieren, damit die Positionsangaben für Textoperationen für alle Teilnehmer gültig sind, da Zeilenumbrüche mit unterschiedlich vielen Zeichen kodiert werden können.

Eine Alternative wäre es, diese Positionsangaben von einem Versatz in Zeichen vom Dokumentanfang auf eine Zeile/Spalte-Angabe umzustellen, um diese Angabe unabhängig von der Wahl der Kodierung der Zeilenumbrüche zu machen. Das hat allerdings einen höheren Verwaltunsgaufwand zur Folge, weil diese Informationen für die Anwendung der Operationen dann wieder in den bisher verwendeten Versatz umgewandelt werden müssten. Trotzdem müsste man beim Erstellen von Prüfsummen über Dateiinhalte die Zeilenumbrüche noch in eine kanonische Form umwandeln.

Das Problem mit der Zeilenumbruchkodierung ist verwandt mit den Problemen bei unterschiedlichen Zeichensatzkodierungen bei den Teilnehmern und der Verwendung von Zeichen ausserhalb des ASCII-Zeichensatzes. Oder in ganz pathologischen Fällen auch dann, falls bei den unterschiedlichen Teilnehmern einer *Saros*-Sitzung Zeichensatzkodierungen verwendet werden, die nicht alle ASCII als Untermenge enthalten. Auch hier könnte man den Teilnehmern die Kodierung des Hosts aufzwingen, oder man müsste für die Prüfsummenberechnung eine kanonische Form der Texte verwenden und beim Übertragen von Texten in Aktivitäten transparent umkodieren.

Umsetzung

Für die Umsetzung habe ich als erstes geschaut, wie und wo die Einstellung der Kodierung der Zeilenumbrüche von *Eclipse* gespeichert wird, und wie die Kodierung ermittelt wird, wenn man eine neue Textdatei anlegt. Es muss sichergestellt werden, dass alle Teilnehmer einer *Saros*-Sitzung auf diesem Wege die selbe Kodierung für Zeilenumbrüche bekommen.

Die Stelle, an der die Hilfsfunktion *convertLineDelimiters()* bedingungslos für jede geöffnete Datei aufgerufen wurde, konnte entfernt werden; die Funktion selbst ist aber auf Bitte der technischen Projektleitung im Quelltext verblieben und nur mit einem Kommentar versehen, dass und seit wann sie nicht mehr verwendet wird.

Für die Darstellung der Cursor-Annotation muss ermittelt werden, ob die Schreibmarke eventuell am Ende einer nicht-leeren Zeile steht, damit in diesem Fall die Position der Annotation angepasst werden kann, so dass sie sichtbar bleibt. Diese Positionsüberprüfung ging davon aus, dass Zeilenumbrüche mit `'\n'` oder `'\r'` kodiert sind. Damit sind zwar momentan die Einsatzzwecke auf den gängigen Plattformen abgedeckt, aber es ist trotzdem unschön diese hart kodierten Werte im Quelltext zu haben. Zudem ist

die Implementierung auch relativ komplex, wie man in Listing 5.1 sehen kann. Es werden dort mehrere Sonderfälle behandelt. Bei einem komplett leeren Dokument kann man keine Cursor-Annotation anzeigen. Innerhalb des Dokuments wird geschaut, ob sich an der aktuellen Position ein Zeichen befindet, das einen Zeilenumbruch darstellt, oder im Falle der Windows-Konvention einen Teil davon, und dass eine Position davor *kein* solches Zeichen steht. Denn dann befindet sich "nichts" zwischen zwei Zeilenumbrüchen und in einer komplett leeren Zeile kann man keine Cursor-Annotation darstellen. Davor wird noch der Fall abgehandelt, dass sich die Schreibmarke ganz am Ende des Dokuments befindet. Das ist ein Sonderfall, weil sich am Ende einer Datei kein Zeichen für einen Zeilenumbruch befinden muss.[7]

Listing 5.1: Alte, zeilenkodierungsabhängige Implementierung von *EditorAPI.setSelection() (r1646)*

```
1   // ...
2   // Adjust for cases in which the cursor is at the end of a line
3   if (isCursor) {
4       IDocument document = docProvider.getDocument(input);
5       if (document == null) {
6           // try with length == 1 maybe it works...
7           length = 1;
8       } else {
9           int n = document.getLength();
10          if (n == 0) {
11              // document has no chars -> cannot adjust anything
12          } else {
13              if (offset == n) {
14                  // we are at the end of the document
15                  if (!isLineEnd(document, offset - 1)) {
16                      length = 1;
17                      offset--;
18                  }
19              } else {
20                  // We are inside the document
21                  length = 1;
22                  if (offset > 0 && isLineEnd(document, offset)
23                      && !isLineEnd(document, offset - 1)) {
24                      offset--;
25                  }
26              }
27          }
28      }
29  }
30  // ...
```

Die neue Implementierung in Listing 5.2 auf der nächsten Seite verwendet eine Methode aus der *Eclipse*-API für *IDocument*-Exemplare, die zu einem gegebenen Versatz ein

[7] Da gibt es unterschiedliche "Philosophien", je nachdem ob man die entsprechenden Zeichen als Zeilen*trenner* oder als Zeilen*ende* betrachtet. Vergleichbar zum Beispiel mit der Rolle des Semikolon bei *Pascal* (Trennen von Anweisungen) oder *Java* (Abschliessen von Anweisungen).

Listing 5.2: Neue, zeilenkodierungs*un*abhängige Implementierung von *EditorAPI.setSelection() (r1676)*

```
1   // ...
2   // Adjust for cases in which the cursor is at the end of a line.
3   if (isCursor) {
4       IDocument document = docProvider.getDocument(input);
5       if (document != null && document.getLength() != 0) {
6           /*
7            * If line not empty and offset at line end then move cursor
8            * one position to the left.
9            */
10          try {
11              IRegion lineInfo = document
12                      .getLineInformationOfOffset(offset);
13              int lineLength = lineInfo.getLength();
14              if (lineLength != 0
15                  && offset == (lineInfo.getOffset() + lineLength)) {
16                  length = 1;
17                  offset--;
18              }
19          } catch (BadLocationException e) {
20              // Ignored intentionally.
21          }
22      }
23  }
24  // ...
```

Objekt liefert, das den Start der Zeile, in dem sich der Offset befindet, und deren Länge enthält. Damit lassen sich die Bedingungen, die für das Anpassen des Versatzes für die Cursor-Annotation gelten müssen, nicht nur unabhängig von der Kodierung der Zeilenumbrüche angeben, sondern sie sind auch noch deutlich kürzer und lesbarer. Ob die Zeile leer ist, kann man einfach über die Zeilenlänge abfragen. Die Schreibmarke steht am Zeilenende, wenn der Versatz des Zeilenanfangs zuzüglich der Zeilenlänge genau den Versatz angibt, an dem sich die Schreibmarke befindet. Der Sonderfall der Schreibmarke am Ende eines Dokuments ohne abschliessenden Zeilenumbruch ist in dieser Bedingung schon enthalten und braucht nicht mehr extra überprüft werden.

Die Befürchtung, dass die Zuordnung vom Versatz in das Dokument zu Zeilenanfang und -länge die Laufgeschwindigkeit bei Dokumenten mit sehr vielen Zeilen signifikant beeinträchtigen könnte, hat sich in der Praxis nicht bewahrheitet. Getestet wurde dies beispielsweise mit einer 65.336 Zeilen langen Textdatei [Tol01]. Das könnte zum Beispiel passieren, wenn intern dafür eine Liste mit Zeileninformationen linear durchsucht würde. Die Standardimplementierung der Methode *getLineInformationOfOffset()* für *IDocument*-Exemplare von *Eclipse* 3.4 und 3.5 wendet eine binäre Suche auf einem Array mit Zeileninformationen an. Die binäre Suche hat eine Laufzeit von $O(\lg n)$ mit n gleich der Anzahl der Zeilen [CLR90].

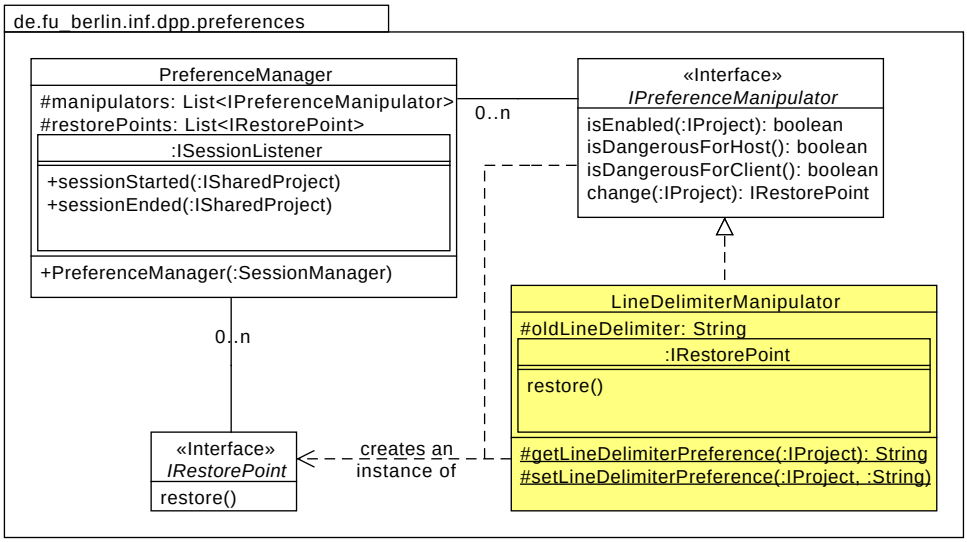


Abbildung 5.1: Klassenrahmen, in den sich die neue Klasse *LineDelimiterManipulator* einfügt.

Zum Ändern der Projekteinstellungen am Anfang einer *Saros*-Sitzung und dem Wiederherstellen am Ende der Sitzung, existiert mit dem *PreferenceManager* bereits ein kleines Rahmenwerk, welches bisher nur für das Deaktivieren der *"Save Actions"* verwendet wurde. Eine Übersicht der Klassen, mit dem von mir neu eingeführten *LineDelimiterManipulator* ist in Abbildung 5.1 zu sehen.

Der *PreferenceManager* registriert sich beim Plugin-Start als Listener um den Start und das Ende einer Sitzung zu erfahren und befragt dann ihm bekannte *IPreferenceManipulator*-Exemplare, ob sie für eine Einstellung verantwortlich sind, die für den Host oder den Client, je nachdem auf welchem der *PreferenceManager* selbst gerade läuft, "gefährlich" sind. Falls ja, wird das *IPreferenceManipulator*-Exemplar dazu veranlasst eine Änderung der Projekteinstellungen vorzunehmen und ein *IRestorePoint*-Exemplar zurückzugeben. Am Ende der Sitzung werden diese gesammelten *IRestorePoint*s verwendet, um die alten Einstellungswerte wiederherzustellen. In Abbildung 5.2 auf der nächsten Seite sind diese beiden Vorgänge als Sequenzdiagramm dargestellt.

Die Einstellung der Zeilenumbruchkodierung muss nur auf der Host-Seite vorgenommen werden, und dort auch nur, falls noch keine Einstellung für das Projekt besteht. Deshalb gibt die Methode *isDangerousForClient()* konstant den Wert **true** und die Methode *isDangerousForHost()* konstant den Wert **false** zurück. Und *isEnabled()* prüft, mittels *getLineDelimiterPreference()*, ob eine Einstellung für das Projekt existiert. Die Durchführung der Änderung wird in Listing 5.3 auf Seite 69 gezeigt. Falls weder projektspezifisch noch im *Eclipse*-Workspace eine Einstellung vorgenommen wurde, wird die von der Java-Laufzeitumgebung als Systemeinstellung ermittelte Kodierung verwendet.

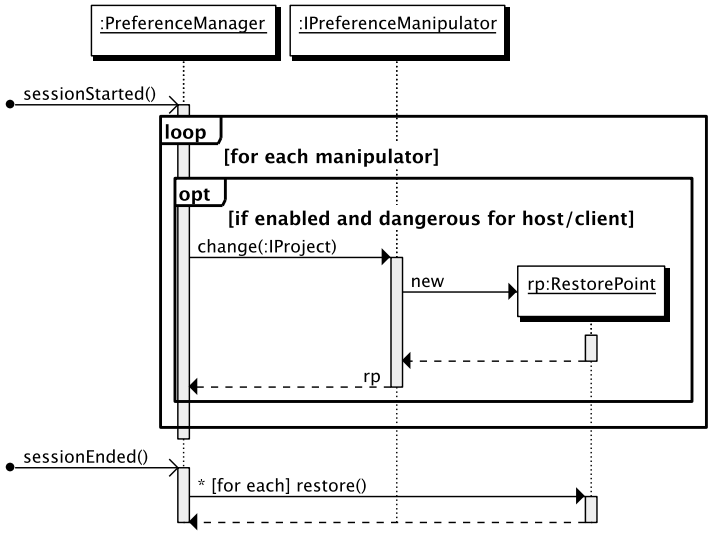


Abbildung 5.2: Ablauf beim Start beziehungsweise Ende einer Saros-Sitzung aus Sicht des *PreferenceManager*.

Saros sollte nun mit beliebigen Kodierungen für Zeilenumbrüche klarkommen, ob das nun Konventionen von alten 8-Bit-Systemen, beispielsweise in einer IDE für Atari-Rechner ist, oder auch Plugins, die ausgiebigeren Gebrauch vom Unicode-Zeichenvorrat machen, wo auch die Zeichen in Tabelle 5.2 einen Zeilenumbruch kodieren.

Codepoint	Bedeutung
U+000C	Seitenvorschub
U+0085	Neue Zeile
U+2028	Zeilentrenner
U+2029	Absatztrenner

Tabelle 5.2: Unicode-Zeichen, die neben U+000A und U+000D, einen Zeilenumbruch implizieren.

Wenn Zeilenumbrüche mit mehr als einem Zeichen kodiert werden, scheint das kein Problem für *Saros* beziehungsweise den verwendeten Jupiter-Algorithmus darzustellen. Ein entsprechender, manueller Belastungstests, den ich zusammen mit Sandor Szücs und Tas Sóti durchgeführt habe, wo wir absichtlich versuchten Text und Zeilenumbrüche simultan an den selben Stellen im Text zu schreiben, zeigte kein Fehlverhalten. Die Zeichenkombination für den Zeilenumbruch blieben immer zusammen und in der richtigen Reihenfolge erhalten. Etwas anderes wurde auch bei regulären Tests vor Releases nicht beobachtet, und auch nicht von Benutzern über den Bugtracker gemeldet. Bei dem Belastungstest sind allerdings Probleme mit der Zeichensatzkodierungen aufgetreten, als Tas im Quelltext eine Klasse erstellte und *Eclipse* in die *JavaDoc*-Vorlage automatisch seinen Namen einfügte und das *ó* in *Sóti* bei ihm anders kodiert war, als bei den anderen Teilnehmern des Tests.

Listing 5.3: *LineDelimiterManipulator.change()* trägt die Zeilenumbruchkodierung des Workspace oder des Betriebssystems in die Einstellungen des übergebenen Projekts ein und liefert einen *IRestorePoint*, der diese Änderung wieder rückgängig macht

```
1   public IRestorePoint change(final IProject project) {
2
3       /* Read the line delimiter from the project specific configuration. */
4       oldLineDelimiter = getLineDelimiterPreference(project);
5       log.debug("Project line delimiter: "
6           + StringEscapeUtils.escapeJava(oldLineDelimiter));
7       /*
8        * If the host has no project specific line delimiter set, we do this
9        * here, so it will be transfered to the clients.
10       */
11      IScopeContext[] scopeContexts = new IScopeContext[] {
12          new ProjectScope(project), new InstanceScope() };
13      IPreferencesService preferencesService = Platform
14          .getPreferencesService();
15      String platformLineDelimiter = System.getProperty("line.separator");
16      String lineDelimiter = preferencesService.getString(
17          Platform.PI_RUNTIME, Platform.PREF_LINE_SEPARATOR,
18          platformLineDelimiter, scopeContexts);
19      putLineDelimiterPreference(project, lineDelimiter);
20
21      return new IRestorePoint() {
22          public void restore() {
23              putLineDelimiterPreference(project, oldLineDelimiter);
24          }
25      };
26  }
```

Vorher verdeckte Defekte

Es traten zwei kleine Defekte zum Vorschein, die vorher noch nicht aufgefallen waren. Zum Einen, dass auch bei über Aktivitäten übertragenen Texten die Zeilenumbrüche in die Unix-Konvention umgewandelt wurden. Das lag daran, dass die Zeichen von Textänderungen in *TextEditActivity*s und in den *Jupiter*-Aktivitäten *DeleteOperation* und *InsertOperation* "ungeschützt" als XML-Textknoten serialisiert wurden, es XML-Parsern vom Standard aber freigestellt ist, alle Vorkommen von Zeilenumbrüchen, die in XML-Dokumenten als Bytewerte 13, 10, und 13;10 kodiert werden dürfen, durch *eine* dieser Varianten zu ersetzen. Das der verwendete Parser alle in die Unix-Konvention umwandelt, ist natürlich in einem Umfeld, welches selbst nur diese Konvention verwendet, nicht aufgefallen. Der Text muss also vor dem XML-Parser geschützt werden, was durch eine zusätzliche Kodierung als *Uniform Resource Identifier* (*URI*) [BLFM05] erreicht werden kann. Dazu wurden die entsprechenden Felder in den Aktivitäten mit einer Annotation für *XStream* versehen, wie Listing 5.4 auf der nächsten Seite anhand von *DeleteOperation.text* stellvertretend für alle betroffenen Felder und Aktivitäten zeigt. Dazu kam noch ein einfacher Unittest zum Sicherstellen, dass eine Aktivität mit einer Zei-

chenkette, welche alle drei Möglichkeiten den Zeilenumbruch in XML auszudrücken beinhaltet, die Abfolge Serialisieren und Deserialisieren unverändert übersteht.

Listing 5.4: *DeleteOperation.text* mit der zusätzlichen Annotation um die Zeilenumbrüche vor dem XML-Parser zu schützen. *(r1696)*

```
1     /**
2      * the text to be deleted.
3      */
4     @XStreamConverter(UrlEncodingStringConverter.class)
5     private String text;
```

Der zweite Defekt war eine falsche Fehlermeldung im Log, in Fällen wo eine Inkonsistenz zwischen Client und Host aufgelöst wurde, welche nur in unterschiedlich kodierten Zeilenumbrüchen bestand. Nach einer Konsistenzwiederherstellung wird beim Client der empfangene korrekte Text, mit dem dazu inkonsistenten Lokalen verglichen und ein Diff ins Log geschrieben, damit man bei der Fehlersuche nachvollziehen kann, wie die Unterschiede genau aussehen, und dadurch Rückschlüsse auf die Ursache ziehen, oder zumindest die Fehlerquelle enger eingrenzen kann. Der verwendete Diff-Algorithmus aus einer externen Bibliothek[8] erwartet dazu beide Texte als Array von Zeilen. Die Hilfsmethoden zum Umwandeln der Texte in diese Form haben die Zeilenumbrüche entfernt, und so bekam der Diff-Algorithmus zwei wertegleiche Arrays und hat deshalb keine Unterschiede gemeldet. Das wurde von *Saros* geprüft und als Fehler im Log gemeldet, denn bei einer Inkonstenz darf das Diff nicht leer sein. Es hätte genügt die Zeilenumbrüche nicht von den Zeichenketten in den Arrays zu entfernen, dann wäre allerdings ein Diff ausgegeben worden, das in vielen Umgebungen zum Anzeigen (der Log-View in *Eclipse*, Konsolen, E-Mails, und so weiter) für den Betrachter in den unterschiedlichen Zeilen augenscheinlich gleich aussieht. Deshalb sind nun die Zeilenumbrüche nicht als solche in den Diff-Zeilen, sondern als sichtbare Zeichenfolgen `'\\r'` und `'\\n'` kodiert. Anstelle der Funktion *IOUtils.readLines()* aus dem *Commons IO*-Paket[9] vom Apache-Project wird nun die Funktion in Listing 5.5 auf der nächsten Seite zum Einlesen verwendet. Sie hat den Nachteil, dass die Datei komplett in den Speicher gelesen wird, und dort zeitweise mindestens doppelt so viel Speicher benötigt, wie die Daten in der Datei. Bei moderaten Dateigrössen ist das kein Problem, aber bei sehr grossen Dateien, zum Beispiel Logdateien mit Grössen im oberen zweistelligen Megabytebereich, könnte der Arbeitsspeicher knapp werden. Bei Textdateien im Bereich um drei Megabyte hat die Funktion in Tests keine Probleme gezeigt, und Quelltexte sind in aller Regel deutlich kleiner. Deshalb ist diese Einschränkung meiner Meinung nach akzeptabel.

Das Rahmenwerk rund um den *PreferenceManager* berücksichtigt bisher noch keine Änderungen der Einstellungen *während* einer Sitzung. Dazu müsste man auch während einer Sitzung gegebenenfalls geänderte Einstellungen vom Host zum Client übertragen,

[8] GNU Diff for Java: `http://www.bmsi.com/java/#diff`, basiert auf [Mye86].
[9] Commons IO: `http://commons.apache.org/io/index.html`

Listing 5.5: *Util.readLinesAndEscapeNewlines() (r1734)*

```
1   /**
2    * Reads the given stream into an array of lines while retaining and
3    * escaping the line delimiters.
4    */
5   public static String[] readLinesAndEscapeNewlines(InputStream stream)
6       throws IOException {
7
8       List<String> result = new ArrayList<String>();
9       String data = IOUtils.toString(stream);
10      Matcher matcher = Pattern.compile("\\r\\n|\\r|\\n").matcher(data);
11      int previousEnd = 0;
12      while (matcher.find()) {
13          // Extract line and escape line delimiter.
14          result.add(data.substring(previousEnd, matcher.start())
15              + escapeForLogging(matcher.group()));
16          previousEnd = matcher.end();
17      }
18      // If there is no line delimiter after the last line...
19      if (previousEnd != data.length()) {
20          result.add(data.substring(previousEnd, data.length()));
21      }
22      return result.toArray(new String[result.size()]);
23  }
```

zumindest für die Einstellungen bei denen der Wunsch des Benutzers berücksichtigt werden kann. Im Gegensatz beispielsweise zum Abschalten der *"Save Actions"*, die *Saros* absichtlich bei allen Teilnehmern für die Dauer einer *Saros*-Sitzung deaktiviert.

5.2 Jabber-Präsenzinformation

In den *Teles*-Sitzungen konnte man durch den Videomitschnitt vom Desktop von Entwickler #1 gut sehen, wann er *Eclipse*, und damit in der Regel auch *Saros* verwendete, und wann nicht. Diese Information fehlte von Entwickler #2 sowohl für uns als Beobachter der Sitzungen, als auch für Entwickler #1. Benutzer von *Jabber*-Chat-Clients sind gewohnt, dass man über längere Abwesenheit eines Gegenübers informiert wird. Das wird in der Regel an einer längeren Periode des Nichtbenutzens der Eingabegeräte festgemacht. Wenn man zum Beispiel fünf oder zehn Minuten lang weder Tastatureingaben tätigt, noch die Maus bewegt, sendet der eigene Chat-Client die Information, dass man abwesend (englisch "away") ist. Im Rahmen von *Saros* scheint es sinnvoll die Bedingungen für "Abwesenheit" zeitlich und "örtlich" enger zu fassen, also schneller so eine Meldung auf den Weg zu bringen. Nicht erst wenn man vermutet die betreffende Person sässe nicht mehr vor dem Rechner, sondern schon wenn man weiss, dass *Eclipse* nicht mehr die aktive Anwendung ist. So erhalten die Teilnehmer eine Zusatzin-

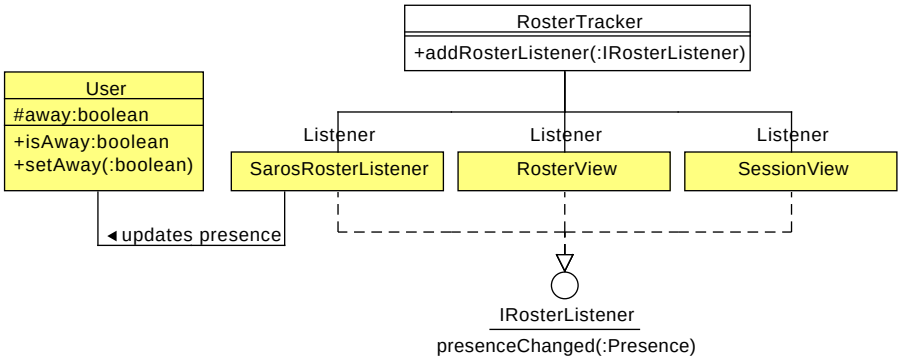


Abbildung 5.3: Klassen, die an der Verarbeitung und Anzeige von *Jabber*-Präsenzinformationen beteiligt sind. *(r1684)*

formation, die zum Beispiel Cursor- oder Viewport-Annotationen ergänzen kann, denn eine Cursor-Annotation alleine sagt nur wo der Cursor des anderen Teilnehmers einer *Saros*-Sitzung steht, nicht aber ob er nicht gerade eine völlig andere Anwendung benutzt. Die beiden Informationen, *Eclipse* ist die aktive Anwendung und die Position der Cursor-Annotation, garantieren natürlich nicht, dass die Aufmerksamkeit auch auf die entsprechende Stelle im Quelltext gerichtet ist.

Um die Erfassung und das Versenden der Information hat sich Christopher Oezbek gekümmert. Erfasst und versendet wird die Präsenz vom *LocalPresenceTracker* und auf der Empfängerseite kann man sich beim *RosterTracker* als Listener registrieren, um über Änderungen von anderen Teilnehmern informiert zu werden. Meine Aufgabe bestand darin, diese Information in die Geschäftslogik und die GUI von *Saros* weiterzutragen. Dazu musste eine entsprechende Behandlung in den *SarosRosterListener* für die Geschäftslogik, und in die beiden GUI-Klassen *RosterView* und *SessionView* für die Anzeige implementiert werden (Abbildung 5.3).

Teilnehmer werden in *Saros* durch die Klasse *User* modelliert, die alle wichtigen Eigenschaften eines Teilnehmers hält, wie zum Beispiel den Namen, die Jabber-ID, die aktuelle Rolle, den Verbindungsstatus, und so weiter. Diese Klasse hat jetzt zusätzlich ein **boolean** Feld mit dem Namen *away* bekommen, in dem die An- beziehungsweise Abwesenheit, nach der Definition am Anfang dieses Abschnitts, vom *SarosRosterListener* vermerkt wird. Vorher wurde dort in der *presenceChanged()*-Methode nur der Verbindungsstatus des Teilnehmers (online, offline, oder unbekannt) vom übergebenen *Presence*-Objekt erfragt und im *User*-Objekt hinterlegt.

Die Listener in den beiden GUI-Klassen sorgen in der *presenceChanged()*-Methode dafür, dass die entsprechenden Anzeigen aktualisiert werden, was in beiden Fällen letztendlich dazu führt, dass eine *getImage()*-Methode eines *LabelProvider*-Exemplars aufgerufen

wird.

Listing 5.6: Die *ViewLabelProvider.getImage()*-Methode im *RosterView* gibt je nach Jabber-Präsenz des Teilnehmers das normale, oder ein "Away"-Icon für *RosterEntry*s zurück. *(r1684)*

```
1       public Image getImage(Object element) {
2
3           if (element instanceof RosterEntry) {
4               RosterEntry entry = (RosterEntry) element;
5
6               Presence presence = roster.getPresence(entry.getUser());
7
8               if (presence.isAvailable()) {
9                   return presence.isAway() ? personAwayImage : personImage;
10              } else {
11                  return personOfflineImage;
12              }
13          }
14          return groupImage;
15      }
```

Im *RosterView* wird die Präsenzinformation, wie man in Listing 5.6 sehen kann, nicht aus einem *User*-Objekt geholt, denn die im Roster angezeigten Kontakte können völlig unabhängig von *Saros*-Sitzungen sein. Listing 5.7 zeigt die Implementierung im *SessionView*. Hier wird auf die Information im *User*-Objekt zurückgegriffen, da diese Objekte im Modell für die Tabelle in der Anzeige hinterlegt sind, und man sehr einfachen Zugriff darauf hat. Bei der Anzeige des "away"-Icons wird an dieser Stelle für den Benutzer die Information verdeckt, ob der betreffende Teilnehmer Driver ist oder nicht. Allerdings steht in dem Fall bei dem Text in der Tabellenzeile noch der Zusatz "(Driver)" hinter dem Namen des Teilnehmers.

Listing 5.7: Die *SessionLabelProvider.getImage()*-Methode im *SessionView* liefert je nach Jabber-Präsenz des Teilnehmers ein "away"-Icon oder eines, welches den Driver-Status anzeigt. *(r1684)*

```
1       public Image getImage(Object obj) {
2           User user = (User) obj;
3           if (user.isAway()) {
4               return awayImage;
5           } else {
6               return user.isDriver() ? driverImage : userImage;
7           }
8       }
```

Die Funktionalität ist noch nicht perfekt, da auch die Anzeige von modalen Dialogen von *Eclipse* selbst, als Abwesenheit von *Eclipse* gemeldet wird. Aus diesem Grund ist das Ticket im Bugtracker noch nicht geschlossen. Allerdings ist dem Eintrag die niedrigste Priorität zugewiesen, da meine Suche nach einer Lösung zu erfolglos und mit

fast einem Tag lang genug war, und es meiner Meinung nach vorerst gut genug funktioniert.

6 Outreach

Für die Verbreitung der frohen *Saros*-Kunde und damit der stetigen Vergrösserung des Bekanntheitsgrades von *Saros* und des Auffrischens der Erinnerung von Leuten, die *Saros* bereits kennen, habe ich mir die Website *Freshmeat.net*[1] ausgesucht. Die Website stellt nach eigenen Angaben[2] den grössten Index von Software für Unix-Derivate und plattformunabhängiger Software im *World Wide Web*. Die Einträge in diesem Online-Katalog werden von Freiwilligen erstellt und gepflegt und Veröffentlichungen von neuen Versionen werden auf der Einstiegsseite angekündigt, als Newsfeed im *Atom Syndication Format* (*ASF*) [NS05] angeboten[3], und auf Wunsch als täglicher Newsletter in Form einer E-Mail an registrierte Benutzer versandt. Newsfeeds können auch für einzelne Projekteinträge bezogen werden.

Neben einer Beschreibung, Bildschirmfotos, Schlagworten (“tags”), und den Ankündigungen zu neuen Versionen eines katalogisierten Projekts, bietet die jeweilige Projektseite auch Graphen an, die wiedergeben, wie viele Leute sich in der näheren Vergangenheit den Projekteintrag angesehen und dem Verweis zur angegebenen Homepage gefolgt sind (Abbildung 6.1). Ausserdem wird ein Index für die Popularität und die Vitalität des Projekts aus den gesammelten Daten berechnet (Abbildung 6.2 auf der nächsten Seite), und Benutzer können für das Projekt eine positive oder negative Stimme abgeben.

1 `http://freshmeat.net/`
2 `http://freshmeat.net/about`
3 `http://freshmeat.net/?format=atom`

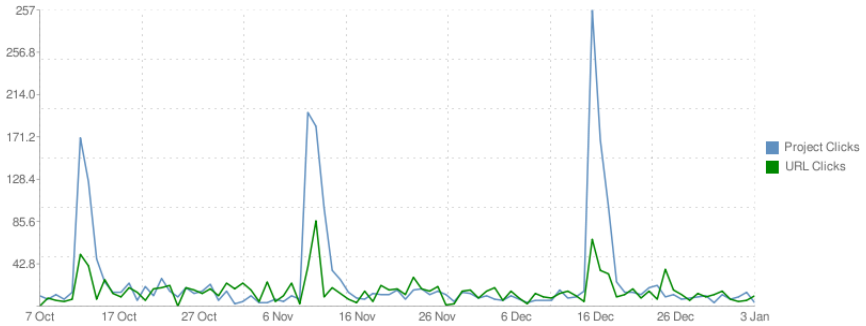


Abbildung 6.1: Traffic-Graph vom *Freshmeat.net*-Eintrag für *Saros*, Stand 2010-01-04. Man sieht deutlich den Anstieg der Besucher der Projektseite und des Folgens der URL zur *Saros*-Projektseite, immer wenn eine neue Version über *Freshmeat.net* angekündigt wurde.

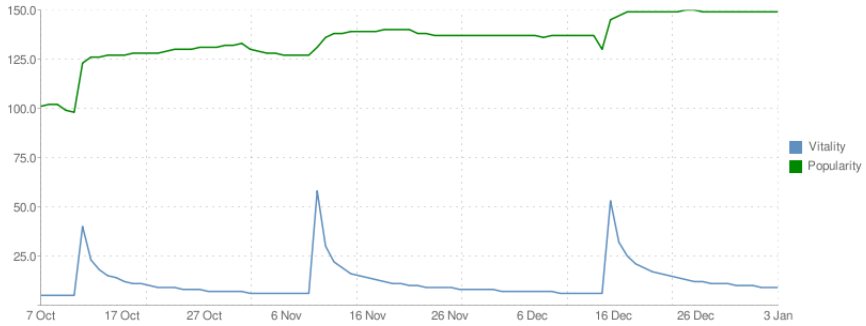


Abbildung 6.2: "Heartbeat"-Graph vom *Freshmeat.net*-Eintrag für *Saros*, Stand 2010-01-04.

Meine Wahl fiel auf *Freshmeat.net*, weil ich dort bereits ein Benutzerkonto besass, und bei der Betreuung eines anderen Projekteintrags[4] schon Erfahrung im Umgang mit der Website gesammelt hatte.

Den Eintrag für Saros[5] kann jeder bei *Freshmeat.net* angemeldete Benutzer bearbeiten. Die Gefahr von Vandalismus oder Spam ist dabei recht gering, da Änderungen an den Texten nicht sofort wirksam werden, sondern erst von *Freshmeat.net*-Mitarbeitern redaktionell bearbeitet und freigeschaltet werden. Was auch den Vorteil hat, dass manch unglückliche englische Formulierung noch einmal von jemandem korrekturgelesen wird, der die Sprache gut beherrscht.

Im Gegensatz zu *Freshmeat.net* war das Thema *Microblogging* neu für mich, aber eine Radiosendung zum Thema [KPh09] brachte mich auf die Idee, dass Dienste wie *Identi.ca*[6] und *Twitter*[7] auch geeignet sein können *Saros* bekannter zu machen und neue Versionen anzukündigen. Zudem bieten diese Plattformen auch gleich einen Rückkanal von Benutzern. *Identi.ca* hat den Vorteil, dass man dort auch Nachrichten an Gruppen adressieren kann, also zum Beispiel an alle Benutzer, welche die bereits vorhandene *Eclipse*-Gruppe[8] verfolgen, während man bei *Twitter* mehr darauf angewiesen ist, dass Interessierte den Feed von sich aus finden und ihm folgen. Die Betreuung des "Outreach" mittels Microblogging hat Robert Kunze übernommen. Der Name des "Benutzers" lautet bei beiden Diensten *Saros Team*.[9]

[4] cc65 – ein nahezu ISO C konformer C-Cross-Compiler, der Binärcode für Rechner erzeugt, die auf dem MOS 6502-Mikroprozessor oder Varianten davon basieren.

[5] *Saros* bei *Freshmeat.net*: `http://freshmeat.net/projects/saros-2`

[6] `http://www.identi.ca/`

[7] `http://www.twitter.com/`

[8] `http://identi.ca/group/eclipse`

[9] *Saros* bei *Twitter* und *Identi.ca*: `http://identi.ca/saros` und `http://twitter.com/saros_plugin`

7 Nachgedanken

7.1 Rückblick

Es gab leider nur zwei "vollwertige" *Teles*-Sitzungen, an denen ich teilnehmen konnte, statt der von mir erwarteten ungefähr 16 Sitzungen. Die ursprünglich für diese Arbeit geplante Erfassung und Auswertung von Datenpunkten, an denen man Aussagen über mögliche technische Probleme treffen kann, wie sie sich über einen längeren Zeitraum entwickeln, war deshalb nicht mehr möglich.

Durch *Saros*-Sitzungen bei OpenSource-Projekten aus Eike Starkmann's Arbeit [Sta09] konnten die *Teles*-Sitzungen nicht ersetzt werden, da keines der kontaktierten Projekte im nötigen Umfang in eine Zusammenarbeit einstieg. Entwickler von einigen Projekten haben sich *Saros* angeschaut und auch Defekte gemeldet, allerdings reichte das meines Ermessens nach nicht für einen eigenen Abschnitt in dieser Arbeit aus.

Eine für mich persönlich bereichernde Erfahrung war die gesprochene englische Sprache. Selbstverständlich kommunizierte ich auch vor der Arbeit schon über das Internet auf Englisch, aber immer nur in schriftlicher Form und überwiegend auch nicht in Echtzeit. Es blieb dabei immer Zeit an den Formulierungen zu arbeiten und Vokabeln nachzuschlagen. Durch Filme und Serien im englischen Originalton hatte ich die Befürchtung, dass es durch Dialekte zu Verständigungsschwierigkeiten kommen könnte. Diese Befürchtung bewahrheitete sich auch bei den *Teles*-Sitzungen – der Entwickler am Standort Indien war für mich teilweise schwer zu verstehen.

Mit dem neuen technischen Projektleiter Karl Beecher erhöhte sich die Anzahl der auf Englisch geführten Gespräche deutlich. Es gab mit ihm – zumindest von meiner Seite aus – keine Verständnisprobleme durch einen Dialekt.

7.2 Ausblick

Die Kooperation mit *Teles* sollte nach meiner Meinung falls möglich wieder aufgenommen werden. Beobachtungen und Bug-Meldungen in diesen regelmässig abgehaltenen Sitzungen sind eine verlässlichere und präzisere Quelle als die Rückmeldungen, die bisher über den Bug-Tracker oder die Online-Umfragen von aussen an das *Saros*-Team herangetragen wurden. Man kann bei den Entwicklern von *Teles* Rückfragen stellen,

wohingegen die Kommunikation mit externen Nutzern von *Saros* in der Regel nur in eine Richtung verläuft – vom Nutzer zum *Saros*-Team.

Die grafische Schnittstelle von *Saros* zum Benutzer müsste überarbeitet werden. Hier wurden bisher noch keine Überlegungen bezüglich der "Usability", also Benutzbarkeit, angestellt. Jedes Team-Mitglied hat isoliert Komponenten hinzugefügt, die für seine Arbeit nötig erschienen. Das schlägt sich zum Beispiel darin nieder, wie bei Menüpunkten im *Roster*- und *Shared Project Session*-View auf eine Mehrfachauswahl reagiert wird, oder wenn im *Roster* nicht nur Kontakte, sondern auch Gruppen ausgewählt wurden. In solchen Fällen sollten die im Menü ausgewählbaren Aktionen einheitlich verfahren. Es gab Aktionen, die bei Mehrfachauswahl nichts taten, solche die auf alle ausgewählten Daten wirkten, und welche die nur auf das zuerst ausgewählte Datum angewendet wurden. In den Kontextmenüs sollten vielleicht keine Menüpunkte aufgeführt werden, die grundsätzlich immer "ausgegraut" sind. Zum Beispiel können Clients keine Rollen vergeben und entziehen – sie können diese Punkte aber trotzdem im Kontextmenü vom *Shared Project Session*-View finden. Die teilweise unpassenden Icons wurden in den Abschnitten zu den Views schon angesprochen (Abschnitte 2.2.2 auf Seite 13 und 2.2.3 auf Seite 14).

Literaturverzeichnis

[BA04] BECK, KENT und CYNTHIA ANDRES: *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[Bal98] BALZERT, HELMUT: *Lehrbuch der Softwaretechnik : Software Management, Software-Qualitätssicherung, Unternehmensmodellierung*, Band 2. Spektrum, Akademischer Verlag, Heidelberg; Berlin, 1998.

[BLFM05] BERNERS-LEE, T., R. FIELDING und L. MASINTER: *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986 (Standard), Januar 2005.

[BLMM94] BERNERS-LEE, T., L. MASINTER und M. MCCAHILL: *Uniform Resource Locators (URL)*. RFC 1738 (Proposed Standard), Dezember 1994. Obsoleted by RFCs 4248, 4266, updated by RFCs 1808, 2368, 2396, 3986.

[Cha85] CHADWICK, IAN: *Mapping The Atari*. COMPUTE! Publications, Greensboro, North Carolina, überarbeitete Auflage, 1985.

[CLR90] CORMEN, THOMAS H., CHARLES E. LEISERSON und RONALD L. RIVEST: *Introduction to Algorithms*. The MIT electrical engineering and computer science series. MIT Press, 1990.

[Cri03] CRISPIN, M.: *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*. RFC 3501 (Proposed Standard), März 2003. Aktualisiert durch RFCs 4466, 4469, 4551, 5032, 5182.

[DB92] DOURISH, PAUL und VICTORIA BELLOTTI: *Awareness and coordination in shared workspaces*. In: *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, Seiten 107–114, New York, NY, USA, 1992. ACM.

[Dje06] DJEMILI, RIAD: *Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammierung*. Diplomarbeit, Freie Universität Berlin, 2006.

[Doh09] DOHRMANN, LISA: *Erhebung von Benutzerfeedback aus der Nutzung eines Werkzeugs zur verteilten Paarprogrammierung*. Bachelorarbeit, Freie Universität Berlin, Juli 2009.

[FB96] FREED, N. und N. BORENSTEIN: *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. RFC 2045 (Draft Standard), November 1996. Aktualisiert durch RFCs 2184, 2231, 5335.

[Fow04] FOWLER, MARTIN: *Inversion of Control Containers and the Dependency Injection pattern*. http://martinfowler.com/articles/injection.html, Januar 2004. Abgerufen: 2010-03-01.

[GGR96] GUTWIN, CARL, SAUL GREENBERG und MARK ROSEMAN: *Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation*. In: , Seiten 281–298. Springer-Verlag, 1996.

[GHJV95] GAMMA, ERICH, RICHARD HELM, RALPH JOHNSON und JOHN VLISSIDES: *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[GN99] GANSNER, EMDEN R. und STEPHEN C. NORTH: *An Open Graph Visualization System and Its Applications to Software Engineering*. Software - Practice and Experience, 30:1203–1233, 1999.

[Gus07] GUSTAVS, BJÖRN: *Weiterentwicklung des Eclipse-Plug-Ins Saros zur Verteilten Paarprogrammierung*. Studienarbeit, Freie Universität Berlin, Oktober 2007.

[Jac09] JACOB, CHRISTOPH: *Weiterentwicklung eines Werkzeuges zur verteilten, kollaborativen Softwareentwicklung*. Diplomarbeit, Freie Universität Berlin, April 2009.

[Kiw10] KIWITT, ALENA: *Supporting Various Programming Languages and Plug-Ins Within a Tool For Collaborative Software Development*. https://www.inf.fu-berlin.de/w/SE/ThesisDPPXVI, Januar 2010. Abgerufen: 2010-02-20.

[KPh09] KRANZ, JAKOB, TIM PRITLOVE und HUKL: *Twitterdemokratie – Microblogging und Zensur ringen um die Macht*. Radiosendung/Podcast, http://chaosradio.ccc.de/cr147.html.

[KSA09] KARNEGES, JUSTIN und PETER SAINT-ANDRE: *In-Band Bytestreams*. XEP-0047 Standards Track, März 2009. http://xmpp.org/extensions/xep-0047.html.

[Lau10] LAU, STEPHAN: *Verbesserte Präsenz durch Screensharing für ein Werkzeug zur verteilten Paarprogrammierung*. https://www.inf.fu-berlin.de/w/SE/ThesisDPPXIII, Februar 2010. Abgerufen: 2010-02-20.

[LBSA+09] LUDWIG, SCOTT, JOE BEDA, PETER SAINT-ANDRE, ROBERT MCQUEEN, SEAN EGAN und JOE HILDEBRAND: *Jingle*. XEP-0166 Standards Track, Dezember 2009. http://xmpp.org/extensions/xep-0166.html.

[Lew48] LEWIN, KURT: *Die Lösung sozialer Konflikte*, Kapitel Tat-Forschung und Minderheitenprobleme, Seiten 278–298. Christian-Verlag, Bad-Neuheim, 1948.

[Log10] LOGA, OLAF: *Verbesserung der Kommunikationsmöglichkeiten in Saros*. https://www.inf.fu-berlin.de/w/SE/ThesisDPPXV, Februar 2010. Abgerufen: 2010-02-20.

[Mye86] MYERS, EUGENE: *An O(ND) Difference Algorithm and its Variations*. Algorithmica, 1(2):251, 1986.

[NCDL95] NICHOLS, DAVID A., PAVEL CURTIS, MICHAEL DIXON und JOHN LAMPING: *High-latency, low-bandwidth windowing in the Jupiter collaboration system*. In: *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, Seiten 111–120, New York, NY, USA, 1995. ACM.

[Nos98] NOSEK, JOHN T.: *The case for collaborative programming*. Commun. ACM, 41(3):105–108, 1998.

[NS05] NOTTINGHAM, M. und R. SAYRE: *The Atom Syndication Format*. RFC 4287 (Proposed Standard), Dezember 2005.

[Ray96] RAYMOND, ERIC S.: *The new hacker's dictionary*. MIT Press, 3 Auflage, 1996.

[Ray99] RAYMOND, ERIC S.: *The Cathedral & the Bazaar*. O'Reilly, 1999.

[Rie08] RIEGER, OLIVER: *Weiterentwicklung einer Eclipse-Erweiterung für verteilte Paar-Programmierung im Hinblick auf Kollaboration und Kommunikation*. Diplomarbeit, Freie Universität Berlin, Juli 2008.

[Rin09] RINTSCH, MARC: *Agile Weiterentwicklung eines Software-Werkzeuges zur verteilten, kollaborativen Programmierung in Echtzeit*. Studienarbeit, Freie Universität Berlin, Juni 2009.

[Ros09] ROSEN, EDNA: *Verteilte Paarprogrammierung im industriellen Umfeld, Etablierung und qualitative empirische Untersuchung*. Diplomarbeit, Freie Universität Berlin, 2009.

[SA04] SAINT-ANDRE, P.: *Extensible Messaging and Presence Protocol (XMPP): Core*. RFC 3920 (Proposed Standard), Oktober 2004.

[SHE99] SCHNELL, RAINER, PAUL B. HILL und ELKE ESSER: *Methoden der empirischen Sozialforschung*. R. Oldenbourg Verlag, 1999.

[Sit08] SITSKY, DAVID: *The Codestriker Guide : Version 1.9.5*. `http://codestriker.sourceforge.net/codestriker.html`, 2008. Abgerufen: 2010-02-20.

[SS09] SCHINDLER, SIGRAM und OLAF SCHULZ: *Geschäftsbericht 2008 der TELES AG Informationstechnologien*, Juli 2009.

[ST10] SAROS-TEAM: *Developer Guide*. `https://www.inf.fu-berlin.de/w/SE/DPPHowToDevel`, Januar 2010. Abgerufen: 2010-02-20.

[Sta09] STARKMANN, EIKE: *Verteilte Paarprogrammierung in Open Source Projekten*. `https://www.inf.fu-berlin.de/w/SE/ThesisDPPIX`, Juli 2009. Abgerufen: 2010-02-20.

[Sta10] STAIB, HENNING: *Verbesserung einer XMPP-Bibliothek für den Einsatz in verteilter Paarprogrammierung*. https://www.inf.fu-berlin.de/w/SE/ThesisDPPX, Februar 2010. Abgerufen: 2010-02-20.

[Sz"09] SZÜCS, SANDOR: *Behandlung von Netzwerk- und Sicherheitsaspekten in einem Werkzeug zur verteilten Paarprogrammierung*. https://www.inf.fu-berlin.de/w/SE/ThesisDPPVI, Juli 2009. Abgerufen: 2010-02-20.

[Só09] SÓTI, TAS: *Einladungsprozess in Saros*. Bachelorarbeit, Freie Universität Berlin, Oktober 2009.

[TDM97] TROOST, R., S. DORNER und K. MOORE: *Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field*. RFC 2183 (Proposed Standard), August 1997. Aktualisiert durch RFCs 2184, 2231.

[Tol01] TOLSTOY, LEO: *War and Peace*. Project Gutenberg Literary Archive Foundation, Salt Lake City, 2001. http://www.gutenberg.org/etext/2600.

[Wil00] WILLIAMS, LAURIE: *The Collaborative Software Process*. Doktorarbeit, University of Utah, 2000.

[WM] WANG, DAVID und ALEX MAH: *Google Wave Operational Transformation*. http://www.waveprotocol.org/whitepapers/operational-transform. Stand 2010-02-09.

[Zil09] ZILLER, SEBASTIAN: *Behandlung von Nebenläufigkeitsaspekten in einem Werkzeug zur Verteilten Paarprogrammierung*. Diplomarbeit, Freie Universität Berlin, Oktober 2009.

Anhang A

Vorlage für Notizen zu Teles-Sitzungen

```
1  ==============================
2   Notizen zu Teles-Sitzung 000
3  ==============================
4
5  :Autor: Marc 'BlackJack' Rintsch
6  :Kontakt: marc@rintsch.de
7  :Datum: 2009-00-00
8  :Version: 0.1
9
10 .. sectnum::
11 .. contents::
12
13
14 Sitzungsdaten
15 ==============
16
17 :Nummer: 000
18 :Datum: 2009-00-00
19 :Laenge: 0 Minuten
20 :Typ:
21 :Entwickler:
22
23
24 Videoauswertung
25 ================
26
27 Zeitangaben in Minuten:Sekunden seit Aufzeichnungsbeginn von X's Desktop.
28
29 * 00:00 ...
30
31 * 00:00 **Ende**
32
33 :Dauer der Videoauswertung: 0 Minuten
34
35
36 Generelle Beobachtungen und Anmerkungen
37 ======================================
38
39 * ...
40
41
```

```
42 Logs
43 ====
44
45 Benutzer 1
46 ----------
47
48 ...
49
50
51 Benutzer 2
52 ----------
53
54 ...
55
56
57 Zusammenfassung
58 ---------------
59
60 ...
```

Anhang B

E-Mail Konversation zu Ctrl + Esc

In diesem Abschnitt sind alle vier Mails zum Thema “Autovervollständigung mittels Ctrl + Esc ” abgedruckt, damit sich der Leser selbst ein Urteil über die Zusammenfassung in Abschnitt 3.6.9 auf Seite 38 bilden kann.

Die Namen der Entwickler sind durch die in dieser Arbeit verwendeten Pseudonyme Entwickler #1 und Entwickler #2 ersetzt, und E-Mail-Adressen und Binäranhänge sind entfernt worden. Ansonsten sind die Mails inhaltlich unverändert so wiedergegeben, wie sie vom E-Mail-Programm angezeigt wurden.

E-Mail #1 von mir an beide Entwickler

```
1  Logs / Ctrl+Esc shortcut
2  From: Marc 'BlackJack' Rintsch <marc@rintsch.de>
3    To: Entwickler #1, Entwickler #2
4    CC: Edna Rosen
5  Date: 2009-09-18 11:11
6
7  Hi Entwickler #1,
8  Hi Entwickler #2,
9
10 Could you please mail the Saros logs to me.  Thanks in advance.
11
12 For the next session you should go to the Saros feedback preferences
13 (Window > Preferences... > Saros > Feedback) and activate both check
14 boxes under "Error Log Submission".  Then the logs will be send
15 automatically when you leave the session.
16
17 @Entwickler #2: I have tried to reproduce your troubles with the Ctrl+Esc
18 keyboard shortcut but was not able to do so.  At least under Windows XP
19 this key combination is the same as pressing the "Windows" key, i.e. it
20 opens the start menu in the task bar and never reaches the active
21 application.  How did you configure your Windows and Eclipse to get this
22 keyboard shortcut working?
23
24 An easy work around is the combination Ctrl+Space that opens the auto
25 completion drop down menu at the cursor position.
```

```
26
27 Ciao,
28         Marc 'BlackJack' Rintsch
```

E-Mail #2 von Entwickler#2 an mich

```
1  RE: Logs / Ctrl+Esc shortcut
2  From: Entwickler #2
3    To: "Marc 'BlackJack' Rintsch" <marc@rintsch.de>, Entwickler #1
4    CC: "Edna Rosen"
5  Date: 2009-09-18 11:17
6  Attachments: 2009-09-16.7z
7
8  HI Marc
9  I need this thing
10
11 An easy work around is the combination Ctrl+Space that opens the auto
12 completion drop down menu at the cursor position.
13
14 But it was not working for me when I was in saros session.
15
16 --Entwickler #2
17
18 -----Original Message-----
19 From: Marc 'BlackJack' Rintsch [mailto:marc@rintsch.de]
20 Sent: Friday, September 18, 2009 2:41 PM
21 To: Entwickler #1; Entwickler #2
22 Cc: Edna Rosen
23 Subject: Logs / Ctrl+Esc shortcut
24
25 Hi Entwickler #1,
26 Hi Entwickler #2,
27
28 Could you please mail the Saros logs to me.  Thanks in advance.
29
30 For the next session you should go to the Saros feedback preferences
31 (Window > Preferences... > Saros > Feedback) and activate both check
32 boxes under "Error Log Submission".  Then the logs will be send
33 automatically when you leave the session.
34
35 @Entwickler #2: I have tried to reproduce your troubles with the Ctrl+Esc
36 keyboard shortcut but was not able to do so.  At least under Windows XP
37 this key combination is the same as pressing the "Windows" key, i.e. it
38 opens the start menu in the task bar and never reaches the active
39 application.  How did you configure your Windows and Eclipse to get this
40 keyboard shortcut working?
41
42 An easy work around is the combination Ctrl+Space that opens the auto
43 completion drop down menu at the cursor position.
44
45 Ciao,
```

```
46          Marc 'BlackJack' Rintsch
```

E-Mail #3 von mir an Entwickler#2

```
1  Re: Logs / Ctrl+Esc shortcut
2  From: Marc 'BlackJack' Rintsch <marc@rintsch.de>
3    To: Entwickler #2
4  Date: 2009-09-21 15:14
5
6  Hi Entwickler #2,
7
8  Entwickler #2 wrote:
9  > HI Marc
10 > I need this thing
11 >
12 >> An easy work around is the combination Ctrl+Space that opens the auto
13 >> completion drop down menu at the cursor position.
14 >
15 > But it was not working for me when I was in saros session.
16
17 Do you have made any changes to the key mapping in the preferences or do
18 you use any plug-in to simulate the behaviour of a specific editor?
19 Because I cannot reproduce the problem with Ctrl+Space and I can't get
20 Ctrl+Esc working in the first place, because Windows grabs that key
21 combination.
22
23 Hard for me to fix a problem without being able to reproduce it.  :-(
24
25 Ciao,
26          Marc 'BlackJack' Rintsch
```

E-Mail #4 von Entwickler#2 an mich

```
1  RE: Logs / Ctrl+Esc shortcut
2  From: Entwickler #2
3    To: "Marc 'BlackJack' Rintsch" <marc@rintsch.de>
4  Date: 2009-09-22 07:28
5
6  HI marc,
7
8  It has been a confusion for ctlr+Esc.Instead I wanted cltr+space.
9  Cltr+ Esc doesnot really exist for Eclipse.
10
11 --Entwickler #2
12
13 -----Original Message-----
14 From: Marc 'BlackJack' Rintsch [mailto:marc@rintsch.de]
```

```
Sent: Monday, September 21, 2009 6:45 PM
To: Entwickler #2
Subject: Re: Logs / Ctrl+Esc shortcut

Hi Entwickler #2,

Entwickler #2 wrote:
> HI Marc
> I need this thing
>
>> An easy work around is the combination Ctrl+Space that opens the auto
>> completion drop down menu at the cursor position.
>
> But it was not working for me when I was in saros session.

Do you have made any changes to the key mapping in the preferences or do
you use any plug-in to simulate the behaviour of a specific editor?
Because I cannot reproduce the problem with Ctrl+Space and I can't get
Ctrl+Esc working in the first place, because Windows grabs that key
combination.

Hard for me to fix a problem without being able to reproduce it. :-(

Ciao,
        Marc 'BlackJack' Rintsch
```