Diplomarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Verbesserung und Pflege der Dokumentation der DPP-Software Saros

Meike Johannsen

Berlin, 10. Oktober 2011

Gutachter:

Prof. Dr. Lutz Prechelt

Prof. Dr. Elfriede Fehr

Betreuer:

Dr. Karl Beecher

Danksagung

Bei Herrn Prof. Dr. Lutz Prechelt bedanke ich mich für den Anstoß, meine Diplomarbeit in der Arbeitsgruppe Software Engineering zu schreiben. Ich habe so die Chance bekommen, im Saros-Team mitzuarbeiten.
Zudem gilt mein Dank Frau Prof. Dr. Elfriede Fehr dafür, dass sie sich bereit erklärt hat, die Zweitkorrektur meiner Arbeit zu übernehmen.
Dr. Karl Beecher möchte ich für seine Unterstützung, Anregungen und Betreuung danken. Auch wenn ich ab und zu nach dem passenden Vokabular gesucht habe, war es immer sehr anregend, die zahlreichen Gespräche in Englisch führen zu können.
Des Weiteren danke ich dem Saros-Team für die Hilfsbereitschaft.
Mein ganz besonderer Dank geht an meine Eltern, die mich während der Zeit meines Studiums immer unterstützt haben.
Nicht zuletzt möchte ich mich bei meiner Schwester bedanken für ihre guten Ratschläge und bei meinem Freund für seine Geduld.

Eidesstattliche Erklärung

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

Berlin, den 10. Oktober 2011

Kurzzusammenfassung

Die Dokumentation des Eclipse-Plugins Saros ist im Laufe der Zeit veraltet und unvollständig geworden. Dies bezieht sich zum einen auf die Webseite, die das Projekt vorstellt und Informationen für Benutzer und Entwickler von Saros liefern soll. Zum anderen geht es um die Dokumentation bzw. Kommentierung des Quelltextes.

Die vorliegende Arbeit beschäftigt sich mit der Umstrukturierung der Saros-Webseite und der Ergänzung von Informationen zur Dokumentation, um eine bessere Übersichtlichkeit zu schaffen. Dies geschieht sowohl durch Erweiterung von Inhalten zur Webseite und zu den Quelltext-Kommentaren als auch durch Einführung neuer Vorgehensweisen zum Umgang mit bzw. zu der Aktualisierung der Dokumentation.

Um die Dokumentation zu verbessern, wurde sie durch neue Informationen ergänzt. Dazu gehört beispielsweise eine Architekturübersicht mit Modulbeschreibungen wie auch eine Quelltext-Dokumentation. Die Webseite wurde zudem überarbeitet und neu strukturiert. Zusätzlich wurden Richtlinien festgelegt, wie der Quelltext kommentiert werden soll. Darüber hinaus wurde die Rolle eines Dokumentationsmanagers eingeführt.

Inhaltsverzeichnis

1 Einleitung **7**
1.1 Aufbau dieser Arbeit . . . 7
1.2 Notationen . . . 8

2 Softwaredokumentation: Konzepte und Ideen **9**
2.1 Dokumentationsstruktur . . . 9
2.1.1 Probleme mit der bisherigen Saros-Webseite . . . 9
2.1.2 Gliederung der Webseite . . . 11
2.1.3 Das Hamburger Verständlichkeitsmodell . . . 14
2.1.4 Kategorisierung von Quellcode-Kommentaren . . . 14
2.2 Dokumentationsverhalten . . . 16

3 Umfrage zum aktuellen Stand (im Mai 2011) **19**
3.1 Erstellung des Fragebogens . . . 19
3.2 Auswertung des Fragebogens . . . 20
3.2.1 Allgemeine Frage . . . 20
3.2.2 Fragen zur Webseite . . . 21
3.2.3 Fragen zum Quellcode . . . 24
3.2.4 Fragen zum Dokumentationsverhalten . . . 26
3.2.5 Schlussfolgerung . . . 28
3.3 Interview mit Christopher Oezbeck . . . 28

4 Umstrukturierung und Erweiterung der Dokumentation **31**
4.1 Umstrukturierung der Webseite . . . 31
4.1.1 Erster Ansatz: Webseiten-Generierung mittels Doxygen . . . 33
4.1.2 Zweiter Ansatz: Umstrukturierung der vorhandenen TWiki-Webseiten . . . 34
4.2 Erweiterungen der Technischen Informationen . . . 37
4.2.1 Überblick über die Saros Architektur . . . 37
4.2.2 Exkurs zu SVG-Graphiken . . . 40
4.2.3 Beschreibung der einzelnen Module . . . 41
4.3 Javadoc Generierung mit Hilfe von Doxygen . . . 44
4.4 Javadoc Paketbeschreibungen . . . 47

5 Entwicklung und Einführung von Dokumentations-Richtlinien **51**
5.1 Neue Richtlinien für die Dokumentation . . . 51
5.2 Einführung des *Documentation Issue Trackers* . . . 52
5.2.1 Exkurs: Prioritäten . . . 55
5.3 Einführung der Rolle des Dokumentationsmanagers . . . 56

6 Auswertung **59**
6.1 Auswertung der zweiten Umfrage (im September 2011) 59
6.1.1 Fragen zur Webseite 60
6.1.2 Fragen zum Quellcode 63
6.2 Schlussfolgerung 65

7 Fazit **67**

Anhang **69**

A Fragebögen **71**
A.1 Ankündigung zum Fragebogen (Mai 2011) 71
A.2 Fragebogen zum Thema Dokumentation von Saros (Mai 2011) 71
A.3 Ankündigung zum Fragebogen (September 2011) 78
A.4 Umfrage zur Dokumentation im Saros-Projekt (September 2011) . . . 78

B Modulbeschreibungen **81**
B.1 *Software Architecture* 81
B.2 *Shared Session* 83
B.3 *Eclipse Bridge* 85
B.4 *UI – User Interface* 86
B.5 *Concurrency Control* 88
B.6 *Invitation* 90
B.7 *Network Layer* 92
B.8 *Feedback* 94

C Kommentare: Regeln und Paketbeschreibungen **97**
C.1 Teilabschnitt der *Code Rules*: *Language & Documentation* 97
C.2 Quelltext-Paketbeschreibungen 100
C.2.1 Beschreibung des Pakets `concurrent` 100
C.2.2 Beschreibung des Pakets `concurrent.jupiter` 101
C.2.3 Beschreibung des Pakets `concurrent.management` 102
C.2.4 Beschreibung des Pakets `concurrent.undo` 102

D Regeln zur Behandlung von Dokumentations-Problemen **103**
D.1 *Documentation Issue Procedures* 103
D.2 *Tasks of the DM* 104

Literatur **107**

Abbildungsverzeichnis **111**

1 Einleitung

Saros ist ein Eclipse-Plugin für die Programmierung mit verteilten Beteiligten (engl. Distributed Party Programming – DPP). Es wurde 2006 im Rahmen der Diplomarbeit von Riad Djemili [Dje06] in der Arbeitsgruppe Software Engineering (AG SE) entwickelt und seitdem in verschiedenen weiteren Abschlussarbeiten von Studenten des Instituts für Informatik an der Freien Universität Berlin erweitert. Die Software wird Eclipse-Entwicklern als Open-Source-Projekt zur Verfügung gestellt. Die Webseite zu dem Projekt ist unter der Adresse `http://www.saros-project.org/` zu finden.

Die Webseite beinhaltet die Dokumentation des Saros-Projekts, sowohl für den Nutzer als auch für den Entwickler des Plugins. Im Laufe der Zeit wurde die Webseite vielfach ergänzt und erweitert, was jedoch dazu geführt hat, dass sie unübersichtlich geworden ist und Informationen unvollständig oder nicht mehr aktuell sind. Darüber hinaus fehlen im Quellcode Javadoc-Kommentare, die den Entwicklern des Plugins beim Verstehen des Codes helfen sollen.
Ziel dieser Arbeit ist es, die Dokumentation umzustrukturieren und Informationen zu ergänzen, damit sie übersichtlicher und aktueller wird, sowie Methoden zu entwickeln, die es ermöglichen, dass dies auch in Zukunft so bleibt.

1.1 Aufbau dieser Arbeit

Meine Arbeit gliedert sich in drei Teile: die Recherchen zu Dokumentation und deren Aktualisierung (Kapitel 2 und 3), die Umsetzung der Neustrukturierung und Ergänzungen der Dokumentation von Saros (Kapitel 4 und 5) und die Auswertung der von mir vorgenommenen Änderungen (Kapitel 6 und 7):

Kapitel 2 umfasst meine Konzepte und Ideen zur Dokumentation von Saros. Ich beschreibe den Zustand der bisherigen Webseite und meine Änderungspläne für eine neue Gliederung. Außerdem gehe ich kurz auf die Verständlichkeit von Texten ein. Darüber hinaus beschreibe ich, welche Arten von Kommentaren inwiefern hilfreich sind. Ferner beschäftige ich mich mit Studien zur Pflege bzw. Aktualisierung von Softwaredokumentation und welche Methoden im Saros-Team angewendet werden sollen.

Kapitel 3 beschreibt das Ergebnis einer ersten Umfrage, die ich zu Beginn meiner Arbeit (im Mai 2011) durchgeführt habe, um den Zustand der Saros-Dokumentation zu beurteilen, und das Interview mit Dr. Christopher Oezbeck.

In Kapitel 4 gehe ich auf die Änderungen ein, die ich zur Verbesserung der Dokumentation durchgeführt habe. Dies betrifft die Gliederung und Umstrukturierung der Webseite, das Hinzufügen neuer Informationen wie eine Saros-Architekturübersicht,

sowie eine neu erzeugte Quellcode-Dokumentation. Zudem gehe ich auf die von mir vorgenommenen Paketbeschreibungen ein.

Kapitel 5 befasst sich mit den Verfahren, die ich entwickelt habe, damit die Dokumentation weiterhin aktualisiert wird. Ich beschreibe die Richtlinien, wie mit Problemen bezüglich der Dokumentation umgegangen werden soll sowie den dafür vorgesehenen und neu erzeugten *Documentation Issue Tracker*. Des Weiteren erläutere ich die Aufgaben des von mir neu eingeführten Dokumentationsmanagers.

In Kapitel 6 werte ich meine Neuerungen aus. Dazu habe ich eine zweite Umfrage im September 2011 durchgeführt und die Ergebnisse mit denen der ersten verglichen.

Kapitel 7 beinhaltet ein Resümee darüber, was ich während meiner Arbeit erreicht habe und wie man die Pflege der Dokumentation fortsetzen kann.

1.2 Notationen

Für meine Diplomarbeit habe ich folgende typografische Konventionen verwendet:

- Durch *Kursivschrift* werden Begriffe hervorgehoben, die Webseiten-Titel oder -überschriften bezeichnen, wie beispielsweise *Developer Guide* oder *Bug Tracker*. Bei den Modulbeschreibungen werden zusätzlich die englischen Modulbezeichnungen bzw. Komponenten hervorgehoben (z. B. *Session* oder *Jupiter*).
- `Maschinenschrift` wird für Bezeichnungen des Quellcodes verwendet. Alle genannten Pakete, Interfaces und Klassen von Saros beziehen sich auf den Namensraum `de.fu_berlin.inf.dpp`.

  ```
  Quellcode-Abschnitte werden zwischen zwei horizontale
  Linien gesetzt.
  ```

- Gemäß den Regeln der deutschen Grammatik und den allgemein üblichen Vorgaben zum wissenschaftlichen Zitieren wird bei der Wiedergabe von Aussagen anderer Autoren (indirekte Rede) der Konjunktiv verwendet. Falls aus dem Satz ersichtlich ist, dass die Aussage von einem bestimmten Autor stammt, wird der Indikativ verwendet (vgl. [Bed04], [vdD09, S. 533]).

2 Softwaredokumentation: Konzepte und Ideen

Zu Beginn meiner Arbeit war die Dokumentation von Saros unvollständig und veraltet. Im Quellcode fehlten viele Kommentare und die Webseite zum Projekt war unübersichtlich. Dies sollte im Laufe meiner Arbeit verändert werden. Da eine gute Dokumentation hilft, die Software und den Umgang mit ihr zu verstehen, soll diese Arbeit zur Vervollständigung, Ergänzung und Übersichtlichkeit beitragen.

> „Document quality is as important as program quality. Without information on how to use a system or how to understand it, the utility of that system is degraded. [...] Producing good documents is neither easy nor cheap, and many software engineers find it more difficult that [sic] producing good quality programs.“ [Som02, S. 7]

Um die Dokumentation des Projekts zu verbessern, habe ich zwei Bereiche näher untersucht: die Dokumentationsstruktur und das Verhalten der Entwickler bezüglich des Aktualisierens der Dokumentation (Dokumentationsverhalten). Durch eine überarbeitete Struktur soll die vorhandene Dokumentation übersichtlicher werden und der Leser sich somit besser zurechtfinden. Zusätzlich müssen fehlende Informationen ergänzt und falsche bzw. nicht mehr aktuelle Angaben überarbeitet werden. Um zu gewährleisten, dass die Dokumentation auch in Zukunft aktualisiert wird, muss auch das Dokumentationsverhalten der Entwickler untersucht und Methoden zu dessen Verbesserung entwickelt werden.

In diesem Kapitel werde ich die Konzepte und Ideen vorstellen, die ich auf Basis meiner Recherchen zu Dokumentationsstrukturen und -verhalten entwickelt habe.

2.1 Dokumentationsstruktur

Softwaredokumentation lässt sich in die externe Dokumentation für Softwareanwender einerseits und die interne Entwicklungsdokumentation andererseits gliedern. Die externe Anwenderdokumentation soll dem Benutzer die Software näher bringen und ihm helfen, diese zu benutzen [Grü07].
Die Entwicklungsdokumentation umfasst verschiedene Bereiche. Zum einen geht es um die Beschreibung der Arbeitsabläufe wie dem Release Schedule, zum anderen sind auch technische Beschreibungen wichtig, die den Entwicklern helfen, die Software zu verstehen und ihren Aufbau nachzuvollziehen.

2.1.1 Probleme mit der bisherigen Saros-Webseite

Die Saros-Webseite hat drei Aufgaben zu erfüllen: sie präsentiert das Projekt nach außen, sie enthält hilfreiche Informationen für die Verwendung von Saros und sie soll den Entwickler am Saros-Projekt bei seiner Arbeit anleiten bzw. unterstützen.
Somit richtet sich die Webseite an zwei Gruppen von Programmierern: diejenigen,

die mit Eclipse arbeiten und das Plugin zum Programmieren mit verteilten Beteiligten verwenden wollen, und diejenigen, die das Plugin weiterentwickeln.
Die Anwender von Saros brauchen eine Beschreibung, aus der hervorgeht, worum es sich bei dem Plugin handelt und wie sie es verwenden können. Für die (Weiter-) Entwickler von Saros spielen neben dieser Beschreibung vor allem die Informationen über einzelne Arbeitsabläufe während der Entwicklung und Richtlinien, die eingehalten werden sollen, eine entscheidende Rolle.
Des Weiteren können den Entwicklern neben den Informationen auf der Webseite die Kommentare im Quellcode beim Programmieren bzw. Verstehen des Codes helfen, sofern sie ausreichend vorhanden sind. Hierauf gehe ich in Abschnitt 2.1.4, S. 14 näher ein.

Abbildung 1 – *Navigation der Saros-Webseite zu Beginn meiner Arbeit (im April 2011)*

Auf der bisherigen Webseite war es aus Sicht eines Anwenders oder Entwicklers schwierig, einen Überblick über die für ihn relevanten Themen zu bekommen. Dies begann bereits mit der Navigation, die eine Strukturübersicht über die Inhalte der Unterseiten lieferte. Alle Inhaltstitel waren in einem einzigen zusammenhängenden Block hintereinander aufgeführt (siehe Abbildung 1). Somit war keine klare Trennung ersichtlich, welche der Unterseiten für den Anwender und welche für den Entwickler von Saros bestimmt sind.
Außerdem waren fast alle relevanten Informationen für den Entwickler auf einer einzigen Seite zu finden, dem *Developer Guide*. Dies hatte zwar den Vorteil, dass er auch alles Wichtige dort fand, aber den großen Nachteil, dass diese Seite sehr umfangreich und damit unüberschaubar war.

> „Zu achten ist hier im besonderen Maße auf die ‚Lesefähigkeit‘, das heißt eine sinnvolle Einteilung der Inhalte. Unnötiges Scrollen – das führt schnell zu einem hohen Maß an Unübersichtlichkeit – ist zu vermeiden.“ [Kry11, S. 20]

Somit habe ich mich dazu entschieden, die Navigation der Webseite zu gruppieren, um dem Leser die Orientierung zu erleichtern und den *Developer Guide* thematisch aufzugliedern, um den Bereich für die Entwickler übersichtlicher zu strukturieren. Auch Godehard Pötter empfiehlt für eine bessere Orientierung eine konsequente Trennung der Inhalte in Kapitel. Des Weiteren spricht er sich für eine klare Gliederung und eindeutige Formulierungen aus, um einen Text verständlicher zu gestalten (vgl. [Pöt94, S. 118, 53]).

2.1.2 Gliederung der Webseite

Jede der beiden Zielgruppen der Software – Anwender und Entwickler – braucht einen optisch klar abgegrenzten Bereich, in dem sich die jeweils für sie wesentlichen Themen befinden. Als dritte Kategorie kommen noch allgemeine Informationen hinzu, die sich an jeden Besucher der Webseite wenden.
Entsprechend der von Dietrich Juhl beschriebenen Leistungsbeschreibung sollte die Startseite einen ersten Eindruck darüber verschaffen, was Saros ist und was man damit machen kann. Diese ermögliche es einem Nutzer der Seite, nicht nur einschätzen zu können, um welche Art von Projekt es sich handelt und wozu die Software dient, sondern könne ihn auch durch den beschriebenen Nutzen motivieren, die Software – hier Saros – zu verwenden (vgl. [Juh05, S. 25–31]). Dabei können kleine vorgestellte Anwendungsbeispiele, wie die auf der bisherigen Webseite von Saros vorhandenen Szenarien, helfen. Eine gesammelte Auflistung aller Features, die das Eclipse-Plugin bietet, kann dem Leser auch ein besseres Verständnis für die Software geben. Eine solche Liste fehlte in dieser Form noch auf der Webseite.

Die einzelnen Unterseiten der bisherigen Webseite werden für die Neustrukturierung wie folgt den drei Kategorien zugeordnet:

Allgemeiner Bereich: Hierzu zähle ich die Startseite (*Home*), die *Screenshots*, die Kontaktinformationen (*Contact*) und die Seite zu Publikationen und Forschungen (*Research on Saros*).

Anwenderbereich: Der Anwender braucht Anleitungen zur Installation (*Installation & Set-up*) und für die erste Verwendung (*Getting Started*), um sich in der Software zurechtzufinden. Die Informationen zum Umgang (*How to be Happy with Saros*) sowie die *FAQs* sind ebenfalls Teil des Anwenderbereichs.

Entwicklerbereich An die Entwickler von Saros richten sich der *Developer Guide*, die *Technical Information* wie auch die beiden externen Links zum *Bug Tracker*[1] und der Projekt-Präsentation bei Sourceforge[2] (*Saros @sourceforge*).

Um den Bereich für den Entwickler übersichtlicher zu gestalten, plante ich – wie eingangs erwähnt –, den *Developer Guide* auf verschiedene Unterseiten aufzuteilen, deren Inhalte jeweils thematisch zusammengehören. Als ich zum Saros-Team kam, bestand der *Developer Guide* aus einer langen Liste von verschiedenen Handlungsanweisungen und Richtlinien, ohne dass eine tatsächliche Gliederung oder thematische Gruppierung erkennbar war (siehe Abbildung 2a auf der nächsten Seite). Da am Projekt Saros aktiv entwickelt wird, unterliegt auch die Webseite stetiger Veränderung. So gab es schon Änderungen am *Developer Guide*, bevor ich beginnen konnte, die Webseite zu bearbeiten. Für die Umstrukturierung im Rahmen dieser Diplomarbeit lag somit eine neu sortierte Auflistung und thematische Gliederung vor, an der ich mich orientieren konnte (siehe Abbildung 2b auf der nächsten Seite):

Getting Started with Developing enthielt die Anleitung zum Einrichten und Auschecken von Saros für die Entwicklung an der Software. Weiter fanden sich in dem Abschnitt die Richtlinien, an die sich der Entwickler halten soll. Zu diesen gehörte unter anderem die Anleitung, wie er Quellcode anderer Entwickler begutachten und selbst Code in das Saros-Projekt einbringen soll.
In Zukunft soll der erste Teil zum Einrichten der Entwicklungsumgebung Eclipse und zum Auschecken des Codes separat als Installationsanweisungen (*Set-up and Check-out Saros*) auf eine eigene Seite gestellt werden. Die restlichen Punkte dieses Abschnitts sollen als Richtlinien (*Guidelines*) zusammen bleiben.

Testing umfasste alle Anleitungen zum Testen (manuelles Testen, JUnit-Test, usw.) und zum Debugging.

Bug tracking procedures enthielt die Informationen, wie man mit Problemen in der Software umgehen soll. Dazu gehörte auch die Prioritäten-Liste, um die ‚Bugs' kategorisieren zu können.

Usability Engineering war ein erst kürzlich neu hinzugekommener Abschnitt darüber, wie man die Software anwenderfreundlicher gestalten kann.

[1] http://sourceforge.net/tracker/?group_id=167540&atid=843359
[2] http://sourceforge.net/projects/dpp/

Developer Guide

If you're a student and just starting with development Started guide, which will answer most of your questio

- Getting started with Developing
- Setting run configurations
- Coding Conventions
- How to contribute code to Saros
- How to Peer Review a Patch
- Developing in a branch
- About Refactor->Rename
- Get source code from subversion repository
- Java packages
- Bug tracking procedures
  - Bug priorities
- Our Release Schedule
  - How to announce a new release
- How to branch
- How to merge
- How to create a ssh key for Sourceforge on Windows
- How to create a new release
- How to make a re-release
- Updating Dependencies
  - How to build Limewire version 4 with java 1.5
- How to create an Architecture Diagram
- How to run a manual test
- How to debug
- How to run the JUnit test-cases
- How to run test coverage analysis locally

(a) *Ungegliedertes Inhaltsverzeichnis des* Developer Guides *im März 2011.*

Developer Guide

If you're a student and just starting with development Started guide, which will answer most of your questio

- Getting started with Developing
  - Setting run configurations
  - Coding Conventions
  - How to contribute code to Saros
  - How to Peer Review a Patch
  - Developing in a branch
  - About Refactor->Rename
  - Get source code from subversion repository
  - Java packages
- Testing
  - How to run a manual test
  - How to debug
  - How to run the JUnit test-cases
  - How to run test coverage analysis locally
- Bug tracking procedures
  - Bug priorities
- Usability Engineering
- Our Release Schedule
  - How to announce a new release
  - How to branch
  - How to merge
  - How to create a ssh key for Sourceforge on Windows
  - How to create a new release
  - How to make a re-release
  - Updating Dependencies
  - How to build Limewire version 4 with java 1.5
- How to create an Architecture Diagram

(b) *Thematisch gegliedertes Inhaltsverzeichnis des* Developer Guides *im Mai 2011.*

Abbildung 2 – *Inhaltsverzeichnis des* Developer Guides *zu Beginn meiner Arbeit.*

Our Release Schedule beschrieb alle Abläufe, die für ein Release der Software nötig sind:
Von Saros wird, soweit es möglich ist, monatlich eine neue Version herausgegeben. In der Release-Woche werden an die Entwickler die Aufgaben des Test- und Release-Managers vergeben. Der Testmanager soll dabei vor allem die Änderungen an der Software testen und ihre Funktionsfähigkeit überprüfen. Der Release-Manager ist dafür verantwortlich, die Software für den Release vorzubereiten und, wenn alle Tests zufriedenstellend waren, die neue Softwareversion herauszugeben. Für jeden dieser beiden Aufgabenbereiche gibt es jeweils einen Assistenten, der die Abläufe kennt und den Entwickler, der die Managerrolle übernommen hat, unterstützen bzw. ihn in die Arbeitsabläufe einführen soll.

Die Inhalte der bisherigen Seite *Developer Guide* sollen somit auf sechs Unterseiten verteilt werden: *Set-up & Check-out*, *Guidelines*, *Testing*, *Bug tracking procedures*, *Usability Engineering* und *Our Release Schedule*.
Auf die Umsetzung der Webseiten-Umstrukturierung gehe ich in Abschnitt 4.1, S. 31

näher ein. In Abbildung 11, S. 35 zeige ich die neue, umgruppierte Navigation im Vergleich zu der alten.

2.1.3 Das Hamburger Verständlichkeitsmodell

Um die Verständlichkeit von Texten zu analysieren, hat Pötter das Hamburger Verständlichkeitsmodell[3] vorgeschlagen. Dabei gebe es vier Dimensionen, anhand derer der Grad der Verständlichkeit eines Textes ausgedrückt werden kann. Durch ein Beurteilungsfenster lasse sich die Verständlichkeit eines Textes grob einstufen. Es würden jeweils gegensätzliche Eigenschaften gegenübergestellt und der Text in Hinblick auf diese Eigenschaften dahingehend bewertet, wie stark diese auf den Text zutreffen. Die Dimensionen lauten:

- Einfachheit – Kompliziertheit
- Gliederung/Ordnung – Unübersichtlichkeit
- Kürze/Prägnanz – Weitschweifigkeit
- zusätzliche Stimulanz – keine zusätzliche Stimulanz

Ein Text ist laut Pötter dann gut verständlich, wenn er einfach geschrieben ist und eine gute Gliederung bzw. Ordnung besitzt. Weiterhin solle der Text weder extrem knapp noch besonders weitschweifig sein – die Mitte aus diesen Eigenschaften sei optimal. Die zusätzliche Stimulanz hänge von den anderen Faktoren bzw. Eigenschaften ab: Anregende Zusätze könnten in einem ungegliederten Text verwirrend, in einen gegliederten Text allerdings motivierend wirken.
Da sich dieses Bewertungsmodell dazu eignet, existierenden Text zu überprüfen, habe ich es für meine Umfragen (in Abschnitt 3, S. 19 und Abschnitt 6, S. 59) übernommen, in denen ich feststellen wollte, wie die Inhalte der Webseite bzw. der Quelltext von den Entwicklern eingestuft werden.

2.1.4 Kategorisierung von Quellcode-Kommentaren

Neben der für den Entwickler wichtigen Dokumentation auf der Webseite gibt es die Dokumentation in Form von Kommentaren im Quellcode. Auch wenn sich Steve McConnell dafür ausspricht, dass der Hauptbeitrag einer guten Quellcode-Dokumentation in einem guten Programmierstil liegt und nicht in den Kommentaren, erklärt er dennoch, wie man effektive Kommentare schreibt (vgl. [McC04, chapter 32: Self-Documenting Code]). Dazu klassifiziert er sie in sechs Kategorien und beschreibt sie folgendermaßen:

[3] Inghard Langer, Friedemann Schulz von Thun und Reinhard Tausch: *Sich verständlich ausdrücken.* Ernst Reinhardt Verlag, München, 2. Auflage, 1981. Zitiert in [Pöt94, S. 164–167].

Wiederholungen des Codes liefern dem Leser keine neuen Informationen, sondern geben den Code nur in anderen Worten wieder.

Erklärungen des Codes werden häufig verwendet, wenn er unübersichtlich ist. In diesem Fall sei es besser, den Code neu zu schreiben, als ihn zu erklären. Erklärungen seien jedoch dann sinnvoll, wenn sie knifflige oder sensible Codeabschnitte verdeutlichen.

Markierungen im Code sind Hinweise für den Entwickler, die anzeigen, was noch erledigt werden muss. Sie seien nicht dafür vorgesehen, im Code zu verbleiben. Um sie leichter auffinden zu können, sollten sie vereinheitlicht werden, z. B. durch Voranstellen der Bezeichnung „ToDo“.

Zusammenfassungen des Codes erfassen den Inhalt eines Codeabschnitts in wenigen Sätzen. Sie seien besonders für Entwickler hilfreich, die den Code nicht selbst geschrieben haben. Da sie sich schneller lesen ließen als der eigentliche Code, ließe sich ein gesuchter Codeabschnitt schneller finden.

Beschreibungen der Absicht des Codes erklären den Zweck, den der Code erfüllen soll. Das helfe anderen Entwicklern zu verstehen, was der Autor des Codes beabsichtigt hat. Dadurch werde auch die Gefahr minimiert, dass ein Entwickler durch Modifizieren des Codes den ursprünglichen Zweck ändert.

Informationen, die nicht durch Code ausgedrückt werden können, gehören zur letzten Kommentar-Kategorie. Zu ihnen zählten u. a. Versionsnummern, Angabe des Autors, Urheberrecht oder Notizen über das Design des Codes.

Nach McConnell gibt es nur drei akzeptable Arten von Kommentaren: Zusammenfassungen des Codes, Beschreibung der Absicht des Codes und Informationen, die nicht durch Code ausgedrückt werden können.
Die Kategorisierung nach den sechs Typen habe ich sowohl für meine Umfragen (in Abschnitt 3, S. 19 und Abschnitt 6, S. 59) verwendet, als auch für die Ergänzung der *Coding Guidelines and Rules* (*Code Rules*)[4] herangezogen (siehe dazu Abschnitt 5.1, S. 51 und Anhang C.1, S. 97).

[4] https://www.inf.fu-berlin.de/w/SE/CodeRules

2.2 Dokumentationsverhalten

Eine übersichtliche Struktur ist wichtig für eine gute Dokumentation. Doch ohne einen vernünftigen Umgang mit ihr kann eine Dokumentation auf Dauer nicht aktuell gehalten werden. Verschiedene Wissenschaftler haben sich mit dem Thema der Softwaredokumentation und ihrer Relevanz beschäftigt, wobei allgemein festgestellt wird, dass die Dokumentation wichtig ist und gepflegt werden muss. Es gibt jedoch nur wenige Untersuchungen darüber, wie man sie tatsächlich aktuell halten kann.

Timothy C. Lethbridge et al. stellen hierzu in ihrer Studie fest, dass Softwareentwickler allgemein selten Dokumentationen pflegen (mit Ausnahme der von ihnen bezeichneten „testing and quality documentation“). Eine Dokumentation werde aber öfter aktualisiert, wenn diese in den Arbeitsprozess eingebunden sei. Es herrsche zudem die verbreitete Meinung, Softwaredokumentation sei immer veraltet, gemessen an dem aktuellen Stand des Softwaresystems. Ihrer Studie zufolge vertrauen Entwickler einer Dokumentation eher, wenn sie ein bestimmtes Feature oder eine Systemarchitektur beschreibt. Je abstrakter eine Beschreibung sei, desto eher würde sie als nützlich und richtig betrachtet, selbst wenn sie veraltet sei (vgl. [LSF03]).

Nach Joachim Bayer und Dirk Muthig gibt es verschiedene Gründe, wieso häufig erst nach Abschluss der Implementierung und oft auch nur das äußerste Minimum dokumentiert wird. Zum einen gebe es bei den meisten Organisationen keinen etablierten Dokumentationsprozess, der die Dokumentation als einen integralen Bestandteil der Entwicklung ansieht. Zum anderen werde die Dokumentation häufig als nicht so wichtig angesehen wie das tatsächliche Produkt (der Quellcode), so dass die Motivation, eine Dokumentation zu erstellen und auch zu pflegen, bei den Entwicklern sinke. Bayer und Mutig zufolge kann für den Entwickler zudem der Eindruck entstehen, dass er zu viel bzw. unnötigerweise dokumentiere, wenn nicht klar werde, wem die Dokumentation nützt bzw. von wem sie hinterher verwendet wird (vgl. [BM06]).

Wie wichtig die Dokumentation ist, stellen auch Marcello Visconti und Curtis R. Cook fest. Dokumentation sei eine Schlüsselkomponente der Softwarequalität; eine Verbesserung des Dokumentationsprozesses werde einen starken Einfluss auf die Verbesserung der Softwarequalität haben. Die meisten Organisationen stimmten überein, dass es ein Problem bezüglich des Dokumentationsprozesses gebe, aber es werde wenig getan, daran etwas zu ändern. Die große Herausforderung sehen Visconti und Cook also darin, sich ernsthaft zu bemühen, den Dokumentationsprozess zu verbessern, indem explizite Grundsätze zur Prozessverbesserung angenommen und umgesetzt werden (vgl. [VC02]).

Software Redokumentation nach dem Schema von Alexander J. Rostkowycz et al. soll zu einer schrittweisen Erneuerung bzw. Verbesserung der Dokumentation führen. Teilabschnitte des Quellcodes, an denen gearbeitet werde, müssten vom Entwickler nach Abschluss des Änderungsprozesses hinreichend dokumentiert werden. Zusätzlich sollten auch Teile des Codes überarbeitet werden, die der Entwickler genutzt, aber

nicht verändert hat. Für beide Bereiche habe der Entwickler kürzlich ein Verständnis entwickelt und sei somit in der Lage, diese zu dokumentieren. Diese Redokumentation solle fest in einen Ablaufplan zur Änderung des Codes eingegliedert werden – zum Beispiel als letzter Teil.

> „When consistently applying the technique, documentation is garanteed to be up-to-date. [...] troublesome components are garanteed to receive a lot of needed documentation, since they will be frequently revisited.“ [RRM04, (S. 10)]

Durch das inkrementelle Vorgehen muss nach Rostkowycz et al. jeweils nur ein kleiner Teil überarbeitet werden. So werde der Entwickler nicht auf einmal mit Arbeit überhäuft und geneigter sein, diese zu erledigen. Die schrittweise Redokumentation könne sehr effizient sein, wenn sie diszipliniert durchgeführt werde.
Rostkowycz et al. ermittelten in ihrer vierjährigen Studie nicht nur, dass sich der durchschnittliche Redokumentationsaufwand um 71% verringerte, sie stellten auch fest, dass sich der Programmieraufwand („design + implementation + testing“) nach 18 Monaten so weit reduziert hat, dass sich der Redokumentationsaufwand ausgezahlt hat.
Nach Beendigung ihrer Studie mussten die Autoren jedoch auch erkennen, dass es immer noch undokumentierte Teile gab. Abschnitte, an denen nicht gearbeitet wurde, seien auch nicht dokumentiert worden. Außerdem könne diese Technik nichts daran ändern, dass es Entwickler gebe, für die es grundsätzlich eine Belastung sei, Dokumentation zu schreiben (vgl. [RRM04]).
Darüber hinaus hatten Lethbridge et al. in ihrer oben genannten Studie angemerkt, dass nicht belegt sei, ob Projekte davon profitieren, wenn die Softwareentwickler dazu gezwungen würden, die Dokumentation akribisch auf dem neuesten Stand zu halten.

Trotz der Einschränkungen halte ich den Ansatz von Rostkowycz et al. für sehr gut. Da es bei der Saros-Software viele Fehlstellen bezüglich der Dokumentation gibt, erscheint eine schrittweise Redokumentation sinnvoll. Es wird kaum möglich sein, die komplette Dokumentation (Webseite und Quellcode-Kommentare) auf einmal zu überarbeiten. Nicht nur, weil jeder Bereich erst eine Einarbeitung erfordert, sondern auch, weil die Software von Studenten geschrieben wird, die ihre Abschlussarbeiten bei der AG SE anfertigen und deren Fokus daher verstärkt auf der eigenen Arbeit liegt. Sie sind keine reinen Softwareentwickler in einem festen Team, in dem man die Implementierungs-Phase für einen längeren Zeitraum unterbrechen und sich nur auf die Redokumentation des Projekts konzentrieren könnte (sofern dies die Projektplanung zeitlich zulässt). Ein schrittweises Vorgehen ermöglicht es daher, die Dokumentation nach und nach zu überarbeiten. Der Aufwand dürfte sich im Laufe der Zeit reduzieren, so dass schlussendlich nur noch die Teile dokumentiert werden müssen, an denen sowieso gearbeitet wird.

Im Saros-Team gibt es die zuvor erwähnten Aufgabenbereiche des Test- und Releasemanagers. Es bietet sich daher an, einen weiteren Verantwortungsbereich – den des Dokumentationsmanagers – einzuführen, dem analog zu den anderen Bereichen

ebenfalls ein Entwickler bzw. eine Entwicklergruppe zugeordnet wäre. Der Dokumentationsmanager kann feststellen, wo die Dokumentation aktualisiert werden soll und überprüfen, ob dies geschehen ist. Wenn er Mängel in der Dokumentation findet, kann er sie dem Team mitteilen, damit diese beseitigt werden.
Nachdem ich diesen Vorschlag dem Team gemacht und positive Resonanz erhalten hatte, habe ich die Frage nach dem Dokumentationsmanager auch in meine erste Umfrage (in Abschnitt 3 auf der nächsten Seite) aufgenommen.

3 Umfrage zum aktuellen Stand (im Mai 2011)

Neben den referierten Möglichkeiten zur Strukturierung und Aktualisierung der Dokumentation, wie die von Rostkowycz et al. untersuchte schrittweise Redokumentation, war mir die Meinung der Entwickler im Saros-Team wichtig. Um festzustellen, wo die Schwächen und Probleme in der Saros-Dokumentation (sowohl bei der Webseite als auch im Quellcode) sind und wie das Dokumentationsverhalten zur Zeit aussieht, habe ich einen Fragebogen (siehe Anhang A, S. 71) entwickelt und ihn an die Saros Mailingliste geschickt.

3.1 Erstellung des Fragebogens

Um einschätzen zu können, über welche Wege Informationen zu Saros genutzt werden, hat mich im Fragebogen zunächst interessiert, woher die Teilnehmer ihre Informationen über Saros bezogen haben; und zwar einerseits zum Zeitpunkt ihrer Einarbeitung in das Projekt und andererseits zum Zeitpunkt der Umfrage.
Basierend auf meiner Vorgehensweise bei der Recherche habe ich die restlichen Fragen in drei Blöcke eingeteilt: Fragen zur Webseite, zum Quellcode (bzw. zu den Kommentaren darin) und zum Dokumentationsverhalten.

Zur Webseite: Aus vorhergehenden Gesprächen mit einigen Entwicklern hat sich herauskristallisiert, dass nicht jeder die Webseite benutzt oder aber nur kurz angesehen hat. Hinzu kommt, dass die mittlerweile unübersichtlich gewordene Webseite die Entwickler nicht zu einer Beschäftigung mit ihr anregte. Deshalb hat mich interessiert, welche Themen angenommen werden, d. h. als wichtig bzw. hilfreich empfunden und genutzt werden, und für welche dies überhaupt nicht zutrifft. Weiter wollte ich herausfinden, welche Themen fehlen und somit unbedingt der Webseite hinzugefügt werden sollten.

Um festzustellen, wie die Teilnehmer den Inhalt und die Gliederung der Webseite vor der Umstrukturierung sehen, und die Möglichkeit eines Vergleichs zu haben, habe ich das von Godehard Pötter beschriebene Hamburger Verständlichkeitsmodell [Pöt94, S. 164–167] von Abschnitt 2.1.3, S. 14 herangezogen. Um herauszubekommen, wieso einige Themen als hilfreich empfunden wurden und andere nicht, habe ich mich dazu entschlossen, jeweils drei Bereiche einschätzen zu lassen: die Webseite ganz allgemein, die Themen, die als besonders hilfreich empfunden wurden, und die Themen, die als nicht hilfreich empfunden wurden.

Zum Quellcode: Um die Qualität der vorhandenen Kommentare beurteilen zu können, habe ich mich an der Klassifizierung von Kommentaren von Steve McConnell [McC04, S. 786–788] orientiert (siehe hierzu Abschnitt 2.1.4, S. 14). Dazu habe ich eine Einteilung vorgenommen zwischen den Kommentaren an den Teilen, an denen

der Entwickler selbst entwickelt, und den Kommentare, die er gelesen hat, um fremd entwickelten Quellcode zu verstehen. Auf diese Weise wollte ich sehen, wie verständlich fremde Kommentare sind und wie die eigenen Kommentare subjektiv beurteilt werden.

Zum Dokumentationsverhalten: Der letzte Fragenabschnitt bezog sich auf das Dokumentationsverhalten der Entwickler. Aufgrund der Erkenntnisse der von Lethbridge et al. durchgeführten Studie [LSF03, S. 38] gehe ich davon aus, dass sich eine Verbesserung der Dokumentation am besten erreichen lässt, wenn Vorgaben oder Richtlinien, die eingeführt werden müssen, von den Entwicklern selbst vorgeschlagen bzw. abgesegnet worden sind. Zu zwei im Ansatz bereits entwickelten und besprochenen Ideen sollte die Meinung der Entwickler erfasst werden: zur Einführung eines Dokumentationsmanagers und zur Einbindung der Webseiten-Information in den Quelltext (siehe hierzu Abschnitt 3.2.4, S. 26).

3.2 Auswertung des Fragebogens

Von zwölf Personen erhielt ich Antworten auf meinen Fragebogen, darunter sieben von Studenten, die ihre Abschlussarbeit bei der AG SE schreiben bzw. geschrieben haben und vier Antworten von anderen AG SE Mitarbeitern, die mit Saros arbeiten. Eine Antwort kam von einem Studenten, der sich auf der Benutzerseite mit Saros beschäftigt und versucht hat, Saros anderen Studenten näherzubringen.

Nicht bei allen Antworten ließen sich klare Tendenzen bzw. Aussagen erkennen. Aus diesem Grund werde ich in meiner Auswertung solche Antworten nicht näher betrachten.

Die Teilnehmer werde ich später in Programmierer (acht Personen) und Nicht-Programmierer (vier Personen) kategorisieren. Dabei verstehe ich unter den Programmierern diejenigen, die am Quellcode von Saros arbeiten bzw. gearbeitet haben und unter Nicht-Programmierern alle anderen, die an der Umfrage teilgenommen haben.

3.2.1 Allgemeine Frage

Auf die allgemeine Frage zu Beginn meines Fragebogens, welche Informationsquellen bezüglich Saros die Teilnehmer nutzten, zeigte sich, dass es unterschiedliche Herangehensweisen gab, die zu einem Einstieg in das Projekt geführt hatten: Zu gleichen Teilen hatten die zwölf Teilnehmer der Umfrage ihre ersten Informationen aus der Webseite, dem Quellcode oder von anderen Entwicklern bzw. AG SE Mitarbeitern bezogen.
Bei der aktuellen Arbeit zeigte sich hingegen, dass vor allem die Programmierer ihre Informationen hauptsächlich aus dem Quellcode bzw. von anderen Entwicklern

bekamen, während die Nicht-Programmierer angaben, ihre Informationen aus der Webseite oder von den Entwicklern erhalten zu haben.

3.2.2 Fragen zur Webseite

Meine nächsten Fragen behandelten die Webseite von Saros. Generell ist die Webseite den Teilnehmern bekannt, wobei drei der vier AG SE Mitarbeiter angaben, an ihr mitgearbeitet zu haben.

Folgende Themen wurden als wichtig bzw. hilfreich betrachtet:

- *Developer Guide* bzw. der Release-/Testplan daraus
- *Technical Information*
- Installationsanweisungen und *Getting Started*
- der Link zum *Bug Tracker*
- der Link zu den *Code Rules*

Auch die Frage nach den Themen, die genutzt wurden, hat dies noch einmal bestätigt. Als nicht hilfreich wurde lediglich von zwei Teilnehmern *How to be Happy with Saros* aufgeführt.

Die Frage „Welche Themen fehlen dir?“ sollte Auskunft darüber geben, wo die Webseite inhaltlich ergänzt werden soll. Fehlende Themen wurden wie folgt benannt:

- eine Übersicht/Einführung, was Saros ist bzw. bietet
- die wesentlichen Neuerungen/Features von Saros
- eine Beschreibung der Architektur, des Pico-Containers und der Netzwerkschicht
- Paket- bzw. Klassenbeschreibungen

Um einen leichten Einstieg in Saros zu bekommen, wurde an der Webseite stark bemängelt, dass diese zu unübersichtlich sei. Dies bestätigte auch noch einmal die erste Frage, bei der die Programmierer angaben, zum Zeitpunkt der Umfrage ihre Informationen über Saros nicht von der Webseite sondern aus dem Quellcode bzw. von anderen Entwicklern zu beziehen (siehe Abschnitt 3.2.1 auf der vorherigen Seite).

Bei den sich anschließenden Fragen zur Text-Verständlichkeit, auf die ich im Folgenden näher eingehe, sollten jeweils die Webseite allgemein als auch die hilfreichen sowie die nicht hilfreichen Themen einzeln beurteilt werden. Zur Webseite allgemein und zu den hilfreichen Themen gab es jeweils elf bzw. zehn Antworten pro Frage. Da kaum einer der Teilnehmer Themen als nicht hilfreich erachtet hat, gab es in diesem Bereich nur sechs bzw. sieben Antwortende.

In der Beantwortung der Frage, ob der Text gegliedert ist bzw. eine Ordnung hat oder unübersichtlich bzw. zusammenhangslos ist, ergab sich keine klare Tendenz. Die nicht hilfreichen Themen wurden aber als unübersichtlicher eingestuft als die hilfreichen Themen (siehe Abbildung 3). Hilfreiche Themen wurden eher am erwarteten Ort gefunden als nicht hilfreiche Themen, jedoch zeichnete sich in den Antworten nicht klar ab, ob Informationen schnell wiedergefunden werden können oder nicht.

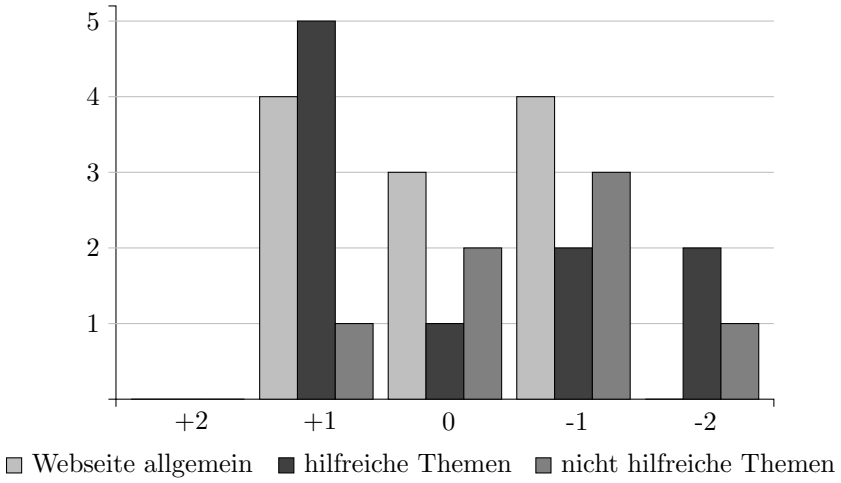


Abbildung 3 – *Antworten zur Frage 10: „Der Text ist gegliedert/hat eine Ordnung. (+2) ↔ Der Text ist unübersichtlich/zusammenhangslos. (-2)“*

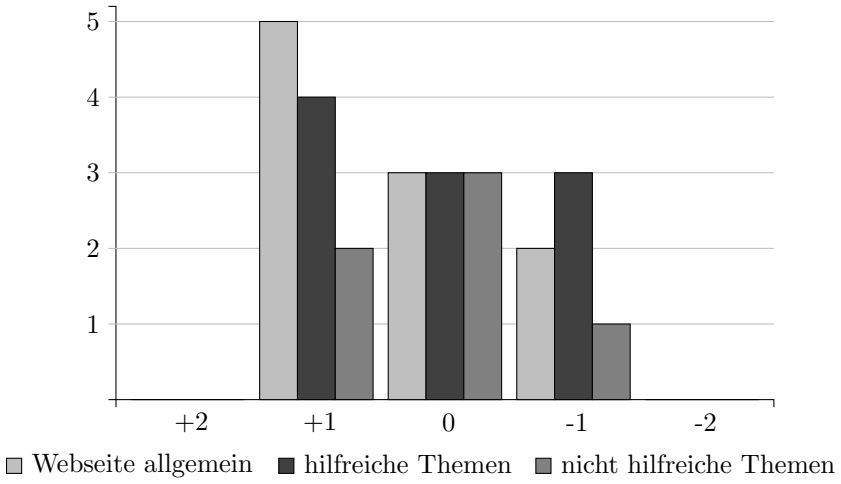


Abbildung 4 – *Antworten zur Frage 13: „Der Text ist einfach geschrieben. (+2) ↔ Der Text ist kompliziert geschrieben. (-2)“*

Der Text der Webseite wurde von den Teilnehmern weder als einfach noch als kompliziert geschrieben beurteilt (siehe Abbildung 4). Dabei gab es keinen signifikanten Unterschied zwischen den hilfreichen und nicht hilfreichen Themen. Allerdings kann

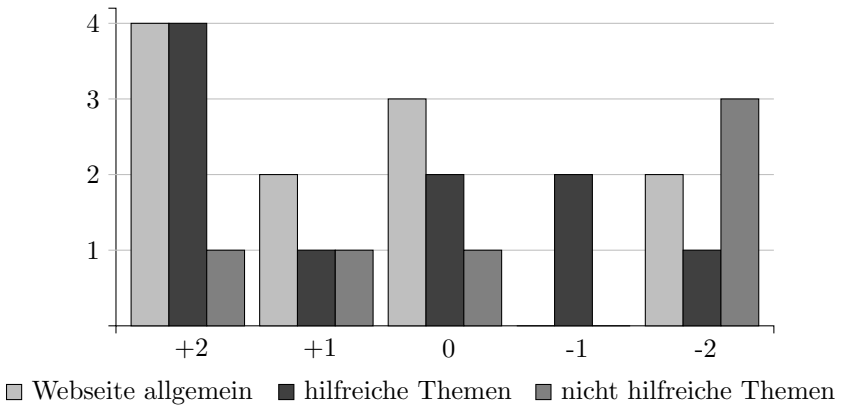


Abbildung 5 – *Antworten zur Frage 14: „Der Text ist kurz und prägnant. (+2) ↔ Der Text ist weitschweifig gefasst. (-2)“*

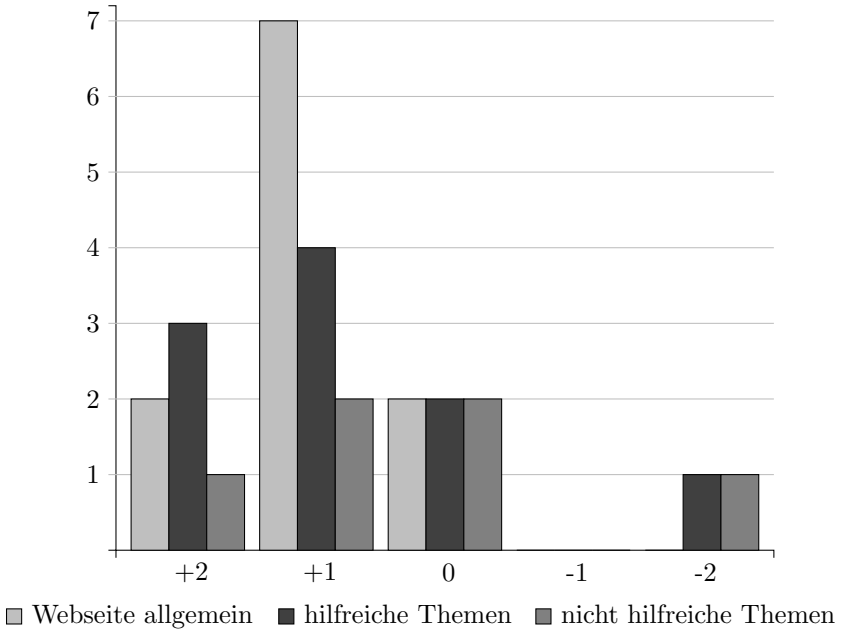


Abbildung 6 – *Antworten zur Frage 15: „Der Text ist aussagekräftig. (+2) ↔ Der Text enthält keine relevanten Informationen. (-2)“*

ich keine Aussage darüber treffen, in wie weit das Verständnis der englischen Sprachen in die Beurteilung dieser Frage mit eingeht.

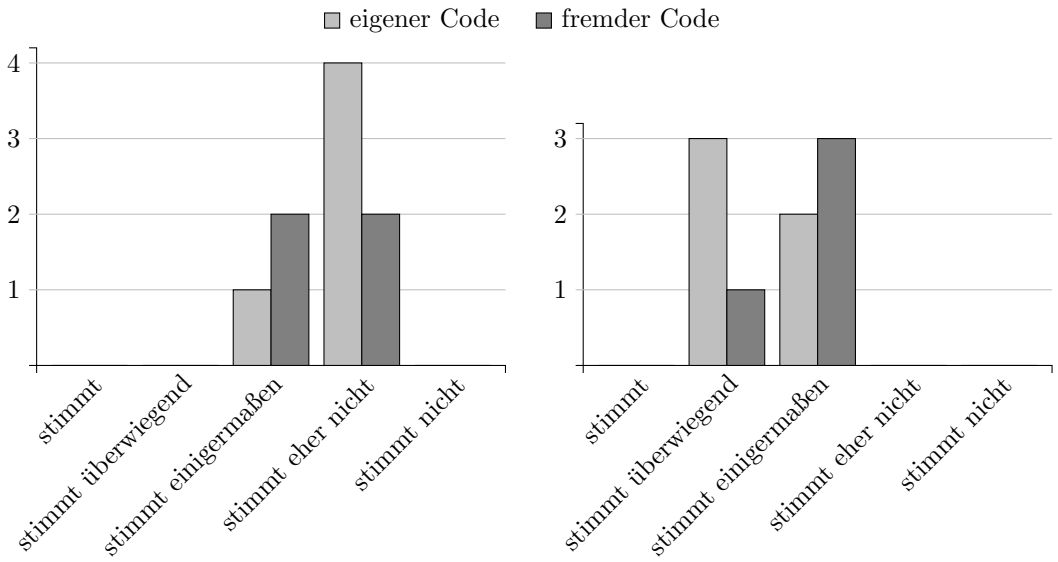
Die Teilnehmer waren der Meinung, dass der Text der Webseite allgemein kurz und prägnant geschrieben ist. Bei den hilfreichen Themen zeigte sich eine Tendenz in Richtung kurz und prägnant, wohingegen sich bei den nicht hilfreichen Themen keine klare Aussage ergibt. Dies kann auch an den wenigen Antworten liegen (siehe Abbildung 5).

Nach Ansicht der Teilnehmer sind die Texte aussagekräftig (siehe Abbildung 6 auf der vorherigen Seite) und korrekt, aber die Angaben der Webseite im Allgemeinen unvollständig, was die aufgeführte Liste der fehlenden Informationen auf der Webseite bestätigt – als fehlende Themen wurden zum Beispiel eine Featureliste bzw. eine Beschreibung der Architektur angemerkt.

Zusammenfassend lässt sich sagen, dass es der Webseite an einer Gliederung mangelt und dass wichtige Informationen fehlen.

3.2.3 Fragen zum Quellcode

Fünf Programmierer haben den Fragenblock zu den Kommentaren im Quellcode von Saros beantwortet. Die Teilnehmer sollten die Kommentare getrennt bewerten: zum einen sollte der Codeabschnitt betrachtet werden, an dem der Befragte selbst gearbeitet hat, zum anderen die Kommentare im fremden Code, die für ihn relevant waren. Da es einen Programmierer gab, der an allen Teilen des Saros-Codes arbeitete, habe ich fünf Antworten zum ‚eigenen Code' und vier Antworten zum ‚fremden Code' bekommen.



(a) *Frage 20a: „Die Kommentare wiederholen den Code."*

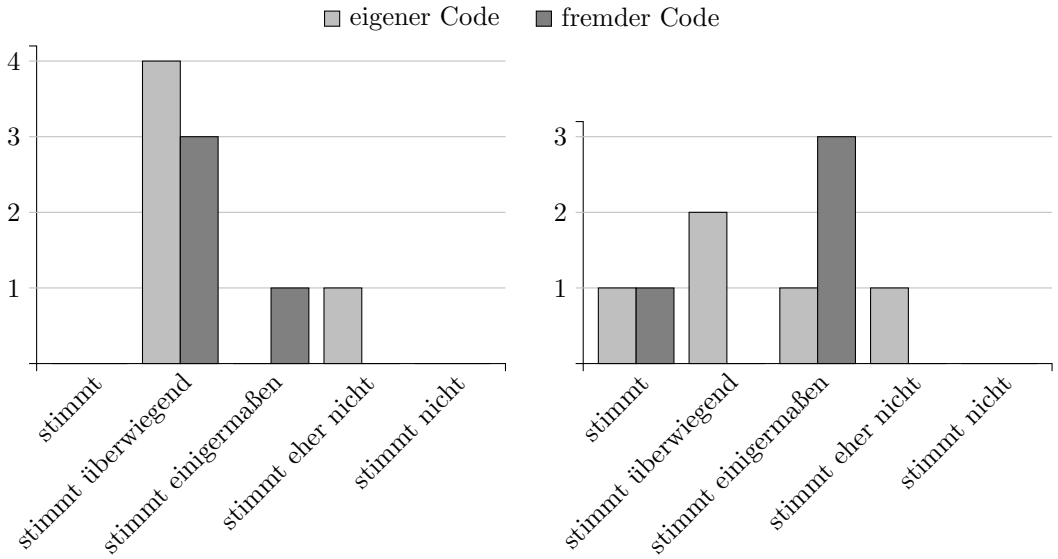
(b) *Frage 20b: „Die Kommentare erklären den Code."*

Abbildung 7 – *Antworten zu den Fragen 20a und b*

Die Umfrage ergab, dass nach Ansicht der Programmierer die Kommentare den Code eher nicht wiederholen sondern erklären. Diese Tendenz war bei der Bewertung des eigene Kommentars deutlicher vorhanden als bei der eines fremden (siehe Abbildung 7). Auch wenn McConnell beschreibt, dass Code, der erklärt werden muss,

nach Möglichkeit neu geschrieben werden sollte, ist es dennoch besser, den Code zu erklären als ihn zu wiederholen (vgl. [McC04, S. 786]).

McConnell empfiehlt, dass Kommentare den Code zusammenfassen bzw. den Zweck beschreiben sollen. Die Programmierer stimmten überwiegend damit überein, dass die Kommentare den Code zusammenfassen (siehe Abbildung 8a). Bei der Beschreibung des Zwecks waren sie sich nicht ganz so einig. Die eigenen Kommentare wurden wieder besser beurteilt, als die fremden, von denen sie zudem nicht immer überzeugt waren, dass der Zweck des Codes wiedergegeben wurde (siehe Abbildung 8b).



(a) *Frage 20d: „Die Kommentare fassen den Code zusammen.“*

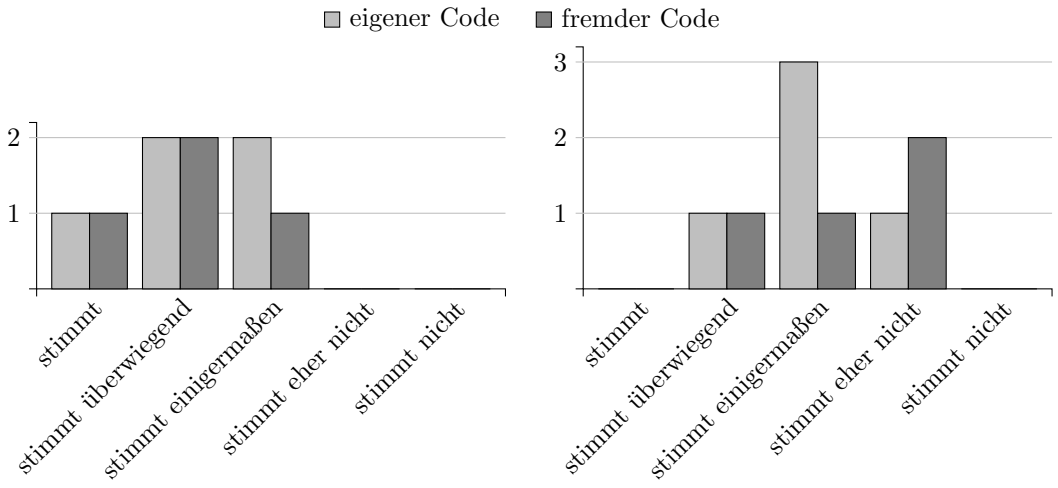
(b) *Frage 20e: „Die Kommentare beschreiben den Zweck/was der Code bewirken soll.“*

Abbildung 8 – *Antworten zu den Fragen 20d und e*

Dass die Frage „Die Kommentare stellen Markierungen im Code dar“ zustimmend beantwortet wurde (siehe Abbildung 9a auf der nächsten Seite), lässt den Schluss zu, dass es recht viele Markierungen im Code gibt, die abgearbeitet werden sollten.
Ob die Kommentare gut sind, so wie sie sind, wurde eher durchschnittlich beantwortet (siehe Abbildung 9b auf der nächsten Seite).

Zu den Fragen, an welchen Stellen die Programmierer eine Kommentierung für nötig bzw. für unnötig halten, wurde fast einstimmig geantwortet, dass eine Kommentierung auf höherer Ebene (Pakete, Interfaces, Klassen) wichtig ist, jedoch auch komplexe Methoden kommentiert werden sollten. Kleinere Methoden sollten durch ihre Namen selbsterklärend sein.

Bei der Frage „An welchen Stellen im Code fehlt dir eine Kommentierung?“ benannten viele Entwickler den Teil, mit dem sie bei ihrer Arbeit am meisten zu tun haben



(a) *Frage 20c: „Die Kommentare stellen Markierungen im Code dar.“*

(b) *Frage 20j: „Die Kommentare sind gut so, wie sie sind.“*

Abbildung 9 – *Antworten zu den Fragen 20c und j*

bzw. hatten (z. B. Netzwerkschicht, Einladungsprozess), so dass hier kein Quellcode-Abschnitt besonders heraus stach.

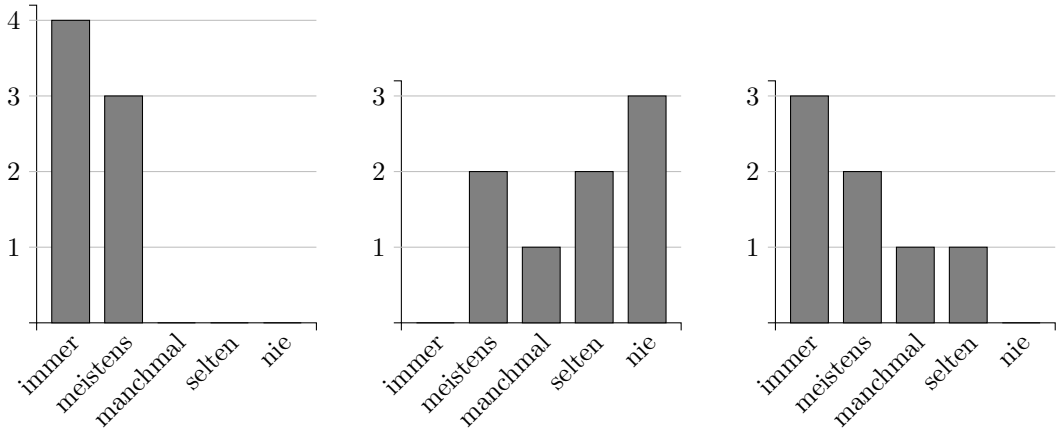
3.2.4 Fragen zum Dokumentationsverhalten

Die Fragen zum Dokumentationsverhalten wurden nicht von allen Teilnehmern beantwortet. Fast alle Antworten kamen von den Studenten, die an Saros mitentwickeln bzw. mitentwickelt haben.

Die Disziplin, den Quellcode zu kommentieren, ist nach Aussage der Programmierer gut (siehe Abbildung 10a auf der nächsten Seite).
Bei der Aktualisierung der Webseite ergab sich jedoch ein anderes Bild. Die studentischen Programmierer bestätigten, dass sie eher selten oder nie die Informationen auf der Webseite aktualisieren, die mit ihrem Teil der Arbeit zu tun haben (siehe Abbildung 10b auf der nächsten Seite). Teilweise wurde diese Aussage damit begründet, dass es keine Bereiche auf der Seite gebe, die mit ihren Teilen zu tun hätten. Die Webseite wird also zur Zeit fast nur von den wissenschaftlichen Mitarbeitern der AG SE gepflegt.

Ein Großteil der Programmierer bewertete während eines Reviews die Kommentare im Code ihrer Mitentwickler, es gab aber auch zwei, die angegeben haben, dies selten oder nur manchmal zu tun (siehe Abbildung 10c auf der nächsten Seite).

Die Frage „Würdest du Mängel an der Dokumentation korrigieren, wenn dich jemand darauf aufmerksam machte (etwa ein Dokumentationsmanager analog zu Release-



(a) *Frage 24: „Kommentierst du deinen Quellcode?“*

(b) *Frage 25: „Ergänzt bzw. aktualisierst du die Informationen auf der Webseite, die mit deinem Teil der Arbeit an Saros zu tun haben?“*

(c) *Frage 26: „Bewertest du während eines Reviews auch die Kommentare im Code der anderen?“*

Abbildung 10 – *Antworten zu den Fragen 24, 25 und 26*

und Testmanager)?“ zielte auf die Rolle eines Dokumentationsmanagers hin, welche ich schon in der AG SE mit einigen Entwicklern besprochen hatte. Die Teilnehmer der Umfrage haben eindeutig zugestimmt, Mängel an der Dokumentation zu korrigieren, wenn sie darauf aufmerksam gemacht werden.

Da ich in Betracht gezogen hatte, die Informationen für die Webseite im Quellcode abzulegen und daraus mit Hilfe von Doxygen die Saros-Webseite generieren zu lassen (siehe Abschnitt 4.1.1, S. 33), habe ich zu der Frage „Was würde es dir erleichtern, Änderungen an der Webseite vorzunehmen?“ zwei Auswahlmöglichkeiten vorgegeben: zum einen, dass die Webseite extern zum Quellcode vorliegt (d. h. wie bisher als TWiki-Seiten) und zum anderen, dass die Informationen, aus denen die Webseite generiert wird, an den Quellcode gebunden werden und somit mit dem Saros-Projekt aus- bzw. eingecheckt werden. Beide Möglichkeiten erhielten gleich viele Stimmen.

Von den Teilnehmern der Umfrage wurden auch Vorschläge gemacht, wie die Dokumentation aktuell gehalten werden könnte. Die Idee des Dokumentationsmanagers wurde wiederholt bestätigt. Ebenso wurde sich dafür ausgesprochen, dass die Erstellung der Dokumentation möglichst leicht gemacht werden sollte. Es sollte klar erkennbar sein, wie die Dokumentation im Quellcode und auf der Webseite auszusehen hat. Ein weiterer Vorschlag war, dass die Informationen bei Änderungen nicht überschrieben, sondern ergänzt werden sollten und es zu jedem Fragment einen Zeitstempel geben sollte, wann dieses von wem erstellt wurde.

3.2.5 Schlussfolgerung

Aufgrund der zustimmenden Antworten des Fragebogens habe ich beschlossen, die Rolle eines Dokumentationsmanagers in den Release-Prozess mit einzuführen. Auf dessen Aufgaben sowie auf die Richtlinien, wie Dokumentationsprobleme behandelt werden sollen und wie Kommentare abgefasst werden sollen, gehe ich in Abschnitt 5, S. 51 ein.
Den aus den Antworten des Fragebogens stammenden Vorschlag, Informationen zu ergänzen, anstatt sie zu überschreiben, habe ich dabei jedoch nicht aufgegriffen. Da das Alter einer Information nicht dringend etwas über ihre Aktualität aussagt und ein ausschließlich additives Ergänzen der Dokumentationsfragmente zu einer Flut an Informationen führen kann, die die gesamte Dokumentation noch unübersichtlicher macht, wäre ein solches Vorgehen meiner Ansicht nach kaum sinnvoll. Veraltete bzw. falsche Informationen müssen gelöscht werden, damit der Überblick erhalten bleibt.

Dennoch gab es in den Antworten des Fragebogens aufgeführte fehlende Themen und diese sollten ergänzt werden, um den Leser hilfreiche Informationen zu liefern.
Aus der Liste der fehlenden Themen habe ich der Beschreibung der Architektur (auf die ich in Abschnitt 4.2, S. 37 eingehe) die höchste Priorität gegeben. Sie bietet den Entwicklern einen Einstieg in den Aufbau der Software und durch ihre höhere Abstraktionsebene bleibt die Information lange gültig.
Eine Auflistung der Features bietet eine Art Werbung für das Projekt und ist von daher sehr nützlich.
Auf der bisherigen Webseite gab es von zwanzig Hauptpaketen des Saros-Quellcodes nur sieben kurze Paketbeschreibungen. Auch hier schien es notwendig, sie in dieser Form von der Webseite zu entfernen. Stattdessen sollen sie mit den restlichen fehlenden Kommentaren im Quellcode ergänzt werden. Da diese Aufgabe sehr umfangreich ist und die jeweilige Einarbeitung in den Codeabschnitt bedarf, sehe ich dies als längeren Prozess, der mit Unterstützung des Dokumentationsmanagers erfolgen kann. Um für die Dokumentation im Quellcode zu demonstrieren, wie die Kommentare aussehen sollten – wie es in meiner Umfrage angemerkt wurde – habe ich mich dazu entschieden, dies anhand von Paketbeschreibungen zu zeigen (siehe hierzu Abschnitt 4.4, S. 47).
Eine erste Übersicht über Saros ist in soweit gegeben, dass mögliche Szenarien auf der Startseite beschrieben werden. Außerdem finden sich Erläuterungen zur ersten Verwendung von Saros sowie Beschreibungen zum Saros-View auf der *Getting Started* Seite, so dass ich diesem Punkt keine höhere Priorität gegeben habe.

3.3 Interview mit Christopher Oezbeck

Dr. Christopher Oezbeck war von September 2004 bis Oktober 2009 ein Mitglied der AG SE. Im Saros-Team hat er in der Zeit Studenten betreut und die techni-

sche Projektleitung übernommen. Im Mai 2011 habe ich ihn angeschrieben und um Beantwortung meines Fragebogens gebeten, wodurch sich die Gelegenheit ergab, ein Interview mit ihm zu führen und ihn nach seiner Meinung zur Dokumentation von Saros und zum Dokumentationsverhalten zu befragen.

Der Einschätzung von Christopher Oezbeck zufolge brauche es Zeit, die volle Komplexität von Saros zu erfassen. Jeder Entwickler benötige eine Eingewöhnungsphase, um ein Modul zu begreifen und zu verstehen, was dieses tut und welche Anforderungen es hat. Ein umfassendes Verständnis müsse daher erstmal „ein Semester reifen“. Das von Dr. Karl Beecher angebotene Softwareprojekt sei dabei eine gute Möglichkeit, Studenten an Saros heranzuführen. Sie hätten dann bereits Erfahrungen mit der Software gemacht, wenn sie ihre Abschlussarbeit im Saros-Team schreiben.

Bezogen auf die Dokumentation beschrieb Christopher Oezbeck in unserem Gespräch verschiedene Ebenen, durch die ein Student ein Verständnis für Saros gewinnen kann. Zum einen sei ein guter Aufbau der Dokumentation wichtig, wobei es vor allem auf die Dokumentation selbst ankomme, die die Anleitung liefere, wie man in das Projekt einsteigt und wie die Abläufe funktionieren. Andererseits sei die Darstellung der Architektur mindestens genauso relevant, um zu verstehen, welches die wichtigen Komponenten sind und wie sie zusammenhängen. Er stimmte mit mir darin überein, bei der technischen Beschreibung ‚von oben‘ anzufangen. Nicht nur, weil eine Architekturbeschreibung nicht so schnell ihre Aktualität verliert, sondern auch, weil man ohne das Verständnis der Architektur die Einzelteile nur schwer erfassen kann.

Die Rolle eines Dokumentationsmanagers einzuführen, traf bei Christopher Oezbeck auf Zustimmung. Zur Ergänzung der Kommentare schlug er vor, sich während einer Release-Woche zunächst die (Top-Level-) Pakete vorzunehmen, welche noch nicht dokumentiert sind und diese zu dokumentieren. Wenn nach und nach alle Pakete dokumentiert seien, könne eines herausgegriffen und gegebenenfalls überarbeitet werden, falls die Paketbeschreibung nicht mehr aktuell ist. Zusätzlich könne eine Klasse, die im letzten Release-Zyklus viel editiert wurde, dahingehend überprüft werden, ob die Dokumentation verständlich ist.
Dieses Vorgehen entspricht auch der von Rostkowycz et al. untersuchten schrittweisen Redokumention, welche ich in Abschnitt 2.2, S. 16 beschrieben habe und bestärkte mich darin, diesen Ansatz für die Aufgaben des Dokumentionsmanagers zu verwenden.

4 Umstrukturierung und Erweiterung der Dokumentation

In diesem Abschnitt beschreibe ich die Umsetzung der Neu- bzw. Umstrukturierung der Dokumentation auf der Webseite. Die Auswertung des Fragebogens hatte diesbezüglich vor allem gezeigt, wie wichtig es sein würde, eine übersichtliche Struktur zu schaffen und bislang fehlende Informationen zu ergänzen. Da auch das Gespräch mit Christopher Oezbeck diesen Ansatz bestätigt hatte, betrafen meine Änderungen einerseits die Navigation, die neu gruppiert wurde, und andererseits die Aufteilung der Informationen auf thematisch geordnete Seiten. Dabei wurde hauptsächlich der *Developer Guide* auf verschiedene Seiten verteilt.

Des Weiteren wurden der Webseite neue Informationen hinzugefügt. Um den Entwicklern einen besseren Überblick zu verschaffen, wurden die *Technical Information* um einen Übersicht über die Saros Architektur und einzelne Modulbeschreibungen erweitert.

Mit Hilfe von Doxygen wurde eine Quellcode-Dokumentation aus den Javadoc-Kommentaren erzeugt und mit auf der Webseite verlinkt. Außerdem sind die Kommentare mit Paketbeschreibungen ergänzt worden.

4.1 Umstrukturierung der Webseite

An vielen Stellen war die Orientierung auf der alten Saros-Webseite unbefriedigend. Bei der Navigation auf der linken Seite gab es keine klare Trennung zwischen den Teilen, die für einen Benutzer des Saros-Plugins interessant sind und den Bereichen, die sich an die Entwickler von Saros wenden. Des Weiteren waren alle wichtigen Informationen für den Entwickler auf der *Developer Guide* Seite zusammengetragen, was sie schwer verständlich machte.

Damit der Leser sich auf der Webseite leichter zurechtfinden kann, habe ich die Navigation in drei Blöcke unterteilt: einen allgemeinen Bereich, einen Bereich für die Benutzer und einen Bereich für die Entwickler von Saros.
Der allgemeine Bereich umfasst die Startseite, die einen ersten Überblick über das Eclipse-Plugin geben soll. Sie beinhaltet Informationen zu Saros-bezogenen Publikationen und Forschungen, zu Kontaktmöglichkeiten sowie zu Screenshots zum Projekt.
Der Bereich für den Benutzer soll einen Einblick in die Verwendung von Saros geben. Dazu gehören die Installationsanleitung, kurze Einführungs-Tutorials, Tipps im Umgang mit Saros und die FAQs.
Für den Entwickler gibt es einen eigenen Bereich mit Produkt- und Prozessinformationen. Der *Developer Guide* wird hierzu inhaltlich in Anleitungen zum Einrichten und Auschecken von Saros, Richtlinien für den Entwickler, Informationen zum Usability Engineering und Testen, Organisatorisches für den Release und Vorschriften zum Umgang mit Problemen mit der Software sowie der Dokumentation (siehe Abschnitt 5.1, S. 51) aufgespalten. Die Technischen Informationen habe ich um eine

Beschreibung der Architektur von Saros erweitert, worauf ich in Abschnitt 4.2, S. 37 eingehe. Außerdem gehören zu dem Entwicklerbereich die externen Links zum *Bug Tracker*, zur Sourceforge-Seite von Saros, zum *Review Board*[5], sowie die Links zu dem *Documentation Issue Tracker*, den ich in Abschnitt 5.2, S. 52 beschreibe, und zu den aus dem Quellcode und Javadoc generierten HTML-Seiten (siehe Abschnitt 4.3, S. 44).

Als Orte, an denen die Informationen, aus denen die Webseite generiert wird, abgelegt werden können, hatte ich in meiner Umfrage zwei Möglichkeiten vorgeschlagen: Die Webseite liegt weiterhin im TWiki-System vor und wird dort editiert oder die Informationen werden in einem Verzeichnis im Quelltext gespeichert und können im Eclipse-Editor verändert werden, woraufhin aus diesen Informationen mit Hilfe von Doxygen die Saros-Webseite (inklusive der Quellcode-Dokumentation) generiert wird (siehe Abschnitt 4.1.1 auf der nächsten Seite).
Das Ergebnis der Umfrage war nicht aussagekräftig bezüglich dieser Einbindung der Webseite in den Quelltext: Die eine Hälfte der Befragten war dafür, die andere Hälfte dagegen. Es gibt Argumente sowohl für als auch gegen eine Einbindung der Webseiten-Informationen in den Quellcode:

Vorteile:

- Die Webseiten-Informationen und der Quellcode befinden sich an einem Ort.
- Die Entwickler haben einen einfacheren Zugriff auf die Webseiten-Informationen, da sie bei jedem Check-out mit dem Saros-Projekt lokal abgespeichert werden.
- Durch automatisches Erzeugen der gesamten Dokumentation in einem Schritt lassen sich die alte Webseiten-Dokumentation und die Quellcode-Dokumentation besser zu einer Einheit verbinden.

Nachteile:

- Um Änderungen an der Webseite vorzunehmen, muss man Saros aus- und einchecken, d. h. nur Saros-Entwickler können die Dokumentation bearbeiten.
- Wenn ein Entwickler Informationen für die Webseite hinzufügt oder ändert, kann es bis zu 24 Stunden dauern, bis die Änderungen auf der Webseite sichtbar werden (bei täglicher Neugenerierung).
- Das kann auch bedeuten, dass der Entwickler unter Umständen sich nicht selbst ansehen kann, wie seine Änderungen aussehen, bevor er sie eincheckt – für eine lokale Ansicht der Webseite muss zusätzliche Software installiert werden (z. B. Doxygen).

Ich war daher unschlüssig, welche Variante besser sei: Weiterhin die TWiki-Seiten zu verwenden oder die Dokumentation der Webseite im Quelltext von Saros vorzunehmen. Für mich waren die Vor- und Nachteile ausgewogen und es gab keine eindeutige

[5] http://saros-build.imp.fu-berlin.de/reviews/

Meinung von Seiten des Saros-Teams. Obwohl die Entwickler bisher kaum Ergänzungen an der Webseite vorgenommen hatten, wäre dies dennoch ihre Aufgabe und nach Aussage meines Betreuers wäre es in Ordnung, wenn nur die Entwickler Zugriff zum Bearbeiten der Seiten hätten. Deswegen habe ich mich zunächst dazu entschieden, die Generierung der Webseite mit Hilfe von Doxygen vornehmen zu lassen.

4.1.1 Erster Ansatz: Webseiten-Generierung mittels Doxygen

Doxygen [vH11] ist ein Werkzeug zur Software-Dokumentation, welches mehrere Programmiersprachen unterstützt (C++, C, Java, Objective-C, Python, IDL, Fortran, VHDL, PHP, C#, sowie teilweise D). Es wird von Dimitri van Heesch unter der GNU General Public License[6] seit 1997 entwickelt.
Doxygen erkennt Javadoc- (`/** ... */`) und Qt-Kommentare (`/*! ... */`) und generiert eine zusammenhängende Dokumentation aus dem Quelltext und den Kommentaren. Diese Ausgabe kann als HTML oder LaTeX erfolgen. Weitere unterstützte Ausgaben sind Rich Text Format (RTF), XML und Unix Manpages.

Im Gegensatz zur HTML-Erzeugung mit Javadoc ermöglicht Doxygen die Generierung und Einbindung verschiedener Graphen in die HTML-Ausgabe mit Hilfe der Beschreibungssprache DOT. Dazu gehören Abhängigkeitsgraphen, Kollaborationsdiagramme, Aufrufgraphen, Verzeichnisstrukturgraphen und Klassenhierarchiegraphen. Um die Graphen erstellen zu können, muss Graphviz[7] – eine Visualisierungs-Software für Graphen (welche unter der Eclipse Public License[8] entwickelt wird) – installiert werden.
Zur Konfiguration wird eine Datei (Doxyfile) verwendet, mit welcher alle Einstellungen für die Ausgabe vorgenommen werden können. Dazu werden die Parameter in der Form `TAGNAME = VALUE` bzw. `TAGNAME = VALUE1 VALUE2` bestimmt. Alternativ kann dafür das Assistenzprogramm Doxywizard zur Hilfe genommen werden, welches eine komfortablere Graphische Oberfläche bietet.

Da man mit Doxygen nicht nur den Quelltext kommentieren, sondern auch unter Verwendung der Javadoc- bzw. Qt-Kommentare eigene Seiten erzeugen kann, habe ich einen Prototypen erstellt, wie die Informationen der Saros Webseite mit Doxygen generiert werden können. In Eclipse habe ich ein neues Testprojekt angelegt, in dem ich die für die Webseiten nötigen Doxygen-Dateien gespeichert habe. So ist es möglich, entweder die Dokumentation als eigenes Projekt parallel zu Saros auszuchecken oder aber den Dokumentationsordner mit den Dateien direkt in Saros zu integrieren. Ich habe die erste Möglichkeit gewählt.

Um HTML-Seiten unabhängig von der Quelltext-Kommentierung zu erzeugen gibt es die Möglichkeit, von der Hauptseite (`@mainpage`) aus weitere Unterseiten zu definieren. Dazu gibt es die Befehle `@subpage` zur Referenzierung weiterer Seiten und

[6] http://www.gnu.org/licenses/gpl-2.0.html
[8] http://www.eclipse.org/org/documents/epl-v10.php

`@page` zur Zuordnung derselbigen. Dadurch wird außerdem die Navigation im linken Bereich der Webseite erweitert.

Ich habe relativ schnell feststellen müssen, dass Doxygen in einigen Bereichen noch nicht ausgereift ist. Es eignet sich nur bedingt zur Erzeugung von Seiten, die nicht direkt zur Code-Dokumentation gehören. Ebenso wie andere Software-Dokumentationswerkzeuge (Javadoc[9] oder Natural Docs[10]) ist es dazu ausgelegt, eine Dokumentation aus dem Quelltext und den dazugehörigen Kommentare zu erzeugen. Seine Aufgabe besteht jedoch nicht darin, normale bzw. eigene Webseiten damit zu gestalten. So gibt es z. B. keine Einstellungsmöglichkeiten, die Navigation zu gruppieren oder externe Links dort einzubinden (wie den Link zum *Bug Tracker* oder zu den Sourceforge-Seiten von Saros), um sie immer sichtbar zu haben. Diese beiden Punkte sehe ich jedoch als essentiell an, um sich auf einer Webseite gut zurechtzufinden.

Für die Navigation wird eine Javascript-Datei erzeugt. Es ist prinzipiell möglich, diese nach der Erzeugung zu ändern. Jedoch haben wir es nicht mit rein statisch erzeugten Webseiten zu tun. Vielmehr sollen die Seiten jede Nacht neu generiert werden, damit die aus dem sich ständig ändernden Quelltext stammenden Informationen jeden Tag aktuell angezeigt werden. Somit müsste man auch nach jeder Generierung der HTML-Seiten die zugehörige Javascript-Datei anpassen. Dabei muss man aber bedenken, dass der Aufbau der Navigation jederzeit komplett geändert werden kann, was eine automatische Anpassung der Javascript-Datei erschwert.

Auch sind die wenigen Möglichkeiten zwar dokumentiert aber teils nicht oder nur unzureichend implementiert: die initiale Breite der Navigation ist trotz dazu vorgesehener Einstelloption nicht änderbar. Auch wenn dies teils nur kosmetische Probleme sind, haben sie doch entscheidenden Einfluss auf die Benutzerfreundlichkeit und damit auf die Akzeptanz des Dokumentationssystems durch die Entwickler, welche ohnehin oft schwer zu erreichen ist.

4.1.2 Zweiter Ansatz: Umstrukturierung der vorhandenen TWiki-Webseiten

Doxygen zeigte viele Nachteile, wenn man es für die Erstellung ‚normaler' Webseiten verwenden wollte. Aus diesem Grund habe ich mich dann doch dazu entschlossen, den Ansatz, Doxygen für die Generierung der Saros-Webseite einzusetzen, nicht länger zu verfolgen und stattdessen die vorhandenen TWiki-Seiten zu verwenden und zu überarbeiten.

Wie Barthélémy Dagenais und Martin P. Robillard feststellen, bietet ein Wiki eine sehr leichte Möglichkeit, eigene Dokumentations-Beiträge zu leisten. Die Syntax zur Bearbeitung sei sehr einfach gestaltet und es gebe eine automatische Versionshistorie bzgl. Änderungen. Sie geben auch an, dass die von ihnen untersuchten Projekte,

[9] http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html
[10] http://www.naturaldocs.org/

die mit einem Wiki begonnen hatten, später zu anderen Dokumentationsmedien gewechselt seien. Dies läge hauptsächlich an Spam-Einträgen, da die Beiträge zu einem Wiki nicht so gut kontrolliert werden könnten (vgl. [DR10]). Dieses Problem dürfte im TWiki-System von Saros nicht oder nur bedingt auftreten, da Änderungen nur von den internen Seiten des Instituts aus mit Zugriffsauthorisierung erfolgen können. Dabei überwiegt der Vorteil der einfachen Editierbarkeit und es ergibt sich die Möglichkeit, dass jeder aus dem Saros-Team – egal ob Entwickler oder nicht – die Seite ergänzen kann.

Abbildung 11 – *Vergleich der alten mit der neuen, umgruppierten Navigation am Beispiel der neuen einleitenden Seite* Developer Guide.
Die Informationen des *Developer Guides* wurden auf verschiedene neue Unterseiten verteilt. Grün markiert sind die Webseiten, die von mir inhaltlich komplett neu angelegt wurden.

Zur Umstrukturierung der vorhandenen TWiki-Seiten habe ich die einzelnen Webseiten gruppiert und hinsichtlich der Reihenfolge ihrer Verwendung sortiert. Hierzu habe ich mich an der von Dietrich Juhl [Juh05, S. 25–97] vorgenommenen Grobgliederung für Anleitungen orientiert:

Eine Leistungsbeschreibung führt den Leser in die Materie ein. Sie klassifiziert das Gerät – bzw. bei Saros die Software – und beschreibt den Nutzen, den der Anwender hat. Die Beschreibung, worum es sich bei Saros handelt, steht ebenso auf der Startseite (*Home*), wie Beschreibungen von einzelnen Szenarios, die erklären, wie man Saros verwenden kann. Dazu kommt eine gesammelte Beschreibung der *Features* von Saros, um die ich die Webseite im späteren Verlauf meiner Arbeit ergänzt habe.

Die Gerätebeschreibung soll dem Leser unter Verwendung von *Screenshots* zeigen, wie das Gerät aussieht.

Diese ersten beiden Beschreibungen geben ein allgemeines Verständnis für Saros und gehören von daher auch in den allgemeinen Bereich. Da es sich nicht um eine reine Anleitung handelt, sondern um die Web-Präsentation eines Software-Plugins, gehören meiner Ansicht nach auch die Kontaktinformationen (*Contact*) sowie *Research on Saros*, d. h. die Saros betreffenden Publikationen und Forschungen, zu den allgemeinen Informationen der Webseite.

Die Tätigkeitsbeschreibungen sollen den Leser zum Handeln anleiten. Für den Benutzer von Saros bedeutet dies, hier Anleitung darüber zu finden, wie er die Software installiert (*Installation & Set-up*), wie er sie einrichtet (*Getting Started*) und was er beachten muss (*How to be happy with Saros*). Um weitere Fragen zu klären gibt es die Seite *FAQ*. Für den Entwickler von Saros stellen sich ähnliche Fragen. Auch er muss wissen, wie er an die Entwickler-Version des Plugins herankommt (*Set-up & Check-out*) und was er beachten muss (*Guidelines*, *Usability Engineering*). Des Weiteren sind an ihn die Anleitung zum Testen (*Testing*) gerichtet und die Anweisungen, was während eines Release durchgeführt werden muss (*Our Release Schedule*). Der Umgang mit Problemen in der Software oder der Dokumentation ist in den *Bug Tracking Procedures* und *Documentation Issue Procedures* beschrieben.

Die Beschreibung der Funktionsweise findet sich in den *Technical Information*, die ich im Laufe meiner Diplomarbeit erweitert habe (siehe hierzu Abschnitt 4.2 auf der nächsten Seite).

Abschließend habe ich fünf externe Links dem Entwicklerbereich zugeordnet: die vorher vorhandenen Links zur Sourceforge-Präsenz von Saros (*Saros @Sourceforge*)

und zum *Bug Tracker*, den Link zum *Review Board* (welches vorher nicht auf der Webseite verlinkt war) und zwei neue Links zum *Documentation Issue Tracker* (siehe diesbezüglich Abschnitt 5.2, S. 52) und zur Javadoc-basierten Code-Dokumentation von Saros (*Saros Javadoc* – in Abschnitt 4.3, S. 44 gehe ich näher darauf ein).

Schon diese Gliederung der Webseite hatte zufolge, dass sie von den Nutzern als deutlich übersichtlicher empfunden wurde. Da durch die Aufteilung des *Developer Guides* und durch die hinzugefügten Modulbeschreibungen dieser Abschnitt jedoch sehr lang geworden war – und damit erneut zur Unübersichtlichkeit neigte – habe ich alle Entwickler-bezogenen Seiten, die zur sogenannten Tätigkeitsbeschreibung gehören zu Unterseiten zusammengefasst und eine neue einleitende Seite „*Developer Guide*“ geschrieben, die dem Leser helfen soll, sich auf den Unterseiten zurechtzufinden (siehe Abbildung 11, S. 35).

4.2 Erweiterungen der Technischen Informationen

In meiner Umfrage wurde als ein fehlender Punkt eine Beschreibung der Architektur genannt. Um diese einzufügen, musste zuvor die Form einer solchen Beschreibung definiert werden. Sie sollte auf einer höheren Abstraktionsebene erfolgen, damit sie nicht durch Änderungen an der Architektur sofort wieder an Aktualität verliert. Dies schien besonders deshalb wichtig, da in dem Zeitraum, in dem ich meine Diplomarbeit schrieb, ein anderer Student, Wjatscheslaw Belousow, in seiner Arbeit eine Verbesserung der Architektur von Saros erarbeitete.

4.2.1 Überblick über die Saros Architektur

Obwohl Saros zur Zeit noch nicht komplett modular aufgebaut ist, gibt es doch logische Konzepte bzw. Module, aus denen Saros zusammengesetzt ist. In einer von Stephan Salinger et al. verfassten Publikation wurde eine Architekturübersicht geliefert, in der das Plugin in fünf Module (*Network Layer*, *Concurrency Control*, *Eclipse Bridge*, *Consistency Watchdog* und *Feedback Module*) eingeteilt wird (vgl. [SOBS10]). Die diese Übersicht veranschaulichende Graphik haben mein Betreuer, Dr. Karl Beecher, und ich herangezogen, um eine detailliertere Architekturübersicht für Saros zu entwickeln, die an den aktuellen Stand der Software angepasst ist. In der neuen Version setzt sich Saros aus acht Modulen zusammen, auf die ich im Folgenden etwas näher eingehen werde.

Drei der von Salinger et al. beschriebenen Module wurden übernommen. Dazu zählt der *Network Layer*, der die Verbindung der Saros-Nutzer und den Austausch der Daten unter Verwendung des XMPP[11]-Protokolls ermöglicht. Mit Hilfe der *Eclipse Bridge* verwendet Saros bestehende Eclipse-Komponenten (z. B. den Editor) und

[11] XMPP steht für Exensible Messaging and Presence Protocol.

erweitert diese. Das *Feedback*-Modul leitet den Saros-Nutzer zu einer Umfrage-Webseite und sammelt statistische Daten während einer Sitzung (sofern diese Optionen ausgewählt wurden).

Die beiden von Salinger et al. getrennt definierten Module *Consistency Watchdog* und *Concurrency Control* wurden für die überarbeitete Architekturübersicht von Saros zu einem Modul *Concurrency Control* zusammengefasst, da beide das Ziel haben, die einzelnen Kopien der von den Nutzern geteilten Projekte zu synchronisieren und konsistent zu halten.

Neu zur Architekturübersicht hinzugekommen sind vier weitere Module, die sich durch ihre Funktionalitäten ebenso wie die vier zuvor genannten gut voneinander abgrenzen: Das *User Interface* liefert – im Gegensatz zur Verwendung bestehender Eclipse-Komponenten, wie es in der *Eclipse Bridge* erfolgt – eigene Komponenten für die Kommunikation zwischen dem Benutzer und Saros (z. B. die Saros-eigenen Menüs und Befehle). *Invitation* ist der Bereich, der die Funktionalität für einen Sitzungsaufbau bereitstellt. *Shared Session* umfasst alle wichtigen Informationen für eine Sitzung in Saros. *Utility* beinhaltet Datentypen und Methoden, die häufiger verwendet werden, aber keinen besonderen Platz im Quellcode besitzen.

Die acht Module werden schließlich durch das von Michael Jurke [Jur10] in seiner Diplomarbeit entwickelte *Whiteboard* ergänzt, welches Saros um einen graphischen Editor für das Erstellen von Zeichnungen mit verteilten Beteiligten, erweitert.

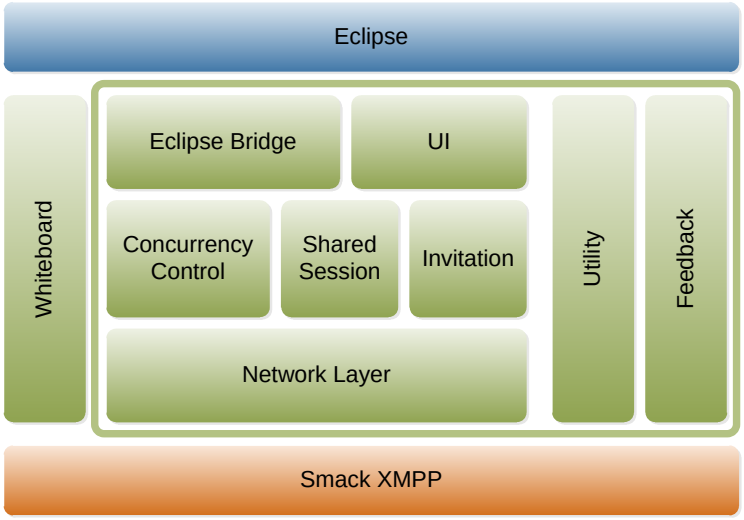


Abbildung 12 – *Erste Version der überarbeiteten Architekturübersicht von Saros*

Meine erste graphische Übersicht der Saros-Architektur (siehe Abbildung 12) war stark an die Version von Salinger et al. angelehnt. Sie konzentrierte sich darauf, zu verdeutlichen, welche Module miteinander interagieren. Um dies zu visualisieren sind

interagierende Module so angeordnet, dass sie jeweils aneinander angrenzen.
Startet ein Saros-Nutzer beispielsweise einen Einladungsprozess, so verwendet er hierfür das *User Interface* (*UI*), welches die Saros-eigenen Menüs in Eclipse zur Verfügung stellt. Somit ist das *User Interface* zwischen *Invitation* und *Eclipse* angeordnet. Der *Network Layer* wiederum grenzt an *Invitation*, da er die Daten-Verbindung zum eingeladenen Saros-Nutzer unter Verwendung des XMPP-Protokolls herstellt.

Diese erste Version einer Architekturübersicht fand keine Zustimmung unter den Saros-Entwicklern, da sie nicht intuitiv verständlich war. Vielmehr wurde die Graphik dahingehend interpretiert, dass Eclipse auf Saros aufbaue.

Um deutlich erkennen zu lassen, welches Modul auf welchem logisch aufbaut, habe ich in einer zweiten Version die graphische Architekturübersicht als Schichtenmodell dargestellt. Zusätzlich habe ich die Graphik um die Java Virtual Machine (JVM) und das darunter liegenden Betriebssystem (OS) ergänzt (siehe Abbildung 13[12]).

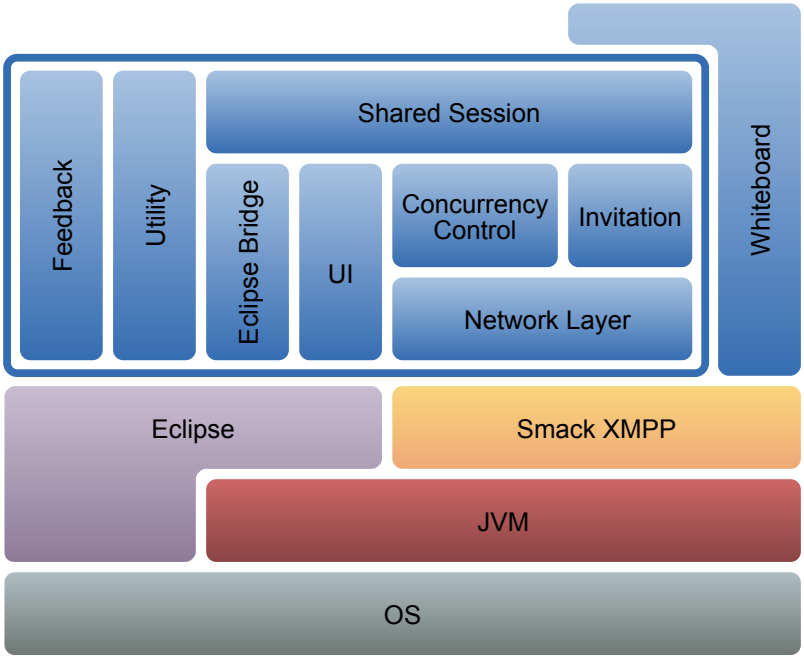


Abbildung 13 – *Zweite Version der überarbeiteten Architekturübersicht von Saros*

Zusammen mit kurzen Beschreibungen der einzelnen Module habe ich anschließend die graphische Architekturübersicht der Webseite *Technical Information* hinzugefügt. Detaillierte Beschreibungen der Module, auf die ich in Abschnitt 4.2.3, S. 41 eingehe,

[12] Die Farben für Eclipse, Smack und der Java Virtual Machine (JVM) sind der besseren Orientierung halber von den jeweiligen Webpräsenzen übernommen. Bei den Saros-Modulen habe ich mich entsprechend der Farben des Logos für Blau entschieden, um es von Smack (Orange) abzugrenzen.

sind zusätzlich jeweils auf den Unterseiten von *Technical Information* verlinkt. Alle Beschreibungen befinden sich auch in Anhang B, S. 81.

4.2.2 Exkurs zu SVG-Graphiken

Für die graphische Darstellung der Architektur und der Module sollten zwei Anforderungen erfüllt werden: Zum einen sollte jedes detailliert beschriebene Modul in der graphischen Architekturübersicht anklickbar sein, so dass man darüber die Webseite mit den näheren Information zum Modul erreicht (in Abschnitt 4.2.3 auf der nächsten Seite gehe ich näher darauf ein). Zum anderen sollten die Graphiken stufenlos und verlustfrei skalierbar sein, so dass sie lesbar bleiben, wenn ein Leser zur besseren Lesbarkeit des Textes die Webseite vergrößert.

Meine Wahl fiel dabei auf die Verwendung von Scalable Vector Graphics (SVG) [SVG11]. Es handelt sich hierbei um eine von der World Wide Web Consortium (W3C) SVG Working Group[13] entwickelte und empfohlene Spezifikation. Sie ist eine auf XML basierende Beschreibungssprache für zweidimensionale Vektorgraphiken. Da die Beschreibung textbasiert ist, sind die erzeugten Dateien zudem sehr klein.

Die SVG-Graphik beispielsweise der *Eclipse Bridge* und des *User Interface* aus der Architekturübersicht wird durch folgenden Quellcode-Abschnitt definiert:

Quellcode 1 – *gekürzte Darstellung des Saros-Architektur (mit* Eclipse Bridge *und* UI*)*

```
101     <g transform="translate(164,132)">
102       <a xlink:href="http://www.saros-project.org/
103                      EclipseBridge"
104          xlink:title="Eclipse Bridge" target="_top">
105         <rect rx="8" ry="8" width="64" height="152"
106               class="gSaros"/>
107         <text x="0" y="8" transform="translate(32,76)
108                                      rotate(270,0,0)"
109               class="textHeading">Eclipse Bridge</text>
110       </a>
111     </g>
112     <g transform="translate(236,132)">
113       <a xlink:href="http://www.saros-project.org/UI"
114          xlink:title="User Interface" target="_top">
115         <rect rx="8" ry="8" width="64" height="152"
116               class="gSaros"/>
117         <text x="32" y="92" class="textHeading">UI</text>
118       </a>
119     </g>
```

[13] http://www.w3.org/Graphics/SVG/

Die aus dem Quellcode 1 auf der vorherigen Seite erzeugte graphische Darstellung der *Eclipse Bridge* und des *User Interface* aus der Saros-Architekturübersicht ist in Abbildung 14 zu sehen. Jedes der beiden Module ist in einer eigenen Gruppe definiert, in der auch der Link zur jeweiligen detaillierten Modulbeschreibung angegeben ist. Der Farbverlauf der blauen Rechtecke (Module) ist in der Klasse `gSaros` und die Schriftart (Arial) und -größe (16pt) der Bezeichner in der Klasse `textHeading` definiert.

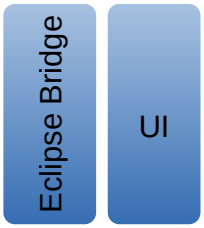


Abbildung 14 – *Darstellung der* Eclipse Bridge *und des* User Interface *der Architekturübersicht*

Für das Anzeigen der SVG-Graphiken wird in dem TWiki-System ein Plugin benötigt, welches zunächst nicht installiert war. Obwohl das TWiki-System noch in diesem Jahr auf ein vergleichbares FosWiki-System[14] umgestellt werden soll, wurde für die Übergangsphase das ObjectPlugin[15] zum Einbinden der SVG-Graphiken installiert. Im FosWiki wird es das MediaPlugin[16] geben, welches diese Aufgabe erfüllen kann.

4.2.3 Beschreibung der einzelnen Module

Da die Kurzbeschreibungen der Module nur einen kleinen Einblick über die Saros-Architektur bieten, sollten durch detailliertere Modulbeschreibungen Aufbau und Funktionalität der einzelnen Module näher erklärt werden. Dazu habe ich gemeinsam mit Dr. Karl Beecher die einzelnen Module näher untersucht.

Um den Aufbau der Module genauer zu beschreiben und die einzelnen Funktionalitäten näher zu erläutern, wurde jedes der Module *Shared Session*, *Eclipse Bridge*, *User Interface*, *Concurrency Control*, *Invitation*, *Network Layer* und *Feedback* weiter in logisch zusammengehörige Komponenten aufgeteilt. Diese Aufteilung habe ich vergleichbar der Architekturübersicht graphisch dargestellt.

Die Beschreibung der Module und ihrer Komponenten fußt auf Informationen, die ich dem Quelltext bzw. den Kommentaren entnommen habe. Zusätzlich konnte ich einige Abschlussarbeiten der AG SE heranziehen, in denen einzelne Teile von Saros

[14] http://foswiki.org/
[15] https://www.mi.fu-berlin.de/w/TWiki/ObjectPlugin
[16] http://foswiki.org/Extensions/MediaPlugin

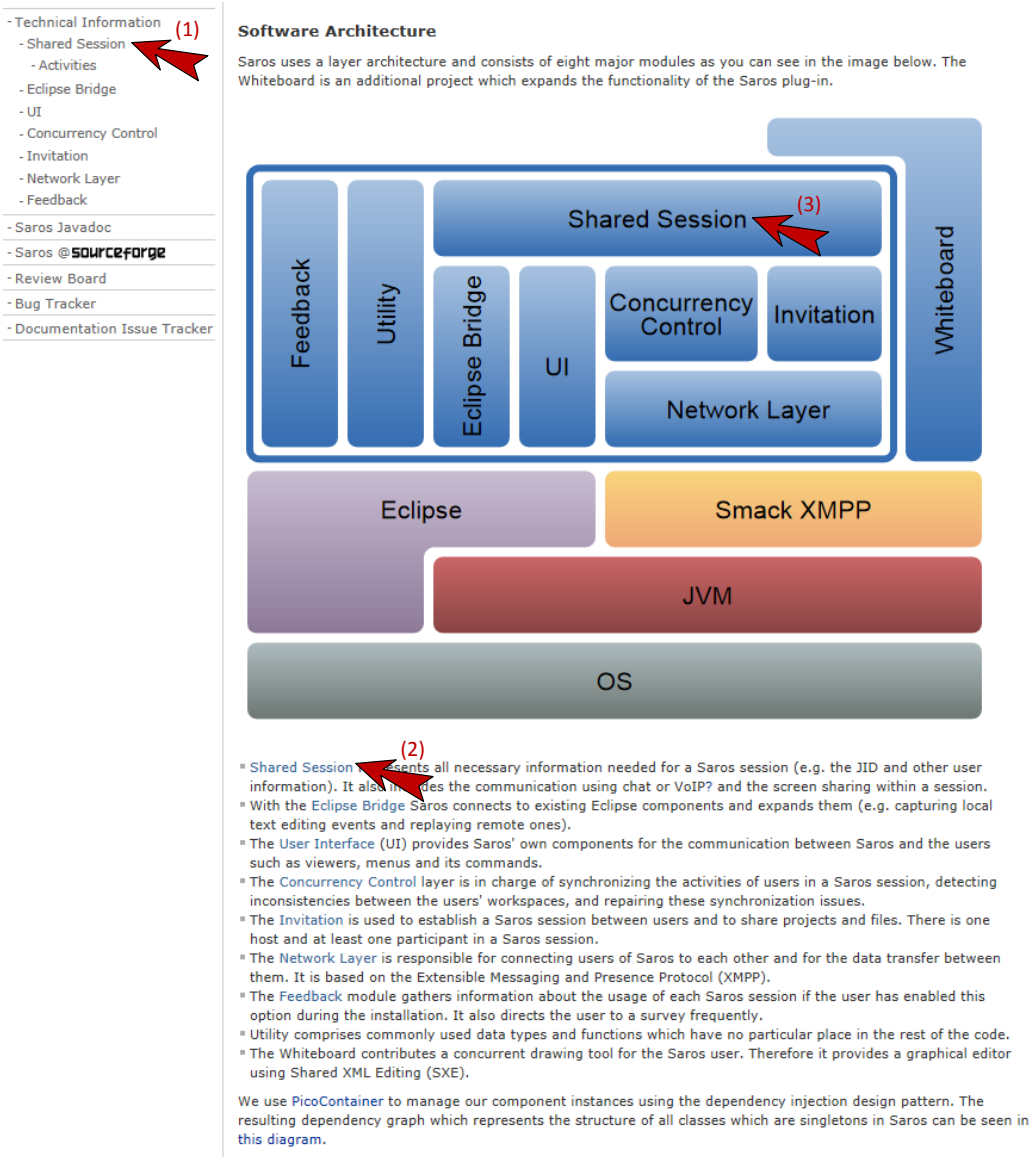
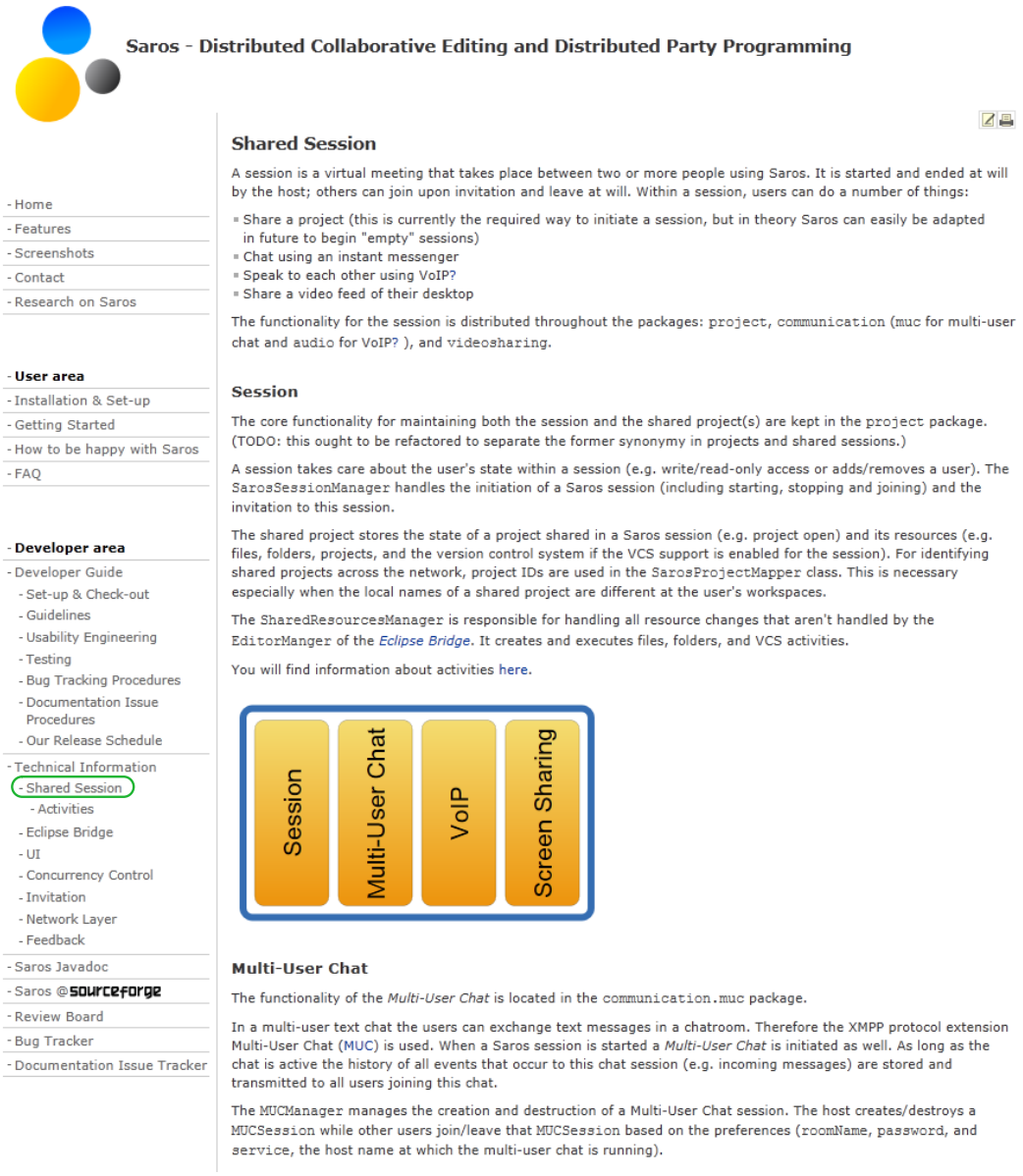


Abbildung 15 – *Ausschnitt aus der Webseite* Technical Information *mit Optionen, wie die detaillierten Modulbeschreibungen aufgerufen werden können (am Beispiel* Shared Session*):* (1) über die Navigation, (2) über die Links in den Kurzbeschreibungen der Module, (3) über die Graphik der Saros-Architektur.

gut erklärt waren. Über die Komponenten, an denen gerade gearbeitet wurde, konnte ich die Informationen direkt von den Entwicklern beziehen. Dazu habe ich eine Vorversion erstellt und diese dann den Entwicklern vorgestellt.

Die fertigen Versionen der Modulbeschreibungen habe ich auf der Webseite verlinkt. Dazu habe ich für jedes Modul eine eigene Webseite erzeugt und die Beschreibun-



Saros - Distributed Collaborative Editing and Distributed Party Programming

- Home
- Features
- Screenshots
- Contact
- Research on Saros

- **User area**
- Installation & Set-up
- Getting Started
- How to be happy with Saros
- FAQ

- **Developer area**
- Developer Guide
  - Set-up & Check-out
  - Guidelines
  - Usability Engineering
  - Testing
  - Bug Tracking Procedures
  - Documentation Issue Procedures
  - Our Release Schedule
- Technical Information
  - Shared Session
    - Activities
  - Eclipse Bridge
  - UI
  - Concurrency Control
  - Invitation
  - Network Layer
  - Feedback
- Saros Javadoc
- Saros @sourceforge
- Review Board
- Bug Tracker
- Documentation Issue Tracker

Shared Session

A session is a virtual meeting that takes place between two or more people using Saros. It is started and ended at will by the host; others can join upon invitation and leave at will. Within a session, users can do a number of things:

- Share a project (this is currently the required way to initiate a session, but in theory Saros can easily be adapted in future to begin "empty" sessions)
- Chat using an instant messenger
- Speak to each other using VoIP?
- Share a video feed of their desktop

The functionality for the session is distributed throughout the packages: `project`, `communication` (muc for multi-user chat and `audio` for VoIP?), and `videosharing`.

Session

The core functionality for maintaining both the session and the shared project(s) are kept in the `project` package. (TODO: this ought to be refactored to separate the former synonymy in projects and shared sessions.)

A session takes care about the user's state within a session (e.g. write/read-only access or adds/removes a user). The `SarosSessionManager` handles the initiation of a Saros session (including starting, stopping and joining) and the invitation to this session.

The shared project stores the state of a project shared in a Saros session (e.g. project open) and its resources (e.g. files, folders, projects, and the version control system if the VCS support is enabled for the session). For identifying shared projects across the network, project IDs are used in the `SarosProjectMapper` class. This is necessary especially when the local names of a shared project are different at the user's workspaces.

The `SharedResourcesManager` is responsible for handling all resource changes that aren't handled by the `EditorManger` of the *Eclipse Bridge*. It creates and executes files, folders, and VCS activities.

You will find information about activities here.

Multi-User Chat

The functionality of the *Multi-User Chat* is located in the `communication.muc` package.

In a multi-user text chat the users can exchange text messages in a chatroom. Therefore the XMPP protocol extension Multi-User Chat (MUC) is used. When a Saros session is started a *Multi-User Chat* is initiated as well. As long as the chat is active the history of all events that occur to this chat session (e.g. incoming messages) are stored and transmitted to all users joining this chat.

The `MUCManager` manages the creation and destruction of a Multi-User Chat session. The host creates/destroys a `MUCSession` while other users join/leave that `MUCSession` based on the preferences (`roomName`, `password`, and `service`, the host name at which the multi-user chat is running).

Abbildung 16 – *Ausschnitt aus der Webseite* Shared Session, *welche die detaillierten Beschreibungen dieses Moduls enthält.*

gen zusammen mit der graphischen Darstellung des jeweiligen Moduls dieser neu erzeugten Webseite hinzugefügt. Alle Modul beschreibenden Webseiten wurden als Unterseiten der Saros-Webseite *Technical Information* integriert.

Von der Saros-Webseite *Technical Information* gibt es drei Wege, wie man zu den detaillierten Modulbeschreibungen gelangen kann: über die Navigation, über die Links, welche sich in den Kurzbeschreibungen der Module finden und über die graphische

Darstellung der Saros-Architektur, indem man auf das passende Modul klickt. Die Informationen über die *Shared Session* beispielsweise erreicht man jeweils über einen der drei oben beschriebenen Wege (siehe Abbildung 15, S. 42).

Die jeweilige Unterseite führt dann die detaillierte Beschreibung der Module auf, wie in Abbildung 16 auf der vorherigen Seite wiederum am Beispiel *Shared Session* zu sehen ist.

Die detaillierten Beschreibungen der anderen Module lassen sich analog dazu aufrufen. Sie finden sich jeweils auf den Unterseiten der Saros-Webseite `http://www.saros-project.org/Technicals` und in Anhang B, S. 81.

4.3 Javadoc Generierung mit Hilfe von Doxygen

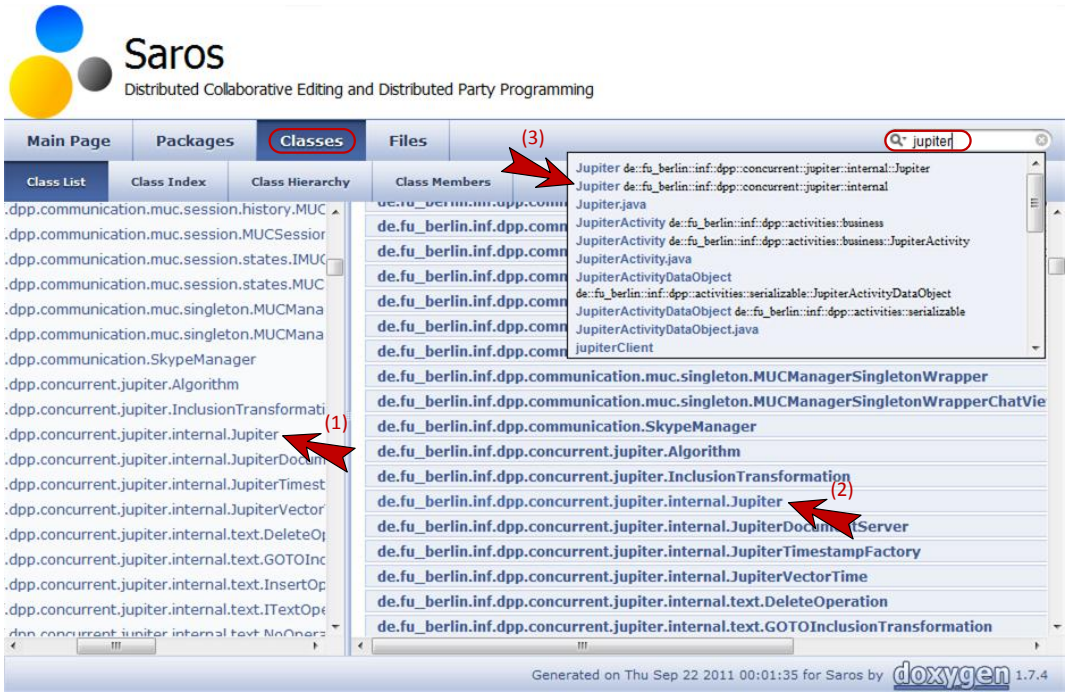


Abbildung 17 – *Ausschnitt aus der Saros Quelltext-Dokumentation mit verschiedenen Wegen, eine Klassen-Referenz aufzurufen (am Beispiel der Klasse Jupiter):* (1) über die Navigation, (2) über den Index, (3) über die Suchfunktion.

Auch wenn ich mich aufgrund der in Abschnitt 4.1.1, S. 33 beschriebenen Schwierigkeiten (z.B. lässt sich die Navigation nicht gruppieren und es lassen sich keine externen Links einbinden) letztlich dagegen entschieden habe, die Webseite mit Doxygen generieren zu lassen, sollte immer noch die Quelltext-Dokumentation auf der Webseite verlinkt werden. Für diesen Zweck bietet sich Doxygen als Software-Dokumentationswerkzeug an, da es mehr Möglichkeiten bietet als Javadoc. Dazu gehört

die in Abschnitt 4.1.1, S. 33 erwähnte Möglichkeit, z. B. mit Hyperlinks versehene Vererbungs- oder Kollaborationsdiagramme zu jeder Klasse anzeigen zu lassen. Ein weiterer Vorteil gegenüber der durch Javadoc erzeugten Quelltext-Dokumentation ist die vorhandene Suchfunktion.

Da zum derzeitigen Stand noch einige Kommentare fehlen oder unvollständig sind, ist es sehr praktisch, dass man mit Doxygen auch unkommentierten Quelltext in die HTML-Übersicht aufnehmen kann. Auch ohne erklärende Kommentare kann dies einen groben Überblick liefern und Zusammenhänge im Quelltext aufzeigen.
Zumal Doxygen die vorhandenen Javadoc-Kommentare erkennt und verarbeiten kann, muss der Quelltext in dieser Hinsicht auch nicht überarbeitet werden.

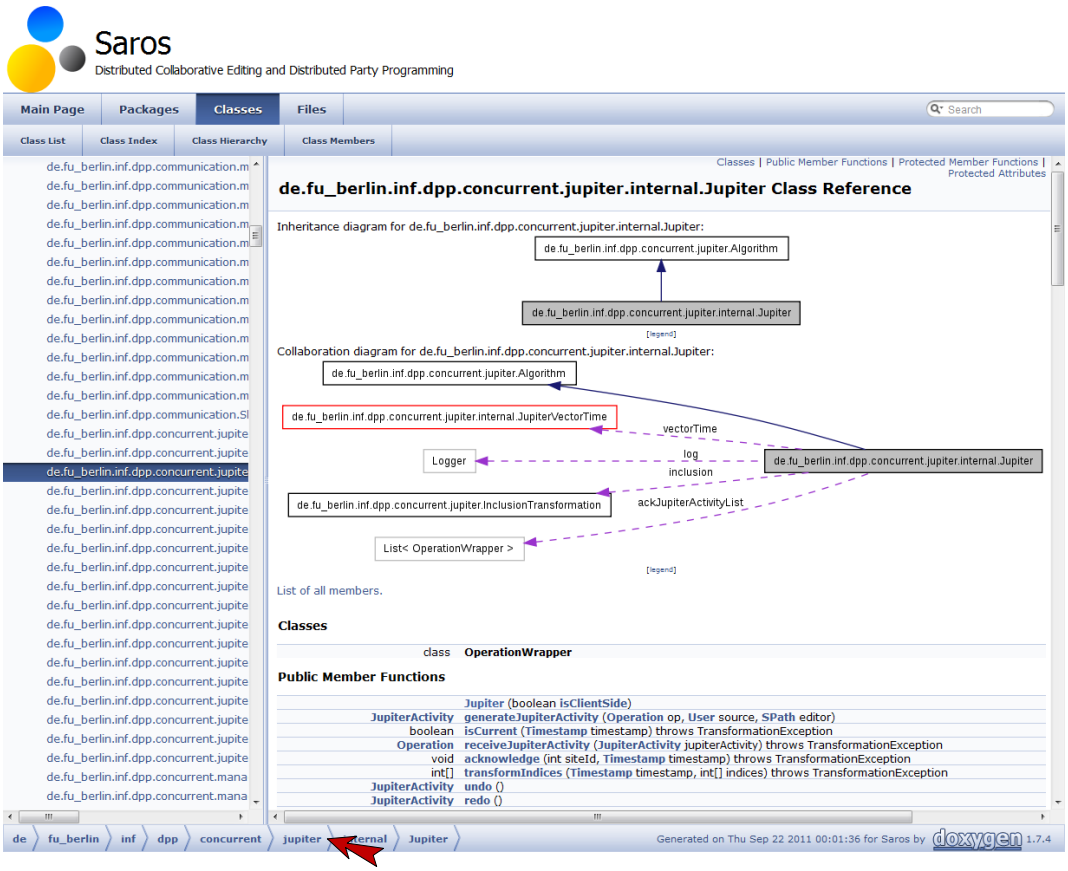


Abbildung 18 – *Ausschnitt aus der Jupiter Klassen-Referenz mit Vererbungs- und Kollaborationsdiagramm.*
Unten auf der Seite ist der vollständige Pfad zur aktuellen Klasse `Jupiter` angezeigt, über den die darüber liegenden Pakete erreicht werden können.

Doxygen erkennt die meisten HTML-Kommandos. Obwohl sich beispielsweise Aufzählungslisten unter Verwendung von Gedankenstrichen (-) sehr einfach schreiben lassen, erscheint der in Javadoc gewohnte Ausdruck über HTML-Tags sinnvoller –

nicht nur, weil die Entwickler daran gewöhnt sind, sondern auch, damit die in Eclipse integrierte Javadoc-Ansicht auch weiterhin Aufzählungslisten richtig anzeigen kann.

Saros
Distributed Collaborative Editing and Distributed Party Programming

Main Page | Packages | Classes | Files | Search

Packages | Package Functions

- Saros
  - Class List
  - Class Index
  - Class Hierarchy
  - Class Members
  - Packages
  - Package Functions
  - File List

Packages | Classes

Package de.fu_berlin.inf.dpp.concurrent.jupiter

Packages

package	internal

Classes

interface	Algorithm
interface	InclusionTransformation
interface	Operation
interface	Timestamp
interface	TimestampFactory
class	TransformationException
interface	VectorTime

Detailed Description

Jupiter Algorithm Overview

The Jupiter Architecture is the central concept for Distributed Party Programming in Saros. Each client administers its own copy of a shared artifact (resource). Local Operations will be performed immediately and communicated to a central server. From there all other clients get informed about the operation which they will transform in order to perform it on their own copies.

The transformation is needed to keep all copies consistent. Consistency maintenance is made up of three components:

- Convergence --- all copies of a document have the same final state after performing a set of operations to it
- Causality-preservation --- for each pair of operations O1 and O2 applies that if O1 precedes O2 then the execution of O1 must take place prior to the execution of O2 for all copies
- Intention-preservation --- for each operation O the result must be identical to the original intent when performing O for all copies. The result may not affect the results of other independent operation

Important components of the Jupiter package:

- internal.Jupiter is the central class of the Jupiter Algorithm
- internal.text.GOTOInclusionTransformation is an implementation of the transformation function of the Jupiter Algorithm

de | fu_berlin | inf | dpp | concurrent | jupiter — Generated on Thu Sep 22 2011 00:01:36 for Saros by doxygen 1.7.4

Abbildung 19 – *Ausschnitt aus der Paketbeschreibung des Pakets Jupiter.*

Zur Konfiguration einer Quelltext-Dokumentation besitzt Doxygen umfangreich einstellbare Optionen. Ich habe das Doxyfile[17] konfiguriert, um eine möglichst übersichtliche und dennoch umfangreiche Quelltext-Dokumentation von Saros zu erzeugen. Mit Hilfe dieser Konfigurationsdatei lässt sich aus dem Quelltext eine Software-Dokumentation erstellen. Dies erfolgt auf dem Saros-Server und kann dort unter `http://saros-build.imp.fu-berlin.de/sarosDoc/` eingesehen werden. Die Quelltext-Dokumentation wird jede Nacht neu generiert, so dass diese bei den stetigen Änderungen am Quelltext aktuell bleibt und neue Elemente (z. B. Methoden,

[17] Zur näheren Erläuterung eines Doxyfiles siehe Abschnitt 4.1.1, S. 33.

Klassen, etc.) und deren Kommentare übernommen werden. Für einen leichteren Zugriff darauf habe ich diese Seiten auf der Saros-Webseite verlinkt (*Saros Javadoc*).

Abbildung 17, S. 44 zeigt drei mögliche Wege, wie beispielsweise die Klassen-Referenz von `concurrent.jupiter.internal.jupiter` aufgerufen werden kann: Über die Navigation und über den Index (unter Auswahl von „Classes“) lässt sich jeweils die Liste der Klassen öffnen und die gesuchte Klasse auswählen. Als dritte Möglichkeit gibt es die Suchfunktion nach dem Klassennamen. Nach Auswahl der Klasse wird deren Referenz angezeigt (siehe Abbildung 18, S. 45). Unten auf der Seite ist der vollständige Pfad zur Klasse `concurrent.jupiter.internal.jupiter` angegeben. Über diese Statuszeile lässt sich beispielsweise die Paketbeschreibung des Pakets `concurrent.jupiter` aufrufen (siehe Abbildung 19 auf der vorherigen Seite). Eine andere Möglichkeit, zur Paketbeschreibung zu gelangen, ist analog zum oben beschriebenen Aufruf einer Klasse über die Navigation oder den Index möglich.

4.4 Javadoc Paketbeschreibungen

Saros wird immer wieder von neu zum Projekt hinzukommenden Studenten weiterentwickelt. Daher ist es besonders wichtig, dass sie den vorhandenen Quelltext verstehen. Dabei können die Kommentare helfen, welche die Studenten entweder direkt im Quellcode oder aber in der neu erzeugten und in Abschnitt 4.3, S. 44 beschriebenen Quellcode-Dokumentation (*Saros Javadoc*) lesen können.

Der Quellcode ist jedoch nur unvollständig kommentiert; die Kommentarabdeckung lag bei 42%[18]. Aus diesem Grund sollte im Rahmen dieser Arbeit nicht nur definiert werden, welche Art von Kommentaren geschrieben werden sollen – wie ich in Abschnitt 2.1.4, S. 14 beschrieben habe und auf die ich in Abschnitt 5.1, S. 51 weiter eingehe –, sondern gleichzeitig auch Beispiele erstellt werden, die zeigen, wie die Kommentare aussehen könnten. Dabei haben mich besonders die Paketbeschreibungen interessiert, da es zwar Beschreibungen auf der Webseite gab, diese jedoch veraltet und sehr kurz waren. Wie ich in Abschnitt 3.2, S. 20 erwähnt habe, sollen die Informationen in den Quelltext eingebunden werden.

Im Saros-Quelltext gab es nur eine einzige Paketbeschreibung und zwar die „*Consistency Watchdog Module Overview*“ zum Paket `concurrent.watchdog`. Ich habe mich dazu entschlossen, das `concurrent`-Paket mit Paketbeschreibungen zu vervollständigen. Ohne dieses Paket und den darin enthaltenen Jupiter-Algorithmus ist eine gemeinschaftliche Arbeit an einem gemeinsamen Projekt in Saros nicht möglich. Es stellt also die Kern-Funktionalität von Saros dar und es wird für viele Studenten wichtig sein, dieses Paket zu verstehen.

[18] laut Doc Check Doclet v 1.2 Beta 2 (http://www.oracle.com/technetwork/java/javase/documentation/index-141437.html) – bestimmt an einer Saros-Version vom 25.03.2011

Für die Beschreibungen der vier von mir ausgewählten Pakete `concurrent` und dessen Unterpakete `concurrent.jupiter`, `concurrent.management` und `concurrent.undo` habe ich den Quelltext (bzw. die vorhandenen Kommentare) studiert und die Diplomarbeiten von Oliver Rieger [Rie08, S. 63–85] und Sebastian Ziller [Zil09, S. 13–35] herangezogen.

Um eine solche Paketbeschreibung im Quelltext anzulegen, wird pro Paket eine `package-info.java` Datei benötigt.[19] Diese wird in dem jeweiligen Paket-Ordner abgespeichert.

Die Paketbeschreibungen in Anhang C.2, S. 100 können im Quelltext bzw. der Quelltext-Dokumentation unter http://saros-build.imp.fu-berlin.de/sarosDoc/ angesehen werden. In Quellcode 2 ist als Beispiel die Beschreibung des Pakets `concurrent` aufgeführt.

Quellcode 2 – *Beschreibung des Pakets* `concurrent`

```
1  /**
2   * <h1>Concurrency Control Overview</h1>
3   *
4   * The Concurrency Control supports writing collaborations
5   * of multiple users. All participants of a Saros session
6   * shall have a consistent copy of all shared resources.
7   *
8   * The Concurrency Control comprises of four subpackages:
9   *
10  * <ul>
11  * <li>jupiter --- this package contains the Jupiter
12  * Algorithm used for collaborative text editing</li>
13  *
14  * <li>management --- manages all JupiterActivities
15  * (operations processed by the users of a Saros session
16  * with WRITE_ACCESS)</li>
17  *
18  * <li>undo --- package for undo/redo operations within a
19  * Saros session</li>
20  *
21  * <li>watchdog --- since inconsistencies might appear the
22  * Consistency Watchdog takes care of them</li>
23  * </ul>
24  */
25 package de.fu_berlin.inf.dpp.concurrent;
```

Bedauerlicherweise können die Javadoc-Informationen der Pakete nicht, wie der Rest der Javadoc-Kommentare, im Eclipse-Editor angezeigt werden, wenn man die Maus

[19] Früher hat man dafür eine `package.html` Datei geschrieben, aber seit dem Java Development Kit (JDK) 5.0 wird eine `package-info.java` verwendet.

über den Namen im Quelltext bewegt (integrierte Javadoc-Ansicht). Diese Problematik wurde schon 2006 im Eclipse Bug Tracker adressiert[20], bis heute aber nicht gelöst. Dennoch lassen sich die Paketbeschreibungen auf zwei Wegen gut erreichen: über die in Abschnitt 4.3, S. 44 beschriebene Quellcode-Dokumentation (*Saros Javadoc*) und direkt im Eclipse-Editor, wenn die `package-info.java` Datei geöffnet wird. Mithilfe dieser Beschreibungen können die Entwickler hilfreiche Informationen über die Pakete erhalten, weshalb die fehlende Javadoc-Ansicht für Pakete in Eclipse keinen Hinderungsgrund darstellen sollte, diese Beschreibungen anzufertigen.

[20] https://bugs.eclipse.org/bugs/show_bug.cgi?id=163633

5 Entwicklung und Einführung von Dokumentations-Richtlinien

Damit meine Verbesserung der Dokumentation auch in Zukunft weitergeführt werden kann, habe ich Methoden entwickelt, wie die Webseite und der Quelltext auch in Zukunft aktualisiert werden können. Mit meinen Ergänzungen habe ich einen Anfang geschaffen; dennoch gib es zu viele ‚Dokumentations-Baustellen', als dass diese während einer Diplomarbeit vollständig hätten überarbeitet und auf den neuesten Stand gebracht werden können. Der Quellcode z. B. ist sehr umfangreich und es gibt viele Stellen, an denen Kommentare fehlen oder überarbeitet werden müssen. Dies zu beheben ist nur möglich, wenn man sich in den jeweiligen Codeabschnitt eingearbeitet hat und somit sehr zeitintensiv.

In diesem Kapitel beschreibe ich, welche Aufgaben bzw. Richtlinien ich entwickelt habe, wie die Entwickler mit dem Thema Dokumentation umgehen sollten und wie sie aktuell gehalten werden kann. Dazu habe ich einen neuen *Documentation Issue Tracker* eingeführt, die Rolle eines Dokumentationsmanagers vorgestellt und seine Aufgaben definiert.

5.1 Neue Richtlinien für die Dokumentation

In einem ersten Schritt habe ich für die Entwickler eine Übersicht erstellt, die beschreibt, was sie kommentieren sollen und was dabei zu beachten ist. In den *Code-Rules*, welche die von den Entwicklern einzuhaltenden Richtlinien zur Programmierung festlegen, gibt es einen Abschnitt zum Thema *Language & Documentation*. Diesen habe ich hierzu verwendet und erweitert.

Damit die Entwickler ein besseres Verständnis für die verschiedenen Möglichkeiten der Kommentierung entwickeln können, habe ich alle sechs von Steve McConnell definierten Kommentararten beschrieben und erklärt, welche verwendet werden sollen (Zusammenfassungen, Beschreibungen der Absicht des Codes und Informationen, die nicht durch Code ausgedrückt werden können; siehe dazu auch Abschnitt 2.1.4, S. 14). Kurze Beispiele, die ich den Beschreibungen hinzugefügt habe, sollen eine bessere Vorstellung und die Möglichkeit zum Vergleich liefern.

Des Weiteren habe ich beschrieben, was kommentiert werden soll, wobei die Kommentierung des Quellcodes vom höheren zum niedrigeren Hierarchie-Level erfolgt.
Jedes Paket soll eine Beschreibung in Form einer `package-info.java` Datei enthalten. Mit Hilfe dieser Beschreibung kann sich der Entwickler einen ersten Überblick verschaffen, worum es in dem Paket geht. Außerdem wird der Zweck beschrieben, damit die Idee, die hinter dem Paket steht, nicht beim Programmieren zerstört wird (dies gilt natürlich für alle Kommentare).
Jedes Interface und jede Klasse soll dokumentiert werden. Bei Interfaces helfen die Kommentare zu verstehen, wie und wofür man die Methoden verwenden kann, ohne

dabei die eigentliche Implementierung zu beschreiben (information hiding). Klassenbeschreibungen liefern einen kurzen Überblick über die Klasse und erklären eventuelle Designentscheidungen.
Methoden sollen vor allem dann kommentiert werden, wenn sie sehr komplex sind. Eine Zusammenfassung bzw. Beschreibung der Absicht, die hinter der Methode steckt, hilft dem Entwickler, die Methode zu verstehen. Wichtige Objekte bzw. Variablen werden ebenfalls kurz und knapp dokumentiert.

Die vollständige Beschreibung der Richtlinien und wie mit Kommentaren umgegangen werden soll findet sich auf der Webseite unter https://www.inf.fu-berlin.de/w/SE/CodeRules#Language_Documentation und in Anhang C.1, S. 97.

5.2 Einführung des Documentation Issue Trackers

Wie geht man mit Problemen bezüglich der Dokumentation um? Während meiner Arbeit musste ich vielfach feststellen, dass die Dokumentation veraltet und unvollständig war, ohne dass daran etwas geändert wurde. Probleme scheinen zwar festgestellt und teilweise auch adressiert worden zu sein, nicht immer erfolgte aber eine Lösung des Problems und damit eine Aktualisierung der Dokumentation.

Meine Umfrage und auch Gespräche mit Entwicklern verdeutlichten mir, dass die Entwickler das Dokumentieren nicht immer als ihre Aufgabe ansahen. Nicht für jeden gehörte das Kommentieren zum Programmieren und oftmals wurden Kommentare und Quellcode getrennt betrachtet. In Bezug auf die Webseite wurde dies teilweise damit begründet, dass sie Angst hätten, Inhalte „kaputt zu machen“, wenn sie etwas an ihnen ändern würden; selbst wenn ihnen bewusst war, dass die vorhandenen Informationen veraltet oder falsch waren.

Dennoch konnte man sehen, dass es Bereiche gab, in denen sehr wohl dokumentiert wurde. Sobald beim Ausführen der Software etwas nicht funktionierte, wurde dieses Problem im *Bug Tracker* adressiert. Das bedeutete für mich, dass eine Dokumentation von Software-Problemen von Entwicklern offensichtlich durchaus vorgenommen wurde, sofern es eine Möglichkeit gab, dies schnell und unkompliziert tun zu können. Da das Dokumentieren der Software-Probleme wie auch deren Behebung, die natürlich umso wichtiger ist, mit dem *Bug Tracker* recht gut funktionierte, habe ich untersucht, ob es nicht möglich ist, ein ähnliches System für die Probleme mit der Dokumentation (Webseite bzw. Kommentare) einzuführen.

Es gibt verschiedene Tracking Systeme. In Bug Trackern werden Probleme mit der Software (Bugs) eingetragen und deren Behandlung dokumentiert. In den meisten Fällen hat jeder Zugriff auf die abzuarbeitenden Bugs und kann feststellen, in wie weit sich um das jeweilige Problem gekümmert wird. Erst, wenn der Bug behoben wurde, wird der Eintrag geschlossen. In Issue Tracking Systemen geht es um das Abarbeiten von Problemen oder Anliegen, die einzelne ‚Kunden‘ haben. Für jeden Kunden wird verfolgt, ob sein Anliegen zu seiner Zufriedenheit gelöst wurde und erst

dann geschlossen. Bei einem Issue Tracker kann es demnach mehrere Anfragen von verschiedenen Kunden zu einem Problem geben. Ein Beispiel für ein Issue Tracking System ist der vom Rechnerbetrieb der Informatik verwendete *Request Tracker*[21] (https://rt.imp.fu-berlin.de/rt/). Beispiele für Bug Tracker sind Bugzilla (http://www.bugzilla.org/) und Mantis (http://www.mantisbt.org/) neben den von Saros verwendeten Trackern bei Sourceforge (http://sourceforge.net/tracker/?group_id=167540) für Bugs oder Feature Requests.

Saros - Distributed Party Programming

Share

Summary | Files | Reviews | Support | Develop | Hosted Apps | Tracker | Mailing Lists | Code

Add new **Browse**

Tracker: Documentation Issues

Search: Search Advanced Options RSS

Page: Page 1 Next → 1 - 25 of 91 Results - Display 25

Assignee: Any Status: Any Category: Any Group: Any Submitter: Keyword: Artifact ID:

Filter Reset Permalink

ID	Summary	Status	Opened	Assignee	Submitter	Resolution	Priority
3408099	saros logo malfunction	Open	2011-09-12	nobody	translatemyname	None	5
3381245	DPPTechnicals - update dependency graph	Open	2011-07-28	nobody	meike11	None	5
3381242	rewrite DPPTesting	Pending	2011-07-28	kargor	meike11	Fixed	5
3381241	update DPPUsabilityEngineering	Open	2011-07-28	nobody	meike11	None	3
3381235	DPPGettingStarted - add How to add a new Buddy	Open	2011-07-28	nobody	meike11	None	5
3381233	update DPPCompatiblePlugins	Open	2011-07-28	nobody	meike11	None	7
3381232	DPP & DPPFeatures - mention Whiteboard	Open	2011-07-28	meike11	meike11	Fixed	5
3381230	DPP - move 'related work' and 'blog roll'	Open	2011-07-28	k_beecher	meike11	Fixed	1
3381228	DPPScreenshots - add screen sharing screenshot	Open	2011-07-28	nobody	meike11	None	3
3381225	DPPScreenshots - add whiteboard screenshot	Open	2011-07-28	nobody	meike11	None	3
3381224	DPPFeatures - missing feature page	Open	2011-07-28	meike11	meike11	Fixed	5
3381223	DPP - add illustrations for the scenarios	Open	2011-07-28	nobody	meike11	None	3
3381221	Saros videosharing.source - missing package description	Open	2011-07-28	nobody	meike11	None	7

Abbildung 20 – *Ausschnitt aus dem* Documentation Issue Tracker.

Bei meiner Recherche der Bug Tracking Systeme habe ich herausgefunden, dass es möglich ist, bei Sourceforge eigene Tracker anzulegen. Somit habe ich mich dazu entschieden, dort einen neuen Tracker für die Dokumentation zu erzeugen, da die

[21] Der Request Tracker (http://bestpractical.com/rt/) wird seit 1996 von Jesse Vincent entwickelt und wird unter der General Public Licence bei Best Practical Solutions LLC vertrieben.

Studenten schon an die Verwendung des dort vorhandenen *Bug Trackers* sowie des *Feature Request Trackers* gewöhnt sind. In dem neuen Tracker sollen sowohl das Fehlen von Informationen (z. B. eine Paketbeschreibung im Quellcode), als auch falsche oder veraltete Informationen oder andere Probleme mit der Dokumentation gemeldet werden. Ich habe ihn *Documentation Issue Tracker* genannt. Der Begriff ‚Bug' für ein (Programm-) Versagen ist nicht zutreffend, ‚Issue' für Problem oder Angelegenheit scheint daher besser zu passen.

Nachdem ich den neuen *Documentation Issue Tracker* bei Sourceforge[22] angelegt hatte, habe ich ihn mit Einträgen gefüllt (siehe Abbildung 20 auf der vorherigen Seite). Dazu durchsuchte ich den Quellcode, um herauszufinden, an welchen Stellen Kommentare fehlen. Auf diese Weise konnte ich eine Liste mit über 130 Issues erstellen, die sich hauptsächlich auf fehlende Paket-, Interface- und Klassenbeschreibungen beziehen. Die Untersuchung der Saros-Webseite ergab noch einmal ein Dutzend weiterer Issues, die ich als zweite Kategorie meiner Liste hinzufügen konnte. Die Liste zur Webseite ist um ein Wesentliches kürzer, da ich viele Probleme schon während der Umstrukturierung und Erweiterung der Webseite ergänzen bzw. lösen konnte. Die gesamte Liste habe ich anschließend in den *Documentation Issue Tracker* übertragen, wobei einige Einträge zusammengefasst werden konnten, so dass ich schließlich 90 neue Einträge anlegte.

Für den Umgang mit den Dokumentations-Problemen habe ich eine Anleitung analog zu den *Bug Tracking Procedures* verfasst und als *Documentation Issue Procedures* der Saros-Webseite hinzugefügt. Hiernach sollen neue Einträge jeweils einer der beiden Kategorien ‚Webseite' und ‚Quellcode' zugeordnet werden, wobei der Tracker die Möglichkeit bietet, die Einträge zu priorisieren. Auch bezüglich der Dokumentation gibt es Issues, die wichtiger sind als andere. Also sollen auch den Documentation Issues Prioritäten zugeordnet werden (vgl. auch den Exkurs im Anschluss an diesen Abschnitt).

Der Dokumentationsmanager, dessen Rolle ich in Abschnitt 5.3, S. 56 beschreibe, weist den Entwicklern Issues zu, überprüft, ob das Issue bearbeitet wurde und schließt es dann. Issues mit der höchsten Priorität sollten dabei vorrangig behandelt werden. Deren Bearbeitung sollte, vergleichbar derjenigen bei den Bugs, entsprechend der Einarbeitung des Entwicklers in das jeweilige Thema erfolgen, um somit von seiner Kenntnis im Problembereich profitieren zu können.

Kommentare sind Teil des Quellcodes, aus diesem Grund sollten sie auch als ein solcher behandelt werden. Besonders, wenn die ergänzenden Kommentare umfangreich sind (z. B. bei Paketbeschreibungen), sollten sie von anderen Entwicklern durch ein Review[23] abgesegnet werden. So können die anderen Entwickler auch gleich überprüfen, ob der Kommentar ihnen weiterhilft, den Code zu verstehen.

[22] http://sourceforge.net/tracker/?group_id=167540&atid=2318917
[23] Code-, in diesem Fall genauer: Kommentardurchsicht

Den *Documentation Issue Tracker* (der auf der Webseite verlinkt ist) sowie die neuen Anleitungen und Richtlinien zur Behandlung von Dokumentations-Problemen habe ich dem Saros-Team vorgestellt. Die *Documentation Issue Procedures* finden sich auf der Webseite unter `http://www.saros-project.org/DocuTracking` und in Anhang D.1, S. 103.

5.2.1 Exkurs: Prioritäten

Die Einordnung der verschiedenen Documentation Issues ist nicht ganz einfach. Im *Bug Tracker* kann man zwischen neun verschiedenen Prioritätsstufen (9 – 1) unterscheiden, wobei die Zuordnung nicht für alle Stufen vollständig erläutert ist. Die Stufen 9, 8 und 7 sind definiert: 9 soll für einen Bug ausgewählt werden, der sofort repariert werden muss. (Das sollte also hoffentlich äußerst selten vorkommen.) 8 bedeutet, der Bug muss bis zum nächsten Release gelöst worden sein und ein Bug der Priorität 7 sollte bis zum Release erledigt sein, bzw., falls dies nicht möglich ist, sollte es zumindest einen guten Grund dafür geben. Die Stufen 6 – 2 sind dagegen nicht genauer spezifiziert. Das obliegt der Einschätzung des Entwicklers, der die Priorisierung vornimmt. Stufe 1 steht für Bugs, die noch nicht eingeordnet sind. Obwohl es diese neun Stufen gibt und auch alle im *Bug Tracker* Verwendung finden, basiert die Zuordnung der Prioritäten 6 – 2 doch auf einer eher subjektiven Entscheidung.

Für den *Documentation Issue Tracker* könnte eine ähnliche Einstufung vorgenommen werden, doch stellte sich mir die Frage, ob wirklich neun verschiedene Prioritäten notwendig sind oder nicht vielleicht fünf oder nur drei ausreichen. Meine Recherche zu Einstufungsmöglichkeiten und Stufenanzahl gestaltete sich schwierig. Publikationen, die die Stufenanzahl bei Skalen untersuchen, beziehen sich meist auf die Gestaltung von Umfragebögen oder die Priorisierung im „Requirements Engineering“. Doch denke ich, dass sich diese Ergebnisse für die Einstufung von Documentation Issues verwenden lassen, denn auch hier geht es darum, dass von den Entwicklern abgestufte Einschätzungen vorgenommen werden sollen.

Larry Fellows und Ivy Hooks beziehen sich bei ihrer Prioritätsklassifizierung von Anforderungen auf eine von Ian Sommerville und Pete Sawyer vorgenommene Terminologie, welche drei Stufen vorsieht: „essential“, „useful“ und „desirable“.[24] Darauf Bezug nehmend stellen sie heraus: „It does not pay to have many priority classifications. Three priorities are sufficient to achieve the goals of reflecting the relative importance of requirements.“ [FH98, (S. 3)].

Ebenfalls drei Stufen für die Anforderungspriorisierung empfiehlt Karl E. Wiegers und nennt sie „high“, „medium“ und „low“ oder „essential“, „conditional“ und „optional“. Er merkt hierzu an: „Keep the prioritization as simple as possible to help you make the necessary development choices.“ [Wie99, S. 3].

[24] Ian Sommerville and Pete Sawyer: *Requirements Engineering: A Good Practice Guide*. John Wiley and Sons, West Sussex, England, 1997. Zitiert in [FH98, (S. 3)].

Noriko Iwai untersucht in seiner Arbeit Methoden, wie Daten durch Umfragen erhoben werden. Ein Experiment sollte ermitteln, ob ein Unterschied im Antwortverhalten entsteht, wenn fünf statt drei graduelle Antwortmöglichkeiten vorgegeben werden. Zusätzlich gab es noch die Möglichkeit, „Don't know“ zu wählen. Das Ergebnis zeigte mehr „Don't know“-Antworten bei Fragen mit drei Wahloptionen (24.6%) als bei Fragen mit fünf (17.1%; vgl. [Iwa05, S. 95]).
Ich denke, dass dieser höhere prozentuale Anteil darin zu suchen ist, dass der Antwortende seine Meinung nicht in den Wahlmöglichkeiten wiederfindet und diese daher überarbeitet bzw. auf fünf Antworten verfeinert werden müssen.

So werden beispielsweise in anderen Bug Tracking Systemen fünf Prioritätsstufen verwendet. Dazu gehören z. B. Bugzilla (`http://www.bugzilla.org/`), Roundup (`http://www.roundup-tracker.org/`) und Track (`http://trac.edgewall.org/`). Bei Mantis (`http://www.mantisbt.org/`) gibt es neben den fünf Stufen „immediate“, „urgent“, „high“, „normal“ und „low“ noch eine sechste („none“), wenn noch keine Priorität ausgewählt wurde.

Es scheint also für die Einstufung der Documentation Issues nicht notwendig zu sein, die neun Priotiätsstufen der *Sourceforge Tracker* und damit die fein unterteilte Einstufung zu übernehmen. Drei bis fünf Stufen scheinen hierbei ausreichend zu sein.

Ich habe mich dazu entschieden, unter Verwendung der ungeraden Prioritäten im Sourceforge *Documentation Issue Tracker* fünf Stufen zu realisieren: 9, 7, 5, 3 und 1. Dabei sollen die mittleren drei die meisten Bedürfnisse abdecken: Basierend auf meinen Regeln, was kommentiert werden sollte, lässt sich in der Betrachtung der Quellcode-Kommentare für die Prioritäten eine Einstufung in drei hierarchisch geordnete Stufen vornehmen: 7 für Pakete, 5 für Interfaces bzw. Klassen und 3 für Methoden. Die niedrigste Priorität (1) soll für noch nicht klassifizierte Issues oder für Anmerkungen verwendet werden, die nicht in Vergessenheit geraten sollen. Die höchste Stufe (9) soll nur für ernsthafte Probleme, die sofort gelöst werden müssen, genutzt werden (z. B. wenn die Saros-Webseite nicht erreichbar ist).

5.3 Einführung der Rolle des Dokumentationsmanagers

Unter Eigenverantwortung der Entwickler wurde die Dokumentation nicht regelmäßig aktualisiert. Um daran etwas zu ändern, habe ich die Rolle des Dokumentationsmanagers eingeführt. So wie der Testmanager prüfen soll, ob die durch „fixed“ markierten Bugs tatsächlich gelöst wurden, soll auch der Dokumentationsmanager überprüfen, ob Mängel in der Dokumentation behoben wurden. Wenn dies der Fall ist, sollen diese Issues im *Documentation Issue Tracker* geschlossen werden. Außerdem soll der Dokumentationsmanager nach ‚neuen‘ Problemen mit bzw. Defekten in der Dokumentation suchen, denn bei neu geschriebenem oder verändertem Quellcode kann es vorkommen, dass dieser nicht ausreichend kommentiert wurde. Auch

neu entwickelte Features sollen auf der Webseite beschrieben werden, damit der Leser gleich sehen kann, was ihm Saros bietet. So kann es also Bereiche geben, an denen die Dokumentation überarbeitet oder ergänzt werden muss, und diese sollen nun in den *Documentation Issue Tracker* eingetragen werden, damit sie aufgenommen und anschließend bearbeitet werden können. Ein erster Anhaltspunkt, um festzustellen, wo die Dokumentation überprüft werden muss, kann für den Dokumentationsmanager die vom Releasemanager erstellte Liste von neu reparierten Bugs und neuen Features sein.

Wenn es in der Liste des *Documentation Issue Trackers* Einträge gibt, denen keine Priorität zugeordnet wurde (d.h. sie haben die Priorität 1), so ist es die Aufgabe des Dokumentationsmangers, diese Einträge entsprechend ihrer Wichtigkeit einzuordnen. Die Hauptaufgabe während der Release-Woche ist es jedoch, passende Einträge den jeweiligen Entwicklern zuzuweisen. (Passend heißt hierbei, dass der jeweilige Entwickler das Wissen hat, den Issue zu beheben, da es kaum sinnvoll ist, jemanden, der sich z. B. in den Einladungsprozess von Saros eingearbeitet hat, darum zu bitten, Paketbeschreibungen zum Whiteboard zu schreiben.) Wenn der Dokumentationsmanager nicht genau weiß, wer welches Problem beheben könnte, können ihm die anderen Mitglieder des Saros-Teams helfen. Grundsätzlich sollen dabei zunächst die Einträge mit der höchsten Priorität abgearbeitet werden.

Um die anderen Entwickler darüber zu informieren, wer welche Issues lösen soll, schickt der Dokumentationsmanager eine E-Mail mit den zugewiesenen Issues an die Mailing-Liste. Ein Beispiel dafür habe ich in der Liste der Aufgaben des Dokumentationsmanagers auf der Webseite angegeben.

Alle Entwickler, die Issues zugewiesen bekommen haben, sollen diese bearbeiten. Bei Bugs in der Software ist es wichtig, dass die Probleme sehr schnell gelöst werden, damit die Software in der Release-Woche herausgegeben werden kann. Unvollständige oder fehlende Dokumentation muss überarbeitet werden, sie wird die Software aber nicht direkt zum Abstürzen bringen. Da in einer Release-Woche sehr viel Arbeit anfallen kann, scheint es vertretbar, wenn Probleme mit der Dokumentation nicht sofort behoben werden, sofern die Entwickler etwas Wichtigeres zu tun haben (es sei denn, es handelt sich um eine Angelegenheit von Priorität 9). Dennoch leidet die Qualität der Software darunter, wenn sie nicht genügend dokumentiert ist. Wenn ein Entwickler beispielsweise einen Codeabschnitt überarbeitet, verändert oder verwendet, in dem der Zweck nicht beschrieben wurde, kann es passieren, dass er ihn falsch versteht und möglicherweise Defekte einbaut, die mit ausreichender Dokumentation hätten vermieden werden können. Aus diesem Grund sollten die vom Dokumentationsmanager zugewiesenen Issues bis zum nächsten Release behoben werden. Alle behobenen Issues werden analog zu den behobenen Bugs auf „fixed“ gesetzt, damit sie beim nächsten Release als „neu behoben“ gefunden und überprüft werden können.

Diese Aufgaben des Dokumentationsmanagers habe ich auf der Saros-Webseite *Our Release Schedule*[25] hinzugefügt (sie finden sich auch in Anhang D.2, S. 104). Die neue Managerrolle soll zeitlich parallel zu den beiden anderen Rollen des Test- und Releasemanagers erfolgen.

Da die Aufgaben des Dokumentationsmanagers nicht direkt vom Release abhängig sind, gibt es verschiedene Möglichkeiten, wie dessen Arbeit zeitlich eingeordnet werden könnte: für einen ganzen Monat, innerhalb einer Woche im Monat, die nicht mit der Release-Woche zusammenfällt oder während der Release-Woche.

Wenn – wie in den ersten beiden Fällen – ein Student für einen Monat oder für eine Woche im Monat Dokumentationsmanager ist, könnte es vorkommen, dass er zusätzlich in der Release-Woche nochmal eine andere Rolle (als Test- oder Releasemanager) zugewiesen bekommt. Er wäre dann in dem Monat mehr mit den Managerrollen beschäftigt als mit seiner Abschlussarbeit. Um dies im Vorhinein zu umgehen, sollte der Dokumentationsmanager ebenfalls in der Release-Woche tätig sein. So können die Manageraufgaben gerechter unter allen Team-Mitgliedern verteilt werden.

Diese Variante birgt allerdings den Nachteil, dass drei Rollen gleichzeitig besetzt werden müssen, was problematisch sein kann, wenn zu wenige Team-Mitglieder verfügbar sind. Dies zeigte sich beispielsweise in der letzten Juli-Woche, als die Rolle des Dokumentationsmanagers zum ersten Mal zum Einsatz kam und wir nur drei Studenten waren, die die Managerrollen übernehmen konnten. Normalerweise wird jede Managerrolle von jeweils zwei Leuten besetzt: ein Manager und ein Assistent, der den Manager in seine Rolle einführt, da der Assistent die Aufgaben schon einmal übernommen hat und sich damit auskennt. Während der Bereich des Dokumentationsmanagers von einer Person übernommen werden kann, ist der Assistent für die beiden anderen Rollen sehr wichtig, um die Aufgaben richtig ausführen zu können. Mit nur drei verfügbaren Studenten konnten wir die Managerrollen gerade so besetzen, indem sowohl der Test- als auch der Releasemanager von meinem Betreuer unterstützt wurden.

In den anderen Release-Wochen während meiner Arbeit im Saros-Team gab es keinen so extremen ‚Engpass', weshalb ich es für vertretbar halte, alle Managerrollen parallel in der Release-Woche zu besetzen. Auf diese Weise wird die Arbeit zur Fertigstellung der aktuellen Version gebündelt und die Studenten können sich darauf einstellen, dass sie innerhalb der Release-Woche weniger Zeit für ihre Abschlussarbeiten haben werden. Gleichzeitig wird so ermöglicht, dass sie sich verstärkt um das Beheben von Problemen (sowohl Bugs als auch Documentation Issues) kümmern können.

[25] http://www.saros-project.org/ReleaseSchedule#Tasks_of_the_DM

6 Auswertung

Zum Abschluss meiner Arbeit habe ich eine zweite Umfrage durchgeführt, um festzustellen, inwiefern sich die vorgenommenen Änderungen hinsichtlich der Dokumentation ausgewirkt haben. Dazu habe ich noch einmal Fragen aus meinem ersten Fragebogen im Mai 2011 aufgegriffen, um die Ergebnisse beider Umfragen vergleichen zu können.

Von den ursprünglich 30 Fragen des ersten Fragebogens habe ich 18 für den zweiten Fragebogen ausgewählt, die sich für einen Vorher-Nachher-Vergleich anbieten, und diesen an die Saros-Mailingliste geschickt. Zusätzlich habe ich den Fragebogen von den Studenten des Softwareprojekts ausfüllen lassen, die in diesem Zeitraum gerade an Saros gearbeitet haben. Mit dieser neuen Gruppe sollte die Anzahl der Teilnehmer an der Umfrage erhöht werden. Da sich diese Studenten für vier bis fünf Wochen intensiv mit Saros beschäftigt hatten, waren sie weit genug eingearbeitet, um die Fragen beantworten zu können. Andererseits waren sie noch nicht so tief ins Projekt eingestiegen, so dass sie an die Fragen unbefangener herangehen konnten.

6.1 Auswertung der zweiten Umfrage (im September 2011)

Von den 18 Fragen, die ich für den zweiten Fragebogen ausgewählt habe, betrafen 14 die Webseite und vier den Quelltext von Saros, wobei eine unter diesen vier Quelltext-Fragen in sieben Unterfragen aufgeteilt war (siehe Anhang A.4, S. 78).

Da viele meiner Neuerungen bzw. Änderungen die Webseite betrafen, lassen sich die Aussagen auf die Fragen zur Webseite gut gegenüberstellen. Hinsichtlich des Quelltextes konnten sich dagegen zum Zeitpunkt der zweiten Umfrage die Neuerungen – wie die Einführung des Dokumentationsmanagers und der Richtlinien (wie bzw. was kommentiert werden soll), sowie zum Umgang mit Dokumentationsproblemen – für die Entwickler noch nicht so deutlich auswirken, da zum Beispiel der Dokumentationsmanager in der Zwischenzeit erst ein einziges Mal (beim letzten Release Ende Juli) zum Einsatz gekommen war. Auch meine exemplarischen Paketbeschreibungen zum Paket `concurrent` und dessen Unterpaketen sowie die durch die Einführung des Dokumentationsmanagers von anderen Entwicklern ergänzten Kommentare umfassen auf die Gesamtmenge der fehlenden Quellcode-Kommentare nur einen kleinen Teil. Die Kommentarabdeckung lag im September bei 43%[26], was gegenüber den vorher bestimmten 42% keine signifikante Änderung zeigt. Diese verhältnismäßig wenigen Änderungen an den Kommentaren spiegelten sich in den Antworten zum Quellcode in der Zweiten Umfrage wider, da sie kaum von denen der ersten Umfrage abwichen.

[26] laut Doc Check Doclet v 1.2 Beta 2 (http://www.oracle.com/technetwork/java/javase/documentation/index-141437.html) – bestimmt an einer Saros-Version vom 25.09.2011

Im Gegensatz zu den zwölf Teilnehmern der ersten Umfrage (acht Programmierer und vier Nicht-Programmierer) haben an meiner zweiten Umfrage insgesamt 18 Personen teilgenommen, darunter zwölf Studenten des Softwareprojekts, zwei wissenschaftliche Mitarbeiter der AG SE und vier Studenten, die ihre Abschlussarbeiten im Saros-Team schreiben bzw. geschrieben haben. Von den letztgenannten vier Studenten, die mir den Fragebogen beantwortet haben, waren auch zwei, die neu ins Team gekommen sind und dadurch teilweise sogar noch weniger Zeit hatten, sich ins Projekt einzuarbeiten, als die Softwareprojekt-Studenten.
Zwei Teilnehmer (Nicht-Programmierer) haben keine Antworten zum Quellcode gemacht, so dass ich zu den Quellcode-bezogenen Fragen von 16 Personen (Programmierer) Antworten erhalte habe. Die Fragen zur Webseite wurden von allen Teilnehmern beantwortet.

6.1.1 Fragen zur Webseite

In meiner ersten Umfrage wurden bei den Fragen nach den hilfreichen sowie nach den genutzten Themen hauptsächlich der *Developer Guide*, gefolgt von den *Technical Information* genannt. Hinzu kamen die Angaben der Installationsanweisungen bzw. *Getting Started* und die Links zum *Bug Tracker* und zu den *Code Rules*. In meiner zweite Umfrage gab es auf diese Frage hin sehr viele Nennungen der *Technical Information*, des *Developer Guides* sowie der Quellcode-Dokumentation *Saros Javadoc*. Besonders bei den genutzten Themen wurden die *Technical Information* – teilweise mit zusätzlicher Angabe der genutzten Modulbeschreibung (z. B. *Concurrency Control*, *UI* oder *Network Layer*) – hervorgehoben. Zusätzlich wurden auch die Installationsanweisungen (bzw. *Set-up & Check-out*), der *Bug Tracker* und das *Review Board* aufgeführt.
Somit wurden die durchgeführten Änderungen an der Webseite und vor allem die neu hinzugefügten Informationen (*Saros Javadoc* und die Modulbeschreibungen in *Technical Information*) angenommen.

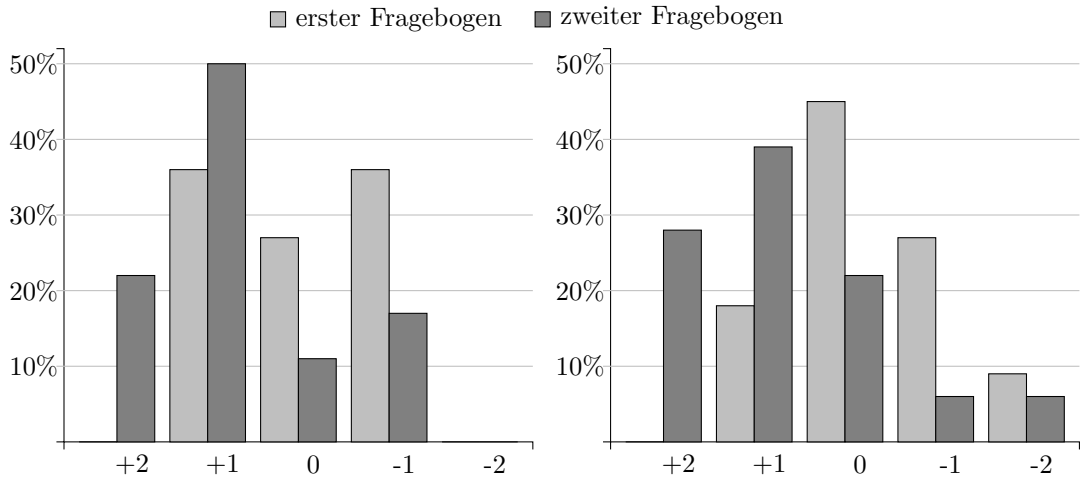
Bei der Frage nach den nicht hilfreichen Themen wurde erneut *How to be Happy with Saros* aufgeführt. Ein Student hat dies damit begründet, dass der Blocktext zu unübersichtlich sei. Somit sollte diese Seite noch überarbeitet werden.

Als fehlende Themen wurden im ersten Fragebogen unter anderem eine Feature-Liste, eine Architekturbeschreibung und Paket- bzw. Klassenbeschreibungen genannt. Diese konnte ich auf der neuen Seite *Features*, in den *Technical Information* sowie den neuen Modulbeschreibungen ergänzen. Zusätzlich wurden seit der Einführung des Dokumentationsmanagers, neben meinen vier, 15 weitere Paketbeschreibungen hinzugefügt, welche in der Quelltext-Dokumentations *Saros Javadoc* eingesehen werden können. Da die zuvor genannten Themen in der zweiten Umfrage nicht mehr aufgeführt wurden, stellen sich meine thematischen Ergänzungen als erfolgreich heraus.

Dafür wurden neue fehlende Themen benannt, wie Tutorials oder Videos, eine Verlinkung zu den Abschlussarbeiten und eine Danksagungs-Seite, auf der die Nutzer leicht ihre Meinung zu Saros abgeben können. Dies sind also Themen, die neu in den Fokus gerückt sind und entsprechend auch noch ergänzt werden sollten.

Die Frage „Hast du mit Hilfe der Webseite einen leichten Einstieg in Saros bzw. in dir vorher unbekannte Aspekte bekommen?“ wurde überwiegend zustimmend beantwortet. Im Gegensatz zu den Antworten im ersten Fragebogen, in dem bei dieser Frage die Unübersichtlichkeit der Webseite verstärkt bemängelt wurde, haben zwei Teilnehmer ihre Antwort durch „Ja, dank der Seite ist ein leichter Einstieg ermöglicht [sic].“ bzw. „Ist schön übersichtlich und ich hab schnell das gefunden, was ich suchte.“ ergänzt.

Die Fragen zur Textverständlichkeit habe ich in meinem zweiten Fragebogen nur allgemein auf die Webseite bezogen und vergleiche sie daher nur mit den Antworten zur ‚Webseite allgemein‘ des ersten Fragebogens. Da es bei den Fragebögen jeweils eine unterschiedliche Anzahl an Teilnehmern gab, habe ich die Balkendiagramme normiert dargestellt, um einen Vergleich zu ermöglichen.



(a) *Frage 6: „Der Text ist gegliedert/hat eine Ordnung. (+2) ↔ Der Text ist unübersichtlich/zusammenhangslos. (-2)“*

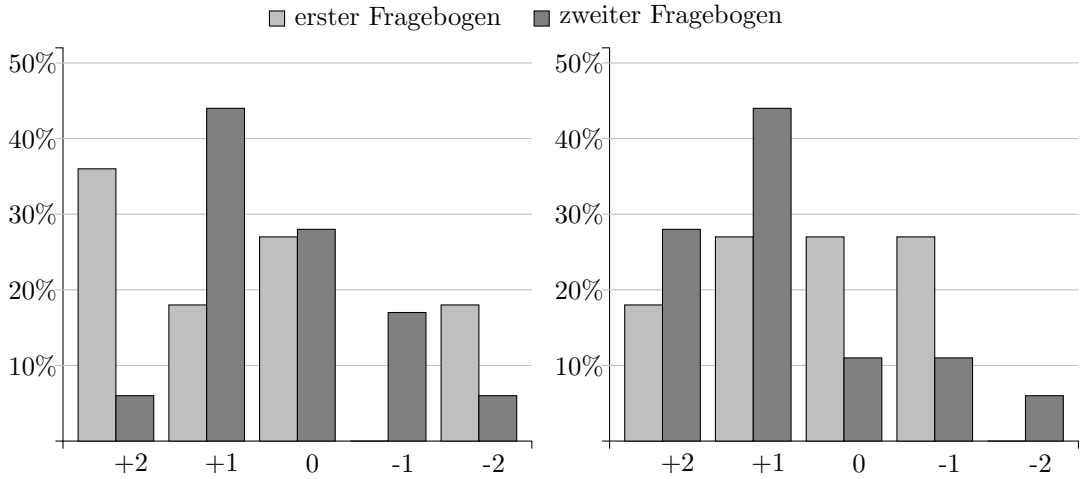
(b) *Frage 7: „Du findest die Informationen dort, wo du sie erwartest. (+2) ↔ Die Information ist an einem Ort, wo sie nicht hingehört. (-2)“*

Abbildung 21 – *Antworten zu den Fragen 6 und 7*

Im Vergleich zu den Antworten in meiner ersten Umfrage, in der es keine klare Tendenz gab, ob der Text gegliedert ist bzw. eine Ordnung besitzt, wurde der Text in der zweiten Umfrage als gegliedert eingestuft (siehe Abbildung 21a). Ebenso wurden Themen dort gefunden, wo sie erwartet wurden (siehe Abbildung 21b).
Wo sich die Teilnehmer in der ersten Umfrage nicht einig waren, ob Informationen schnell wiedergefunden werden können oder nicht, waren in der zweiten prozentual

mehr Teilnehmer vom schnellen Wiederfinden der Informationen überzeugt.
Meine neue Gliederung und thematische Umgruppierung der Navigation mit der neuen Übersichtsseite *Developer Guide* beispielsweise, welche kurz erklärt, was jeweils auf deren Unterseiten zu finden ist, hat das Auffinden der Informationen verbessert.

Anders als zuvor wurde der Text der Webseite verstärkt als einfach geschrieben beurteilt. Bei der Frage, ob der Text kurz und prägnant oder weitschweifig abgefasst ist, ergab sich eine Veränderung in der Bewertung, indem diese nun eher in der Mitte der Auswahlskala vorgenommen wurde (siehe Abbildung 22a). Dies entspricht der Einschätzung von Pötter für einen verständlichen Text. Er merkt an: „Extrem knappe Texte sind nahezu genauso schwer verständlich, wie weitschweifige Textpassagen.“ [Pöt94, S. 167].



(a) *Frage 10: „Der Text ist kurz und prägnant. (+2) ↔ Der Text ist weitschweifig gefasst. (-2)“*

(b) *Frage 12: „Die Angaben sind korrekt (sie stimmen mit der Realität überein). (+2) ↔ Die Angaben sind fehlerhaft. (-2)“*

Abbildung 22 – *Antworten zu den Fragen 10 und 12*

Von den Teilnehmern wurde der Text weiterhin als aussagekräftig angesehen. Im Vergleich zur ersten Umfrage wurden die Angaben verstärkt als korrekt beurteilt (siehe Abbildung 22b). Bei der Frage zur Vollständigkeit wurde der Text tendenziell als vollständiger beurteilt als zuvor, was zeigt, dass es immer noch Angaben bzw. Themen gibt, die ergänzt werden sollten, wie es auch in den Antworten zur Frage nach den fehlenden Themen (z. B. Tutorials und Videos) zu erkennen war.
Durch das Ergänzen der ursprünglichen fehlenden Themen wurden neue Wünsche oder Forderungen aufgedeckt, die vorher nicht erkannt oder als wichtig betrachtet wurden. Es zeigt sich dennoch, dass die Erweiterungen der Webseite diese vollständiger machten und sich zudem meine Korrekturen an den Informationen der alten Seite ausgezahlt haben, da die Angaben nun als korrekt bezeichnet wurden.

Auf die Frage „Ergänzt bzw. aktualisierst du Informationen auf der Webseite, die mit deinem Teil der Arbeit an Saros zu tun haben?“ wurden alle Antwortmöglichkeiten von „meistens“ bis „nie“ genannt, wobei die Studenten des Softwareprojekts einstimmig mit „nie“ geantwortet haben. Dazu lässt sich aber ergänzen, dass es nicht Aufgabe der Softwareprojekt-Studenten war, an der Webseite Veränderungen vorzunehmen und die anderen Antworten von Entwicklern des Saros-Teams stammten, die fast ausschließlich neu im Team sind und infolgedessen nicht alle die Gelegenheit hatten, die Webseite zu bearbeiten.

6.1.2 Fragen zum Quellcode

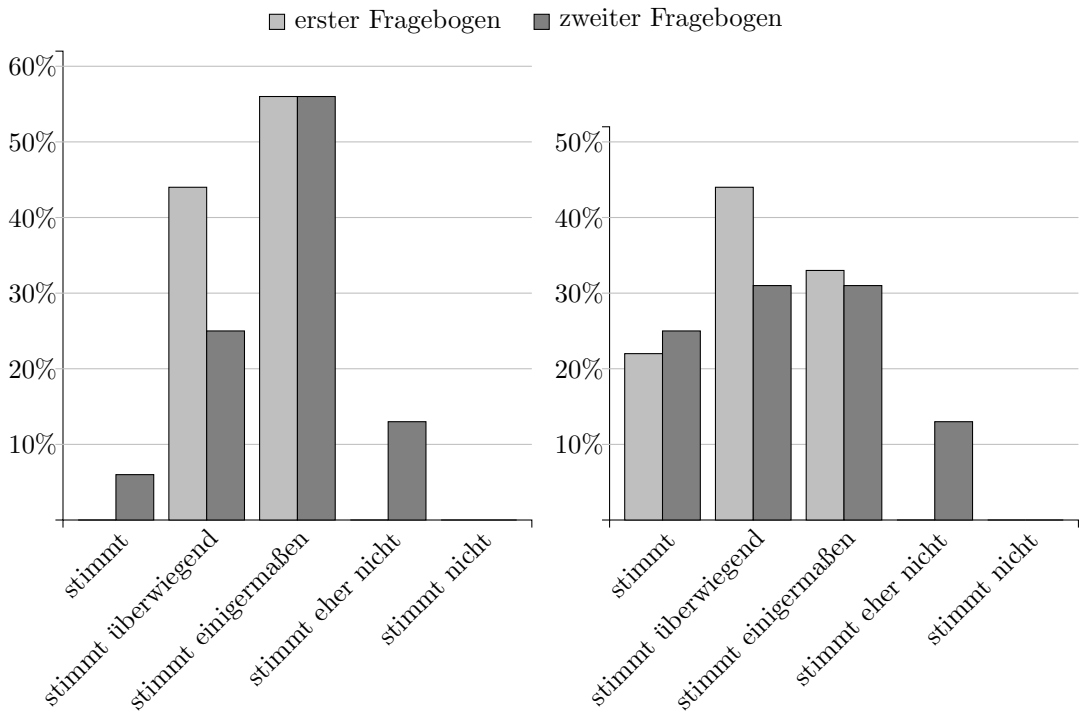
Die den Quellcode betreffenden Fragen wurden in meiner ersten Umfrage nur von fünf der zwölf Teilnehmern beantwortet. Von den 18 Teilnehmern der zweiten Umfrage erhielt ich in diesem Fragenblock 16 Antworten. Aus den wenigen Antworten vom Mai 2011 konnten nur Tendenzen zur Bewertung der Kommentare im Quellcode entnommen werden. Durch die höhere Antwortanzahl der zweiten Umfrage im September lassen sich die Kommentare im Quellcode etwas genauer beleuchten. Dennoch muss bei der Bewertung auch hier berücksichtigt werden, dass viele der Teilnehmer dieser zweiten Umfrage keine so umfassende Einarbeitungszeit in das Projekt und insbesondere in den Quellcode hatten, wie die fünf Entwickler, von denen die Antworten in der ersten Umfrage stammten und die alle schon länger im Saros-Team tätig waren.

Wie schon bei den Fragen zur Textverständlichkeit habe ich für einen Vergleich die Balkendiagramme normiert dargestellt. Die Antworten zum ‚eigenen‘ sowie zum ‚fremden‘ Code aus dem ersten Fragebogen habe ich dabei zusammengefasst, da ich im zweiten Fragebogen meine Fragen allgemein zum Quelltext gestellt hatte.

Auch die zweite Umfrage ergab, dass die Kommentare den Code eher nicht wiederholen sondern erklären. Dabei wurde der Frage „Die Kommentare erklären den Code“ tendenziell aber weniger zugestimmt als in der ersten Umfrage (siehe Abbildung 23a auf der nächsten Seite).

In der Tendenz wurde auch der Frage, ob die Kommentare Markierungen im Code darstellen etwas weniger zugestimmt als zuvor (siehe Abbildung 23b auf der nächsten Seite). Dieser geringe Unterschied erscheint jedoch nicht gravierend genug, um feststellen zu können, ob es im Vergleich zu vorher tatsächlich weniger Markierungen im Code gibt, die abgearbeitet werden müssen.

Die Programmierer der zweiten Umfrage waren sich im Gegensatz zu denen der ersten, die sich länger in das Projekt eingearbeitet hatten, nicht einig, ob die Kommentare den Code zusammenfassen (siehe Abbildung 24a, S. 65). Bei der Frage „Die Kommentare beschreiben den Zweck/was der Code bewirken soll“ gab es hingegen kaum einen Unterschied zur ersten Umfrage (siehe Abbildung 24b, S. 65).



(a) *Frage 15b: „Die Kommentare erklären den Code.“*

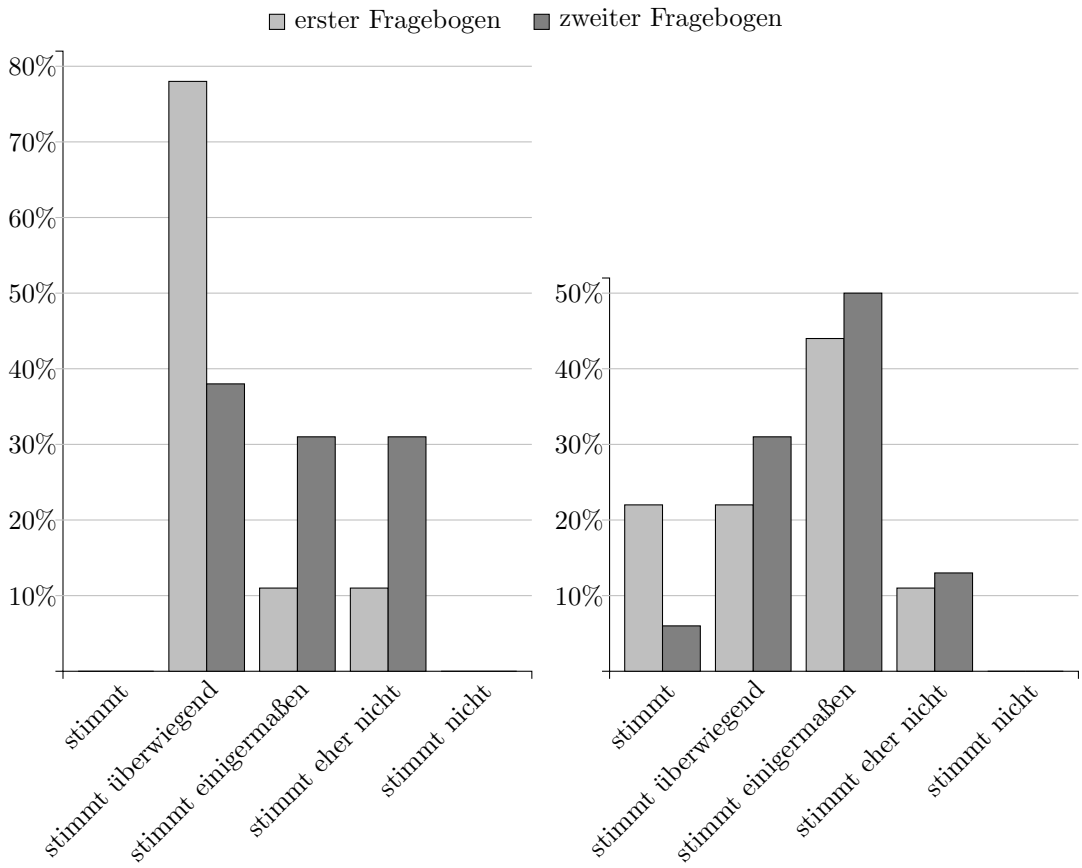
(b) *Frage 15c: „Die Kommentare stellen Markierungen im Code dar (z. B. ToDo's)“*

Abbildung 23 – *Antworten zu den Fragen 15b und c*

Die Unschlüssigkeit bezüglich der Frage, ob die Kommentare den Code zusammenfassen, erschließt sich mir im Vergleich zu den Antworten der ersten Umfrage und der wenigen neuen Änderungen an den Kommentaren nicht. Vielleicht wurden von den Teilnehmern der zweiten Umfrage Codeabschnitte verstärkt betrachtet (wie z. B. das Whiteboard, mit dem sich vier Studenten des Softwareprojekts beschäftigten), in denen besonders die Kommentare bei der Zusammenfassung des Codes abweichen.

Ob die Kommentare die Suche im Code erleichtern, wurde ebenfalls nicht klar beantwortet. Zur Frage „Die Kommentare sind gut so, wie sie sind.“ gab es tendenziell negativere Antworten als zuvor. Die Notwendigkeit, Kommentare zu ergänzen und zu überarbeiten, hatte ich bereits in meiner ersten Umfrage festgestellt und sie bestätigte sich erneut in der zweiten Umfrage. Dass dieses Ergebnis hier noch deutlicher ausfiel, kann darauf zurückgeführt werden, dass nun mehr projektunerfahrene Programmierer teilgenommen haben und verstärkt auf die Kommentare zum Verstehen des Quellcodes angewiesen sind.

Der Frage „Kommentierst du deinen Quellcode?“ wurde erneut mit „immer“ bzw. „meistens“ zugestimmt. Die Frage „Bewertest du während eines Reviews auch die



(a) *Frage 15d: „Die Kommentare fassen den Code zusammen.“*

(b) *Frage 15e: „Die Kommentare beschreiben den Zweck/was der Code bewirken soll.“*

Abbildung 24 – *Antworten zu den Fragen 15d und e*

Kommentare im Code der anderen?“ wurde allerdings weniger zugestimmt als im ersten Fragebogen, was einige Programmierer damit begründet hatten, dass sie bisher keine oder kaum Reviews durchgeführt hätten. Dies passt dazu, dass sich einige Entwickler des Saros-Teams noch in der Einarbeitungsphase von Saros befinden.

6.2 Schlussfolgerung

Insgesamt unterscheiden sich die Antworten der beiden Fragebögen zum Quelltext nur wenig voneinander. Dies kann daran liegen, dass die Richtlinien zur Erstellung der Kommentare und auch der Dokumentationsmanager erst im späteren Verlauf meiner Arbeit eingeführt wurden und die Zeit zwischen der Einführung und der Umfrage nicht lang genug war, um Veränderungen an den Kommentaren im Quelltext

aufzeigen zu können. Wie ich schon in Abschnitt 5.3, S. 56 angemerkt habe, war der Dokumentationsmanager zum Zeitpunkt der Umfrage erst ein einziges Mal zum Einsatz gekommen.

Der Student, der Ende Juli die Rolle des Dokumentationsmanagers übernommen hat, war von der Einführung des *Documentation Issue Tracker* bei Sourceforge sehr angetan. Er hatte allerdings zunächst ein paar Schwierigkeiten, welche Probleme er welcher Person zuweisen könnte. Mit Unterstützung der anderen Mitglieder aus dem Saros-Team, die ebenfalls festlegen konnten, in welchen Bereichen sich jemand besonders gut auskennt, hat dies aber funktioniert. Dass der Dokumentationsmanager seine Team-Kollegen bei der Zuordnung der Documentation Issues fragen soll, habe ich bei dessen Aufgaben vermerkt.

Bei den Paketbeschreibungen hatte es zunächst noch Unsicherheiten gegeben, wie diese auszusehen hätten. Nachdem ich meine Paketbeschreibungen angefertigt hatte (siehe Abschnitt 4.4, S. 47), bekam ich positive Rückmeldungen, dass sie als Vorlage sowie inhaltlich sehr nützlich seien. Einen Verweis darauf, dass man sich das Paket `concurrent` mit seinen Unterpaketen ansehen kann, um ein Beispiel für Paketbeschreibugen zu erhalten, habe ich ebenfalls bei den Aufgaben des Dokumenationsmanagers angemerkt.

Schon bevor ich die zweite Umfrage durchgeführt hatte, wurde mir von Studenten aus dem Saros-Team bestätigt, dass die Webseite viel übersichtlicher geworden sei, was die Auswertung des zweiten Fragebogens bestätigt: Der Text der Webseite wurde nach meinen Änderungen insgesamt als verständlicher eingestuft, die Webseite als gegliedert bezeichnet und meine Ergänzungen wurden positiv angenommen.

Ende September bzw. Anfang Oktober kam der Dokumentationsmanager während eines weiteren Release ein zweites Mal zum Einsatz. Auch wenn der Release zum Zeitpunkt der Abgabe meiner Arbeit noch nicht abgeschlossen ist, gebe ich an dieser Stelle kurz wieder, welche Erfahrung der Student gemacht hat, der in diesem Release die Rolle des Dokumentationsmanagers übernommen hat.
Die thematische Zuordnung von Documentation Issues an die jeweiligen Studenten, die für das Beheben des Issues geeignet sind, fiel ihm leicht und er war sehr erfreut über das Engagement der anderen Studenten, welche die ihnen zugewiesenen Issues schnell und gewissenhaft bearbeitet hätten. Um zu zeigen, wie die Aktualisierung der Dokumentation voran geht und um die Neuerungen bekannt zu machen, hat er vorgeschlagen, am Ende eines Dokumentations-Bearbeitungszyklus eine E-Mail zu schreiben, in der zusammengefasst wird, was in der Dokumentation überarbeitet wurde. Da dies jeweils eine gute Zusammenfassung der Änderungen und Erweiterungen der Dokumentation sein kann, habe ich den Versand einer E-Mail als abschließenden Punkt den Aufgaben des Dokumentationsmanagers noch hinzugefügt.

7 Fazit

Meine Diplomarbeit beschäftigte sich mit der Verbesserung und Pflege der Dokumentation von Saros. Dazu habe ich einige Veränderungen an der Dokumentation vorgenommen und zusätzlich Methoden eingeführt, wie sie weiterhin durch eine schrittweise Redokumentation aktuell gehalten werden kann:

- Die Webseite wurde umstrukturiert, was zu einer neu gegliederten und gruppierten Navigation geführt hat. Dabei wurden u.a. die Informationen von der bisherigen Seite *Developer Guide* thematisch gegliedert und auf verschiedene Unterseiten verteilt. Für einen Überblick wurde die neue Übersichtsseite *Developer Guide* erstellt, welche kurz erklärt, was auf den Unterseiten zu finden ist. Der fehlende Link zum *Review Board* wurde der Navigation hinzugefügt.
- Die Webseite wurde durch neue Informationen ergänzt. Dazu gehören eine Architekturübersicht auf der Seite *Technical Information* und detaillierte Modulbeschreibungen auf deren Unterseiten. Des Weiteren wurde die Webseite um eine Auflistung der Saros-*Features* ergänzt.
- In den *Code Rules* wurde der Abschnitt *Language & Documentation* erweitert, indem nun aufgeführt wird, welche Arten von Kommentaren es gibt, welche davon akzeptabel sind und welche Bereiche des Quellcodes (Pakete, Interfaces, etc.) kommentiert werden müssen.
- Im Quellcode wurden Paketbeschreibungen des Pakets `concurrent` sowie der Unterpakete hinzugefügt. Diese dienen als Vorlage für weitere Kommentare, insbesondere Paketbeschreibungen.
- Eine vorher nicht vorhandene Quelltext-Dokumentation wurde generiert und auf der Saros-Webseite verlinkt (*Saros Javadoc*). Diese Quelltext-Übersicht wird jede Nacht aus den aktuellen Quellcode-Versionen von Saros (inklusive Whiteboard) neu erzeugt.
- Ein *Documentation Issue Tracker* wurde erstellt, um Probleme mit der Dokumentation aufzuführen. Zu deren Behandlung gibt es neu eingeführte Richtlinien (*Documentation Issue Procedures*).
- Um die Dokumentation weiterhin zu aktualisieren wurde als neue Rolle der Dokumentationsmanager eingeführt, dessen Aufgaben in *Our Release Schedule* angegeben sind.

Die Webseite ist durch ihre thematische Gliederung und die Einteilung der Inhalte in Bereiche für Benutzer, für Entwickler sowie für allgemeine Themen viel übersichtlicher geworden. Die neu hinzugefügten Informationen haben dazu beigetragen, dass der Leser ein besseres Verständnis für Saros entwickeln kann.

Seit Einführung des Dokumentationsmanagers wurden von den 90 von mir angelegten Documenation Issues 29 abgearbeitet. Zusätzlich wurden auch eigenständig

Veränderungen bzw. Ergänzungen an der Dokumentation vorgenommen – zu nennen sind hierbei beispielsweise zwei Neuheiten auf der Saros-Webseite: *Coverage of Saros* beinhaltet Links zu Berichterstattungen oder Zeugnissen von Saros-Nutzern außerhalb der Universität und *Data connections in Saros* gibt eine Übersicht über die beiden möglichen Datenverbindungen (Socks5 oder In Band Bytestream) zwischen den Sitzungsteilnehmern und wie sich mögliche Probleme damit lösen lassen.
Auch während der Reviews gab es nach der Einführung der neuen Richtlinien Anmerkungen zu den Kommentaren, wenn sie nicht verständlich waren und ergänzt werden mussten.

Dies sind positive Effekte, die erkennen lassen, dass meine Veränderungen an der Dokumentation Wirkung zeigen und die Dokumentation des Projekts weiterhin aktualisiert wird.
Wenn die Richtlinien eingehalten werden und der Dokumentationsmanager regelmäßig zum Einsatz kommt, um auf Schwachstellen aufmerksam zu machen, wird es möglich sein, dass die Dokumentation Schritt für Schritt auf einen aktuellen Stand gebracht und gehalten wird.

Trotz der beschriebenen Maßnahmen gibt es besonders im Quellcode noch viele Stellen, die bearbeitet werden müssen. Und auch bei der Webseite wurden neue fehlende Themen genannt wie zum Beispiel Tutorials oder Videos, die ergänzt werden sollten, um den Einstieg ins Projekt weiter zu erleichtern. Durch ein Tutorial oder Video kann der Nutzer von Saros ‚an die Hand genommen werden' und ihm Schritt für Schritt gezeigt werden, wie er beispielsweise ein Projekt (oder nur einzelne Dateien) mit einem anderen Teilnehmer gemeinsam bearbeiten kann oder wie das Screen Sharing funktioniert bzw. verwendet wird.
Aber auch für die Entwickler können Tutorials oder Videos sehr hilfreich sein. Mit ihnen lässt sich beispielsweise zeigen, wie ein Peer Review durchgeführt werden kann oder wie bestimmte Codeabschnitte aussehen bzw. verwendet werden. Dazu hat Dr. Christopher Oezbeck mir in dem mit ihm geführten Interview von Screen-Casts berichtet, die er zusammen mit Stephan Salinger aufgezeichnet hat. Mit diesen Video-Code-Walkthroughs sollten die Studenten einen direkten Einblick in die besprochenen Codeabschnitte bekommen und direkt im Code nachvollziehen können. Ein großes Problem bei den aufgezeichneten Videos war jedoch die schlechte Tonqualität. Es wäre sicherlich eine gute Ergänzung der Dokumentation, solche Aufzeichnungen neu zu erstellen und im Anschluss auf der Webseite zu verlinken.

Insgesamt ist es sicher sinnvoll, nach einer gewissen Zeit – beispielsweise nach sechs bis zwölf Monaten – ein weiteres Mal eine Untersuchung zum Zustand der Dokumentation durchzuführen, um sehen zu können, ob die Dokumentation vollständiger bzw. aktueller ist und ob sie dem Leser weiterhilft beim Verstehen bzw. Verwenden der Software.

Anhang

A Fragebögen

Um ein Meinungsbild über die Dokumentation zu Beginn (im Mai 2011) und am Ende (im September 2011) meiner Diplomarbeit zu erhalten, habe ich zwei Fragebögen entwickelt, die im Folgenden aufgeführt werden.

A.1 Ankündigung zum Fragebogen (Mai 2011)

Hallo Sarosianer,

ich heiße Meike Johannsen und bin Diplom-Studentin in der AG SE. Ich arbeite an einer Diplomarbeit, welche die Verbesserung und Pflege der Dokumentation von Saros zum Ziel hat.
Die Dokumentation von Saros bezieht sich auf die Webseite (http://www.saros-project.org) und auf die Kommentierung im Quellcode des Projekts.

Dazu habe ich einen Fragebogen entwickelt, bei dem ich dich bitten möchte, ihn zu beantworten. Dieser Fragebogen richtet sich an die Entwickler von Saros und alle anderen, die sich mit Saros beschäftigen.

Durch die Beantwortung der Fragen würdest du mir helfen, herauszufinden, wie du den Zustand der Dokumentation siehst, in wie weit sie zur Zeit gepflegt wird und wie man dies verbessern könnte.

Ich bitte dich, deine Antworten zu diesem Fragebogen bis Freitag, den 20. Mai 2011 an mich zurückzuschicken. Wenn du die Fragen mit mir gemeinsam durchgehen möchtest, schick mir eine E-Mail, damit wir einen Termin (innerhalb der kommenden zwei Wochen) verabreden können.

Wenn du weitere Anmerkungen hast zu dieser Ankündigung, zu dem Fragebogen oder zur Dokumentation von Saros, kannst du mich gerne kontaktieren.

Vielen Dank im Voraus,
Meike Johannsen

A.2 Fragebogen zum Thema Dokumentation von Saros (Mai 2011)

Durch die Teilnahme an dieser Umfrage hilfst du mit, die Dokumentation zu beurteilen und Wege zu finden, sie zu verbessern. Dies soll (kommenden) Entwicklern den Einstieg ins bzw. den Umgang mit dem Projekt erleichtern.

Bei mehreren Antwortmöglichkeiten kreuze bitte die zutreffende an.

1. Woher beziehst du deine Informationen über Saros?

   A) als du dich zum ersten Mal mit Saros beschäftigt hast
   B) jetzt

   (bitte gib für jeden Punkt den prozentualen Anteil an)

	A)	B)
aus der Webseite	__ %	__ %
aus dem Quellcode	__ %	__ %
von anderen Entwicklern	__ %	__ %
sonstiges: ____	__ %	__ %

Die folgenden Fragen beziehen sich auf die Webseite von Saros (http://www.saros-project.org).

2. Wie gut bist du über die Webseiten-Dokumentation von Saros informiert?

   ☐ kenne ich nicht
   ☐ ich habe davon gehört
   ☐ ich habe mir die Webseite angesehen
   ☐ ich kenne sie gut
   ☐ ich habe an Teilen von ihr mitgearbeitet

3. Welche Themen sind für dich besonders wichtig?

   _____

   _____

4. Welche Themen brauchst du persönlich gar nicht?

   _____

   _____

5. Welche Themen sind für dich besonders hilfreich?

   _____

   _____

6. Welche Themen haben dir nicht weiter geholfen?

   _____

   _____

7. Welche Themen der Dokumentation hast du genutzt?

   _____

   _____

8. Welche Themen fehlen dir?

   _____

   _____

9. Hast du mit Hilfe der Webseite einen leichten Einstieg in Saros bzw. in dir vorher unbekannte Aspekte bekommen?

   _____

Bitte nimm jeweils eine Einteilung vor für:

A) die Webseite ganz allgemein
B) die Themen, die du als besonders hilfreich empfindest
C) die Themen, die du als gar nicht hilfreich empfindest

Hierbei bedeutet:

+2 Die Eigenschaften auf der linken Seite sind deutlich vorhanden.
+1 Die Eigenschaften auf der linken Seite überwiegen.
0 Die Eigenschaften auf der rechten und linken Seite sind etwa gleich stark.
-1 Die Eigenschaften auf der rechten Seite überwiegen.
-2 Die Eigenschaften auf der rechten Seite sind deutlich ausgeprägt.

10. Der Text ist gegliedert/hat eine Ordnung. ↔ Der Text ist unübersichtlich/ zusammenhangslos.

	+2	+1	0	-1	-2
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐
C)	☐	☐	☐	☐	☐

11. Du findest die Information dort, wo du sie erwartest. ↔ Die Information ist an einem Ort, wo sie nicht hingehört.

	+2	+1	0	-1	-2
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐
C)	☐	☐	☐	☐	☐

12. Informationen findest du schnell wieder. ↔ Du musst immer wieder neu nach der Information suchen.

	+2	+1	0	-1	-2
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐
C)	☐	☐	☐	☐	☐

13. Der Text ist einfach geschrieben. ↔ Der Text ist kompliziert geschrieben.

	+2	+1	0	-1	-2
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐
C)	☐	☐	☐	☐	☐

14. Der Text ist kurz und prägnant. ↔ Der Text ist weitschweifig gefasst.

	+2	+1	0	-1	-2
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐
C)	☐	☐	☐	☐	☐

15. Der Text ist aussagekräftig. ↔ Der Text enthält keine relevanten Informationen.

	+2	+1	0	-1	-2
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐
C)	☐	☐	☐	☐	☐

16. Die Angaben sind korrekt (sie stimmen mit der Realität überein). ↔ Die Angaben sind fehlerhaft.

	+2	+1	0	-1	-2
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐
C)	☐	☐	☐	☐	☐

17. Die Angaben sind vollständig. ↔ Die Angaben sind unvollständig (es fehlen wichtige Informationen).

	+2	+1	0	-1	-2
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐
C)	☐	☐	☐	☐	☐

Die folgenden Fragen beziehen sich auf den Quellcode von Saros.

18. An welchen Paketen/welchen Klassen arbeitest du?

    _____

    _____

19. Die Kommentare in welchen Teilen des Quellcodes sind für dich interessant?

    ☐ nur der Bereich, an dem ich arbeite
    ☐ weitere Teile, die mit ‚meinen' Klassen/Paketen zusammengehören: ____
    ☐ auch andere Teile: ____

Bitte bewerte im Folgenden die Kommentare des Quellcodes,

A) an dem du arbeitest
B) mit dem du ansonsten zu tun hast

20. Die Kommentare ...

a) ... wiederholen den Code.

	stimmt	stimmt überwiegend	stimmt einigermaßen	stimmt eher nicht	stimmt nicht
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐

b) ... erklären den Code.

	stimmt	stimmt überwiegend	stimmt einigermaßen	stimmt eher nicht	stimmt nicht
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐

c) ... stellen Markierungen im Code dar (z. B. ToDo's).

	stimmt	stimmt überwiegend	stimmt einigermaßen	stimmt eher nicht	stimmt nicht
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐

d) ... fassen den Code zusammen.

	stimmt	stimmt überwiegend	stimmt einigermaßen	stimmt eher nicht	stimmt nicht
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐

e) ... beschreiben den Zweck/was der Code bewirken soll.

	stimmt	stimmt überwiegend	stimmt einigermaßen	stimmt eher nicht	stimmt nicht
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐

f) ... erleichtern die Suche im Code.

	stimmt	stimmt überwiegend	stimmt einigermaßen	stimmt eher nicht	stimmt nicht
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐

g) ... beschreiben die einzuhaltenden Vorbedingungen.

	stimmt	stimmt überwiegend	stimmt einigermaßen	stimmt eher nicht	stimmt nicht
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐

h) ... beschreiben die Nachbedingungen.

	stimmt	stimmt überwiegend	stimmt einigermaßen	stimmt eher nicht	stimmt nicht
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐

i) ... beschreiben die Invarianten des Codes.

	stimmt	stimmt überwiegend	stimmt einigermaßen	stimmt eher nicht	stimmt nicht
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐

j) ... sind gut so, wie sie sind.

	stimmt	stimmt überwiegend	stimmt einigermaßen	stimmt eher nicht	stimmt nicht
A)	☐	☐	☐	☐	☐
B)	☐	☐	☐	☐	☐

21. An welchen Stellen hältst du eine Kommentierung für nötig?

_____

_____

22. An welchen Stellen findest du eine Kommentierung unnötig?

_____

_____

23. An welchen Stellen im Code fehlt dir eine Kommentierung?

_____

_____

Die folgenden Fragen beziehen sich auf das Dokumentationsverhalten.

24. Kommentierst du deinen Quellcode?

immer	meistens	manchmal	selten	nie
☐	☐	☐	☐	☐

25. Ergänzt bzw. aktualisierst du die Informationen auf der Webseite, die mit deinem Teil der Arbeit an Saros zu tun haben?

immer	meistens	manchmal	selten	nie
☐	☐	☐	☐	☐

26. Bewertest du während eines Reviews auch die Kommentare im Code der anderen?

immer	meistens	manchmal	selten	nie
☐	☐	☐	☐	☐

27. Würdest du Mängel an der Dokumentation korrigieren, wenn dich jemand darauf aufmerksam machte (etwa ein Dokumentationsmanager analog zu Release- und Testmanager)?

    _____

28. Was würde es dir erleichtern, Änderungen an der Webseite vorzunehmen?

    ☐ wenn sie extern zum Quelltext vorliegt (so wie jetzt als TWiki-Seite)
    ☐ wenn sie an den Quellcode gebunden ist (mit dem Saros-Projekt aus-/eingescheckt wird)

29. Welche Vorschläge hast du, wie die Dokumentation (Kommentare im Quellcode und Webseite) aktuell gehalten werden kann?

    _____

    _____

30. Hast du Wissen, welches andere Entwickler weiterhelfen würde, was jedoch nicht niedergeschrieben ist und somit verloren geht, wenn du das Saros-Team verlässt?

    _____

    _____

A.3 Ankündigung zum Fragebogen (September 2011)

Hallo Sarosianer,

einige von euch hatten meinen ersten Fragebogen beantwortet. Auf diesem Wege nochmal danke dafür.

Nun bitte ich euch alle (sowohl Alt- als auch Neu-Sarosianer) darum, meinen zweiten Fragebogen zu beantworten. Ihr würdet mir dabei helfen, einen Vorher-Nachher-Vergleich zu ziehen, um die Verbesserungen an der Dokumentation auswerten zu können.

Bitte nehmt euch ein paar Minuten Zeit, diesen Fragebogen auszufüllen und schickt ihn mir bis Freitag, den 16. September zurück.

Vielen Dank im Voraus,
Meike Johannsen

A.4 Umfrage zur Dokumentation im Saros-Projekt (September 2011)

Durch die Teilnahme an dieser Umfrage hilfst du mit, die Veränderungen an der Dokumentation von Saros zu beurteilen. Deine Antworten werden dabei anonym behandelt.

Vielen Dank, dass du an der Umfrage teilnimmst!

Die folgenden Fragen beziehen sich auf die Webseite von Saros (http://www.saros-project.org).

1. Welche Themen sind für dich besonders hilfreich?

   _____

   _____

2. Welche Themen haben dir nicht weiter geholfen?

   _____

   _____

3. Welche Themen hast du genutzt?

   _____

   _____

4. Welche Themen fehlen dir?

   _____

   _____

5. Hast du mit Hilfe der Webseite einen leichten Einstieg in Saros bzw. in dir vorher unbekannte Aspekte bekommen?

   _____

Bitte beurteile im folgenden die Saros-Webseiten nach folgendem Schema (zutreffendes ankreuzen):

+2 Die Eigenschaften auf der linken Seite sind deutlich vorhanden.
+1 Die Eigenschaften auf der linken Seite überwiegen.
0 Die Eigenschaften auf der rechten und linken Seite sind etwa gleich stark.
-1 Die Eigenschaften auf der rechten Seite überwiegen.
-2 Die Eigenschaften auf der rechten Seite sind deutlich ausgeprägt.

	+2	+1	0	-1	-2
6. Der Text ist gegliedert/hat eine Ordnung. ↔ Der Text ist unübersichtlich/zusammenhangslos.	☐	☐	☐	☐	☐
7. Du findest die Information dort, wo du sie erwartest. ↔ Die Information ist an einem Ort, wo sie nicht hingehört.	☐	☐	☐	☐	☐
8. Informationen findest du schnell wieder. ↔ Du musst immer wieder neu nach der Information suchen.	☐	☐	☐	☐	☐
9. Der Text ist einfach geschrieben. ↔ Der Text ist kompliziert geschrieben.	☐	☐	☐	☐	☐
10. Der Text ist kurz und prägnant. ↔ Der Text ist weitschweifig gefasst.	☐	☐	☐	☐	☐
11. Der Text ist aussagekräftig. ↔ Der Text enthält keine relevanten Informationen.	☐	☐	☐	☐	☐
12. Die Angaben sind korrekt (sie stimmen mit der Realität überein). ↔ Die Angaben sind fehlerhaft.	☐	☐	☐	☐	☐
13. Die Angaben sind vollständig. ↔ Die Angaben sind unvollständig (es fehlen wichtige Informationen).	☐	☐	☐	☐	☐

14. Ergänzt bzw. aktualisierst du die Informationen auf der Webseite, die mit deinem Teil der Arbeit an Saros zu tun haben?

☐ immer
☐ meistens
☐ manchmal
☐ selten
☐ nie

Die folgenden Fragen beziehen sich auf den Quellcode von Saros.

Bitte bewerte im Folgenden die Kommentare des Quellcodes (zutreffendes ankreuzen).

15.	Die Kommentare ...	stimmt	stimmt überwiegend	stimmt einigermaßen	stimmt eher nicht	stimmt nicht
a)	... wiederholen den Code.	☐	☐	☐	☐	☐
b)	... erklären den Code.	☐	☐	☐	☐	☐
c)	... stellen Markierungen im Code dar (z. B. ToDo's).	☐	☐	☐	☐	☐
d)	... fassen den Code zusammen.	☐	☐	☐	☐	☐
e)	... beschreiben den Zweck/was der Code bewirken soll.	☐	☐	☐	☐	☐
f)	... erleichtern die Suche im Code.	☐	☐	☐	☐	☐
g)	... sind gut so, wie sie sind.	☐	☐	☐	☐	☐

16. An welchen Stellen im Code fehlt dir eine Kommentierung?

_____

_____

		immer	meistens	manchmal	selten	nie
17.	Kommentierst du deinen Quellcode?	☐	☐	☐	☐	☐
18.	Bewertest du während eines Reviews auch die Kommentare im Code der anderen?	☐	☐	☐	☐	☐

B Modulbeschreibungen

Im Folgenden sind die Texte aufgeführt, mit denen ich die Technischen Information auf der Saros-Webseite erweitert habe, beginnend mit der Software-Architekturübersicht, welche eine Kurzbeschreibung des logischen Aufbaus (Module) von Saros geben soll, und gefolgt von den Beschreibungen der einzelnen Module.

Für die Beschreibungen habe ich neben den Kommentaren im Quelltext und persönlichen Gesprächen mit den Entwicklern des Saros-Teams verschiedene Abschlussarbeiten, die in der AG SE entstanden sind, herangezogen.

Das *VoIP* bzw. der *Chat* aus *Shared Session* (Anhang B.2, S. 83) orientiert sich an den Beschreibungen in den Bachelorarbeiten von Olaf Loga [Log10, S. 10–33] und Florian Pütz [Püt10, S. 12–22]. Für das *Screen Sharing* aus *Shared Session* konnte ich auf die Bachelorarbeit von Stephan Lau [Lau10, S. 20–30] zurückgreifen.

Für die Beschreibung des *User Interface* (Anhang B.4, S. 86) habe ich zunächst die Masterarbeit von Björn Kahlert [Kah11, S. 65–81] herangezogen und anschließend mit ihm persönlich besprechen können. Dabei habe ich noch zusätzliche Vorschläge bekommen, wie man die Beschreibung optimieren kann.

Zur *Concurrency Control* (Anhang B.5, S. 88) habe ich in den Diplomarbeiten von Oliver Rieger [Rie08, S. 63–85] und Sebastian Ziller [Zil09, S. 13–35] hilfreiche Informationen zu *Jupiter* finden können.

Zu *Invitation* (Anhang B.6, S. 90) hat mir Karl Held – ein Student aus dem Saros-Team – ein Ablaufdiagramm erstellt, welches ich in meine SVG-Graphiken überführt habe.

Der *Network Layer* (Anhang B.7, S. 92) wurde im Zeitraum meiner Diplomarbeit von Björn Gustavs – ebenfalls ein Student aus dem Saros-Team – überarbeitet. Somit konnte er mir zu einigen Komponenten Informationen liefern. Für die Beschreibung des *Stream Service* im *Network Layer* habe ich wiederum die Bachelorarbeit von Stephan Lau [Lau10, S. 31–54] herangezogen.

Das *Feedback*-Modul (Anhang B.8, S. 94) wurde von Lisa Dohrmann [Doh09, S. 14–30] in ihrer Bachelorarbeit entwickelt.

B.1 Software Architecture

Saros uses a layer architecture and consists of eight major modules as you can see in the image below. The Whiteboard is an additional project which expands the functionality of the Saros plug-in.

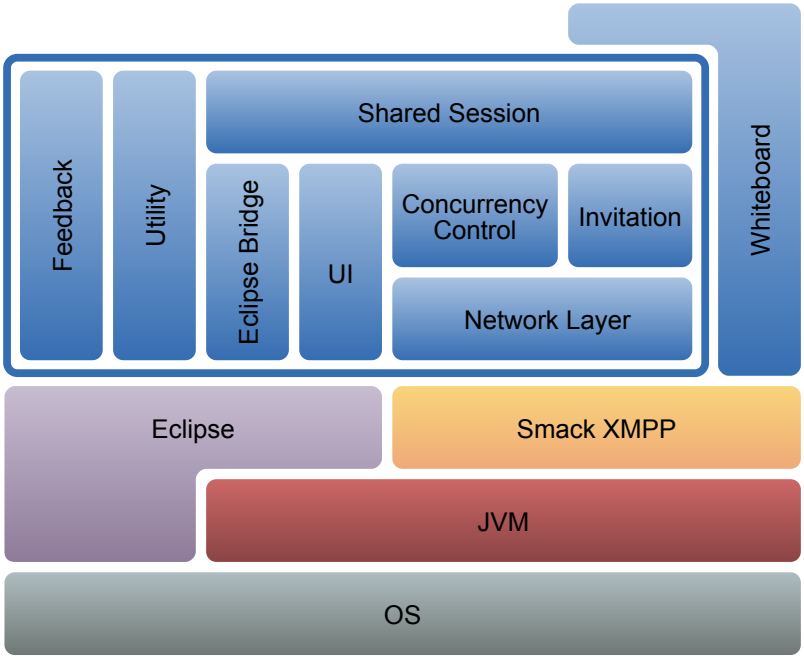


Abbildung 25 – *Saros high-level architecture*

- *Shared Session* represents all necessary information needed for a Saros session (e.g. the JID and other user information). It also includes the communication using chat or VoIP and the screen sharing within a session.
- With the *Eclipse Bridge* Saros connects to existing Eclipse components and expands them (e.g. capturing local text editing events and replaying remote ones).
- The *User Interface* (UI) provides Saros' own components for the communication between Saros and the users such as viewers, menus and its commands.
- The *Concurrency Control* layer is in charge of synchronizing the activities of users in a Saros session, detecting inconsistencies between the users' workspaces, and repairing these synchronization issues.
- The *Invitation* is used to establish a Saros session between users and to share projects and files. There is one host and at least one participant in a Saros session.
- The *Network Layer* is responsible for connecting users of Saros to each other and for the data transfer between them. It is based on the Extensible Messaging and Presence Protocol (XMPP).

- The *Feedback* module gathers information about the usage of each Saros session if the user has enabled this option during the installation. It also directs the user to a survey frequently.
- *Utility* comprises commonly used data types and functions which have no particular place in the rest of the code.
- The *Whiteboard* contributes a concurrent drawing tool for the Saros user. Therefore it provides a graphical editor using Shared XML Editing (SXE).

We use PicoContainer to manage our component instances using the dependency injection design pattern. The resulting dependency graph which represents the structure of all classes which are singletons in Saros can be seen in this diagram.

B.2 Shared Session

A session is a virtual meeting that takes place between two or more people using Saros. It is started and ended at will by the host; others can join upon invitation and leave at will. Within a session, users can do a number of things:

- Share a project (this is currently the required way to initiate a session, but in theory Saros can easily be adapted in future to begin "empty" sessions)
- Chat using an instant messenger
- Speak to each other using VoIP
- Share a video feed of their desktop

The functionality for the session is distributed throughout the packages: `project`, `communication` (`muc` for multi-user chat and `audio` for VoIP), and `videosharing`.

Session

The core functionality for maintaining both the session and the shared project(s) are kept in the `project` package. (TODO: this ought to be refactored to separate the former synonymy in projects and shared sessions.)

A session takes care about the user's state within a session (e.g. write/read-only access or adds/removes a user). The `SarosSessionManager` handles the initiation of a Saros session (including starting, stopping and joining) and the invitation to this session.

The shared project stores the state of a project shared in a Saros session (e.g. project open) and its resources (e.g. files, folders, projects, and the version control system if the VCS support is enabled for the session). For identifying shared projects across

the network, project IDs are used in the `SarosProjectMapper` class. This is necessary especially when the local names of a shared project are different at the user's workspaces.

The `SharedResourcesManager` is responsible for handling all resource changes that aren't handled by the `EditorManager` of the *Eclipse Bridge*. It creates and executes files, folders, and VCS activities.

You will find information about activities here.

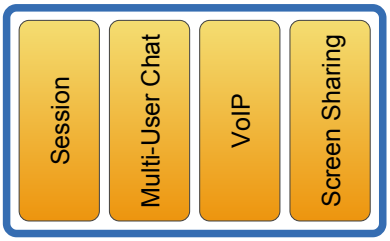


Abbildung 26 – Shared Session *view*

Multi-User Chat

The functionality of the *Multi-User Chat* is located in the `communication.muc` package.

In a multi-user text chat the users can exchange text messages in a chatroom. Therefore the XMPP protocol extension Multi-User Chat (MUC) is used. When a Saros session is started a *Multi-User Chat* is initiated as well. As long as the chat is active the history of all events that occur to this chat session (e.g. incoming messages) are stored and transmitted to all users joining this chat.

The `MUCManager` manages the creation and destruction of a Multi-User Chat session. The host creates/destroys a `MUCSession` while other users join/leave that `MUCSession` based on the preferences (`roomName`, `password`, and `service`, the host name at which the multi-user chat is running).

VoIP

The *VoIP* functionality is located in the `communication.audio` package.

The `AudioServiceManager` manages a VoIP session (inviting a user, starting/stopping a session). Within a Saros session users with a properly configured playback device can start a VoIP session for voice communication. This session will be started even if one user does not have a record device since there might be one user who does.

The used audio codex is Speex where the bitrate depends on the selected audio quality level (0 – 10 with 10 as best quality). Speex supports constant and variable bitrates.

Note: The VoIP module is under construction right now.

The `communication` package also contains the SkypeManager which allows the user to initiate a Skype VoIP session to a given XMPP entity if it supports Skype.

Screen Sharing

The Video-sharing Framework (VSF) for *Screen Sharing* in Saros is located in the `videosharing` package.

There are two options for sharing a screen:

- full screen shares the current screen of the host and
- follow mouse shares the surrounding area (specified by width and height) of the host's mouse pointer.

`VideoSharing` is the main class for video sharing sessions. It takes care of setting up a session with a given user on the host's side if possible and receiving the host's screen on the user's side. If no Video-sharing session could be established the user will be informed.

Note: Currently only one session at a time between two users is possible.

During a screen sharing session the client can perform different activities on the shared screen of the host like mouse click, mouse move, pause session, or stop session. The `ActivityManager` receives and responds to these activities.

The recommended codec for coding and encoding video data is Xuggler (it needs to be installed). The basic parameters of a coding process (e.g. frames per second, resolution, and maximum bitrate) are set at the Encoder class.

B.3 Eclipse Bridge

The *Eclipse Bridge* contains components that enable integration of Saros into Eclipse. Contrary to the *UI* which implements Saros' own user interface the *Eclipse Bridge* uses Eclipse' existing user interface and tells the API what to do.

The packages `editor` and `preferences` belong to the *Eclipse Bridge*.

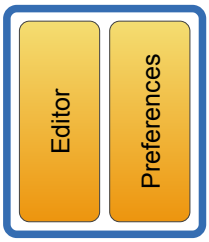


Abbildung 27 – Eclipse Bridge *view*

Editor

The `EditorManger` and the `EditorAPI` are the most important classes in the `editor` package.

The `EditorManager` expands the capability of the existing Eclipse editors during a Saros session. It listens for user input into an editor and fires an activity about what the user has changed concerning the editor (such as an opened editor, the position of the viewport, or changed text).

The `EditorAPI` encapsulates the interaction with the Eclipse editors (e.g. opens an editor, sets or gets a selection).

The awareness techniques implemented by annotations are also controlled in this module. They include the user's viewports, text selections and contributions to the text and make them aware to all users of that session (marked with the user's color).

Preferences

In the `preference` package Saros' specific preferences (`auto_accept_invitation`, `auto_follow_mode`, `local_socks5_proxy_disabled`) are defined and default values are set for initialization. These preferences can be manipulated. A restore point is set to allow restoring the preferences changed for a project after the end of a session.

B.4 UI – User Interface

Important parts of the *UI* are commands and *menus*, *widgets*, *viewers* and *models*, *wizards*, and *dialogs*.

Commands and Menus

A command is an abstraction of an action since it allows for multiple implementations depending on the content (e.g. copy is a command as multiple implementations exist depending on the currently opened editor). Another advantage is that commands can be embedded at the same time in different menus, context menus and toolbars.

`Commands`, `commandHandlers` and `menus` are declared in the `plugin.xml` file as the extension of the extension points `org.eclipse.ui.commands`, `org.eclipse.ui.handlers`, and `org.eclipse.ui.menus`.

The `commandHandlers` contain the implementations of commands that can be triggered by the user.

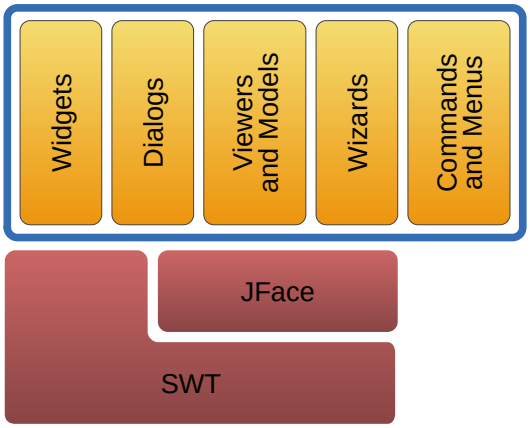


Abbildung 28 – User Interface *view*

Widgets

The `widgets` package contains all custom *UI* elements that are not part of the Standard Widget Toolkit (SWT). This covers a wide range from rounded composites to fully functional chat controls.

It is highly recommended for you to download the Saros Widget Gallery (`https://dpp.svn.sourceforge.net/dpp/trunk/dpp/de.fu_berlin.inf.dpp.ui.widgetGallery`) in order to get an overview of the various applications of Saros' custom widgets.

You will find more information on Usability Engineering.

Viewers and Models

JFace Viewers represent our domain model, namely the

- roster,
- an integrated view of the roster and the session, and
- the workspace.

The actual composites that contain these viewers are located in the `widgets.viewer` package.

The `model` package contains the wrapped domain objects in order to be displayed by the viewers. They encapsulate the knowledge on what they should look like (especially the caption and the icon). The most important domain object representatives are the `RosterEntryElement` and the `UserElement`.

Wizards

A wizard is a collection of several wizard pages and the logic of what to do if the wizard is finished or canceled. The `wizard` package itself contains all wizards whereas the sub package pages contains all wizard pages. Examples are wizards for creating a new XMPP Account, for adding buddies to the roster, for the invitation, for sharing projects, and many more.

Note: `WizardUtils` contains convenient methods for a simple and consistent start of wizards.

Dialogs

The `dialog` package contains some dialog classes that will pop up to inform the user like warning messages or the feedback dialog.

B.5 Concurrency Control

To support a writing collaboration of multiple users, Saros uses the Jupiter algorithm for concurrency control. The *Concurrency Control* keeps all shared data of the participants of a Saros session consistent.

Jupiter

The functionality of the `Jupiter` package is based on the Jupiter algorithm taken from the Open Source project ACE (A Collaborative Editor).

`Jupiter` is the most important class of this package. It implements the client-side core of the Jupiter control algorithm. It generates `JupiterActivities` for local operations and transforms received `JupiterActivities` into operations that need to be executed locally. Operations are changes made on a document. An operation contains what has changed and the position where it was changed. These changes made by multiple participants of a Saros session need to be executed in the correct order and position by all participants so that they all have the same consistent copy of this changed workspace.

For example: Both participants of a Saros session C and S start with the text `"abc"`. C performs an insert operation `insert(0,"def")`and S performs an delete operation `delete(1,"b")`. The resulting text is `"defac"`. Since the local operation is performed immediately at each participants side, the received operation of the other participant is transformed to ensure the consistency. Therefore for participant C the delete operation is transformed into `delete(4,"b")`. For participant S the transformation doesn't change the insert operation.

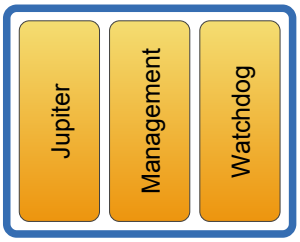


Abbildung 29 – Concurrency Control *view*

Management

The subpackage `management` is responsible for managing the state of the shared workspace in a Saros session. Every user with write-access sends their `JupiterActivities` to the `ConcurrentDocumentServer` located on the host side where they are transformed and sent to everybody else. The transformed `JupiterActivities` are received and transformed to `TextEditActivities` in the `ConcurrentDocumentClient` for execution on the client side.

The host of the session is always responsible for managing the state of all projects (or files), even when a client adds projects or files to the Saros session.

Watchdog

The consistency *Watchdog* is responsible for monitoring the states of all participant's workspaces and to recover from inconsistent states between the participants of a Saros session.

For all opened files anywhere in the shared session (host and clients) checksums are generated periodically (every 10 seconds) on the host side. These checksums are sent to all clients and the clients check their locally generated checksums against them. If an inconsistency is detected the inconsistency state is set and the user of the session gets informed (the yellow triangle on the Saros View). When the user wants to recover from that inconsistency a `ChecksumError` is sent to the host and waited for the reply. The user receives the file changes from the host. The host can never be in an inconsistent state.

B.6 Invitation

The `invitation` package consists of two areas: the *Session Invitation* and the *Project Invitation*.

During the *Session Invitation* a Saros session will be established between the host and the invitee.

Within the *Project Invitation* all shared projects/files will be synchronized that all participants (including host) have the same project data.

The invitation process is shown in the image below.

Session Invitation

The list of `ProjectExchangeInfos` will be generated based on the resources selected by the host. They contain information regarding the project (`projectID`, `projectName`, and description of the project) and a `FileList` which holds information about the files and folders belonging to the project (path and checksum).

When the host sends a session invitation a wizard will open on the invitee's side. This wizard invites the invitee to this Saros session and shows the projects which need to be synchronized.

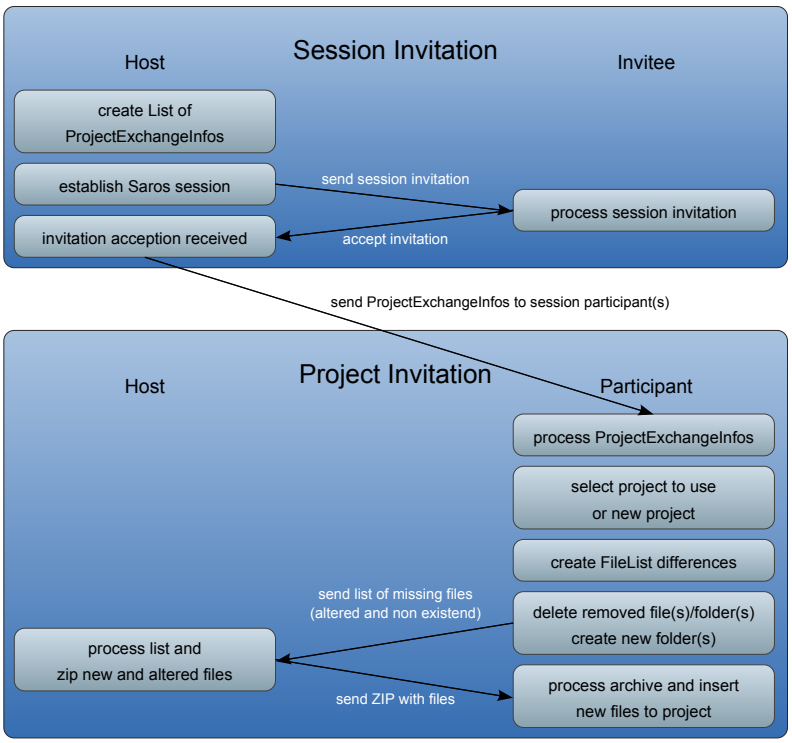


Abbildung 30 – Invitation *process view*

Project Invitation

After the session is established another wizard will open at the participant's side. For each project of the session the participant can choose to use an existing project out of her/his workspace or to create a new project.

Based on the `ProjectExchangeInfo` the differences to the host project will be detected. Files or folders that don't exist in the host project will be deleted on the participant's side. Missing or different files on the participant's side will be added/updated.

For all data transfer between the host and the participant the `XMPPTransmitter` of the *Network Layer* will be used.

Once the session is created every participant of this session is able to add new projects or files to this session.

B.7 Network Layer

The important concepts of the *Network Layer* are shown in the image below.

Discovery

The *Discovery* component consists of the `DiscoveryManager` and the `DiscoveryManagerListener`. It will request and keep track of the peers supported features (e.g. is Saros supported) and stores the presence information for each user.

Stream Service

The three classes of the *Stream Service* are `StreamSession`, `StreamServiceManager` and `StreamService`. The *Stream Service* component concerns the stream-based communication between the users within a session. It supports VoIP, screen sharing and stream invitation.

When starting an instance of a `StreamSession` you may register one listener to this session which will be called if this session has to be stopped (by user or if an error occurred). `StreamSession` also contains `StreamSessionOutputStream` and `StreamSessionInputStream`. These virtual streams behave like *java.io.InputStream* and *java.io.OutputStream*. For using streaming you have to implement the abstract class `StreamService`.

Note: The streaming is currently not available because it is faulty.

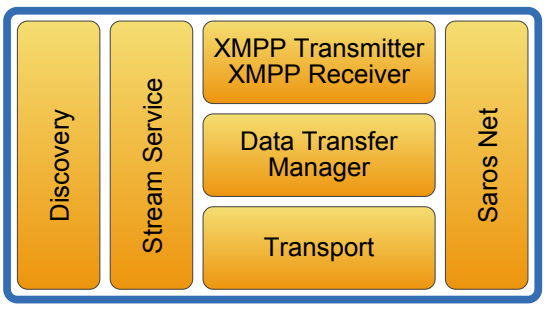


Abbildung 31 – Network Layer *view*

Transport

The classes `Socks5Transport`, `IBBTransport` and the `BinaryChannel` belong to the Transport concept. `Socks5Transport` and `IBBTransport` both extend the abstract class `BytestreamTransport` and use `ByteStreamManager` from Smack to establish a bytestream connection to exchange data. There are two types of bytestream connections that can be established: Socks5 and IBB.

Socks5 can be either direct (peer-to-peer) or mediated (through a proxy server). TCP is the typically used transport protocol. The host sends a connection request to the participant and the participant tries to establish that connection. To back up the connection establishment in Saros, the participant also sends an connection request and the host tries to establish that connection. If the first connection is established and it is bidirectional then this one will be kept and the second will be discarded. If both connections are established but they are uni-directional then both connection will be kept and wrapped into one bidirectional one.

If no Socks5 connection can be established a slower In Band Bytestream (IBB) connection will be established and used. IBB is an XMPP protocol extension that establishes a one-by-one bytestream between any two peers. The data is divided into smaller chunks and transported over XMPP.

`BinaryChannel` is a class that encapsulates a bidirectional communication between two participants. The `BinaryChannel` is working on a `Bytestreamsession` which is either a `Socks5Transport` or a `IBBTransport`.

Data Transfer Manager

The `DataTransferManager` is the major class for sending arbitrary data. It is responsible for handling all transfers of binary data via `ByteStreamTransport`. If possible it reuses an established bytestream connection.

XMPP Transmitter / XMPP Receiver

The `XMPPTransmitter` and `XMPPReceiver` are important classes when sending, receiving and processing data through XMPP.

The `XMPPReceiver` is a Facade for receiving XMPP packages. It receives and passes on packets and `IncomingTransferDescription` to registered listeners with accepting filters.

The `XMPPTransmitter` is the central class for sending object (e.g. activities, filelists) to peers by XMPP and receiving objects by registering a listener on the `XMPPReceiver`. A `TransferDescription` contains all necessary information for dealing with `FileTransfers` to a buddy via Jingle, IBB or Chat.

SarosNet

SarosNet encapsulates XMPP connection related objects and functions.

B.8 Feedback

Feedback comprises two ways of gathering user feedback data: directly via a *User Survey* and indirectly via *Statistics* by collecting user's session data.

User Survey

At the end of a Saros session the user is sometimes asked to participate in a *User Survey*. This happens the first time when a user ends his first session and after that in an interval what can be configured by the user. He also has the option to turn this request on and off.

The main class of the *User Survey* is `FeedbackManager`. It determines the user's preferences (participation option turned on/off and the request interval) and shows a `FeedbackDialog`. Apart from the user's preferences, the `FeedbackDialog` is displayed only if a Saros session lasted at least 5 minutes. If the user accepts the feedback request the survey opens in an external browser. If this fails, Eclipse's internal browser is used. If both methods fail a message dialog containing the survey's URL is shown.

The survey is kept at: http://saros-build.imp.fu-berlin.de/phpESP/public/survey.php?name=SarosFastUserFeedback_1.

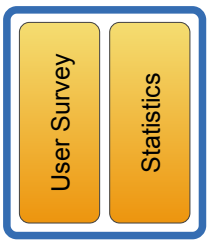


Abbildung 32 – Feedback *view*

Statistics

The *Statistic* framework gathers statistic data (e.g. session duration, number of participants, and the user's role as well as the used operation system and the installed versions of Java, Eclipse, and Saros) and send it to our Apache Tomcat server. Like

the user survey, the user has the option to turn the statistic data collection on and off.

For gathering that data collector classes which extend the abstract class `AbstractStatisticCollector` are used. Examples of these collector classes:

- the ParticipantCollector gathers information about the number of participants in a Saros session and the time they were present in the session (accumulated),
- the PermissionChangeCollector collects permission changes of the local user, e.g. how often they are changed and how long the user kept what permission (write/read-only access),
- the SessionDataCollector gathers general session data like session time, session ID, and session count as well as platform information.

The `StatisticManager` receives the gathered data of all collectors at the end of a session, combines them to one file and sends it to our server (Saros-build).

C Kommentare: Regeln und Paketbeschreibungen

Hier sind die Regeln zur Kommentierung im Quellcode aufgeführt, wie ich sie im Abschnitt *Language & Documentation* der *Code Rules* ergänzt habe. Anschließend folgen die Paketbeschreibungen, die ich für das Paket `concurrent` und dessen Unterpakete (`jupiter`, `management` und `undo`) geschrieben habe. Diese Beschreibungen finden sich als `package-info.java` Datei in den jeweiligen Paketordnern des Saros-Quelltextes.

C.1 Teilabschnitt der Code Rules: Language & Documentation

All source documents are expected to be documented and written in English.

- JavaDoc documentation and names should be meaningful and make the programming element understandable with minimum insight into the code. If your comments make the code more difficult to understand, remove them.

- Don't use single line comments (starting with //) for multi line text, use instead

```
/*
 * one comment that takes up at least
 * two lines
 */
```

- Comments should describe complex code in **shorter** text. Comments like "Create HashMap", "Set Value", "Loop here", or "else" should be removed.

Kinds of comments

This section is based on Steve McConnell's Code Complete, Chapter 32.

There are six categories of comments:

- **Repeat of the Code**

  If something is not complex and thus documentation repeats code, you should not document at all. For instance, don't document getters and setters (an exception would be to explain what the variable actually contains that you are getting).

A counter-example:

```
1 /**
2  * return a stack of String,
3  * @param text
4  * @return Stack
5  */
6 public Stack getFormatJavadocLine(String text) {
7   StringTokenizer st = new StringTokenizer(text,"\n")
      ;
8   Stack stack = new Stack();
9   while (st.hasMoreTokens()) {
10    stack.add(st.nextToken());
11  }
12  return stack;
13 }
```

The documentation is absolutely useless as it just repeats the signature and even fails to explain the method name. Furthermore the method name is wrong for the task that is actually performed. The important question whether this method returns a stack of the individual lines in text from top to bottom or bottom to top remains unanswered.

- **Explanation of the Code**

  If the code needs to be explained, it is usually better to improve the code instead of adding comments.

- **Marker in the Code**

  Marker comments are notes for the developers that the work isn't done yet. They should not be left in the code. If you have to mark a section in the code, use "TODO". This way all marker comments will be standardized and it is easier to locate them.

```
1 /*
2  * TODO FIX: Our caller should be able to distinguish
3  * whether the query failed or it is an IM client
4  * which sends back the message
5  */
```

- **Summary of the Code**

  Comments that summarize the code in a few sentences can be valuable especially for readers who haven't written the code. They can scan these comments more quickly than the code.

```
1 /*
2  * move all chess pieces to start position
3  */
```

- **Description of the Code's Intent**

  Intent comments explain the purpose of a section of code. In contrast to the summary comments which operate at the level of the solution the intent comment operates at the level of the problem.

```
/*
 * initialize a new chess game
 */
```

- **Information that cannot possibly be expressed by the Code Itself**

  Some information needs to be written in comments since they cannot be expressed in code. This can be copyright notices, version numbers, notes about the code's design, optimization notes and so on.

Acceptable code comments are summary comments, intent comments and information that cannot be expressed in code. Markers are acceptable during development but should be removed before release. Try to avoid comments that repeat or explain the code.

What to comment

Generally you should document the code starting from the highest level of code hierarchy. This means that all packages need a documentation followed by interfaces and classes. All documentation should be in JavaDoc comments in order to automatically generate HTML source code documentation.

- Each package should be documented in a `package-info.java` file located in the package folder.

  The package description should help the reader to understand, what the package is about and what its intent is. It should also inform about terms to adhere when using or modifying this package. Important relations to other packages should be described here.

- Each interface should be documented.

  The comments in the interface should provide a short description of what you can use it for. For all exposed routines you should at a minimum document each parameter and each return value but hide implementation details.

- Each class should be documented.

  The description of the class should provide a short overview of what this class is about. Design decisions and limitations should be mentioned as well.

- Methods should be documented, if they are complex enough and it will be helpful for other readers to summarize or explain the purpose of these methods.

  Important objects and variables should also be briefly documented.

C.2 Quelltext-Paketbeschreibungen

C.2.1 Beschreibung des Pakets `concurrent`

```
1 /**
2  * <h1>Concurrency Control Overview</h1>
3  *
4  * The Concurrency Control supports writing collaborations
5  * of multiple users. All participants of a Saros session
6  * shall have a consistent copy of all shared resources.
7  *
8  * The Concurrency Control comprises of four subpackages:
9  *
10 * <ul>
11 * <li>jupiter --- this package contains the Jupiter
12 * Algorithm used for collaborative text editing</li>
13 *
14 * <li>management --- manages all JupiterActivities
15 * (operations processed by the users of a Saros session
16 * with WRITE_ACCESS)</li>
17 *
18 * <li>undo --- package for undo/redo operations within a
19 * Saros session</li>
20 *
21 * <li>watchdog --- since inconsistencies might appear the
22 * Consistency Watchdog takes care of them</li>
23 * </ul>
24 */
25 package de.fu_berlin.inf.dpp.concurrent;
```

C.2.2 Beschreibung des Pakets `concurrent.jupiter`

```
1 /**
2  * <h1>Jupiter Algorithm Overview</h1>
3  *
4  * The Jupiter Architecture is the central concept for
5  * Distributed Party Programming in Saros. Each client
6  * administers its own copy of a shared artifact
7  * (resource).
8  * Local {@link Operation}s will be performed immediately
9  * and communicated to a central server. From there all
10 * other clients get informed about the operation which
11 * they will transform in order to perform it on their own
12 * copies.
13 *
14 * The transformation is needed to keep all copies
15 * consistent.
16 * Consistency maintenance is made up of three components:
17 *
18 * <ul>
19 * <li>Convergence --- all copies of a document have the
20 * same final state after performing a set of operations
21 * to it</li>
22 *
23 * <li>Causality-preservation --- for each pair of
24 * operations O1 and O2 applies that if O1 precedes O2
25 * then the execution of O1 must take place prior to the
26 * execution of O2 for all copies</li>
27 *
28 * <li>Intention-preservation --- for each operation O the
29 * result must be identical to the original intent when
30 * performing O for all copies. The result may not affect
31 * the results of other independent operation</li>
32 * </ul>
33 *
34 *
35 * Important components of the Jupiter package:
36 *
37 * <ul>
38 * <li>{@link internal.Jupiter} is the central class of
39 * the Jupiter Algorithm</li>
40 *
41 * <li>{@link internal.text.GOTOInclusionTransformation}
42 * is an implementation of the transformation function of
43 * the Jupiter Algorithm</li>
44 * </ul>
45 */
46 package de.fu_berlin.inf.dpp.concurrent.jupiter;
```

C.2.3 Beschreibung des Pakets `concurrent.management`

```
1 /**
2  * <h1>Management Overview</h1>
3  *
4  * The Management package manages {@link JupiterServer}
5  * and {@link JupiterClient} instances and all Jupiter
6  * activities.
7  * Local Operations on each client site will be
8  * transformed into JupiterActivities (by {@link
9  * ConcurrentDocumentClient}) and sent to the host ({@link
10 * ConcurrentDocumentServer}).
11 * The JupiterActivity will be transformed into its
12 * corresponding operation and sent to all clients. This
13 * process has to be atomic and may not be interrupted.
14 *
15 * The {@link ConcurrentDocumentServer} exists only on the
16 * host site of a Saros session, even when a client adds
17 * resources to this session.
18 */
19 package de.fu_berlin.inf.dpp.concurrent.management;
```

C.2.4 Beschreibung des Pakets `concurrent.undo`

```
1 /**
2  * <h1>Undo Module Overview</h1>
3  *
4  * The undo module takes care of undoing the last
5  * operation made by the user within a Saros session.
6  * In order to not destroy the work made by another user
7  * in Saros it is important to only undo the last own
8  * operation made by the user.
9  * Because Eclipse only provides a linear undo mechanism
10 * this module is used to keep track of all operations
11 * made by the user for undo/redo purposes.
12 *
13 * <ul>
14 * <li>the {@link OperationHistory} of the user's own
15 * operations is stored</li>
16 *
17 * <li>the {@link UndoManager} only works during a running
18 * Saros session</li>
19 * </ul>
20 */
21 package de.fu_berlin.inf.dpp.concurrent.undo;
```

D Regeln zur Behandlung von Dokumentations-Problemen

In diesem Abschnitt habe ich die Richtlinien aufgeführt, die ich entwickelt habe. Im ersten Teil ist beschrieben, wie man mit Problemen mit der Dokumentation umgehen soll. Im zweiten Teil sind die Aufgaben des Dokumentationsmanagers (DM) aufgelistet, die dieser während des Release ausführen soll.

D.1 Documentation Issue Procedures

How to handle documentation issues

1. New documentation issues should be listed into the *Documentation Issue Tracker*. There are two categories: website and source code.
   - set the status to "Open"
   - documentation issues can be:
     - missing information
     - wrong or outdated information
     - other problems with existing documentation, for instance if it's not understandable
   - if you can correct the issue right away just do so
2. Each documentation issue should be prioritized (see *Documentation Issue Priorities*).
3. The developer who has the knowledge to correct that issue is responsible to do so.
   - please concentrate on the issues with the highest priority first
   - if the issues concerns source code comments ask for a review especially if you have larger comment changes
     - write comments that summarize the code or describe the code's intent (for more information see *Language & Documentation in Code Rules*)
   - after correcting or completing a documentation issue set the resolution to "**Fixed**"
4. The *documentation manager* should close corrected documentation issues entries after s/he confimed the correction.

Documentation issue priorities

Since there are only five priorities needed right now and all sourceforge tracker provide nine the odd priority numbers should be used:

Priority	Meaning
9	This issue has to be corrected/completed immediately!
7	This is a very important documentation issue, e. g. a missing package description of a major package or a very important missing or wrong information on the website.
5	This is an important documentation issue, e. g. an interface or a class description is missing/incomplete or information on the website which is not understandable.
3	This is a documentation issue what needs to be taken care of, e. g. a method that needs to be commented.
1	Not prioritized yet - or documentation issues that aren't important but should be noted.

D.2 Tasks of the DM

1. DM checks all "Fixed" list entries in the *Documentation Issue Tracker* with the status not "Closed"

   - please check also assigned documentation issues which are not marked as "Fixed" – the assignee might just have forgotten to do that – and assign it again if it hasn't been resolved
   - if the documentation issue is resolved the DM **must** close the entry

2. DM adds new documentation issues

   - use the list of newly fixed bugs and features as a starting point
   - the Changelog from SVN could also be helpful

3. DM prioritizes entries if they are not already prioritized (see *Documentation Issues Priorities*)

4. DM assigns entries to someone who has the knowledge to correct that documentation issue

   - start with the issues with the highest priority
   - if you don't know someone to assign it to ask your team members
   - if there really is no one right now, leave it at "none"

5. DM sends a list with the assigned documentation issues to the mailing-list

   - try not to scare off the responsible person with too many issues at once ;-) (There are a lot of documentation issues that need to be resolved right now. But if we go step by step we will get an updated documentation eventually.)
   - example for the **email**:

assignee1:	[Code] Saros editor package: package description (package-info.java) is missing [Website] DPPGettingStarted: add instructions on how to add a Buddy to the Buddy-list (roster)
assignee2:	...

6. all team members should take the time to correct/complete the documentation issues (see also *How to handle documentation issues*)
   - comment changes should be handled the same as source code changes since they are part of it (ask for a review especially if you have larger comment changes)
     - write comments that summarize the code or describe the code's intent (for more information see *Language & Documentation in Code Rules*)
     - if you want to know how a package description could look like see the `package-info.java` files within the `concurrent` package in the Saros source code
     - commit comment changes to the trunk
   - mark the documentation issue entry as "**Fixed**" when you're done
   - if you won't find the time to take care of the issue for this release please do so in the following week(s) that it's done for the next release
7. DM announces the new documentation changes to the mailing list
   - don't forget to tell everybody that they did a good job

Literatur

[Bed04] Thomas Bedorf: *Hinweise zum wissenschaftlichen Arbeiten: Zitieren und Paraphrasieren*, 2004. http://www.fernuni-hagen.de/imperia/md/content/philosophie/textdokumente/bedorf_zitieren.pdf, (20.09.2011).

[BM06] Joachim Bayer and Dirk Muthig: *A View-Based Approach for Improving Software Documentation Practices*. In *13th Annual IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'06)*, 2006.

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

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

[DR10] Barthélémy Dagenais and Martin P. Robillard: *Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors*. In *ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pages 127–136, November 2010.

[FH98] Larry Fellows and Ivy Hooks: *A Case for Priority – Classifying Requirements*. In *Proceedings of the 8th Annual International Symposium of the National Council on Systems Engineering*, pages (1–7), Vancouver, British Columbia, Canada, 1998.

[Grü07] Gertrud Grünwied: *Software-Dokumentation*. expert verlag, Renningen, 2. Auflage, 2007.

[Iwa05] Noriko Iwai: *Japanese General Social Surveys (2) – Methodological Experiments in Administering the Questionnaire, Incentives, Scales and Wording*. ZA-Information / Zentralarchiv für Empirische Sozialforschung, (57):83–102, 2005.

[Juh05] Dietrich Juhl: *Technische Dokumentation: Praktische Anleitungen und Beispiele*. Springer Verlag, Berlin, Heidelberg, New York, 2., neu bearbeitete Auflage, 2005.

[Jur10] Michael Jurke: *Iterative, prototype-driven development of a whiteboard feature for the Distributed Party Programming tool Saros*. Diplomarbeit, Freie Universität Berlin, Dezember 2010.

[Kah11] Björn Kahlert: *Verbesserung der Out-Of-Box-Experience in Saros mittels Heuristischer Evaluation und Usability-Tests.* Masterarbeit, Freie Universität Berlin, Mai 2011.

[Kry11] Veikko Krypczyk: *Wer schreibt, der bleibt.* dotnetpro, 5:20–25, 2011.

[Lau10] Stephan Lau: *Verbesserte Präsenz durch Screensharing für ein Werkzeug zur verteilten Paarprogrammierung.* Bachelorarbeit, Freie Universität Berlin, April 2010.

[Log10] Olaf Loga: *Verbesserung der Kommunikationsmöglichkeiten in einem Werkzeug zur verteilten Paarprogrammierung (Saros).* Bachelorarbeit, Freie Universität Berlin, April 2010.

[LSF03] Timothy Lethbridge, Janice Singer, and Andrew Forward: *How Software Engineers Use Documentation: The State of the Practice.* IEEE Software, 20:35–39, 2003.

[McC04] Steve McConnell: *Code Complete: A practical handbook of software construction*, chapter 32: Self-Documenting Code, pages 777–817. Microsoft Press, Redmond, Washington, 2nd edition, 2004.

[Pöt94] Godehard Pötter: *Die Anleitung zur Anleitung: Leitfaden zur Erstellung technischer Dokumentation.* Vogel Buchverlag, Würzburg, 1. Auflage, 1994.

[Püt10] Florian Pütz: *Analyse und Erweiterung der Voice over IP Funktionalität in Saros.* Bachelorarbeit, Freie Universität Berlin, Juli 2010.

[Rie08] Oliver Rieger: *Weiterentwicklung einer Eclipse Erweiterung zur Realisierung und Protokollierung Verteilter Paarprogrammierung im Hinblick auf Kollaboration und Kommunikation.* Diplomarbeit, Freie Universität Berlin, Juli 2008.

[RRM04] Alexander J. Rostkowycz, Václav Rajlich, and Andrian Marcus: *A Case Study on the Long-Term Effects of Software Redocumentation.* In *20th IEEE International Conference on Software Maintenance (ICSM'04)*, pages (1–10), 2004.

[SOBS10] Stephan Salinger, Christopher Oezbek, Karl Beecher, and Julia Schenk: *Saros: An Eclipse Plug-in for Distributed Party Programming.* In *Proceedings of the 2010 International Conference on Software Engineering Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '10)*, pages 48–55, Cape Town, South Africa, May 2010.

[Som02] Ian Sommerville: *Software Documentation.* In *In Software Engineering, vol. 2: The supporting Processes. R.H. Thayer and M.I. Christensen (eds), Willey-IEEE*, pages 1–19. Press, 2002.

[SVG11] *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*, 2011. http://www.w3.org/TR/SVG11/, (09.06.2011).

[VC02] Marcello Visconti and Curtis R. Cook: *An Overview of Industrial Software Documentation Practice*. In *XXII International Conference of the Chilean Computer Science Society (SCCC'02)*, 2002.

[vdD09] Herausgegeben von der Dudenredation: *Duden, Die Grammatik: unentbehrlich für richtiges Deutsch*, Band 4. Dudenverlag, Mannheim, Zürich, 8., überarbeitete Auflage, 2009.

[vH11] Dimitri van Heesch: *Doxygen – Generate documentation from source code*, March 2011. http://www.stack.nl/~dimitri/doxygen/, (14.04.2011).

[Wie99] Karl E. Wiegers: *First Things First: Prioritizing Requirements*. Software Development, 7(9), 1999. reprinted version (with modifications) 1999.

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

Abbildungsverzeichnis

1 Navigation der Saros-Webseite zu Beginn meiner Arbeit (im April 2011) 10
2 Inhaltsverzeichnis des *Developer Guides* zu Beginn meiner Arbeit. . . 13
a Ungegliedertes Inhaltsverzeichnis des *Developer Guides* im März 2011. 13
b Thematisch gegliedertes Inhaltsverzeichnis des *Developer Guides* im Mai 2011. 13

Diagramme zur Auswertung des ersten Fragebogens (vom Mai 2011)

3 Antworten zur Frage 10: „Der Text ist gegliedert/hat eine Ordnung. (+2) ↔ Der Text ist unübersichtlich/zusammenhangslos. (-2)“ 22
4 Antworten zur Frage 13: „Der Text ist einfach geschrieben. (+2) ↔ Der Text ist kompliziert geschrieben. (-2)“ 22
5 Antworten zur Frage 14: „Der Text ist kurz und prägnant. (+2) ↔ Der Text ist weitschweifig gefasst. (-2)“ 23
6 Antworten zur Frage 15: „Der Text ist aussagekräftig. (+2) ↔ Der Text enthält keine relevanten Informationen. (-2)“ 23
7 Antworten zu den Fragen 20a und b 24
a Frage 20a: „Die Kommentare wiederholen den Code.“ 24
b Frage 20b: „Die Kommentare erklären den Code.“ 24
8 Antworten zu den Fragen 20d und e 25
a Frage 20d: „Die Kommentare fassen den Code zusammen.“ . . 25
b Frage 20e: „Die Kommentare beschreiben den Zweck/was der Code bewirken soll.“ 25
9 Antworten zu den Fragen 20c und j 26
a Frage 20c: „Die Kommentare stellen Markierungen im Code dar.“ 26
b Frage 20j: „Die Kommentare sind gut so, wie sie sind.“ 26
10 Antworten zu den Fragen 24, 25 und 26 27
a Frage 24: „Kommentierst du deinen Quellcode?“ 27
b Frage 25: „Ergänzt bzw. aktualisierst du die Informationen auf der Webseite, die mit deinem Teil der Arbeit an Saros zu tun haben?“ 27
c Frage 26: „Bewertest du während eines Reviews auch die Kommentare im Code der anderen?“ 27

11 Vergleich der alten mit der neuen, umgruppierten Navigation am Beispiel der neuen einleitenden Seite *Developer Guide*. 35
12 Erste Version der überarbeiteten Architekturübersicht von Saros . . . 38
13 Zweite Version der überarbeiteten Architekturübersicht von Saros . . . 39
14 Darstellung der *Eclipse Bridge* und des *User Interface* der Architekturübersicht 41

15 Ausschnitt aus der Webseite *Technical Information* mit Optionen, wie die detaillierten Modulbeschreibungen aufgerufen werden können (am Beispiel *Shared Session*). 42
16 Ausschnitt aus der Webseite *Shared Session*, welche die detaillierten Beschreibungen dieses Moduls enthält. 43
17 Ausschnitt aus der Saros Quelltext-Dokumentation mit verschiedenen Wegen, eine Klassen-Referenz aufzurufen (am Beispiel der Klasse Jupiter). 44
18 Ausschnitt aus der Jupiter Klassen-Referenz mit Vererbungs- und Kollaborationsdiagramm. 45
19 Ausschnitt aus der Paketbeschreibung des Pakets Jupiter. 46
20 Ausschnitt aus dem *Documentation Issue Tracker*. 53

Diagramme zur Auswertung des zweiten Fragebogens (vom September 2011)

21 Antworten zu den Fragen 6 und 7 61
a Frage 6: „Der Text ist gegliedert/hat eine Ordnung. (+2) ↔ Der Text ist unübersichtlich/zusammenhangslos. (-2)“ 61
b Frage 7: „Du findest die Informationen dort, wo du sie erwartest. (+2) ↔ Die Information ist an einem Ort, wo sie nicht hingehört. (-2)“ 61
22 Antworten zu den Fragen 10 und 12 62
a Frage 10: „Der Text ist kurz und prägnant. (+2) ↔ Der Text ist weitschweifig gefasst. (-2)“ 62
b Frage 12: „Die Angaben sind korrekt (sie stimmen mit der Realität überein). (+2) ↔ Die Angaben sind fehlerhaft. (-2)“ . . . 62
23 Antworten zu den Fragen 15b und c 64
a Frage 15b: „Die Kommentare erklären den Code.“ 64
b Frage 15c: „Die Kommentare stellen Markierungen im Code dar (z. B. ToDo’s)“ 64
24 Antworten zu den Fragen 15d und e 65
a Frage 15d: „Die Kommentare fassen den Code zusammen.“ . . 65
b Frage 15e: „Die Kommentare beschreiben den Zweck/was der Code bewirken soll.“ 65

25 Saros high-level architecture 82
26 *Shared Session* view 84
27 *Eclipse Bridge* view 86
28 *User Interface* view 87
29 *Concurrency Control* view 89
30 *Invitation* process view 91
31 *Network Layer* view 92
32 *Feedback* view 94