Partielle Projektsynchronisation und bedarfsgerechte Dateisynchronisation in Saros

MASTERARBEIT

für die Prüfung zum

Master of Science

im Studiengang Informatik
an der Freien Universität Berlin

von

KARL HELD

Bearbeitungszeitraum	17.03. - 16.09.2011
Matrikelnummer	4404101
Erstgutachter	Prof. Dr. Lutz Prechelt
Zweitgutachter	Prof. Dr. Elfriede Fehr
Betreuer	Dr. Karl Beecher

Kurzfassung

Diese Masterarbeit erweitert das in der Arbeitsgruppe Software Engineering der Freien Universität Berlin gepflegte Werkzeug Saros. Saros dient der Softwareentwicklung mit verteilten Beteiligten (*engl.:* Distributed Party Programming (DPP)). Für eine effektive Zusammenarbeit an großen Projekten ist ein effizientes und flexibles Ressourcenmanagement erforderlich. Es soll um die gemeinsame Nutzung partieller Projekte gemeinsam nutzen zu können und einen bedarfsgerechten Synchronisationsansatz erweitert werden.
Die Heuristische Evaluation wird zur Detektion bestehender Usability-Probleme im Synchronisationsprozess herangezogen. Daraus hervorgehende Ergebnisse werden als Grundlage für die Implementierung genutzt. Sie ermöglichen weiter eine Evaluation der gemachten Änderungen.
Die entwickelte partielle Projektsynchronisation unterstützt auf vielfältige Art die Ressourcenauswahl. Die bedarfsorientierte Dateisynchronisation erweitert zusätzlich intuitiv die Zusammenarbeit der Entwickler.
Große Projekte lassen sich mit den neu verfügbaren Funktionalitäten schneller und flexibler gemeinsam nutzen. Die abschließende Heuristische Evaluation sowie die Aussagen des GQM-Ansatzes untermauern die geforderten Qualitätsansprüche. Die erweiterte Awareness erzeugt zusätzliche Akzeptanz der Saros-Benutzer.

Abstract

This master thesis extends the groupware tool Saros, which is maintained by the software engineering working group of the Freie Universität Berlin. Saros enables distributed software development with multiple persons (DPP). A flexible resource management is mandatory for effective distributed collaboration especially in large projects. For that reason a partial sharing function and a need-based file synchronization has to be implemented.
For detection of usability conflicts in the synchronization process a heuristic evaluation is performed. The following implementation is based on these findings. Additionally the heuristics are used to evaluate the implementation later on.
The partial sharing function enables the selection of resources in various ways. The need-based file synchronization extends the collaborative development intuitively.
The new functionalities allow a more flexible cooperation on larger projects. The final heuristic evaluation in addition to the GQM approach validate given quality requirements. The enhanced awareness induce more acceptance for Saros users.

Erklärung/Declaration

Hiermit versichere ich, dass die vorliegende Arbeit von mir angefertigt wurde und ich keine weiteren als die angegebenen Quellen und Hilfsmittel benutzt habe. Die aus den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen wurden als solche kenntlich gemacht.

I hereby declare to have written this thesis on my own, having used only the listed resources and tools.

Berlin, den 13. September 2011 Karl Held

Inhaltsverzeichnis

Abkürzungsverzeichnis

API Application Programming Interface

CVS Concurrent Versions System

DIN Deutsches Institut für Normung

DPP Distributed Party Programming

ECF Eclipse Communication Framework

EN Europäische Norm

GQM Goal-Question-Metric

IDE Integrated Development Environment

IEC International Engineering Consortium

ISO International Organization for Standardization

JDK Java Development Kit

STF Saros Test Framework

SVN Apache Subversion

VCS Version Control System

XML Extensible Markup Language

XMPP Extensible Messaging and Presence Protocol

Abbildungsverzeichnis

Tabellenverzeichnis

Codesegmentverzeichnis

1 Einleitung

Bei der Softwareentwicklung entsteht immer wieder die Situation, dass viele Personen an einem Projekt arbeiten. Dabei werden häufig gleiche Dateien von mehreren Personen bearbeitet. Eine direkte Interaktion der Softwareentwickler ist in solchen Fällen häufig notwendig (vgl. (Lau00, Seite 2 f.)). Diese Zusammenarbeit kann durch die persönliche Anwesenheit oder durch Softwarehilfsmittel (sog. Groupware) erreicht werden. Falls die genutzte Software nicht speziell hierfür entwickelt wurde, kann es zu Inkonsistenzen in den bearbeiteten Dateien kommen. Bei räumlicher Trennung sollte also eine passende Struktur aufgebaut werden, die die Interaktion von Menschen unterstützt und unerwünschte Nebeneffekte verhindert. (vgl. (Uni10))

Die Arbeitsgruppe Software Engineering des Institutes für Informatik der Freien Universität Berlin[1] ist in der Lehre und Forschung in allen Bereichen der Softwareplanung und Entwicklung engagiert. Eines der ausgeprägtesten Forschungsgebiete ist das Entwicklerwerkzeug Saros[2]. Dieses, vornehmlich auf studentischen Arbeiten basierende, Plugin erweitert die Eclipse[3] Entwicklungsumgebung mit zahlreichen Funktionalitäten und Ansichten. Sie ermöglicht eine Softwareentwicklung mit verteilten Beteiligten (*engl. kurz:* DPP). Eine der Hauptaufgaben von Saros ist die Awarenessunterstützung für die kollaborative Softwareentwicklung.

Saros bietet in seiner Gesamtheit hervorragende Vorteile für eine verteilte Zusammenarbeit und Interaktion von Kollegen, Freunden und Mitarbeitern. Dabei soll diese Softwarelösung für große, wie auch für kleine, Projekte und Anwendungen nutzbar und praktikabel sein.

Gewachsene Projekte zeichnen sich in der Regel durch ihre Komplexität und Vielzahl von Dateien, sowie der daraus resultierenden Größe aus. In einem Unternehmensumfeld sind Projektgrößen zwischen mehreren hundert Megabyte bis in die Gigabyte üblich. Auch in diesem komplexen Umfeld wird verteilte Zusammenarbeit praktiziert.

Daraus ergeben sich Problematiken, die der aktuelle Zustand von Saros noch nicht bewälti-

[1] `http://www.inf.fu-berlin.de/en/groups/ag-se/index.html`
[2] `http://www.saros-project.org/`
[3] `http://www.eclipse.org/`

gen kann, oder nicht effizient löst. Zum Beispiel ist es problematisch, wenn mit einem sehr großen Projekt gearbeitet wird, welches bei der eingeladenen Partei nicht oder nur in Teilen vorhanden ist. In dieser Situation wird eine sehr große Menge von Daten verglichen, verarbeitet, komprimiert und übertragen. Selbst wenn nur an wenigen Dateien dieses Projektes eine Zusammenarbeit erfolgt, resultiert dies in einem hohen Aufwand.
Ziel der folgenden Masterarbeit ist die Lösung dieses Problems. Hierzu wird im Folgenden die Aufgabe näher erläutert, der Zeitplan vorgestellt und die weitere Vorgehensweise beschrieben. Danach wird die Arbeitsumgebung dieser Masterarbeit erläutert und die Struktur der Arbeit vorgestellt.

1.1 Aufgabenstellung und Zielsetzung

Die Projektsynchronisation im Saros-Plugin wird projektbasiert durchgeführt. Das bedeutet, dass die kleinste zu synchronisierende Einheit ein ganzes Projekt ist. Diese Variante hat gewisse Vorteile, die es auch zu behalten gilt. Zum Beispiel sind nach der Synchronisation alle Dateien eines Projektes bei allen Sitzungsteilnehmern identisch. Jedoch kann es für Anwender zeitaufwendig und nicht erforderlich sein, alle Dateien zu synchronisieren. Dies ist der Fall, wenn nur Teile eines Projektes oder einzelne Dateien für eine Zusammenarbeit relevant sind. Aus diesem Grund ist es erstrebenswert, dass Saros die Möglichkeit bietet, auf Dateiebene zu synchronisieren.
Die Umsetzung einer solchen Funktionalität bedarf einer Vielzahl von wichtigen Entwurfs-Entscheidungen. Wenn lediglich einzelne Dateien einer statisch typisierten Sprache synchronisiert werden, ist eine Kontrolle schwieriger, als bei einer dynamisch typisierten. Daraus ergibt sich die Notwendigkeit einer bedarfsgerechten Datenübertragung. Die bedarfsgerechte Dateisynchronisation ermöglicht es Dateien priorisiert übertragen zu können.
Zu diesem Zweck wird in dieser Arbeit für die **partielle Projektsynchronisation** und die **bedarfsgerechte Datenübertragung** Konzept und Umsetzung erarbeitet. Beide Ansätze werden zunächst analysiert und dann evaluiert.
Die partielle Projektsynchronisation soll eine intuitive Möglichkeit sein, dem Benutzer Spielraum bei der Ressourcenauswahl zu lassen. Zudem sollen alle Sitzungsteilnehmer der gemeinsam nutzbaren Ressourcen gewahr sein und diese erweitern können. Die Benutzbarkeit der bedarfsgerechten Dateiübertragung soll gleichwertig zu der partiellen Lösung sein.
Der Benutzer soll immer wissen, was eine Interaktion zwischen ihm und Saros auslöst. Der Anspruch ist, den Nutzern das Leben zu erleichtern, indem die gemeinsame Zusammenarbeit schneller ermöglicht wird. Außerdem soll dem Nutzer stetiges Feedback für anstehende

oder durchgeführte Aktionen gegeben werden.
Ziel ist einerseits die gewünschten Erweiterungen des Saros-Eclipse-Plugins umzusetzen und andererseits dies unter Berücksichtigung wissenschaftlicher Maßstäbe durchzuführen. Die eigentliche Implementierung der Funktionalitäten basiert auf vorheriger Analyse und Planung. Darauf aufbauende Tests erweitern den Arbeitsumfang zusätzlich.
Letztlich ist das Ziel funktionsfähige Erweiterungen für Saros zu implementieren. Dabei darf die Usability („Benutzbarkeit“) und Awareness („Gewahrsamkeit“) nicht außer Acht gelassen werden.

1.2 Herangehensweise

Durch das erfolgreich abgeschlossene Softwareprojekt der Softwaretechnik und der daraus gewonnenen Erfahrung mit der Software Saros, ist keine weitere Einarbeitung nötig.
Das Studium der für diese Arbeit relevanten Klassen und Methoden ergibt, dass kein expliziter Ansatz zur Aufgabenerfüllung gegeben ist. Eine experimentelle partielle Projektsynchronisations-Möglichkeit war vorher vorhanden, diese war jedoch weder verlässlich noch gut nutzbar. Sie war nie über den experimentellen Status hinaus weiterentwickelt worden. Darum ist sie fast vollständig aus dem Code von Saros verschwunden, wodurch kein Zurückgreifen auf vorhandene Codesegmente möglich ist.
Der Ansatz für die bedarfsgerechte Datenübertragung ist gänzlich neu und vollkommen unerprobt. Er muss von Grund auf entwickelt, konzeptioniert und evaluiert werden.
Nach diesen Erkenntnissen wurden genauere Anforderungen definiert und Use Cases entworfen. Außerdem wurde ein Vorgehensmodell gewählt sowie ein Projektstukturplan und Meilensteinplan ausgearbeitet. Im weiteren Verlauf dieses Kapitels werden diese Punkte genauer beschrieben.

1.2.1 Anforderungen

Gemäß der Aufgabenstellung gilt es die Anforderungen für diese Arbeit zu definieren. Dabei wird vor allem auf die funktionalen Anforderungen eingegangen. Beim Sammeln von Anforderungen steht immer die Frage „*Was soll das System leisten?*“ im Vordergrund. Das „*wie*“ wird nicht definiert (vgl. (Fra07)). Aus der Aufgabenstellung und weiteren Vorüberlegungen für die Umsetzung lassen sich folgende funktionelle Anforderungen ableiten:

- Partielle Dateisynchronisation ermöglicht:
  - eine Auswahl beliebiger Ressourcen

  - partielle Integration gemeinsamer Ressourcen in vorhandenes lokales Projekt
  - Konsistenzsicherung der Ressourcen einer Sitzung
  - stabile und schnelle Synchronisierung
  - intuitive Benutzbarkeit und Awarenessgestaltung
  - Verfügbarkeit der Funktionalität für jeden Sitzungsteilnehmer

- Bedarfsgerechte Dateisynchronisation ermöglicht:
  - priorisierte Übertragung von Dateien während des Synchronisationprozesses
  - Erweiterung von partiellen Projekten einfach und schnell um weitere Sitzungsressourcen
  - intuitive Benutzbarkeit und Awarenessgestaltung
  - Verfügbarkeit der Funktionalität nur für Ressourceninhaber

Nach Abschluss der praktischen Umsetzung kann anhand der Anforderungen ein Vergleich auf vollständige Funktionalität der gewünschten Saros-Erweiterungen stattfinden. Außerdem wird in Kapitel 2 auf Seite 16 f. erläutert, wie die wesentlich umfangreichere Evaluation durchgeführt wird.
Den funktionalen Anforderungen gegenüber, stehen die nichtfunktionalen Anforderungen an ein System. Diese beschreiben „*wie gut*“ ein System etwas leisten soll. Dem entsprechende Qualitätsattribute werden beispielsweise im Rahmen des ISO Standards 9126 definiert. (siehe dazu Abschnitt 2.1 auf Seite 16)

1.2.2 Use Cases

Mit der Use Case[4] Analyse sollen alle möglichen Szenarien innerhalb des Projektes untersucht werden. Ein Use Case beschreibt ein mögliches Szenario. In einem solchen Szenario versucht ein festgelegter Akteur ein bestimmtes Ziel in der Software zu erreichen.
Die Use Cases werden entsprechend dem angepassten Schema von Alistair Cockburn[5] (vgl. (Ali98)) erstellt und detailliert beschrieben. Im Folgenden werden entsprechende Use Cases aufgezählt. Die genaue Beschreibung ist im Anhang A auf Seite 85 zu finden. Diese Anwendungsfälle sollen umgesetzt werden:

- Ressourcenauswahl in Saros-View
- Ressourcenauswahl im Projektauswahl-Assistenten
- Ressourcenauswahl im Project Explorer von Eclipse
- beliebige Projekt/Ordner/Datei-Kombinationen in ein vorhandenes Projekt synchronisieren
- beliebige Projekt/Ordner/Datei-Kombinationen in ein nicht vorhandenes Projekt synchronisieren
- beliebige Projekt/Ordner/Datei-Kombinationen in eine laufende Sitzung in ein vorhandenes Projekt synchronisieren
- beliebige Projekt/Ordner/Datei-Kombinationen in eine laufende Sitzung in ein nicht vorhandenes Projekt synchronisieren
- bedarfsgerechte Dateisynchronisation während Projektsynchronisation
- bedarfsgerechte Dateisynchronisation in partiell geteiltes Projekt

1.2.3 Verwendetes Vorgehensmodell

Vorgehensmodelle ermöglichen es, den Entwicklungsprozess in einzelne Phasen aufzugliedern und zu strukturieren. Diese Phasen, welche die zu erledigenden Aktivitäten enthalten, sind zeitlich voneinander abhängig. Für dieses Projekt ist nach (Som06, Seite 101 f.) die Umsetzung in Form einer inkrementell, iterativen Vorgehensweise gewählt. Diese erlaubt es

[4] *deutsch:* Anwendungsfall

[5] Dr. Alistair Cockburn ist ein US-amerikanischer Informatiker und IT Stratege. Er ist ein Experte der agilen Softwareentwicklung und gehört zu den Urhebern des Agilen Manifestes. (vgl. (Dr.08))

dem Entwickler, nacheinander Teilaspekte zu implementieren und diese dann zu einem Gesamtergebnis zusammenzuführen. Wie in Abbildung 1.1 dargestellt, initiiert das Ende einer Iteration die Planung der nächsten Iteration. Der wesentliche Vorteil der sich daraus ergibt ist, dass nach Fertigstellung eines jeden Teilaspektes dieser getestet werden kann. Dies gewährleistet eine gute Qualität der Software. Zudem können Änderungen im Ablauf direkt vorgenommen werden, wodurch dieses Vorgehensmodell an Agilität gewinnt. Dadurch ist nicht erst in der Endphase dieser Arbeit lauffähiger und fehlerfreier Quellcode vorhanden. Somit ist eine vergleichsweise zeitnahe Abschätzung der Machbarkeit der Arbeit möglich. Zudem wird bei diesem Vorgehensmodell darauf Wert gelegt, dass eine rege Kommunikation mit dem „Auftraggeber“ stattfindet. Im Falle dieser Arbeit ist dies der Betreuer Karl Beecher als Projektleiter. Es zeigt sich, dass das Festhalten weder an traditionellen noch agilen Vorgehensmodellen, sowie deren strikte Umsetzung, speziell für dieses Vorhaben nicht praktikabel ist. Jedoch kann in der Strukturierung einer Iteration ein Vergleich zum Wasserfallmodell (vgl. (Seb04)) gezogen werden. Dies wird in Abbildung 1.1 dargestellt.
Grundsätzlich folgt einer Analysephase immer die Entwurfsphase. An diese schließt sich dann die Implementierungs- und zuletzt die Testphase an.

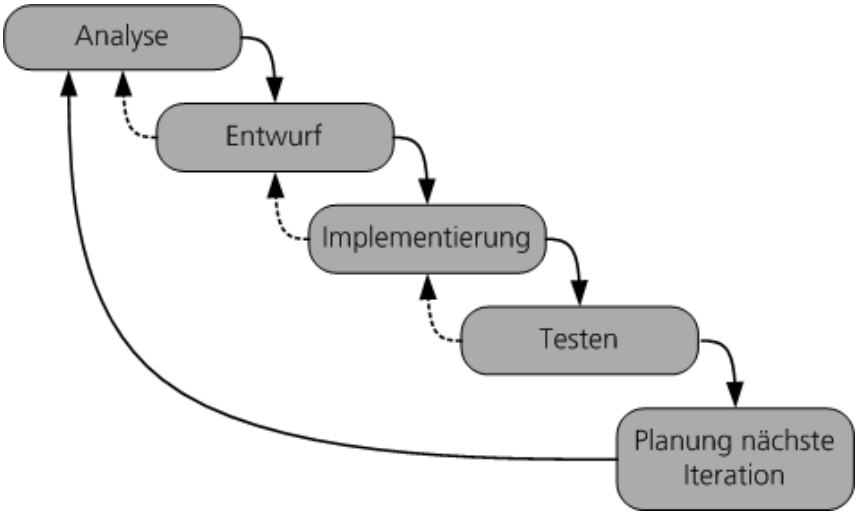


Abbildung 1.1: Inkrementell iteratives Vorgehensmodell

Soweit sind die Analogien zum Wasserfallmodell gegeben (vgl.(Som06, Seite 101)). Jedoch schon bei der Vermischung mit dem iterativen Vorgehen, wird wesentlich vom traditionellen Modell abgewichen. Das Wasserfallmodell sieht keine Erweiterungen oder Änderungen nach dem Abschluss der Implementierungsphase vor. Dies ist jedoch Hauptbestandteil des gewählten inkrementellen Vorgehens. Sie ermöglicht Flexibilität, die für dieses Vorhaben ge-

währleistet sein muss. Das entwickelte inkrementell, iterative Vorgehensmodell gliedert die folgende Ausarbeitung. (Phasen siehe Abbildung 1.3 auf Seite 8)

1.2.4 Projektstrukturplan

Ein Projektstrukturplan dient der optimalen Zeiteinteilung und Organisation aller Aktionen, die für den erfolgreichen Abschluss der Masterarbeit nötig sind. Zu diesem Zweck wird die Arbeit als Projekt gegliedert und in plan- und kontrollierbare Teilaufgaben zerlegt. (vgl. (Som06, Seite 128)) Um Abhängigkeiten darzustellen werden Beziehungen zwischen den einzeln entstandenen Arbeitspaketen definiert. Somit kann ein softwaregestütztes, einfaches Projektcontrolling erfolgen (vgl. (Ins08, Seite 145, 154, 157 f.)). Für die in dieser Masterarbeit gestellte Aufgabe eignet sich besonders gut ein Gantt-Balkendiagramm (siehe Abbildung 1.2). Unter Zuhilfenahme von *Microsoft Project 2010* lässt sich ein solches schnell und flexibel erstellen.

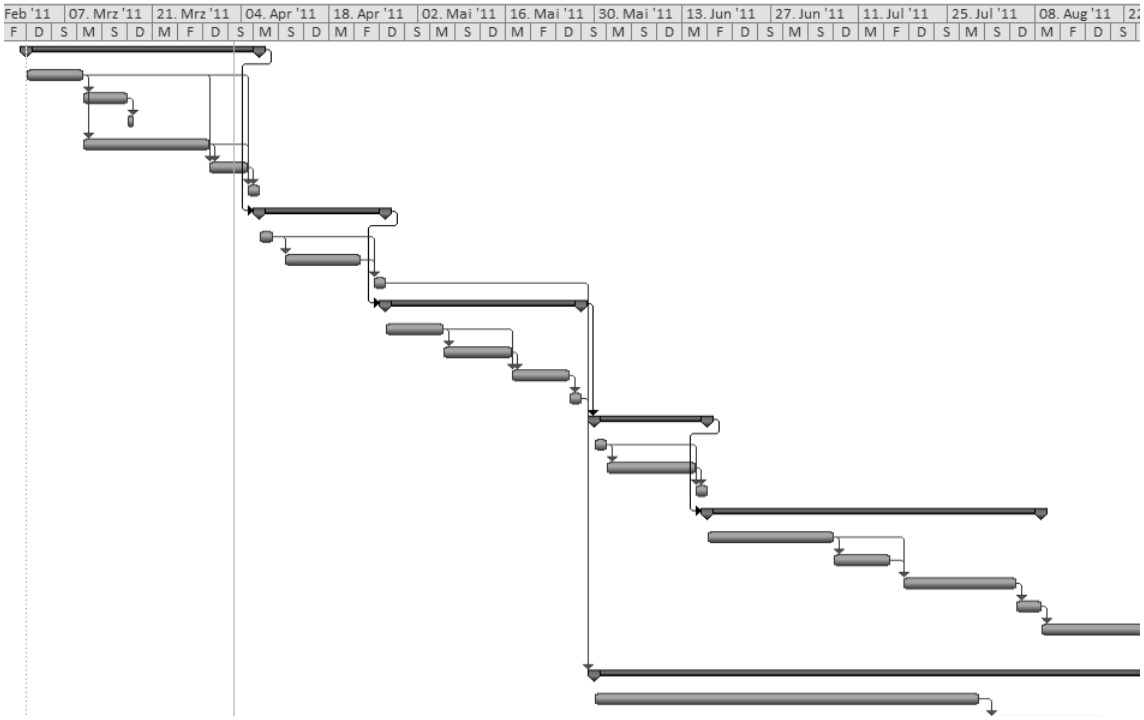


Abbildung 1.2: Gantt-Diagramm dieser Masterarbeit, erstellt mit *Microsoft Project 2010*

In Abbildung 1.3 auf Seite 8 ist zu erkennen, dass die Arbeitspakete als Vorgänge definiert sind. Diese Vorgänge können hierarchisch weiter aufgebrochen werden, so werden weitere Teilaufgaben zu einem Arbeitspaket definiert. Aus definierten Vorgängen wird automatisch ein Gantt-Balkendiagramm, wie in Abbildung 1.2 dargestellt, erzeugt. *Microsoft Project* erkennt automatisch alle Teilaufgaben, die die Mindestprojektdauer bestimmen. Diese kenn-

zeichnen den kritischen Pfad des Projektes. Die Balken definieren über ihre Länge die Zeit des jeweilig daneben stehenden Vorgangs. Ein schwarzer Streifen innerhalb eines Balkens zeigt prozentual den Projektfortschritt an. Vollständig ausgefüllte Balken entsprechen einer fertiggestellten Teilaufgabe.

Die Balken sind miteinander durch Pfeile verbunden. Sie zeigen an, welcher Vorgang von welchem abhängig ist. Zudem lassen sich Meilensteine definieren, welche einem Zwischenziel entsprechen. Diese Meilensteine sind ein gutes Mittel für die Überwachung des Projektverlaufs. (vgl. (Som06, Seite 129 f.))

Vorgangsname	Dauer	Anfang	Fertig stellen	Vorgänger
Einarbeitung	**27 Tage**	**Di 01.03.11**	**Mi 06.04.11**	
Arbeitsplatz/-umgebung einrichten	7 Tage	Di 01.03.11	Mi 09.03.11	
UseCases + Meilensteinplan	5 Tage	Do 10.03.11	Mi 16.03.11	2
Codekonventionen lesen	1 Tag	Do 17.03.11	Do 17.03.11	3
Code studieren (relevante Stellen ausmachen)	14 Tage	Do 10.03.11	Di 29.03.11	2
alte Implementierung kennen lernen/evaluieren	4 Tage	Mi 30.03.11	Mo 04.04.11	2;5
Ergebnisse zusammenfassen	2 Tage	Di 05.04.11	Mi 06.04.11	2;5;6
Lösungsansätze Partial Sharing generieren	**14 Tage**	**Do 07.04.11**	**Di 26.04.11**	**1**
UI-Elemente auswählen	2 Tage	Do 07.04.11	Fr 08.04.11	
Implementierungsverfahren klären	10 Tage	Mo 11.04.11	Fr 22.04.11	9
Ergebnisse zusammenfassen	2 Tage	Mo 25.04.11	Di 26.04.11	9;10
Implementierung Partial Sharing	**23 Tage**	**Mi 27.04.11**	**Fr 27.05.11**	**8**
erste Version implementieren	7 Tage	Mi 27.04.11	Do 05.05.11	
Testen	7 Tage	Fr 06.05.11	Mo 16.05.11	13
schrittweise Verfeinerung	7 Tage	Di 17.05.11	Mi 25.05.11	13;14
Ergebnisse zusammenfassen	2 Tage	Do 26.05.11	Fr 27.05.11	15
Lösungsansätze Lazy Sharing generieren	**19 Tage**	**Mo 30.05.11**	**Do 23.06.11**	**12**
UI-Elemente auswählen	4 Tage	Mo 30.05.11	Do 02.06.11	
Implementierungsverfahren klären	10 Tage	Fr 03.06.11	Do 16.06.11	18
Ergebnisse zusammenfassen	5 Tage	Fr 17.06.11	Do 23.06.11	18;19
Implementierung Lazy Sharing	**38 Tage**	**Fr 24.06.11**	**Di 16.08.11**	**17**
erste Version implementieren	14 Tage	Fr 24.06.11	Mi 13.07.11	
Testen	7 Tage	Do 14.07.11	Fr 22.07.11	22
schrittweise Verfeinerung	14 Tage	Mo 25.07.11	Do 11.08.11	22;23
Ergebnisse zusammenfassen	3 Tage	Fr 12.08.11	Di 16.08.11	24
Empirische Bewertung der Implementierung	14 Tage	Mi 17.08.11	Mo 05.09.11	25
Bugfixing	9 Tage	Di 06.09.11	Fr 16.09.11	26
Ausarbeitung	**77 Tage**	**Mo 30.05.11**	**Di 13.09.11**	**16;11**
Schreiben	45 Tage	Mo 30.05.11	Fr 29.07.11	
Korrrektur lesen	14 Tage	Mo 01.08.11	Do 18.08.11	29
Korrigieren	14 Tage	Fr 19.08.11	Mi 07.09.11	30
Drucken	4 Tage	Do 08.09.11	Di 13.09.11	31

Abbildung 1.3: Projektstrukturplan für diese Masterarbeit

Ein Projektstrukturplan bietet sich bei einer Vielzahl von Aufgabenstellungen an. Meist findet eine zeitliche, wie auch kostenbedingte, Begrenzung eines Projektes statt. Damit kann sehr schnell der Einfluss von Veränderungen auf den Projektfluss erkannt und ausgewertet werden. Vorausgesetzt der Projektstrukturplan wird ständig gepflegt, gibt er immer eine Kontrollmöglichkeit des momentanen Standes der Arbeit. Zudem ist die gute Ablesbarkeit der Strukturierung eines Projektes sehr nützlich.
Die Vorgangsliste dieser Masterarbeit wird in Abbildung 1.3 auf Seite 8 dargestellt. Aus dieser Vorgangsliste lassen sich vier Hauptphasen dieser Arbeit ablesen:

1. **Vorbereitung**
2. **Entwicklung**
3. **Analyse**
4. **Schreiben**

Die **Entwicklung** ist die längste Phase, was durch die hohe Anzahl von Unteraufgaben weiter untermauert wird. Durch die erwartet umfangreichen Implementierungsarbeiten, wird diese Phase so zeitaufwendig eingeschätzt. Die Implementierung an sich erfolgt nach dem im Abschnitt 1.2.3 auf Seite 5 f. beschriebenen Vorgehensmodell.
Ebenso relevant ist der theoretische Anteil an dieser Arbeit, welcher in den Phasen **Vorbereitung** und **Analyse** stattfindet.
Bei dieser Masterarbeit wurde darauf Wert gelegt, dass neben der Implementierung, Zeit für begleitendes Schreiben ist. Das hat den Vorteil, dass immer über den gerade aktuellen Bearbeitungspunkt dokumentiert werden kann. Es zeigt sich schon hier, wie praxisrelevant und hilfreich ein solches Diagramm bei jedem mittleren und größeren Projekt ist.

1.3 Arbeitsumgebung: Distributed Party Programming

Distributed Party Programming (DPP), also die Softwareentwicklung mit verteilten Beteiligten, ist eine Erweiterung der Paarprogrammierung. DPP zeichnet sich durch die verteilte Zusammenarbeit von zwei und mehr Softwareentwicklern aus. In den folgenden zwei Unterkapiteln erfolgt eine kurze Erläuterung der Begriffe: „Paarprogrammierung“ und „Verteilte Programmierung“. Danach wird das Softwarewerkzeug Saros und die Konzepte, auf die es aufsetzt, erklärt.

1.3.1 Paarprogrammierung

Paarprogrammierung ist eine Arbeitstechnik, die vor allem bei agiler Softwareentwicklung angewendet wird. Bei der Paarprogrammierung arbeiten zwei Entwickler zusammen an einer gemeinsamen Problemlösung. Dabei wird typischerweise nur ein Rechner verwendet. Die Arbeitstechnik der Paarprogrammierung sieht zwei Rollen für die Interaktionspartner vor. Da nur einer der Beteiligten Tastatur und Maus bedienen kann (Programmierer), obliegt es dem Anderen den Bildschirm zu beobachten (Beobachter). Die Paarprogrammierung fordert einen regen Gedankenaustausch der gleichberechtigten Interaktionspartner. Der Beobachter kontrolliert den geschriebenen Quelltext, spricht mögliche Probleme an und sucht mit dem Programmierer gemeinsam nach Lösungen. Diese Rollen sind nicht fest zugewiesen. Es empfiehlt sich häufig die Rollen untereinander zu tauschen.
Ziel der Paarprogrammierung ist die Designverbesserung und Fehlervermeidung. Durch das kollaborative Verhältnis wird Wissen geteilt, ausgetauscht und gegebenenfalls die Arbeitsmoral der Beteiligten gesteigert (vgl. (Lau00, Seite 39 f.)). Komplexe Lösungen und neue, einschleichende Probleme sollen durch Paarprogrammierung vermieden werden. Daraus resultiert eine Steigerung der Softwarequalität.
Ein nicht zu vernachlässigender Nebeneffekt, zu den bereits genannten Vorteilen, ist die Verbreitung des Wissens über die Codebasis. Da stets zwei Softwareentwickler an genau derselben Stelle des Codes arbeiten, erlangen beide Wissen über den entsprechenden Code. Dadurch wird das Projektrisiko verringert, dass durch die Fluktuation bzw. die Abwesenheit von Mitarbeitern entsteht. (vgl. (Ste10), (Lau00, Seite 2 f.) und (Til05, Seite 1 f.))
Jedoch gibt es auch Kritikpunkte dieser Arbeitstechnik. Bei einfachen Aufgaben wird die Effizienz bemängelt. Immerhin werden zwei Softwareentwickler an diese Arbeitstechnik gebunden und produzieren somit potentiell weniger Code. Große Unterschiede der fachlichen Kompetenz der Beteiligten oder unterschiedliche Arbeits- und Herangehensweisen können hinderlich für erfolgreiches Paarprogrammieren sein.
Paarprogrammierung birgt also die Gefahr der Ineffizienz. Befürworter hingegen behaupten, dass die Effizienz gesteigert werden kann. Beispielsweise werden Ablenkungen während der Arbeit von zwei Personen besser kompensiert.
Unumstritten ist, dass wenn mehr als ein Augenpaar den Code begutachtet und dessen Entwicklung vorantreibt, die Codequalität erheblich gesteigert werden kann.

1.3.2 Verteilte Programmierung

Die verteilte Programmierung erweitert das Konzept der Paarprogrammierung. Es ermöglicht eine Mehrzahl an Interaktionspartnern, sowie die Arbeit an getrennten Computern. Augenscheinlicher Vorteil ist das Vorhandensein von Tastatur und Maus für jeden Softwareentwickler. Gestützt wird diese verteilte Programmierung durch Software, welche die Interaktionen am Bildschirm allen zugänglich macht. Saros, auf Basis von Eclipse, ist eines der bekanntesten Werkzeuge für verteilte Programmierung. Es hält die Kommunikationsbarriere der Interaktionspartner, welche durch den verteilten Kontext entsteht, möglichst klein.
Die Vorteile der Paarprogrammierung sind auf die verteilte Paarprogrammierung übertragbar. Aus diesem Grund können mehr als zwei Teilnehmer diese Vorteile nutzen. Typische Anwendungsszenarien der verteilten Paarprogrammierung sind das Anlernen von Neulingen, gemeinsames Begutachten, Paarprogrammierung und Side-by-Side-Programmierung[6]. (vgl. (Ste10))

1.3.3 Das Eclipse-Plugin-Konzept

Eclipse ist eine vielseitige und anpassbare Entwicklungsumgebung für die Softwareentwicklung in verschiedenen Programmiersprachen. Sie bietet einen Eclipse-Kernel, auf dem zusätzliche Erweiterungen aufsetzen. Saros ist eine solche Erweiterung, ein so genanntes Plugin. Aus diesem Grund soll an dieser Stelle ein Blick auf Eclipse-Plugins im Allgemeinen geworfen werden.

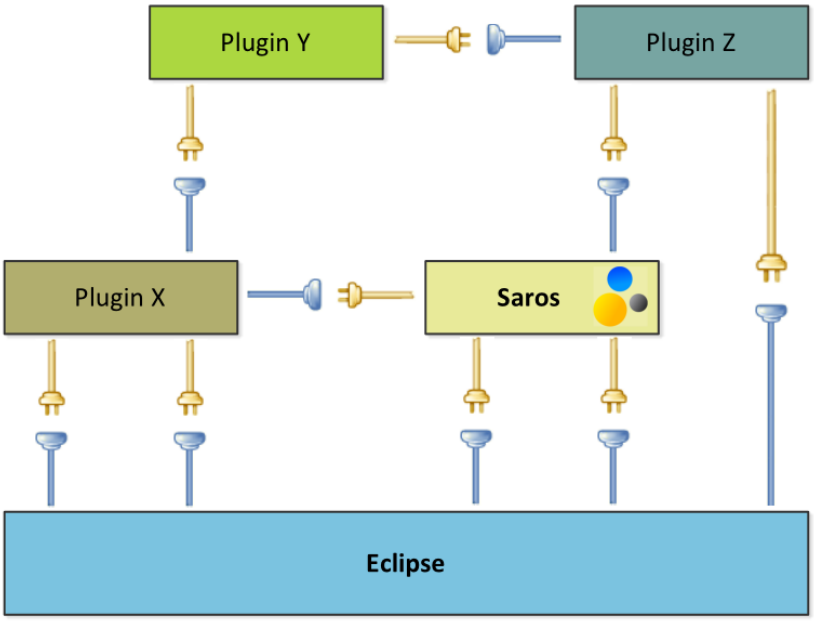


Abbildung 1.4: Schematische Darstellung des Eclipse-Plugin-Konzeptes ((Bjö11, Seite 5))

[6] „Zwei oder mehr Entwickler arbeiten jeder für sich an eigenen Aufgaben [...]. Bei Bedarf kann jeder von ihnen jeden anderen ansprechen und um Hilfe oder Zusammenarbeit bitten [...].“ (Ste10) (vgl. (DASH09))

Plugins sind Komponenten, welche Erweiterungen und Dienste innerhalb der Eclipse Arbeitsumgebung zur Verfügung stellen. Diese Komponenten werden zur Laufzeit in das Eclipse Rahmenwerk integriert. Innerhalb einer Eclipse Instanz wird das Plugin in eine Art Laufzeitklasse eingebettet. Diese bietet Konfigurationsunterstützung der Plugin-Instanz. Diese Plugin-Klasse wiederum erweitert *org.eclipse.core.runtime.Plugin*: eine abstrakte Klasse welche generische Funktionalitäten für das Plugin Management bereitstellt (vgl. (Aza03)).
Der Eclipse-Kernel bietet zur Integration die barrierefreie Grundlage für die Aktivierung und Interaktion mehrerer Plugins. Diese so genannten *Eclipse Extension Points* (in Abbildung 1.4 auf Seite 11 blau) werden vom Plugin durch entsprechende *Extensions* (gelb) erweitert. Zudem können Plugins *Extension Points* bereitstellen, die wiederum durch andere Plugins genutzt werden können. Dieses Konzept ist in Abbildung 1.4 auf Seite 11 dargestellt (vgl. (Bjö11, Seite 4 f.)).

1.3.4 Das Saros-Eclipse-Plugin

Saros ist ein aktiv in der Arbeitsgruppe Software Engineering an der Freien Universität Berlin entwickeltes Open Source[7] Eclipse-Plugin für die verteilte Zusammenarbeit. Das gesamte Basiskonzept, sowie dessen elementare Implementierung entstammt einer Diplomarbeit von Riad Djemili (Ria06a) aus dem Jahre 2006. Seitdem wird es kontinuierlich erweitert und verbessert.
Das Saros-Plugin vergrößert den Funktionsumfang der Entwicklungsumgebung Eclipse um mächtige Werkzeuge zur Echtzeit-Kollaboration und Interaktion von zwei oder mehr verteilten Beteiligten. Den Entwicklern werden zu diesem Zweck verschiedene visuelle Elemente und Funktionen zur Verfügung gestellt. Dies erlaubt bestmögliche Kommunikation und zeigt, *was*, *wo*, durch *wen* verändert wird.
Dazu wird zwischen den Teilnehmern, eine Session (*deutsch:* Sitzung) aufgebaut. Saros verwaltet dazu eine Liste von „*Buddies*“ (*deutsch:* Kumpel/Freund), welche an einer Sitzung teilnehmen können. Eine umfangreiche Sitzungsverwaltung synchronisiert die Dateien und Ordner eines Ursprungs-Projektes. So bekommen alle eingeladenen Teilnehmer den gleichen lokalen Bestand an Ressourcen. Die Sitzungsteilnehmer können miteinander interagieren, sei es zum gemeinsamen Programmieren, Reviewen oder Diskutieren. Im weiteren Verlauf einer Sitzung wird kontinuierlich eine Konsistenzkontrolle auf Basis von Prüfsummen durchgeführt. Diese hält zu jedem Zeitpunkt alle Teilnehmer in einer Sitzung synchron.

[7] „Open Source, das heißt offener Quellcode und meint gemeinhin Software, die jeder nach Belieben studieren, benutzen, verändern und kopieren darf.“(Bun07) Dazu existieren unterschiedlichste Lizensierungsmodelle.

Zusätzlich zur Synchronisierung macht Saros alle Aktionen eines Sitzungsteilnehmers allen Anderen zugänglich. Dabei verarbeitet das Plugin nicht nur Texteingaben in beliebige Dateien und Formate, sondern auch Aktionen wie das Erstellen, Verschieben und Löschen von Ressourcen. Alle Aktionen können gleichzeitig von verschiedenen Teilnehmern durchgeführt werden und unter Umständen auch miteinander konkurrieren.
Neben einer „Echtzeit-Editierbarkeit" von Ressourcen können noch weitere Funktionen zur Kommunikations- und Interaktionsunterstützung verwendet werden. Dazu steht ein Chat, ein Whiteboard und Voice-over-IP-Funktonalität zur Verfügung. Diese dienen zur Untermauerung und Erweiterung der auf den Editor beschränkten Zusammenarbeit innerhalb des Eclipse-Fensters.

1.3.5 Konkurrenzprodukte zu Saros

Nicht nur an der FU Berlin wird an Softwarelösungen zur verteilten Zusammenarbeit gearbeitet. Neben Saros sind noch weitere Konkurrenzprodukte am Markt. Zu nennen sind hier **Sangam**[8], **PEP**[9] - Pair Eclipse Programming, **XPairtise**[10], und **XecliP**[11]. Bei jedem dieser Softwarelösungen handelt es sich um Eclipse Plugins, welche verteilte Paarprogrammierung gewährleisten sollen. Im folgenden Abschnitt werden Ähnlichkeiten und Unterschiede dieser Lösungen mit der von Saros aufgezeigt.

Funktionsumfang

Der Funktionsumfang unterscheidet sich bei den betrachteten Plugins stark. Allen gemein ist die Unterstützung von mindestens zwei Interaktionspartnern. Die grundsätzlichen Funktionalitäten sind wie folgt vorhanden.

- **Sangam:** verteilte Paarprogrammierung, gemeinsames Editieren
- **PEP:** verteilte Paarprogrammierung, Chat, Code (Re-)Synchronisation, kein Server notwendig
- **XPairtise:** verteilte Paarprogrammierung, gemeinsames Editieren, Projektsynchronisation, Chat, Whiteboard, Nutzermanagement
- **XecliP:** verteilte Paarprogrammierung nur auf Java Projekten, Projekt muss über CVS synchronisiert werden

[8] `http://sangam.sourceforge.net/`
[9] `http://pep-pp.sourceforge.net/`
[10] `http://xpairtise.sourceforge.net/`
[11] `http://xeclip.sourceforge.net/`

- **Saros:** verteilte Programmierung mit mehreren Beteiligten, gemeinsames Editieren, gemeinsame Operationen im Project Explorer, Projekt- und Ressourcensynchronisation, Konsistenzmanagement, Nutzermanagement, Whiteboard, Chat, Screensharing, Accountmanagement, Awareness- und Usabilityerweiterungen

Der weitaus größere Funktionsumfang von Saros fällt deutlich auf. Zudem ist Saros das einzige Plugin, welches mehr als zwei Anwender in einer Sitzung verarbeiten kann. Grundsätzlich können die meisten Plugins Projekte synchronisieren und unterstützen gemeinsames Editieren.

Aktualität

Bei Saros handelt es sich um ein gepflegtes sowie weiterentwickeltes Eclipse Plugin. Somit ist eine Kompatibilität zum aktuellesten JDK und Eclipse-Version stets gegeben. Des weiteren werden die gegebenen Funktionalitäten ständig gepflegt und verbessert, sowie neue Erweiterungen hinzugefügt. Mit dieser Strategie versucht das Saros Entwicklerteam jeden Monat ein Release bereitzustellen, welches eine Vielzahl von Verbesserungen beinhaltet.
In Betracht der Aktualität ist Saros den Konkurrenzprodukten weit überlegen. Dies ist schon bei der Pflege der eigenen Webseite und des Releasezyklus zu erkennen. Diese Übersicht spiegelt diesen Eindruck wieder.

- **Sangam** - letztes Update: 11.01.2011
- **PEP** - letztes Update: 17.07.2009
- **XPairtise** - letztes Update: 12.05.2010
- **XecliP** - letztes Update: 08.06.2010
- **Saros** - letztes Update: 29.07.2011

Vor- und Nachteile

Wesentlicher Vorteil von Saros ist eindeutig der überlegene Funktionsumfang. Zudem wird an Saros nach wie vor aktiv verbessert und weiterentwickelt. Keine der gefundenen Konkurenzlösungen unterstützt die Zusammenarbeit mit mehr als zwei Teilnehmern.

1.4 Struktur der Arbeit

Nach dieser Einleitung, wird die weitere Arbeit wie folgt strukturiert. Kapitel 2 ab Seite 16 gibt eine Einführung in die verwendeten Evaluationsmethoden. Darauf folgt in Kapitel 3 ab Seite 28 die Beschreibung des Konzeptes und der Umsetzung der partiellen Projektsynchronisation. Dabei wird auf die wesentlichen Bestandteile und Änderungen, die an Saros gemacht werden, eingegangen. Die bedarfsgerechte Dateisynchronisation wird im darauf folgenden Kapitel 4 ab Seite 45 näher erläutert. In Kapitel 5 ab Seite 52 erfolgt die Anwendung, der in Kapitel 2 eingeführten Evaluationsmethoden. Das Ergebnis der Evaluation wird abschließend in selben Kapitel zusammengefasst. Zuletzt erfolgt ein Fazit in Kapitel 6 ab Seite 79. Dies umfasst eine Zusammenfassung und den abschließenden Ausblick dieser Masterarbeit.

2 Einführung in verwendete Evaluationsmethoden

Für die Evaluation der geleisteten Arbeit sollen spezifische Methoden verwendet werden. Sie sollen eine Aussage über die qualitative Umsetzung der Arbeit geben. Dazu wird einerseits eine heuristische Evaluation und andererseits eine ISO Norm für Softwarequalität verwendet. Diese Auswahl erfolgte aufgrund einer vorhergehenden Recherche nach häufig eingesetzten Standards zur Softwareevaluation. Außerdem wird der Bezug zur vorhergehenden Masterarbeit von Björn Kahlert (Bjö11) hergestellt. Da beide Evaluationsmethoden Überschneidungen vorweisen, wird im folgenden jede Evaluationsmethode für sich erläutert. Darauf folgt eine Analyse von Ähnlichkeiten und Schnittmengen, sowie die Anpassung auf das zu untersuchende Problem. Im Fokus dieser Untersuchungen sollen die als **FURPS** (vgl. Adaption aus Norm ISO/IEC 9126 & (Pet05)) bekannten Qualitätsmerkmale sein. Dieses Akronym fasst die wesentlichen Qualitätsmerkmale zusammen:

- **F**unctionality (Funktionalität)
- **U**sability (Benutzbarkeit)
- **R**eliability (Zuverlässigkeit)
- **P**erformance (Effizienz)
- **S**upportability (Änderbarkeit/Wartbarkeit)

2.1 Evaluation mittels der Norm ISO/IEC 9126 für Softwarequalität

Die Bewertung der Lösung der gegebenen Aufgaben soll anhand der Norm ISO/IEC 9126 (vgl. (Wik11b)), einem Softwarequalitätsmodell, nachvollzogen werden. Diese Norm ist ein weit verbreitetes Modell für die Feststellung von Qualitätsstandards bei Software. Dazu wird die Produktqualität anhand sechs generischer Qualitätsmerkmale festgestellt und gemessen.

Diese Qualitätsmerkmale sind funktions- aber auch implementierungsbezogen und decken somit einen großen Anteil der Umsetzung guter Software ab. Aufgrund ihrer generischen Natur sind häufig Anpassungen notwendig, um sie auf ein reales Projekt anzuwenden (vgl. (P. 04, Seite 1)). Folgende sechs Qualitätsmerkmale bzw. Qualitätsfaktoren werden aufgeführt (adaptiert aus (P. 04) und (Wik11b)):

1. Funktionalität - Inwieweit besitzt die Software die geforderten Funktionen?

2. Zuverlässigkeit - Kann die Software ein bestimmtes Leistungsniveau unter bestimmten Bedingungen über einen bestimmten Zeitraum aufrecht erhalten?

3. Benutzbarkeit - Welchen Aufwand fordert der Einsatz der Software von den Benutzern und wie wird er von diesen beurteilt?

4. Effizienz - Wie liegt das Verhältnis zwischen Leistungsniveau der Software und eingesetzten Betriebsmitteln?

5. Änderbarkeit - Welchen Aufwand erfordert die Durchführung vorgegebener Änderungen an der Software?

6. Übertragbarkeit - Wie leicht lässt sich die Software in eine andere Umgebung übertragen?

Sowohl die Anforderung als auch die Implementierung werden anhand dieser Merkmale bewertet. Dies soll einerseits die Deckung der Anforderungen zur Umsetzung widerspiegeln, andererseits die Qualität der Software bewertbar machen.
Im Folgenden wird jedes Merkmal weiter verfeinert und genauer erläutert. Diese Untermerkmale werden später als Grundlage der Bewertung verwendet.

Funktionalität: Inwieweit sind die geforderten Funktionen vorhanden und Erfüllen die definierten Anforderungen.

- **Angemessenheit:** Eignet sich die Funktion für diese spezifische Aufgabe?
- **Richtigkeit:** Wie ist die Genauigkeit und Angemessenheit zu erwarteten Ergebnissen?
- **Interoperabilität:** Wie ist das Zusammenwirken mit vorgegebenen/anderen Systemen?
- **Sicherheit:** Wird der unberechtigte Zugriff auf Programme/Daten verhindert?
- **Ordnungsmäßigkeit:** Werden anwendungsspezifische Normen und Vereinbarungen erfüllt?

In dem Kontext dieser Arbeit werden lediglich Angemessenheit, Richtigkeit und Sicherheit genauer betrachtet. Diese Einschränkung kann gemacht werden, da nicht alle Untercharakteristiken im Rahmen dieser Arbeit evaluierbar sind. Saros deckt nicht alle Untercharakteristiken ab und kann deswegen dahingehend nicht untersucht werden.

Zuverlässigkeit: Fähigkeit der Software den Leistungsumfang unter festgelegten Bedingungen zu bewahren.

- **Reife:** Führen Fehlerzustände zu geringer Versagenshäufigkeit?
- **Fehlertoleranz:** Erhält die Software den Leistungsumfang bei Fehlern oder Fehlbenutzung aufrecht?
- **Wiederherstellbarkeit:** Wird das Leistungsniveau und die Daten nach Fehlern oder Problemen wieder hergestellt?
- **Konformität:** Welcher Grad der Erfüllung anwendungsspezifischer Normen und Vereinbarungen zur Zuverlässigkeit ist gegeben?

Reife, Fehlertoleranz und Wiederherstellbarkeit sind in dieser Ausarbeitung Bestandteil der Untersuchung.

Benutzbarkeit: Der Aufwand, der für den Benutzer zur Verwendung der Software nötig ist.

- **Verständlichkeit:** Wie ist die Nachvollziehbarkeit des Konzeptes der Umsetzung?
- **Erlernbarkeit:** Wie hoch ist der Aufwand die Anwendung/Funktion zu erlernen?
- **Bedienbarkeit:** Wie hoch ist der Aufwand die Anwendung/Funktion zu bedienen?
- **Attraktivität:** Wie hoch ist die Anziehungskraft der Anwendung/Funktion für den Benutzer?
- **Konformität:** Welcher Grad der Erfüllung anwendungsspezifischer Normen und Vereinbarungen zur Benutzbarkeit ist gegeben?

Der Aufwand soll dem eigenen Anspruch nach möglichst gering sein, aber auch Kriterien der Verständlichkeit, Erlernbarkeit, Bedienbarkeit und Attraktivität erfüllen.

Effizienz: Das Verhältnis des Leistungsniveaus der Software zum aufgewendeten Umfang der Betriebsmittel.

- **Zeitverhalten:** Wie lang ist die Verarbeitungs- und Anwendungsdauer der Funktionalität?

- **Verbrauchsverhalten:** Wie groß ist die Anzahl und Dauer der benötigten Ressourcen bei der Erfüllung der Funktionen?
- **Konformität:** Welcher Grad der Erfüllung anwendungsspezifischer Normen und Vereinbarungen zur Effizienz ist gegeben?

Diese Arbeit umschließt eine Betrachtung des Zeitverhaltens und des Verbrauchsverhaltens.

Änderbarkeit: Aufwand, der zur Durchführung vorgegebener Änderungen (Korrekturen, Verbesserungen oder Anpassungen) notwendig ist.

- **Analysierbarkeit:** Wie hoch ist der Aufwand um Mängel oder Ursachen von Versagen festzustellen oder änderungsbedürftige Teile zu bestimmen?
- **Modifizierbarkeit:** Wie hoch ist der Aufwand um zu verbessern, Fehler zu beseitigen oder an Umgebungsänderungen an zu passen?
- **Stabilität:** Wie hoch ist die Wahrscheinlichkeit des Auftretens unerwarteter Wirkungen von Änderungen?
- **Testbarkeit:** Wie hoch ist der Aufwand zur Prüfung geänderter Software?
- **Konformität:** Welcher Grad der Erfüllung anwendungsspezifischer Normen und Vereinbarungen zur Änderbarkeit ist gegeben?

Wichtige Punkte sind hier die Analysierbarkeit, Modifizierbarkeit, Stabilität und Testbarkeit.

Übertragbarkeit: Eignung der Software, von der Umgebung in eine andere übertragen werden zu können. Umgebung kann organisatorische Umgebung, Hardware- oder Software-Umgebung sein.

- **Anpassbarkeit:** Ist Anpassungsfähigkeit an verschiedene Umgebungen gegeben?
- **Installierbarkeit:** Wie groß ist der Installationsaufwand in festgelegter Umgebung?
- **Koexistenz:** Wie ausgeprägt ist die Fähigkeit neben gleichartiger Software zu arbeiten?
- **Austauschbarkeit:** Gibt es die Möglichkeit diese Software für eine andere nutzbar zu machen? Wie hoch ist der Aufwand dazu?

- **Konformität:** Welcher Grad der Erfüllung anwendungsspezifischer Normen und Vereinbarungen zur Übertragbarkeit ist gegeben?

Diese Merkmale werden bei der partiellen Projektsynchronisation und der bedarfsgerechten Dateiübertragung angewendet um eine qualitätsbezogene Aussage zu treffen. Dazu empfiehlt sich die Anwendung von Metriken, die zum Messen der Gegebenheiten verwendet werden können. Das verwendete Layout der Metriken gleicht dabei dem des Goal-Question-Metric-Ansatzes. (siehe Absatz 2.1.2 auf Seite 22)
Es ergibt sich folgender Arbeitsablauf: Mit der Norm ISO/IEC 9126 wird ein Qualitätsmodell aufgebaut. Dieses dient als Rahmenwerk zur Softwareevaluation. Das Qualitätsmodell fußt auf den gegebenen generischen sechs Qualitätsmerkmalen von Software. Aus denen leiten sich weitere Untercharakteristiken ab. Diese Untercharakteristiken wiederum zerlegen sich in Softwareattribute, welche durch Metriken messbar werden. (vgl. (P. 04, Seite 2))

2.1.1 Verfeinerung und Anpassung der Charakteristiken

Wie bereits erwähnt zeichnet sich die Softwarequalität durch die sechs „high-level“ Qualitätsfaktoren und deren Untercharakteristiken aus. Um diese evaluieren zu können, sieht der 9126 Standard die Definition der „lowest-level“ Qualitätsfaktoren sowie der entsprechenden Metrik vor. Problematisch dabei ist, das typischerweise ISO/IEC Standards sehr generisch gestaltet sind und somit einen weiten Interpretationsspielraum zu lassen. Zudem sind nicht alle Qualitätsfaktoren auf jedes Softwareprodukt anwendbar sind. Aus diesem Grund wird eine Reduktion und Zusammenfassung der gegebenen Merkmale durchgeführt. Dies ermöglicht eine bessere Einschätzung der geleisteten Arbeit.

Funktionalität ist eine wichtige Softwarecharakteristik und wird deshalb hier genauer auf die Untercharakteristiken **Richtigkeit** und **Sicherheit** untersucht.

- **Richtigkeit** - In dieser Untercharakteristik soll die Korrektheit der Implementierung aufgezeigt werden. Dazu wird auf Basis diverser wiederholter Szenarien die partielle Synchronisierung, sowie die bedarfsgerechte Dateisynchronisation untersucht.
- **Sicherheit** - Da beim partiellen und bedarfsgerechten Synchronisieren auch sensible Daten und Projekte verarbeitet werden, muss die Sicherheit betrachtet werden. Dabei wird analysiert, ob unberechtigt Zugriff auf nicht geteilte bzw. gemeinsam genutzte Dateien erlangt werden kann.

Zuverlässigkeit geht einher mit der Funktionalität und ist ein wichtiger Faktor für gute Software. Aus diesem Grund wird **Reife**, **Fehlertoleranz** und **Wiederherstellbarkeit** der Software analysiert.

- **Reife** - Fehlerzustände sollen bei den neuen Funktionen entweder vermieden werden, oder zumindest zu keinem vollständigen Versagen führen. In diesem Hinblick muss explizit versucht werden Fehlerzustände und Extremfälle während der Synchronisation zu provozieren.
- **Fehlertoleranz** - Fehler bzw. Fehlbenutzung können gegebenenfalls zu einem Versagen der Funktionalität führen. Es muss untersucht werden, wie die Software diese behandelt, damit umgeht und sogar die Funktionalität aufrecht erhält. Dazu muss speziell die Fehlbenutzung forciert werden.
- **Wiederherstellbarkeit** - Da sich beide Erweiterungen mit der Synchronisierung von Projekt-Ressourcen beschäftigen, muss die Konsistenzverwaltung zu jedem Zeitpunkt korrekt funktionieren. Zur Überprüfung müssen Inkonsistenzen in jedem Teilprozess verursacht werden und dann lösbar sein.

Benutzbarkeit wird unter Verwendung der heuristischen Evaluation in Kapitel 2.2 auf Seite 24 intensiver evaluiert. Hier ist sie als Erweiterung Teil der Evaluierung. Dabei steht **Verständlichkeit**, **Erlernbarkeit** und **Bedienbarkeit** im Vordergrund.

- **Verständlichkeit** - Das Umsetzungskonzept soll sich an das bereits bestehende Synchronisierungskonzept angliedern. Somit soll getestet werden, ob das vorherige Verständnis der Software weiterhin beim Nutzer besteht. Zudem muss die Erweiterung durch die neuen Funktionen ähnlich verständlich und logisch organisiert sein.
- **Erlernbarkeit** - Die neuen Möglichkeiten sollen sich problemlos und übergangslos in die gewohnte Funktionalität eingliedern. Der Aufwand für das Verständnis und die Anwendbarkeit soll minimal und selbsterklärend sein. Es soll geprüft werden, ob mögliche Probleme der Erlernbarkeit den neuen Funktionalitäten zu Grunde liegen.
- **Bedienbarkeit** - Das Bedienkonzept neuer Funktionen soll sich an vorhergehenden Umsetzungen orientieren. Es gilt die Nutzerinteraktion mit den neuen Funktionen zu analysieren und Barrieren zu finden.

Änderbarkeit beschreibt vor allem das Quelltextstruktur, aber auch die Art und Weise der Umsetzung. Der Aspekt der Änderbarkeit darf nicht vernachlässigt werden und somit wird **Analysierbarkeit** und **Modifizierbarkeit** genauer betrachtet.

- **Analysierbarkeit** - Dokumentation, Umsetzung und Lesbarkeit müssen als wichtige Merkmale der Analysierbarkeit untersucht werden. Dazu ist unter anderem eine statische Codeanalyse geeignet.
- **Modifizierbarkeit** - Die Prämisse ist eine einfache Modifizierbarkeit zu gewährleisten. In diesem Zusammenhang müssen potentielle Möglichkeiten für die Verbesserung der Modifizierbarkeit aufgezeigt werden.

Effizienz und Übertragbarkeit werden in diesem Zusammenhang **nicht** genauer betrachtet.

2.1.2 Der Goal-Question-Metric-Ansatz

Als Erfinder der Goal-Question-Metric (GQM) gelten Prof. Victor R. Basili und Dr. David Weiss. Beide sind Mitarbeiter im Software Engineering Labor, des NASA Goddard Space Flight Centers. Die Idee hinter GQM ist: nicht das zu messen, was einfach messbar ist, sondern das, was man in Bezug auf die eigenen Qualitätsziele wissen will. (vgl. (Chr09, Seite 5)) Dabei handelt es sich bei GQM nicht um einen Bottom-Up-Ansatz. Denn es wird nicht von existierenden Metriken ausgegangen, sondern von den eigenen Qualitätszielen. Es handelt sich also um einen Top-Down Ansatz, bei dem von den Qualitätszielen ausgehend eigene Metriken erzeugt werden. Daraus ergeben sich interpretierbare und auf die Qualitätsziele ausgerichtete Ergebnisse.
Dazu werden bei GQM Ziele[12] einer Software identifiziert und verwendet um Fragen abzuleiten. Diese Fragen charakterisieren wiederum die Art und Weise, wie ein spezifisches Ziel erreicht werden soll. Zu diesen Fragen werden Metriken assoziiert, welche die Fragen quantitativ subjektiv oder objektiv beantworten. (vgl. (P. 04, Seite 6)) Das Verhältnis zwischen „*Zielen* und *Fragen*“ sowie „*Fragen* und *Metriken*“ ist beide Male 1-zu-n. Die Struktur des GQM-Ansatzes ist eine Hierarchie, welche auch als Graph dargestellt werden kann.
Der Aufbau des GQM Graphens wird in 6 Schritten gemacht: (vgl. (Lut10, Slide 33/34) und (DAC08))

1. Schritt: Festlegung des Zieles (**G**oal) das untersucht werden soll

2. Schritt: Formulierung der Fragen (**Q**uestion), welche von Interesse sind

[12](vgl. Abschnitt 1.1 auf Seite 2)

3. Schritt: Definition der konkreten **M**etrik

4. Schritt: Anwendung/Implementierung der manuellen/automatisierten **M**essung

5. Schritt: Sammlung valider Daten aus der **M**essung

6. Schritt: Datenanalyse zur Beantwortung der Fragen (**Q**uestion)

Ergebnis: zielorientierte Aussage

Für den ersten Schritt wird jedes GQM-Ziel durch fünf Aspekte ausgedrückt:

1. **Objekt der Messung**
2. **Zweck**
3. **Qualitätsfokus**
4. **Blickwinkel**
5. **Kontext**

Das **Objekt der Messung** legt fest, welches Objekt zu untersuchen ist. Dies kann ein Prozess, ein Produkt oder Ressourcen sein. Durch eine Beschreibung des **Zwecks** wird von vornherein festgelegt was erreicht werden soll. Der entsprechende **Qualitätsfokus** erzeugt den Bezug zu einem Qualitätsattribut, wie der Wartbarkeit oder Performance. Das Messobjekt wird aus dem gewählten **Blickwinkel** betrachtet. Der Blickwinkel hat eine objektbezogene Rolle, zum Beispiel aus der Sicht des Kunden, des Entwicklers oder des Projektleites. Der **Kontext** beschreibt letztlich die Anwendungsdomäne der Messung. (vgl. (Chr09))
Der zweite Schritt umfasst die Formulierung von Fragen zur genaueren Definition der Ziele. In dieser Phase werden Fragen gestellt, welche helfen dem Ziel näher zu kommen. Dabei muss klar sein, was im Qualitätsfokus zu erfragen gilt. Die Frage muss nicht nur das Ziel treffen, sondern die Bedeutung der Antwort muss klar definiert sein. Da das *Ziel* zu *Fragen* Verhältnis 1-zu-n ist, sollte auch auf eine angemessene Anzahl von Fragen geachtet werden. (vgl. (Chr09))
Der dritte Schritt baut auf dem zweiten auf und soll entsprechende Messziele identifizieren und Metriken ableiten. Im Wesentlichen geht es hier um das Finden der entsprechenden Metrik, die die Fragestellung beantworten kann.
Schritt vier dient der Entwicklung von Mechanismen zur Datensammlung. Hier wird festgelegt, wie die Daten erfasst werden. Gesammelte Messwerte sind dann einheitlich und vergleichbar. Erst durch die Verarbeitung im nächsten Schritt, der sich mit der Analyse und

Interpretation erlangter Daten befasst, werden die Daten repräsentativ. Gewonnene Ergebnisse werden validiert und analysiert, um sie im abschließenden sechsten Schritt zusammenzufassen und auf die Ziele anzuwenden. (vgl. (Chr09))
Da der GQM-Ansatz hier auf die Qualitätsfaktoren der ISO Norm 9126 angewendet werden soll, sind die Ziele vorgegeben. Die Evaluation der erstellten Funktionen erfolgt im Kapitel 5 ab Seite 52. Dort werden relevante Fragen und Metriken ermittelt und ausgewertet.

2.2 Heuristische Evaluation

Die heuristische Evaluation[13] ist eine Methode, die qualitative Aussagen über die Usability von Software ermöglicht. Sie dient zur Bewertung der Gebrauchstauglichkeit einer Benutzeroberfläche vor Fertigstellung des Gesamtsystems. (vgl. (Wik11a))
Die Planung und die Implementierung dieser Arbeit basiert auf einer detaillierten heuristischen Evaluation des Saros Einladungsprozesses (Masterarbeit Björn Kahlert (Bjö11)). Diese Arbeit soll eine weiterführende Usability-Untersuchung durchführen. Vor und nach der Implementierung sollen vorhandene, potentiell neu hinzugekommene und eliminierte Usability Probleme identifiziert werden. Hintergrund dieser spezifischen Evaluation ist der ausgeprägte Mehrwert einer gesteigerten Usability. (vgl. (Som06, Seite 416 f.))
Das erwartete Ergebnis der Evaluation ist eine gleichbleibende oder sogar gesteigerte Usability. Vorteil dieser Evaluationsmethode ist, dass sie in jedem Entwicklungsstadium angewendet werden kann. Sie ist weiter in einem überschaubaren Zeitrahmen durchführbar.

2.2.1 Ablauf der heuristischen Evaluation

Im Folgenden wird der allgemeine Ablauf der heuristischen Evaluation und die Auswahl der untersuchten Usability-Merkmale (Heuristiken) beschrieben.
Begründer der heuristischen Evaluation ist Jakob Nielsen zusammen mit Rolf Molich im Jahre 1990. Nielsen sieht für eine Evaluation eine Gruppe von fünf Usability-Experten vor, um potentielle Probleme späterer realer Nutzer ausfindig zu machen (vgl. (Flo06)). Dies wird an dieser Stelle nicht gemacht, da die Usability-Betrachtung zwar Evaluationsmittel der Implementierung, jedoch nicht Hauptaugenmerk ist. Dazu fehlen zudem Ressourcen in Form von Usability-Experten.

[13] Heuristik (griech. *heuriskein* „finden“) ist in diesem Zusammenhang ein allgemein anerkanntes Prinzip für gute Usability.

Die heuristische Evaluation untersucht das technische System auf die Einhaltung von definierten Heuristiken. Dazu nimmt der Evaluator gezielt die Position eines späteren Benutzers ein. Durch Anwendung von Heuristiken und eigenem Usability-Wissen findet er so etwaige Probleme in der Benutzbarkeit. Dabei entdecken Usability-Experten, die zusätzliches Wissen über die Anwendungsdomäne haben, am häufigsten Probleme der Software. Typische Phasen der Durchführung einer heuristischen Evaluation sind: (vgl. (Flo06, Seite 204 - 212))

1. **Vorevaluatives Training**
2. **Evaluationsdurchgänge**
3. **Auswertung der Usability-Probleme**
4. **Bewertung und Kategorisierung aller gefundenen Hinweise**

Phase eins dient der Einführung in die Methode und die Heuristiken. Zudem wird eine überblicksartige Erläuterung zur Wissensdomäne der Applikation und ihrer grundlegenden Problemstellung gemacht. Dadurch erhalten die Evaluatoren ein grundsätzliches Bild von der Zielpopulation der späteren Nutzer (vgl. (Flo06, Seite 204)). Diese Phase der Sensibilisierung für den Anwendungsbereich ist in dieser Arbeit nicht nötig, da die Rolle des Evaluators durch mich übernommen wird. Nötiges Wissen über die Anwendungsdomäne und etwaige Usability-Ansprüche wird einerseits aus der Literatur und anderseits durch die Lektüre der Masterarbeit von Björn Kahlert ((Bjö11)) gewonnen.

Die **zweite Phase** wird so beschrieben: „Jeder Evaluator geht unter Zuhilfenahme der Heuristiken das System mindestens zweimal [...] durch.“ ((Flo06, Seite 205)) Im ersten Durchgang gewöhnt sich der Evaluator an das System. Der zweite Durchgang enthält eine möglichst vollständige und analytische Erfassung einzelner Probleme und ihrer Ursachen. Als Durchgang wird eine fest spezifizierte Arbeitsaufgabe definiert, welche es zu lösen gilt. Das Auffinden und das Analysieren der Probleme sind zwei unterschiedliche Prozesse. Aus diesem Grund steht in dieser Phase die reine Identifikation von Verstößen gegen die Heuristiken im Mittelpunkt. Die Verstöße werden erst in der nächsten Phase interpretiert.

Phase drei wertet die Evaluationsergebnisse aus. Dazu werden alle in Phase zwei gefundenen Probleme analysiert und entsprechend granular zusammengefasst. Zudem werden in diesem Schritt Dubletten eliminiert.

Die **letzte Phase** bewertet die Schwere (Fatalität) gefundener Probleme. Dazu werden vier Faktoren berücksichtigt: Die **Frequenz des Auftretens**, der **Einfluss auf die Arbeitsabläufe**, die **Persistenz des Auftretens** und der **Markteinfluss**. (vgl. (Nie94))

Diese logische Priorisierung der gefundenen Hinweise spiegelt die Notwendigkeit der Behebung der Problemstellung wider. Bei Sarodnick und Brau wird die Fatalität mittels einer Fünferskala kategorisiert. Unterschieden wird vom „kosmetischen Problem“ bis hin zur „Usability-Katastrophe“ (vgl. (Flo06)).
Diese vier Schritte werden zwei Mal in dieser Arbeit durchgeführt und Ergebnisse präsentiert. Es wird eine Evaluation vor und eine Evaluation nach der Implementierung gemacht. Dabei wird auf Ergebnisse aus einer vorhergehenden Untersuchung durch Björn Kahlert (vgl. (Bjö11, Seite 177)) Bezug genommen.

2.2.2 Heuristiken zur Usability-Auswertung

Bereits erwähnte Heuristiken sollen Usabilityprobleme aufspüren. Dazu werden Verstöße gegen eine Heuristik als potentielle Usability-Probleme kategorisiert.
Folgende Heuristiken basieren auf einem Satz genereller Heuristiken nach DIN EN ISO 9241 Teil 10, Erfahrungen aus Evaluationsprojekten sowie Literaturrecherchen. Sie sollen eine eindeutige Kategorisierung begünstigen und einfach anwendbar sein. (vgl. (Flo06, Seite 140 f.))

1 Aufgabenangemessenheit - Alle benötigten Funktionen für anstehende Aufgaben sind im System vorhanden. Sie sind hinreichend so gestaltet, dass sie den Nutzer unterstützen und bei Routineaufgaben entlasten.

2 Prozessangemessenheit - Das System sollte für die Erfüllung realer Arbeitsaufgaben in typischen Einsatzfeldern optimal sein. Es sollte einen Bezug zum übergeordneten realen Prozessziel haben und auf Qualifikationen und Erfahrungen der realen Benutzer abgestimmt sein.

3 Selbstbeschreibungsfähigkeit ist die einheitliche und unmittelbare Anzeige des Systemstatus. Der Benutzer sollte die Informationstiefe des Systemstatus selbst bestimmen können.

4 Steuerbarkeit beinhaltet die Kontrolle des Nutzers über den Dialog mit dem System. Sie bietet dem Nutzer die Möglichkeit verschiedene Eingabehilfen zu nutzen oder das System ohne Datenverlust zu beenden.

5 Erwartungskonformität - Die Informationsdarstellung sollte systemimmanent und bei plattformspezifischen Konzepten konsistent sein. Bei ähnlichen Aufgaben sollten Dialoge vergleichbar und an erwarteter Position dargestellt werden.

6 Fehlervermeidung - Fehlermeldungen sollten deutlich sein und Hinweise über die Art und Handlungszusammenhang enthalten. Der Nutzer muss über irreversible Handlungen informiert werden.

7 System- und Datensicherheit - Das System sollte bei fehlerhaften Eingaben des Nutzers stabil und ohne Datenverluste arbeiten. Dies gilt auch bei hoher Ressourcenbelastung.

8 Individualisierbarkeit - Das Dialogsystem sollte sich individuell an die Präferenzen der Nutzer anpassen lassen. Dies muss der Effektivität, Effizienz und Zufriedenstellung dienen. Dabei darf es nicht um Widerspruch zu notwendigeren technischen oder sicherheitsrelevanten Begrenzungen stehen.

9 Lernförderlichkeit - Lernstrategien wie „Learning by Doing“ sollten durch schrittweise Anleitung oder Navigationshilfen unterstützt werden.

10 Wahrnehmungssteuerung - Das Layout sollte minimalistisch gehalten werden. Gruppierungen, Farbgestaltung und sinnvolle Informationsreduktion sollten so verwendet werden, dass die Aufmerksamkeit des Nutzers zu relevanter Information gelenkt wird.

11 „Joy of use“ - Arbeitsabläufe und grafische Gestaltung des Systems sollen konsistent sein und dennoch Monotonie vermeiden. Zusätzlich sollten sie zeitgemäß wirken. Metaphern sollten passend und dem Nutzungskontext entsprechend verwendet werden.

12 Interkulturelle Aspekte - Das System sollte auf einen definierten Nutzerkreis und dessen funktionale, organisatorische und nationale Kultur abgestimmt sein.

3 Konzeptionierung und Umsetzung der partiellen Projektsynchronisation

Die partielle Projektsynchronisation beeinflusst große Teile des Saros Synchronisationsprozesses. Diese nötigen Änderungen werden im folgenden Kapitel näher erläutert. Zunächst werden die allgemeinen Anforderungen an die grafische Benutzerschnittstelle und die Logikimplementierungen definiert. In den folgenden Abschnitten wird genauer auf die Umsetzung der partiellen Projektsynchronisation eingegangen. Dabei soll neben dem *WAS* auch das *WARUM* erläutert werden. Bei dieser Beschreibung wird die Modifikation der Benutzerschnittstelle getrennt von der Logikimplementierung betrachtet. Zum Schluss wird die neue Benutzerinteraktion beschrieben und ein Vergleich zu Konkurrenzprodukten der partiellen Projektsynchronisation gezogen.

3.1 Anforderungen an die partielle Projektsynchronisation

Ausgelegt ist die grafische Benutzerschnittstelle, wie auch die Programmlogik, auf das Synchronisieren mehrerer vollständiger Projekte. Die durchzuführenden Änderungen haben den Anspruch, den Benutzer weder zu überfordern noch zu verwirren. Aus diesem Grund sind folgende Anforderungen gestellt.
Die Benutzerschnittstelle soll:

- die Freiheit bieten Projekte, Ordner und/oder mehrere Dateien in beliebiger Kombination zu synchronisieren
- jedem Sitzungsteilnehmer das partielle Synchronisieren zur Verfügung stellen
- die gegebenen grafischen Elemente nicht überladen oder stark verändern
- selbsterklärend und logisch sein
- die Möglichkeit bieten Projektressourcen auch während einer Sitzung zu synchronisieren

- den Sizungsstart nicht verkomplizieren
- sich in das Gesamtkonzept von Saros einpflegen

Auf Basis dieser Überlegungen werden die Änderungen an Logik und Benutzerschnittstelle hergeleitet, geprüft und umgesetzt. Da die Synchronisation in Saros eines der wichtigsten und sensibelsten Prozessbereiche ist, wird auf eine gewissenhafte Umsetzung besonders Wert gelegt. Eine sorgsame Benutzerinteraktion und Awarenessgestaltung[14] ist obligatorisch. Eine fehlerbehaftete Umsetzung kann zu Akzeptanzverlust führen. Es muss explizit darauf geachtet werden, dass:

- nur an sinnvollen und erwartungskonformen Stellen eine Benutzerinteraktion notwendig und/oder möglich ist
- die Benutzerschnittstelle allen Sitzungsteilnehmern visuelle Rückmeldung gibt
- der Benutzer ausreichend bei der Ressourcenauswahl unterstützt wird
- dem Benutzer intuitiv verständliche Freiheitsgrade in der Interaktion zugesprochen werden

Logische Fehler und Defekte im Code müssen vermieden werden, da sie im schlimmsten Fall ein Versagen des gesamten Saros Plugins verursachen.

3.2 Erweiterung der grafischen Benutzerschnittstelle zur freien Ressourcenauswahl

Das übergeordnete Ziel der grafischen Oberfläche ist es, dem Benutzer der neuen Funktionalität mehr Freiheiten und geeignete Hilfestellungen zu geben. Angefangen bei seiner Auswahlentscheidung bis hin zur visuellen Rückmeldung soll die Benutzerschnittstelle unterstützend und informativ sein. Für eine bessere Integration in die grafische Oberfläche von Eclipse wird diese zunächst betrachtet.
Danach werden die Anpassungen am Project Explorer, dem Projektauswahl-Assistenten und den Awarenessfunktionen erläutert.

[14] Awareness entspricht in diesem Zusammenhang dem Gewahrsein der Aktivitäten durch grafische Elemente. (vgl. (Plu09))

3.2.1 Analyse der vorhandenen Benutzerschnittstellen

Wesentlicher Bestandteil jeder Eclipse-Entwicklungsumgebung ist der *Project Explorer*[15]. Der *Project Explorer* ist eine so genannte *View* (*deutsch*: Sicht/Ansicht). Die View ist ein Basiskonzept der Eclipse-Arbeitsumgebung (vgl. (Ecl10)). Der *Project Explorer* ist eine der wichtigsten Navigations- und Auswahlanzeigen. Er stellt Projekte und andere Ressourcen, mit denen gearbeitet werden kann, dar. Dazu bietet er eine hierarchische Sicht (Baumstruktur) auf alle Ressourcen in der Eclipse-Arbeitsumgebung. Von hier aus werden vom Anwender Elemente der Struktur zur Bearbeitung und Interaktion ausgewählt. Abbildung 3.1 zeigt die typische Ressourcendarstellung.

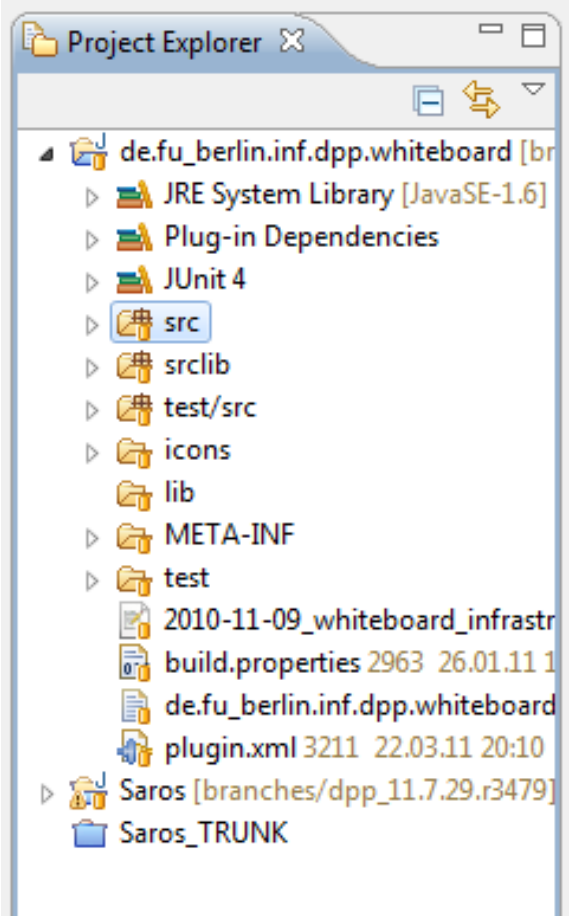


Abbildung 3.1: Ressourcendarstellung im *Project Explorer*

Dabei stützt sich der *Project Explorer* auf ein weiteres wichtiges Konzept: Die Ressource. Das Wort „Ressource“ ist ein Sammelbegriff, der alle existenten Dateien, Ordner und Projekte der Arbeitsumgebung umschreibt. (vgl. (Ecl10))

[15] Der *Project Explorer* dient der Projektrepräsentation. Alle Aussagen gelten gleichfalls für den *Navigator* und den *Package Explorer* von Eclipse.

Eclipse unterscheidet drei Basistypen von Ressourcen:

- **Files** - Dateien
- **Folders** - Ordner
- **Projects** - Projekte

Files sind die Repräsentation der Dateien im Dateisystem des darunter liegenden Betriebssystems.
Folders können Dateien oder weitere Ordner enthalten. Sie sind vergleichbar mit Verzeichnissen auf Betriebssystemebene.
Die generischste Auslegung der Ressource ist das des **Projects**. Projekte enthalten alle Dateien und Ordner und bilden so den Kontext, der ein Projekt vom Anderen trennt. Das Konzept des Projektes ermöglicht die Ressourcenkoordination und -organisation. Es vereinfacht das Versionsmanagement und den Build-Prozess. Projekte besitzen auf Dateisystemebene keine eigene Struktur. Sie werden wie Ordner als Verzeichnis abgelegt, und nur in der Eclipse-Arbeitsumgebung anders interpretiert. (vgl. (Ecl10))
An diese eclipsespezifische Ressourcenorganisation muss sich Saros im Folgenden stets orientieren. Mit diesem Vorwissen wird die grafische Oberfläche modifiziert. Änderungen der Benutzerschnittstelle sind besonders kritisch. Hier entscheidet sich, ob eine Software selbsterklärend und brauchbar für einen gewünschten Zweck ist.
Zielstellung dieser Arbeit ist die projektübergreifende Auswahl von beliebigen Ressourcen zu ermöglichen. Diese sollen anschließend mit den Sitzungsteilnehmern synchronisiert werden. Saros sieht dazu drei unterschiedliche Wege vor, wie ein Nutzer seine Ressourcen (bzw. Projekte) einer Sitzung auswählen kann, siehe dazu auch Abbildung 3.2 auf Seite 32.

- Auswahl eines oder mehrerer Projekte über das Kontextmenü des *Project Explorers* (links)
- Auswahl eines oder mehrerer Projekte aus einer Projektliste durch Anklicken einer Auswahlbox in einem Assistenten (oben rechts: Menüpunkt öffnet diesen Assistenten)
- Auswahl eines Projektes im Kontextmenu eines Buddys der Kontaktliste (unten rechts)

Die ersten beiden Interaktionsmöglichkeiten sind am besten geeignet für die Erweiterung durch partielle Projektsynchronisation. Die letzte eignet sich nur für die Auswahl von vollständigen Projekten. Dies begründet sich darin, dass die Liste der auszuwählenden Ressourcen (in dem Falle Projekte) als Kontextmenü dargestellt wird. Dies ist für eine schnelle Projektauswahl definitiv von Vorteil. Es bietet jedoch nicht die Möglichkeit einzelne Dateien oder

Ordner sinnvoll und übersichtlich darzustellen. Denn es soll besonders auf die übermäßige Verwendung von Scrollbalken oder ähnlichem verzichtet werden. Im Gegensatz dazu bieten Project Explorer wie auch der Projektauswahl-Assistent eine übersichtliche Darstellung und einfache Mehrfachauswahl von verfügbaren Ressourcen.

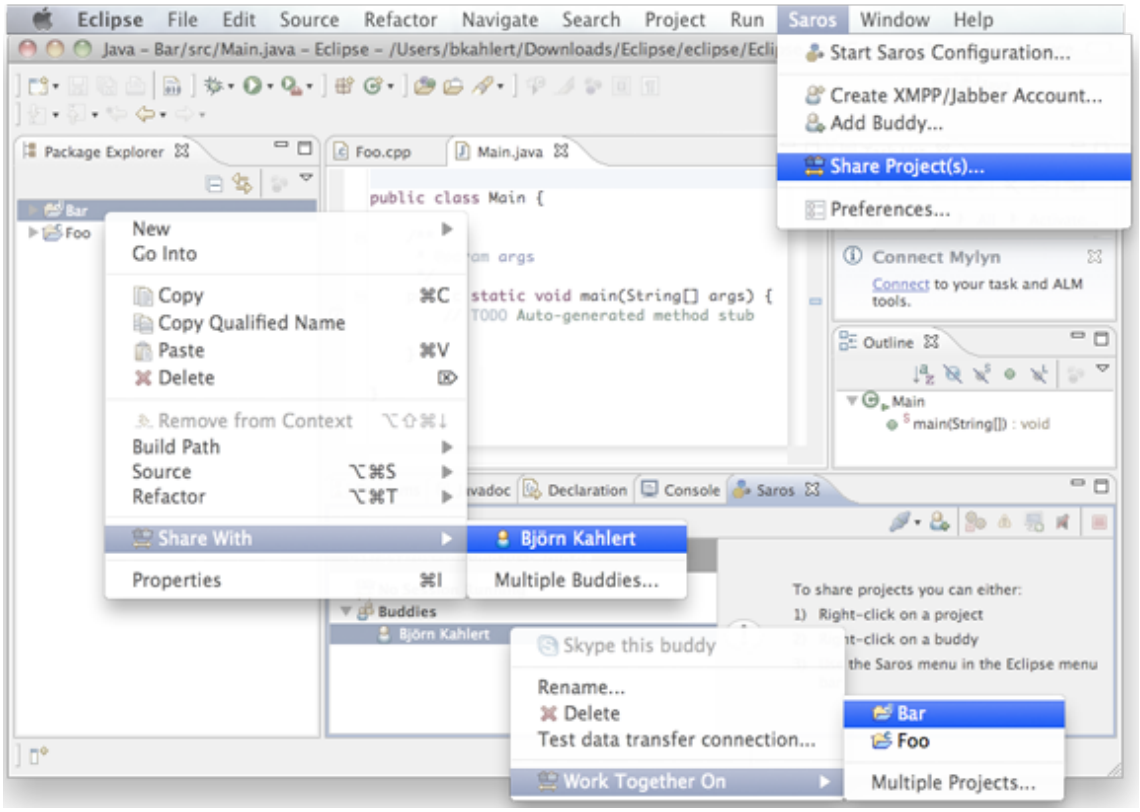


Abbildung 3.2: Drei Wege der Ressourcenauswahl (Getting Started Tutorial (Bjö11, Seite 95))

Saros ist ein Plugin für die Eclipse-Entwicklungsumgebung. Aus diesem Grund werden wesentliche Informationen zur Aktivierung und Anzeige grafischer Elemente in der *plugin.xml*-Datei festgelegt. Daraus ergibt sich, dass Menü- und Kontextmenüeinträge mit denen Saros die Oberfläche von Eclipse erweitert, in diesem XML-Manifest festgelegt werden (vgl. Abschnitt 1.3.3 auf Seite 11). Der Inhalt der XML-Datei wird durch die eclipsespezifische Plugin-Registrierungs-API integriert. Während der Eclipse-Laufzeit wird eine Instanz des Plugins auf Basis dieser Registrierungs-API erstellt. Zudem wird die API zur Gewinnung von Plugin spezifischen Informationen genutzt, wie die dynamischen Anpassung der grafischen Oberfläche. Die *plugin.xml*-Datei bietet das grafische Grundgerüst der Saros Benutzerschnittstelle und bildet somit die Basis notwendiger Modifikationen an der Eclipse-Oberfläche.

3.2.2 Anpassung des *Project Explorers* zur freien Ressourcenwahl

Eine Sitzung kann bisher nur auf Projektebene durch Auswahl im Kontextmenü gestartet werden. Der Kontextmenü-Eintrag wird gänzlich ausgeblendet, wenn kein Projekt ausgewählt ist.

Das neue Benutzungskonzept sieht vor alle Ressourcen, um einem Kontextmenüeintrag für die partielle Synchronisation zu erweitern. Aus diesem Konzept leiten sich spezielle Anforderungen ab:

- Es soll nicht möglich sein bereits geteilte Ressourcen nochmals zu synchronisieren.
- Es darf keine Vermischung von Ressourcen unterschiedlicher Anwender innerhalb eines Projektes stattfinden.
- Es darf in einer Sitzung jeder Sitzungsteilnehmer Ressourcen bereitstellen.
- Erweiterung der Awarenessgestaltung auf Ressourcen- und Projektebene.

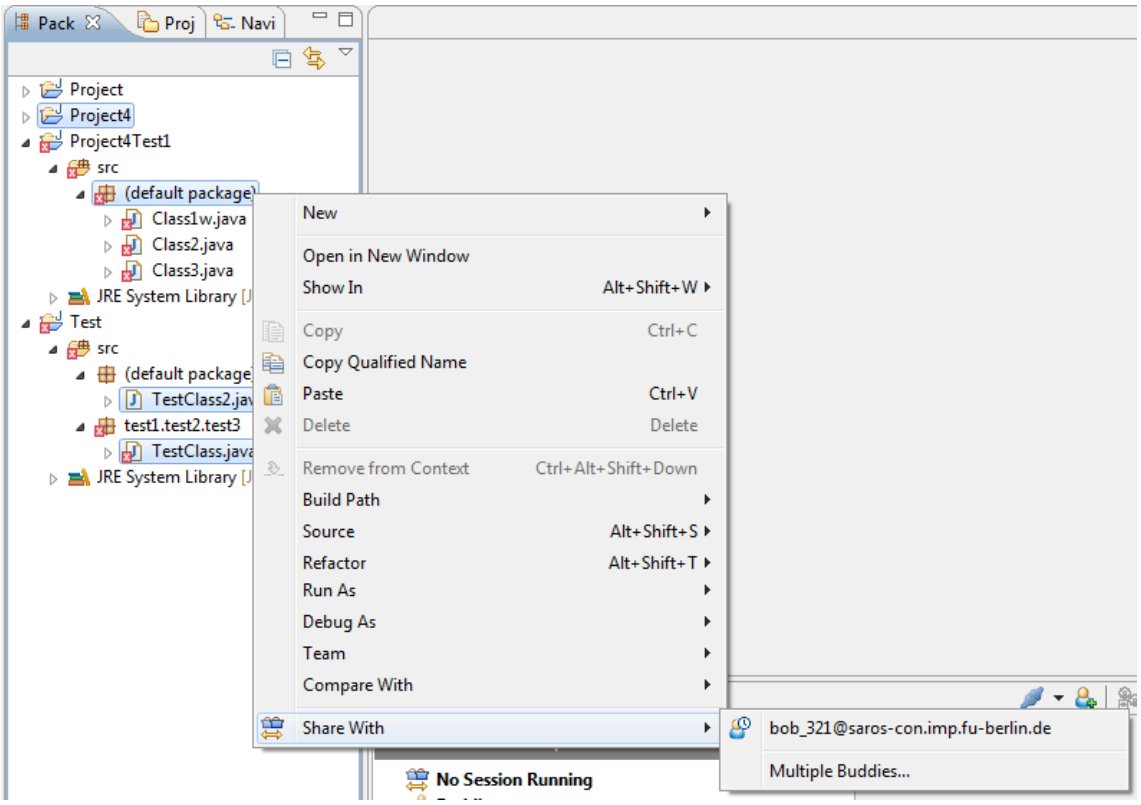


Abbildung 3.3: Partielle Projektauswahl im *Project Explorer*

Es sind Änderungen in der *plugin.xml* notwendig. Die Änderungen ermöglichen die Darstellung gewünschter (Kontext-)Menüeinträge zu bestimmten Benutzerinteraktionen mit Saros. Dadurch wird die Ressourcenauswahl-Funktion erweitert und ein Klicken auf beliebige Projektressourcen ermöglicht. Eine detailliertere Beschreibung der Änderungen befindet sich im

Anhang B auf Seite 94.

Das *Project Explorer Kontextmenü* wie auf Abbildung 3.3 auf Seite 33 bietet dem Anwender zwei Möglichkeiten. Einerseits kann er die von ihm gewählten Ressourcen direkt mit jemandem synchronisieren, indem er den entsprechenden Partner auswählt. Andererseits kann er aber auch weitere Auswahlentscheidungen treffen, indem er „*Multiple Buddies...*" wählt.

Da die partielle Projektsynchronisierung zusätzliche Logik erfordert, ist neben den Anpassungen in der Plugin-Definition eine veränderte Verarbeitung der Benutzerinteraktionen notwendig. Eine genauere Beschreibung dazu befindet sich im Abschnitt 3.3 auf Seite 36.

3.2.3 Anpassung des Projektauswahl-Assistenten zur freien Ressourcenwahl

Saros besitzt bereits einen Assistenten für die Auswahl von Projekten. Klickt der Benutzer im Menü auf „*Saros*" und dann auf „*Share Project(s)...*" wird der Projektauswahl-Assistent geöffnet. Dieser wird in dieser Arbeit benutzt und erweitert. Demnach ähnelt der *neue* Assistent dem bisher verfügbaren, besitzt jedoch mehr Funktionalität und Logik. Abbildung 3.4 zeigt die erweiterte Benutzerschnittstelle. Zu erkennen ist die Darstellung der Ressourcen der Arbeitsumgebung als Baum, wie es aus dem *Project Explorer* bekannt ist.

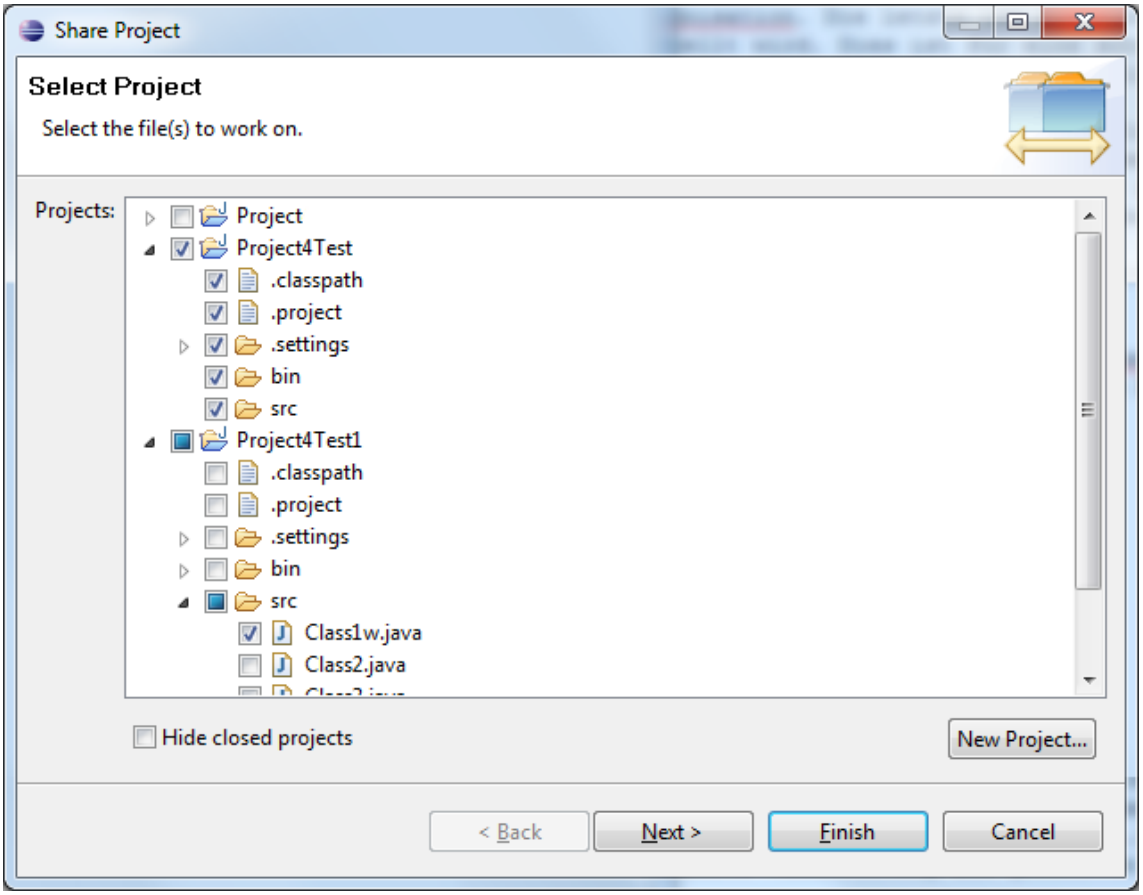


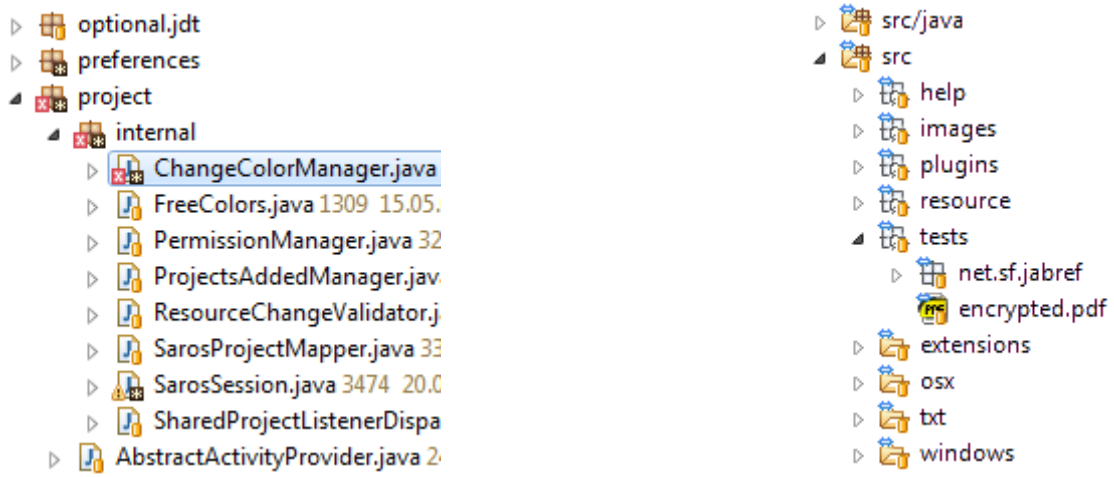
Abbildung 3.4: Partielle Projektauswahl im Projektauswahl-Assistenten

Außerdem wird jedem Element des Baumes eine Auswahlbox zugeordnet. Die Auswahlbox hat zwei Aufgaben zu erfüllen. Einerseits kann der Anwender hier die für eine Sitzung relevanten Elemente bequem auswählen. Andererseits spiegeln die Auswahlboxen den Auswahlstatus in allen Ebenen des Baumes wieder. So ist es dem Anwender sofort ersichtlich, welche Teile eines Projektes bereits ausgewählt sind. Bereits im Sitzungskontext befindliche Ressourcen werden im Auswahl-Assistent nicht mehr dargestellt. Sie werden durch einen Filter ausgeblendet.

Die partielle Projektsynchronisation ist ein tiefer Eingriff in den Prozess des Sitzungsaufbaus und der Ressourcenverwaltung. Dies soll der Benutzer jedoch nicht negativ wahrnehmen, sondern als Erweiterung seiner Möglichkeiten erfassen. Darum muss nicht nur die Benutzbarkeit gut sein, sondern auch die Awareness hilfreich und informativ. Wie diese Prämisse umgesetzt ist, wird im Folgenden genauer betrachtet.

3.2.4 Awareness-Erweiterungen

Für die Gestaltung der Benutzeroberfläche bietet Eclipse eine Vielzahl von Möglichkeiten. Einer der wichtigsten Punkte für Saros ist das Gewahrsein gemeinsam genutzter Ressourcen in einer Sitzung. Dies ist komplexer, wenn nicht nur ganze Projekte, sondern auch Teile davon in einem Sitzungskontext sein können.



(a) *Decorator* auf Ressourcen der Eclipse-Arbeitsumgebung

(b) Saros-*Decorator* auf Ressourcen der Eclipse-Arbeitsumgebung

Abbildung 3.5: Decorator der Eclipse-Arbeitsumgebung

Hierzu wurde ein Overlay-Icon, das bereits zur Projektkennzeichnung verwendet wurde, auf Ordner und Dateiebene übertragen. Dieser so genannte *Decorator* wird in den *Project Explorer* appliziert. Dort wird er auf den entsprechenden Ressourcen der Eclipse-Arbeitsumgebung platziert. Diesen *Decorator* benutzt jedoch nicht nur Saros, sondern auch Eclipse und andere

Plugins. Sie verwenden es zur Sprachendefinition, Fehler- , Informations- und Warnungsdarstellung. (siehe Abbildung 3.5 (a))
Damit es zu keinen *Decorator* Konflikten kommt, wird der Saros Ressourcen-*Decorator* oben links platziert. Dies wird in Abbildung 3.5 (b) gezeigt.

3.3 Logik-Implementierung zur angepassten Ressourcenverarbeitung

Grundsätzlich sollen nicht nur mehrere Projekte, sondern alle Projektressourcen verarbeitet und in eine Sitzung überführt werden. Daraus resultieren vielfältige Änderungen diverser Schnittstellen und Klassen, die bisher auf Objekten des Types *IProject* arbeiteten. *IProject* ist ein Eclipse-Interface[16], welches das Projekt der Arbeitsumgebung repräsentiert. Das Projekt fasst alle zugehörigen Dateien und Ordner zusammen. Mit dem Ziel der partiellen Projektverarbeitung, müssen aber auch Objekte vom Typ *IFile* und *IFolder* verarbeitet werden. Dazu wird die gesamte Logik hinter der Benutzerschnittstelle auf das Superinterface *IRessource* adaptiert. Dieses Interface kann alle Ressourcenarten handhaben. Es stellt alle Ressourcen der Arbeitsumgebung analog zum Dateisystem dar. Es gibt genau vier Ressourcenarten: **Dateiressource**, **Ordnerressource**, **Projektressource** und der **Worspace Root**. Allesamt bis auf der Worspace Root sind Analogien zu den bekannten Dateien, Ordnern und Projekten. Aufgrund dieser Adaption sind weitere Anpassungen des eigentlichen Synchronisationsprozesses notwendig.
Zu diesem Zweck sind, wie im Vorgehensmodell in Abschnitt 1.2.3 auf Seite 5 beschrieben, mehrere Iterationen entstanden. Die verschiedenen Versionen sind getestet und bewertet worden. Danach wurden sie entweder in einem Reviewprozess unterzogen oder gemessen an ihrer Geschwindigkeit verwendet oder verworfen. Im Folgenden wird auf die finale Implementierung Bezug genommen.
Zur Kommunikation zwischen Benutzerschnittstelle und Logik-Schicht wurde das Entwurfsmuster der Fabrikmethode als erzeugendes Muster verwendet (vgl. (Ria06b)).
Problemstellung ist die Erkennung und Rückgabe aller ausgewählten Ressourcen der lokalen Arbeitsumgebung. Ziel ist eine Liste aller ausgewählten Ressourcen des Nutzers zu erhalten. Diese werden dann entweder im Projektauswahl-Assistent angezeigt oder direkt weiterverarbeitet. Die genutzte generische *SelectionRetrieverFactory* findet hier alle selektierten Ele-

[16] Interface (*deutsch:* Schnittstelle) ist eine abstrakte Klassendeklaration und enthält Datenelemente und abstrakte Methoden.(Pro09)

mente des gesamten Eclipse-Fensters. Das Ergebnis wird nach den gewünschten Elementen gefiltert und zurückgegeben. Diese generische Klasse kann nicht nur selektierte Ressourcen des *Project Explorers* ausfindig machen. Sie liefert bei Bedarf auch den selektierten Buddy der eigenen Buddyliste zurück.

Die Erweiterung der Benutzerschnittstelle macht beliebige Ressourcen selektierbar. Aus diesem Grund muss die durch die *SelectionRetrieverFactory* erlangte Liste mit unsortierten Ressourcen nachverarbeitet werden. Hierfür ist nun eine explizite Zuordnung von Ressourcen zu ihren Projekten notwendig. Also wurde ein geeigneter Datentyp zur effizienten Weiterverarbeitung gewählt. Die Implementierung nutzt zu diesem Zweck eine *HashMap*. Sie ermöglicht eine praktische Schlüssel–>Wert(e) Zuordnung (*englisch:* Mapping) von einem Projekt (Schlüssel) auf eine Liste von einer oder vielen Ressourcen (Werte). Die HashMap wird wie in Codesegment 3.1 dargestellt initialisiert.

```
1 /**
2  * mapping of {@link IProject}s --> Lists of project dependent
3  * {@link IResource}s
4  */
5 HashMap<IProject, List<IResource>> resourceMapping =
6                                 new HashMap<IProject, List<IResource>>();
```

Codesegment 3.1: Erzeugung der *HashMap* zur Ressourcenzuordnung

Die selektierten Ressourcen werden so entsprechend ihrem Projekt zugeordnet. Das Projekt muss explizit ermittelt werden, wenn es nicht Bestandteil der Selektion ist. Dies entspricht der Standardannahme der partiellen Projektsynchronisation. Ist das Projekt selektiert, wird immer das gesamte Projekt synchronisiert. Dies entspricht dem Grundsatz der Ressourcenhierarchie. Demnach werden Dateien und Ordner unterhalb selektierter Projekte und Ordner als selektiert verstanden. Auf Basis dieser Zuordnung ist die Portierung der Ressourcen in den Sitzungskontext gewährleistet.

Weitere Probleme resultieren hieraus. Die Konsistenzkontrolle von Saros ist eines der Basismodule zur Interaktion. Sie wird während einer Sitzung immer wieder automatisch vom Host durchgeführt, und stellt sicher, dass alle Sitzungsteilnehmer auf dem selben Stand sind. Die Problematik ist nun, dass nicht immer das gesamte Projekt in der Sitzung verfügbar ist. Alle Ressourcen, die nicht der Sitzung hinzugefügt sind, müssen aus der Konsistenzkontrolle herausgehalten werden. Geschieht dies nicht, würde jeder Sitzungsteilnehmer mit Inkonsistenzwarnungen überhäuft werden. Die Auflösung könnte entweder zu unvorhersagbaren Verhalten oder zu einer ungewollten Dateisynchronisation führen.

3.4 Die „neue“ Benutzerinteraktion in Saros zum Sitzungsaufbau

Dieses Unterkapitel erläutert die veränderte Benutzerinteraktion beim Sitzungsaufbau. Benutzerinteraktion definiert sich im Umfeld von Saros hauptsächlich durch Dialoge und Assistenten. Auf diese muss der Nutzer von Saros reagieren, um eine Sitzung aufzubauen oder ihr beizutreten. Grundsätzlich ist vor jeder gemeinsamen verteilten Zusammenarbeit ein Sitzungsaufbau notwendig. Eine Sitzung wird durch einen Teilnehmer in mehreren Schritten initiiert.
Voraussetzung für die Zusammenarbeit mit Saros ist neben der Nutzung von Eclipse, eine aktuelle Saros-Version und eine Netzwerkanbindung. Die Netzwerkanbindung ermöglicht auf Basis des *Extensible Messaging and Presence Protocol (XMPP)* die Kommunikation mit dem Jabber-Netzwerk. Die Benutzer von Saros müssen alle mit diesem Netzwerk verbunden sein. Es stellt die gegenseitigen Interaktionen aller verbundenen Sitzungsteilnehmer bereit.
Sind diese Voraussetzungen erfüllt, kann es zur eigentlichen verteilten Zusammenarbeit kommen. Die Bereitstellung einer gemeinsamen Sitzung unterteilt sich in zwei Stufen: Einerseits in den Sitzungsaufbau an sich, andererseits in die Ressourcensynchronisation.
In den folgenden Unterkapiteln werden die im Rahmen dieser Arbeit gemachten Änderungen erläutert. Dies erfolgt zuerst mit dem Sitzungsaufbau und danach mit der Ressourcensynchronisation.

3.4.1 Der Sitzungsaufbau

Wie alle Benutzerinteraktionen soll der Sitzungsaufbau durch genügend Rückmeldung des Programmes den Anwender stets informiert halten. Nicht nur Fortschrittsinformationen bei der Verarbeitung von Daten, sondern auch Fehler sollen deutlich gemacht werden.
Für den Aufbau einer Sitzung wählt der Initiator zuerst die zu teilenden Ressourcen in seiner Arbeitsumgebung aus. Dies kann er im Ressourcenauswahl-Assistenten (vgl. Abbildung 3.4 auf Seite 34) machen, oder aber durch die Auswahl im *Project Explorer*. Ist mindestens eine Ressource ausgewählt, so kann der Initiator im nächsten Schritt beliebig viele „Buddies“ einladen. An dieser Stelle ist die gesamte Benutzerinteraktion des Sitzungsinitiators für den Sitzungsaufbau vollständig. Im Folgenden wird eine Sitzung beim Initiator gestartet und die Anderen dazu eingeladen. Es folgt die Ressourcensynchronisation. Dabei werden zunächst die Prüfsummen der Elemente berechnet und dann mit den anderen Sitzungsteilnehmern synchronisiert.

Auf Seiten des Eingeladenen erscheint ein Einladungsdialog. Dieser gibt ihm die Möglichkeit der Sitzung beizutreten oder die Einladung abzulehnen. Tritt er der Sitzung bei, schließt sich der Dialog und ein Assistent öffnet sich zeitversetzt. Dieser fordert ihn zur Auswahl eines Ablageortes der Ressourcen im Arbeitsbereich auf. Der zeitliche Versatz zwischen den beiden Benutzerinteraktion des Eingeladenen liegt an der Berechnung und Übertragung der Ressourcen-Prüfsummen des Initiators.

Bei der Umsetzung der partiellen Projektsynchronisation dieser Arbeit wurde der Ablauf des Sitzungsaufbaus zeitlich umstrukturiert. Der veränderte Ablauf wird in Abbildung 3.6 auf Seite 40 dargestellt. Dadurch wurde der Prozess schneller und die gesamte Interaktion für den Eingeladenen transparenter gestaltet. Bevor nun eine einzige Einladungsinformation an potentielle Sitzungsteilnehmer versendet wird, werden alle notwendigen Informationen vorbereitet und gesammelt. Diese Informationen sind vor allem ressourcenspezifische Daten. Sie werden berechnet und zu so genannten *ProjectExchangeInfos* zusammengefasst. Sie umfassen alles Notwendige, was zur Synchronisation und dem darauf folgenden Projektaustausch gebraucht wird. Vor allem die zeitaufwändige Prüfsummenberechnung ist Hauptfaktor eines langen Sitzungsaufbaus. Diese Berechnungen kommen ohne jegliche Informationen anderer Sitzungsteilnehmer aus.

Einerseits kann davon ausgegangen werden, dass mindestens einer der Eingeladenen die Einladung annimmt. Andererseits müssen diese Informationen problemlos durch den Initiator bereit gestellt werden können. Sollten Probleme während der zeitkritischen Informationssammlung auftreten, so kann der Sitzungsaufbau frühzeitig abgebrochen werden.

Zusammengefasst setzt sich der Sitzungsaufbau aus fünf wesentlichen Schritten zusammen:

1. **Initiator wählt Ressourcen für Sitzung aus**
2. **Initiator wählt Buddies für die Zusammenarbeit aus**
3. ***ProjectExchangeInfos* werden generiert**
4. **Sitzung wird auf Seiten des Initiators gestartet**
5. **Buddies werden eingeladen**

Der **3. Schritt** zur Generierung der *ProjectExchangeInfos* ist eine zeitliche Position nach vorne verschoben. Abbildung 3.6 auf Seite 40 stellt diese Veränderung dar. Dies beschleunigt den Sitzungsstart, da keine Berechnung während einer laufenden Sitzung gemacht wird.

Neben der zeitlichen Verkürzung der Sitzungsetablierung, wird der Synchronisationsprozess für den Benutzer informativer gestaltet. Dem Sitzungs-Initiator wird eine aussagekräftigere

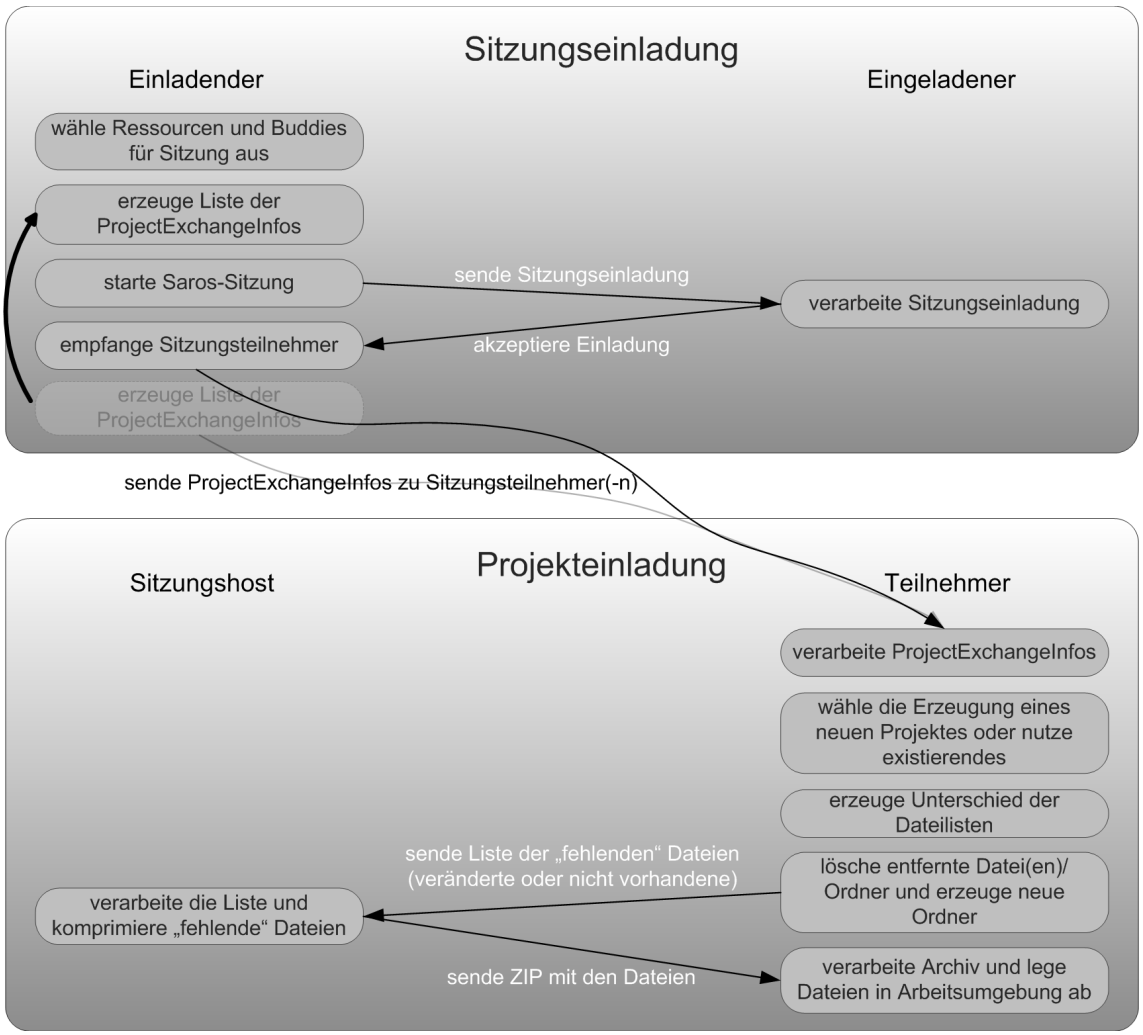


Abbildung 3.6: Zeitlicher Ablauf des neuen Einladungsprozesses

und intuitivere Rückmeldung gegeben.

Für den Sitzungsaufbau wurde auf die *Eclipse Job API* zurückgegriffen. Diese ist speziell auf die Bedürfnisse langwieriger Operationen, die in Threads abgearbeitet werden, ausgelegt. Der Fortschrittsdialog aus Abbildung 3.7 auf Seite 41 zeigt dies exemplarisch. Der Fortschritt ist speziell an die zu verarbeitenden Projekte angepasst. Der Initiator erkennt dadurch, wie viel der Bearbeitung prozentual abgeschlossen ist und was gerade geschieht. Dieser Dialog kann vom Nutzer gesteuert werden und besitzt mehrere Optionen. Ein fortgeschrittener Nutzer kann beispielsweise diesen Dialog minimiert starten. Wurden wichtige Ressourcen vergessen, kann durch einen einfachen Klick auf „*Cancel*" oder das *Stopp*-Symbol der Prozess abgebrochen werden und diese neu hinzugefügt werden.

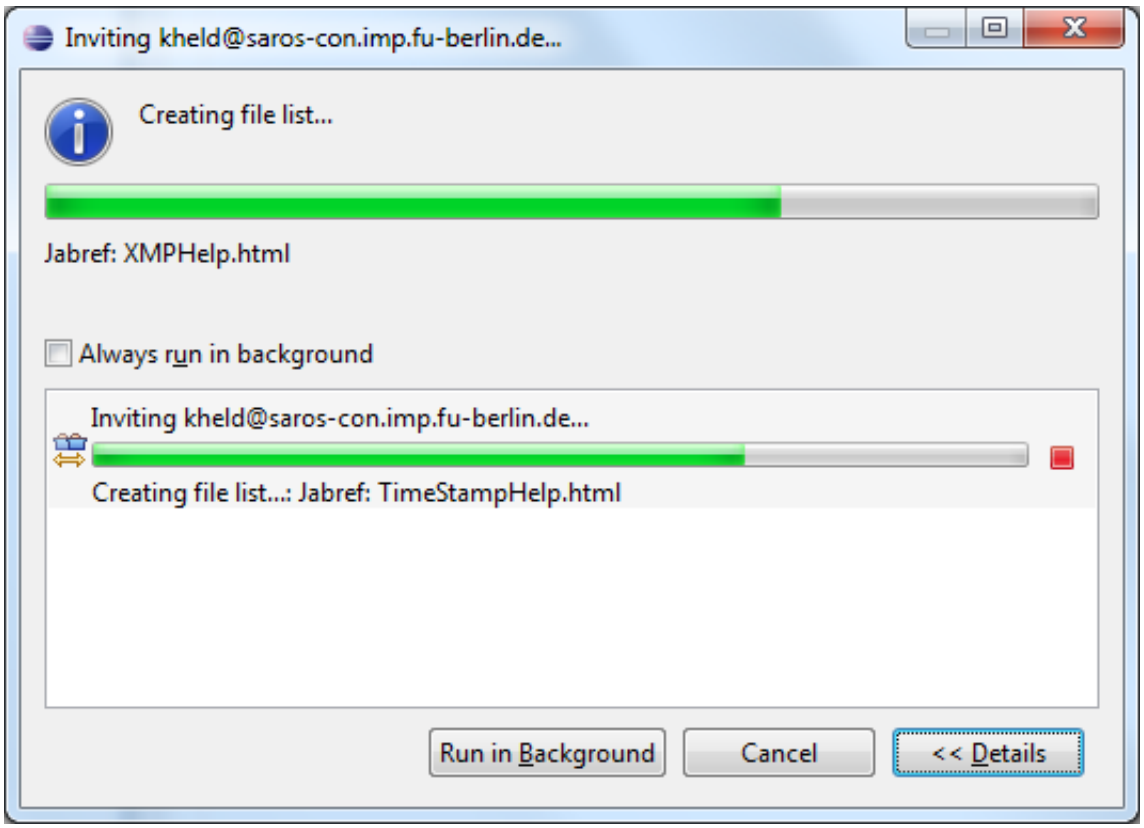


Abbildung 3.7: Fortschrittsanzeige der *Eclipse Job API*

3.4.2 Die Ressourcensynchronisation

Direkt an den Sitzungsaufbau schließt sich der Synchronisationsvorgang der Ressourcen an. Dieser hat als Grundlage die vor dem Sitzungsaufbau generierten *ProjectExchangeInfos*. Da diese nun allen sofort zur Verfügung stehen, ist die Lücke zwischen Sitzungsaufbau und Projektsynchronisation klein gehalten. Dies macht den Einladungsprozess schneller und stabiler. Um dem Benutzer durchgängig Rückmeldung zu geben, wird auch bei der Ressourcensynchronisation die *Job API* von Eclipse verwendet. In dieser Arbeit wurde eine neue Fortschrittsanzeige für alle Sitzungsteilnehmer umgesetzt. Sie beschreibt jeden Schritt der Synchronisation. Alle Interaktionspartner können sich somit bei Bedarf über den aktuellen Stand des Einladungsprozesses informieren und trotzdem agieren. Erfahrene Nutzer können hier wiederum den gesamten Dialog minimieren. So lässt sich Saros auf die Bedürfnisse der Nutzer einstellen.

3.5 Konkurrenzprodukte der partiellen Projektsynchronisation

Dieser Abschnitt fokussiert auf ein bekanntes Konkurrenzprodukt mit dem speziellen Aspekt der partiellen Projektsynchronisation. Es wird kurz beschrieben und grundsätzlich mit der Lösung in Saros verglichen. Der Hintergrund für diese Untersuchung ist eine Grundlage zur Bewertung sowie Einschätzung der neuen Saros-Funktionalität.
Es wird das *DocShare*[17]-Plugin betrachtet. Es ermöglicht die Synchronisation von kleineren Teilen eines Projektes. Das *ECF/DocShare*-Plugin gibt hierzu geöffnete Textfenster frei (Hen11). Dazu setzt *DocShare* auf dem *Eclipse Communication Framework*[18] auf, einem auf Eclipse basierendem Kommunikationsrahmenwerk. DocShare-Entwickler beschreiben es mit: „Das Hauptaugenmerk des ECF-Projekts gilt der Bereitstellung von APIs zur Interprozesskommunikation innerhalb des Java Stacks.“ (M.A09)

3.5.1 Eigenschaften von DocShare

DocShare kann beliebige Editoren eines Eclipse-Projektes mit anwesenden Benutzern einer internen Kontaktliste teilen und gemeinsam bearbeiten. Dazu öffnet der Einladende zuerst die zu teilende Textressource im eclipseinternen Texteditor. Nun kann über das Texteditor eigene Kontextmenü ein Nutzer zur gemeinsamen Bearbeitung angewählt werden. Das Dokument kann nun gemeinsam genutzt werden. Dies geht mit *DocShare* jedoch nur im 1-zu-1-Verhältnis, d.h. ein Einladender und ein Eingeladener können zusammenarbeiten. Dies entspricht dem Konzept der verteilten Paarprogrammierung (siehe Abschnitt 1.3 auf Seite 9 f.).
DocShare ist somit begrenzt auf die gemeinsame Nutzung von Textressourcen, welche einzeln angewählt werden müssen. Zusätzlich wird die Kollaboration auf Projektebene ermöglicht. Dabei muss der entsprechende Punkt im Kontextmenü eines Projektes des *Project Explorers* angewählt werden.
DosShare bietet dem Anwender auch praktische Awarenessinformationen. Sie weisen vor allem darauf hin, welcher Editor gerade mit wem geteilt wird. Dazu wird der Registerreiter (Tab) des entsprechenden Editors manipuliert. Der Benutzername und Kommunikationsserver des Partners werden dort wiedergegeben, wie in Abbildung 3.8 auf Seite 43 zu sehen.

[17] `http://wiki.eclipse.org/DocShare_Plugin`
[18] `http://www.eclipse.org/ecf/`

3.5.2 Vergleich Saros und DocShare

Die partielle Projektsynchronisation bei Saros greift diese Idee auf und erweitert sie. Vor allem im Bereich der Awareness und der Flexibilität geht Saros einen Schritt weiter. So bietet Saros nicht nur das gemeinsame Nutzen von Projekten auf ähnliche Weise wie *DocShare*. Es bietet zusätzlich die Möglichkeit beliebige Unterressourcen eines Projektes zu teilen. *DocShare* hat eine Beschränkung auf Textressourcen. Saros kann im Gegensatz beliebige Ressourcen, wie Ordner und Bibliotheken, verfügbar machen. Zudem ist dies nicht nur mit einer Person möglich, sondern mit mehreren. Alle können wiederum beliebige Ressourcen mit allen anderen teilen. Somit ergibt sich ein flexibleres Ressourcenmanagement als *DocShare* es bietet.

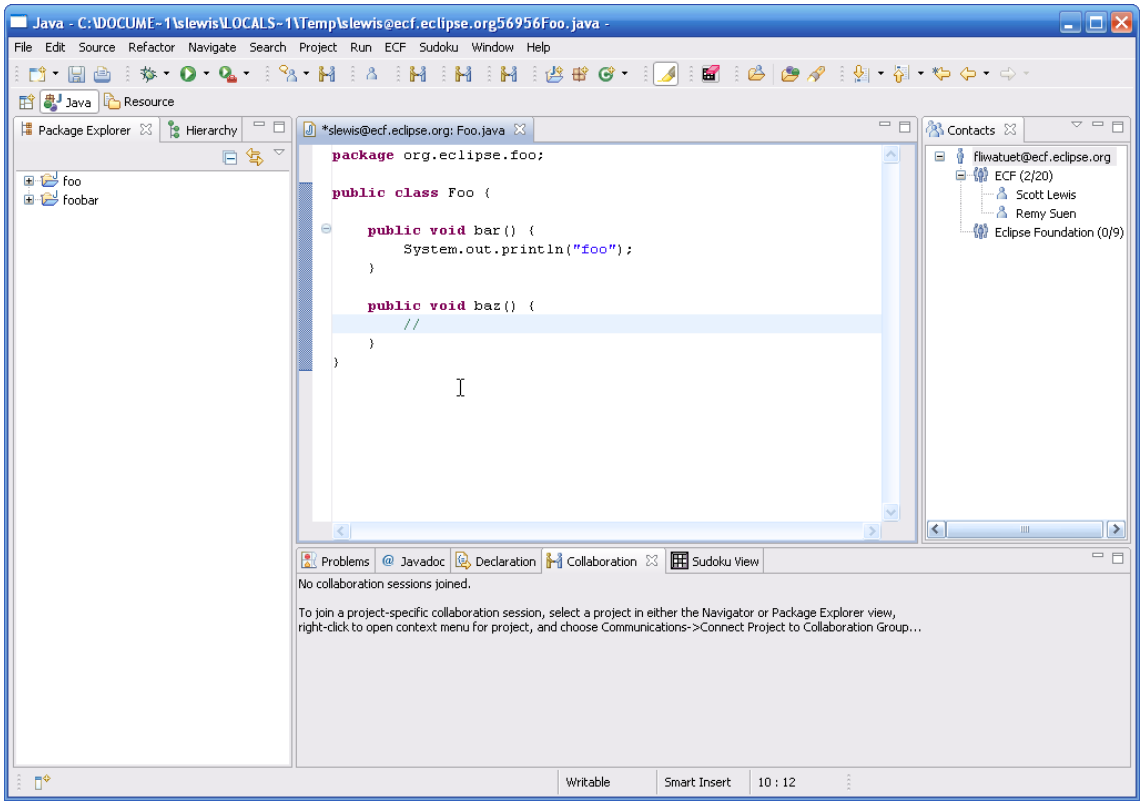


Abbildung 3.8: Ansicht DocShare Plugin (Hen11)

Ein weiterer wichtiger Punkt ist Awareness. Wie bereits erwähnt fügt *DocShare* dem Registerreiter den Namen des entsprechenden Interaktionspartners hinzu. Dies macht Saros nicht, da es in der Tableiste häufig zu abgeschnittenen Dateinamen kommt. Eine Ressourcenauswahl über die Tabs ist dadurch stark erschwert. Saros versucht dieses Problem zu umgehen, indem es entsprechende Rückmeldung auf den Ressourcen im *Project Explorer* gibt. Dazu wird jede geteilte Ressource mit einem blauen Doppelpfeil gekennzeichnet, wie in Abbildung 3.5 (b) auf Seite 35 zu sehen.

3.6 Bewertung und Einschätzung des Konzepts

Die strukturelle Vorgehensweise der partiellen Projektsynchronisation ist im Anhang C auf Seite 96 wiedergegeben. Wie in den vorhergehenden Abschnitten erläutert, ist diese Erweiterung universell gestaltet. Sie ist für jeden Sizungsteilnehmer voll nutzbar. Zudem ist die partielle Synchronisation eines Projektes dem entsprechend schneller als die Synchronisation des Gesamtprojekts. Die Vorabberechnung der *ProjectExchangeInfos* erlauben außerdem die zügigere Abarbeitung des Einladungsprozesses. Dies zeigt sich vor allem im Zeitraum zwischen Sitzungseinladungsdialog und Projekteinladungs-Assistenten. Vor der Implementierung benötigte die Einladung an dieser Stelle deutlich mehr als zehn Sekunden. Die Implementierung konnte diese Zeitspanne auf durchschnittlich unter fünf Sekunden reduzieren.

Das Senden sehr großer Projekte war bislang nicht möglich. Saros geriet dabei schnell an die Grenzen des zugesicherten Speichers (*OutOfMemoryError*) und die Synchronisation musste abgebrochen werden. Die partielle Projektsynchronisation macht auch solche Projekte nutzbar. Es können entweder nur für die Interaktion wichtige Teile übertragen werden, oder es wird das Gesamtprojekt in mehreren Schritten synchronisiert.

In beiden Fällen bietet diese neue Funktionalität eine transparente und flexible Sitzungsorganisation. Sie integriert sich dazu nahtlos in die aktuelle Saros-Oberfläche. Die Wiederverwendung von bekannten Symbolen, Assistenten und Dialogen erzeugen eine gerade Linie in Saros. Sie fördern das angestrebte Prinzip des intuitiven Erlernens („Learning by Doing“). Die Erweiterung des Synchronisationsprozesses um vielseitige visuelle Rückmeldung erzeugt einen soliden Eindruck des Plugins. Der Anwender wird bei rechen- und/oder zeit-intensiven Verarbeitungsschritten über den Fortgang informiert. Zudem bleibt ihm in jedem dieser Schritte die volle Kontrolle. Dies ist ein effektives Mittel die Akzeptanz einer Software zu bewahren oder gar zu steigern.

Eine detaillierte Evaluation dieser Funktionalität erfolgt in Kapitel 5 auf Seite 52.

4 Konzept und Umsetzung der bedarfsgerechten Dateisynchronisation

Das Konzept der bedarfsgerechten (oder auch: bedarfsorientierten) Dateiübertragung baut auf die partielle Projektsynchronisation auf. Das Hauptziel der folgenden Überlegungen und Implementierungen ist eine einfache, intuitive, schnelle und konsistente Dateiübermittlung. Die bedarfsgerechte Dateisynchronisation basiert auf der Ausgangssituation, dass Interaktionspartner häufig große Projekte in ihrer lokalen Arbeitsumgebung haben. Wollen sie nun gemeinsam an einem sehr großen Projekt arbeiten, finden aufwändige und zeitintensive Dateiübertragung bzw. Synchronisation statt. Die bedarfsgerechte Dateisynchronisation soll diese Phase der Synchronisation für verteilte Zusammenarbeit nutzbar machen.
Bedingt durch Bandbreite und vorhandene Ressourcen bei den Beteiligten benötigt der Sitzungsaufbau und die anschließende Synchronisation immer eine projektabhängige Zeitspanne. In dieser werden etwaige Unterschiede in den zu synchronisierenden Projekten festgestellt und behoben. Dieser Zeitraum kann zwischen wenigen Sekunden und mehreren Minuten umfassen. Im Folgenden werden Lösungsstrategien für einen flexibleren und schnelleren Sitzungsstart durch bedarfsgerechte Dateisynchronisation vorgestellt. Außerdem werden die Implementierung und die nötigen Änderungen an Saros beschrieben.

4.1 Ausgangssituation und Anforderungen

Die Idee ist eine bedarfsorientierte Dateiübertragung zu implementieren, um dann eine schnellere Interaktion zu ermöglichen. In diesem Zusammenhang ist bedarfsorientiert bzw. bedarfsgerecht als „*nur wenn nötig*“ zu verstehen. Dies bedeutet, dass Dateien *nur* wenn sie gebraucht werden übertragen werden. Fraglich ist, in welchen Situationen diese Notwendigkeit besteht.
Sitzungsteilnehmer können nur gemeinsam an Dateien eines Projektes arbeiten, wenn diese bei allen identisch sind. „Arbeiten“ kann im Kontext von Saros als das Editieren in einer Datei definiert werden. Beim Editieren verliert diese Datei ihre potentielle Synchronität zu

denen der Anderen. Also muss diese Datei übertragen werden, wenn an dieser gearbeitet wird. Zusätzlich muss darauf geachtet werden, dass nur beabsichtigte Dateien übertragen werden.
Dieses Grundkonzept bedarf jedoch einer Fallunterscheidung, zwischen einer partiellen und einer vollständigen Projektsynchronisation. Diese wird im Folgenden erläutert.

4.1.1 Anwendung bei der partiellen Projektsynchronisation

Die partielle Projektsynchronisation hat den Vorteil, dass vorher ausgewählte Ressourcen eines Projektes in der Sitzung verwendet werden. In den meisten Fällen wird es sich hierbei um wenige einzelne Dateien handeln. Dies hat den Vorteil, dass der Sitzungsaufbau sehr schnell ist. Die Frage ist nun: Wie kann die bedarfsorientierte Dateiübertragung hier Vorteile bringen, wenn die Sitzung bereits schnell aufgebaut wird? Der Fokus ist in diesem Zusammenhang auf die nicht synchronisierten Ressourcen eines partiell verfügbaren Projektes. Während eines Sitzungsverlaufes ist es üblich, dass Dateien benötigt werden, an die eingangs nicht gedacht wurde.
In dieser Situation gibt es drei Möglichkeiten:

1. Ressource wird nicht in den Sitzungskontext übernommen
2. Datei wird explizit über den Projektauswahl-Assistenten zur bestehenden Sitzung hinzugefügt
3. Bedarfsgerechte Dateisynchronisation übernimmt die Aufgabe des zweiten Punktes. Sie fügt die Datei bei der ersten Veränderung automatisch dem Sitzungskontext zu.

Alle drei Möglichkeiten haben Vor- und Nachteile. Die **erste Möglichkeit** ist der am wenigsten anzustrebende Zustand bei einer Zusammenarbeit. Mögliche Zusammenhänge für das Verständnis und die weitere Interaktion können verloren gehen und ein Zusammenarbeiten unmöglich machen.
Bei **Variante zwei** kann die Ressource gemeinsam genutzt werden, da eine explizite Übertragung in den Sitzungskontext vorgenommen wird. Hier ist die Schwierigkeit, dass dies manuell geschieht. Dies kann unter Umständen eine längere Zeit der Arbeitsunterbrechung zur Folge haben. Der gesamte Prozess zwingt alle Sitzungsteilnehmer zu einer erneuten Interaktion über Assistenten. Zudem kommt die notwendige Disziplin des Verantwortlichen diese explizit zu übertragen.
Die **dritte Variante** verwendet eine automatische, bedarfsgerechte Dateisynchronisation. Sie macht aufwändige Interaktionen mit Assistenten aller Sitzungsteilnehmer unnötig und

verhindert fehlende Ressourcen im Sitzungskontext. Ziel ist hierbei eine selbstständige Übertragung der Ressource, falls sie benötigt wird. Alle Sitzungsteilnehmer sollen jedoch hierüber informiert werden.

4.1.2 Anwendung bei vollständiger Projektsynchronisation

Die bedarfsorientierte Dateisynchronisation soll nicht nur bei der partiellen Projektsynchronisation verwendet werden, sondern auch bei vollständiger Projektsynchronisation. In diesem Zusammenhang werden die Dateien bedarfsgerecht übertragen, weil der Sitzungsstart großer Projekte lange dauern kann. Dieser Zeitraum macht eine schnelle Zusammenarbeit unmöglich.
Die Übertragungszeit soll nutzbar gemacht werden, um nicht auf das Ende der Synchronisierung warten zu müssen. Das Konzept beachtet dabei Ressourcen, die während des Synchronisierungsvorgangs durch den Ressourcenhost modifiziert werden. Dies hat den wesentlichen Vorteil, dass die Synchronisierung bei allen Sitzungsteilnehmern beliebig lange im Hintergrund laufen kann. Momentan notwendige Dateien werden sofort übertragen. Die Verwendung der Wartezeit birgt neben den offensichtlichen Vorteilen auch Probleme. Diese müssen bei der Umsetzung gelöst werden. Beispielsweise muss die Konsistenzprüfung stets funktionieren. Hierfür müssen *AktivitätsObjekte* während der Projekt-Synchronisierung übermittelt werden. Dies ist in Saros nicht vorgesehen und unterstützt. Aber es ist unbedingt notwendig, um eine frühzeitige Interaktion zu ermöglichen.

4.2 Konzept zur Umsetzung

Die Umsetzung bedarf vieler Änderungen im Aktivitätsmanagement. Dies ist wiederum ein sehr sensibeles und wichtiges Modul von Saros. Die Übertragung aller Aktionen einer Arbeitsumgebung zu allen anderen Sitzungsteilnehmern wird durch diese gesteuert. Dazu werden so genannte *AktivitätsObjekte* verschickt. Es gibt sie in unterschiedlichsten Ausführungen. (siehe Anhang D auf Seite 97)
Ein solches *AktivitätsObjekt* benötigt die bedarfsgerechte Dateisynchronisation, um Dateien zu übertragen und so die frühzeitige Interaktion zu ermöglichen. Problematisch ist, dass das Senden von *AktivitätsObjekten* erst nach Beendigung der Synchronisation erfolgen sollte. Der Grund dafür ist, dass sich eine frühere Version der Datei noch im eigentlichen Synchonisationsvorgang des Gesamtprojektes befindet. Ein erneutes Überschreiben könnte zu ungewollten Nebeneffekten wie Inkonsistenzen führen. Diese Problemstellung gilt es prag-

matisch und flexibel zu lösen.
Der erste Schritt der Umsetzung versendet *AktivitätsObjekten* der editierten Dateien vor Synchronisationsende. Die Datei befindet sich nach der bedarfsgerechten Übertragung im normalen Sitzungskontext. Daher ist ein Abgleich mit gerade gemeinsam genutzten Ressourcen ausreichend, um ein Überschreiben der bereits geteilten Datei zu vermeiden. Nun ist es notwendig auf die Aktualität der Ressourcenliste gemeinsam genutzter Dateien zu achten. Zu diesem Zweck wurden Funktionen integriert, die es komfortabel erlauben das Projekt-zu-Ressourcen-Mapping ständig zu aktualisieren. Beim Initiator reicht die Überführung der gerade selektierten Datei in den Sitzungskontext aus. Alle anderen müssen auf eine Benachrichtigung dieses Ressourcenhosts warten, der die Datei publiziert.
Da die Konsistenzwiederherstellung von Saros ähnlich funktioniert, wurde diese adaptiert. Inkonsistenzen werden durch Unterschiede auf Basis von Prüfsummen zwischen Sitzungsteilnehmer und Host-Datenbestand ermittelt. Entstandene Inkonsistenzen kann der Saros-Nutzer durch einen Klick auf die entsprechende Schaltfläche lösen. Die Datei des Ressourcenhosts überschreibt dann die inkonsistent gemeldete Datei des Sitzungsteilnehmers. Dieser Übermittlungsprozess wurde dahingehend übernommen, dass ein *AktivitätsObjekt* generiert und dann an alle versendet wird. In dessen Inhalt befindet sich die konkrete Operation zum Überschreiben der gemeinsam genutzten Datei. Zur Kontrolle der Übertragung wird auf Basis von Prüfsummen die Korrektheit und Vollständigkeit sichergestellt.
Vorteile des Konzepts ist die schnelle Bereitstellung der Dateien und die damit einhergehende Möglichkeit der Zusammenarbeit.

4.3 Neues Benutzerrollenkonzept - Ressourcenhost

Die Rolle des Hosts in Saros ist mit der bedarfsgerechten Dateisynchronisation nicht mehr eindeutig definiert. Einst initiierte der Host die Sitzung und seine Rolle im Sitzungskontext war festgelegt. Unter anderem darf der Host Lese- und Schreibrechte vergeben, die Sitzung vollständig beenden oder Projekte der Sitzung hinzufügen. Vor allem im Bezug auf die Projektsynchronisation haben sich die Möglichkeiten grundlegend geändert. Dies führt zu einer grundsätzlichen Überlegung und Hinterfragung der Hostrolle.
Das Grundkonzept von Saros sieht vor, dass eine Person die Sitzung startet und dazu einlädt. Aufgrund dessen erhält diese Person die Verantwortung über die Sitzung und so auch die Rolle des **Sitzungshosts**. Diese Rechte sind exklusiv und können nicht übertragen werden. Die Eclipse Instanz des Sitzungshost führt außerdem die Konsistenzkontrolle für alle Sitzungsteilnehmer automatisiert durch. Dazu ist der Sitzungshost stets die zentrale Instanz

der Ressourcen. Alle Inkonsistenzen werden entsprechend der Ressourcen seiner Arbeitsumgebung detektiert und aufgelöst.

Unter der Voraussetzung, dass der Sitzungshost die Rechte nicht beschränkt hat, sind alle anderen Sitzungsteilnehmer gleichberechtigt. Jeder Teilnehmer kann grundsätzlich die Sitzung verlassen und beliebige Ressourcen erstellen, verschieben und löschen. Entsprechende *AktivitätsObjekte* werden erzeugt und alle Sitzungsteilnehmer schnellstmöglich informiert. Somit wird beispielsweise jede gelöschte Ressource bei jedem Sitzungsteilnehmer gleichermaßen gelöscht. Die Sitzung bleibt so bei allen Parteien konsistent und stabil.

Über diese „Grundrechte“ hinweg, kann jeder Sitzungsteilnehmer, beliebige Ressourcen in den Sitzungskontext überführen. Dementsprechend kann jeder (partielle) Projekte der Sitzung hinzufügen. Genau an dieser Stelle beginnt nun die Rolle des Hosts ungenau zu werden. Da jeder Ressourcen hinzufügen kann, entsteht implizit die Rolle des **Ressourcenhosts** für die von ihm bereitgestellten (Teil-) Projekte. Dies spielt vor allem bei der Umsetzung der bedarfsgerechten Dateisynchronisation eine wichtige Rolle. Hier ist eine Zuordnung der „Quelle“ aller gemeinsam genutzten Ressourcen notwendig.

Deshalb wird die Rolle des Ressourcenhosts eingeführt, um dessen Projekt als Grundlage zu nehmen. Der Ressourcenhost ist für die Ressourcen des gesamten Projektes verantwortlich. Selbst wenn er nur eine Datei des Projekts in den Sitzungskontext eingefügt hat, muss er die restlichen Ressourcen des gesamten Projektes übertragen. Somit ist auch eine Synchronisierung unterschiedlicher partieller Projekte in ein und dasselbe Projekt nicht möglich. Unterschiedliche Ressourcenhosts für ein und dasselbe Projekt werden vermieden. Projekte bleiben aus diesem Grund ausführbar und überschaubar.

4.4 Zusätzliche Awarenessfunktionen

Grundsätzlich steht es jedem Benutzer frei, das bedarfsgerechte Synchronisieren zu aktivieren. Dies lässt sich in den Saros-Einstellungen vornehmen. Der Benutzer wird außerdem bei erstmaliger Benutzung in einer Sitzung informiert, dass bei Bedarf Dateien sofort in den Sitzungskontext übernommen werden könnten. Ein Abfragedialog wie in Abbildung 4.1 auf Seite 50 holt die Erlaubnis dazu ein, oder aber deaktiviert die Funktionalität für diese Sitzung. Dadurch wird das Bewusstsein geweckt, dass die Funktion aktiviert ist.

Bedarfsgerechte Freigaben sind erwartungsgemäß nur auf zu übertragende Ressourcen, oder partiell freigegebenen Projekten erlaubt. Demnach besteht nicht die Gefahr projektfremde Dateien freizugeben.

Für die Übertragung der Datei an sich wird ein Fortschrittsdialog angezeigt. Dieser ist in der

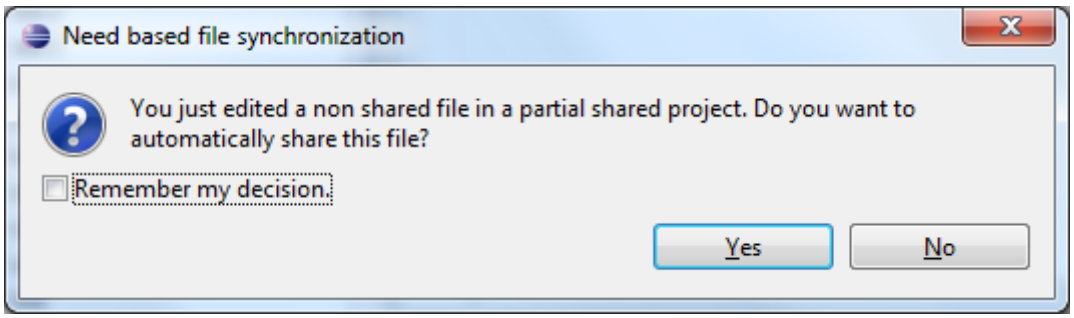


Abbildung 4.1: Aktivierungsdialog der bedarfsgerechten Dateisynchronisation

Regel schnell beendet und spiegelt dem Benutzer den Übertragungsvorgang wider. Während des Übertragungsprozesses werden die Editoren für Eingaben gesperrt, um Inkonsistenzen vorzubeugen. Folglich steht die Datei im Sitzungskontext zur Verfügung und kann gemeinsam genutzt werden. Durch die Implementierung der partiellen Projektsynchronisation wird auf solchen Dateien das Overlay-Icon im *Project Explorer* angezeigt. Dies zeigt allen Anwendern sofort welche neue Datei gemeinsam nutzbar ist. Darüber hinaus wird jeder Sitzungsteilnehmer, der nicht Initiator der bedarfsgerechten Synchronisierung ist, durch eine *BalloonMessage* benachrichtigt. Dies ist exemplarisch in Abbildung 4.2 dargestellt.

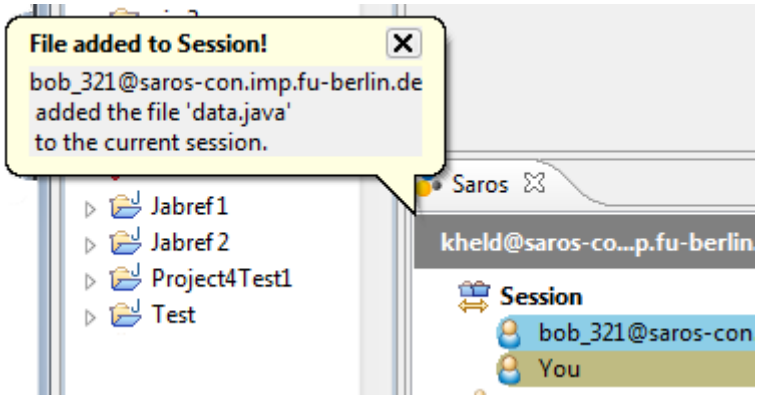


Abbildung 4.2: *BalloonMessage* zur Benachrichtigung der Sitzungsteilnehmer

4.5 Bewertung und Einschätzung des Konzepts

Das gesamte Konzept der bedarfsgerechten Ressourcenfreigabe ist neu und unerprobt. Somit ist fraglich, wie die neue Funktionalität angenommen wird und sich fortentwickelt. Die angestrebten und auch umgesetzten Vorteile dieser Erweiterung sind Zeitersparnis und Komforterhöhung. Der Benutzer wird ausreichend über die Synchronisierungsschritte informiert. Er verliert nie die Kontrolle oder Übersicht über die Sitzung. Der eigentlichen Akzeptanz steht somit nichts im Wege.

Da bei allen Änderungen der Code wartbar und lesbar gehalten wurde, ist eine Weiterentwicklung möglich. Bis dahin kann diese Funktionalität als praktische Alternative zur bekannten vollständigen Ressourcenfreigabe genutzt werden.

5 Evaluation

Die Evaluation findet mit den in Kapitel 2 ab Seite 16 erläuterten Methoden statt.
Um die Qualität der Software gegen die Norm ISO/IEC 9126 zu erfassen wird zuerst die GQM verwendet. Dieser Ansatz bezieht sich auf Funktionalität und Implementierung.
Danach wird die heuristische Evaluation zur Bewertung der Usability benutzt. Abschließend werden die implementierten Funktionalitäten zusammenfassend bewertet.

5.1 Anwendung der Goal-Question-Metric-Methode

Die GQM wird nach dem im Abschnitt 2.1.2 auf Seite 22 beschriebenen Ansatz durchgeführt. Schritt drei bis sechs werden in einem Schritt zusammengefasst, um die Übersichtlichkeit zu bewahren. Dabei ist eine quantitative Untersuchung nicht immer möglich. Zum Beispiel ist die Reife einer Software schlecht zählbar.
Die Vorgehensweise ist hier, dass zunächst ein Ziel definiert wird, danach Fragen formuliert und eine Metrik festgelegt wird.
Jedes betrachtete Ziel wird (wenn möglich) gegen die partielle Projektsynchronisation und gegen die bedarfsorientierte Dateisynchronisation analysiert.
Jedes identifizierte Ziel der ISO/IEC 9126 Norm wird im folgenden Unterkapitel analysiert.

5.1.1 Analyse im Bezug auf Richtigkeit

Ziel: Analysiere die Funktionalität zum Zwecke der Auswertung in Bezug auf **Richtigkeit** vom Blickwinkel des Benutzers im Kontext von Saros.

Fragen: Wurde die Richtigkeit durch die Integration von partieller Projektsynchronisation verändert?
Werden Projekte, Dateien und Ordner vollständig und wiederholbar korrekt übertragen?
Wird in jedem Fall die Konsistenz richtig hergestellt und Inkonsistenzen bemerkt?

Metrik: Identifiziertes Messziel ist die reproduzierbare Synchronität gewählter Ressourcen nach der partiellen Projektsynchronisation. Um dieses Messziel zu erreichen, werden manuell verschiedene Synchronisierungsvorgänge getestet. Dabei wird die Übertragung mit einer einzelnen Ressource/Datei bis zu mehreren vollständigen Projekten durchgeführt. Die Evaluation basiert auf der abgeschlossenen Synchronisierung. Außerdem wird die Funktionalität und Richtigkeit durch das Saros Test Framework (STF)[19] geprüft.

Metrik partielle Projektsynchronisation: Die partielle Projektsynchronisation wird nach einem festgelegten Schema validiert. Dieses Schema spiegelt mögliche Permutationen der partiellen Benutzung wider. Es werden alle Ressourcen der vertikalen Achse mit denen der horizontalen jeweils gemeinsam übertragen. Falls es zu einer Kombination von Ressourcen mit einem Projekt kommt, wird keine projekteigene Ressource gewählt. Es wird eine dem zu synchronisierenden Gesamtprojekt fremde Ressource verwendet. Dies hat den Hintergrund, dass ein selektiertes Projekt in jedem Fall vollständig synchronisiert wird, egal wie viele Unterressourcen noch selektiert sein sollten. Zur Überprüfung wurde das Schema aus Tabelle 5.1 entwickelt und verwendet.

Richtigkeit	Datei	Ordner	Projekt	Projekt + Ordner	Projekt + Datei
Datei					
Ordner					
Projekt					

Tabelle 5.1: Untersuchsungsschema zur Feststellung der **Richtigkeit**

Wie zu erkennen ist, wird einerseits die Synchronisation einzelner Dateien bis hin zu

[19] ehem. Sandors Test Framework

mehreren Projekten getestet. Das gesamte Testszenario wird auf Basis realer, aktiv entwickelter, mittelgroßer Projekte (z.B. Saros Trunk oder JabRef[20]) durchgeführt, um Alltagstauglichkeit zu gewährleisten. Dazu werden diese Projekte ausgecheckt und vom SVN getrennt. Dies soll eine vom Repository gestützte Synchronisation unterbinden und den zu untersuchenden internen Synchronisierungsprozess von Saros anstoßen. Zudem wird das Synchronisieren zwischen zwei Sitzungsteilnehmern bis zu vier Sitzungsteilnehmern jeweils wiederholt, dies ist nicht weiter aufgeschlüsselt. Des Weiteren wird die gesamte Prozedur einmal mit einer neuen Sitzung und einmal mit einer bestehenden Sitzung wiederholt. Zu Testzwecken wird eine lokale Eclipse-Instanz, welche es erlaubt bis zu drei Sitzungsteilnehmer zu erzeugen, und zwei getrennte Rechner für den Versuch herangezogen. Die gewonnenen Ergebnisse werden in das Schema übertragen, wobei alle positiv ausgefallenen Tests (*grün*) nicht genauer kommentiert werden.

Richtigkeit	Datei	Ordner	Projekt	Projekt + Ordner	Projekt + Datei
Datei	**OK**	**OK**	**OK**	**OK**	**OK**
Ordner		**OK**	**OK**	**OK**	**OK**
Projekt			**< X^* MB**	**< X^* MB**	**< X^* MB**

***** abhängig vom Java Heap Space

Tabelle 5.2: Auswertung zur Feststellung der **Richtigkeit**

Das Ergebnisschema in Tabelle 5.2 gibt ein eindeutiges Bild, wenn es um die Synchronisierung vergleichsweise leichtgewichtiger Ressourcenkombinationen geht. Dateien und Ordner jeweils in Kombination mit beliebigen anderen Ressourcen sind unkritisch zu übertragen. Sie sind schnell und zuverlässig bei allen Teilnehmern nach der Synchronisierung vorhanden. Eine Problematik die sich zeigt, ergibt sich vor allem in den *gelb* gekennzeichneten Flächen. Kommt es zu einer Synchronisierung mehrerer Projekte, welche jeweils mehrere hundert Megabyte groß sind, wird die Synchronisierung träge. Dies ist nicht in der partiellen Projektsynchronisation begründet, sondern dass große und viele Dateien generell einen hohen Verarbeitungsaufwand für Saros bedeuten. Im schlimmsten Fall resultiert dies in einem *Java Heapspace Error*. Weniger kritisch, daher sind diese Felder gelb gekennzeichnet, ist die Situation, wenn Projekte bei allen Sitzungsteilnehmern entweder teilweise, oder aber ganz vorliegen. Entsprechende Übertragungsoperationen entfallen in diesem Fall.

[20] JabRef ist ein Open Source Literaturverzeichnis Referenz Manager. siehe: `http://jabref.sourceforge.net/`

Metrik bedarfsgerechte Dateisynchronisation: Im Anbetracht der Richtigkeit muss die bedarfsgerechte Dateisynchronisation anders betrachtet werden als die partielle Projektsynchronisation. Da die bedarfsgerechte Dateisynchronisation auf reinen Dateioperationen basiert, müssen auch nur solche überprüft werden. Grundsätzliche Idee der Messung ist es, während der Übertragung großer Projekte, einzelne Dateien bei Bedarf zu übertragen. Dann soll vor Synchronisierungsende die Konsistenz analysiert werden. Zu diesem Zweck wird die unterste Zeile des Schemas nochmals ausgeführt. Unterschied ist, dass nur Unterressourcen der zu synchronisierenden Projekte übertragen werden. Die Testbedingungen bleiben sonst gleich.

Das Ergebnis dieser Messung ist, dass alle zu übertragenden Dateien ihren Bestimmungsort erreichten und konsistent waren. Zudem konnte vor Synchronisierungsende an allen übertragenen Dateien gemeinsam gearbeitet werden.

Der zweite Aspekt der bedarfsorientierten Dateiübertragung ist die Möglichkeit Ressourcen nachzuladen. Also können während einer Sitzung Dateien, die in einem partiell verfügbaren Projekt editiert werden, dem Sitzungskontext hinzugefügt werden. Es handelt sich hier um reine Dateioperationen, welche durch den Konsistenzmanager von Saros oder das gemeinsame Editieren validiert werden. Ergebnis der Messung ist, dass die vom Ressourcenhost angestoßenen bedarfsgerechten Synchronisierungen, bei allen gemachten Versuchen erfolgreich verliefen.

Analyse und Interpretation: Die gewonnenen Ergebnisse zeichnen ein eindeutiges Bild. Das partielle, wie auch das bedarfsgerechte Synchronisieren arbeiten richtig. Dieses Ergebnis wird gleichermaßen von den STF-Tests repliziert. Probleme bei zu großen zu übertragenden Dateien bzw. Projekten sind nicht auf die entwickelten Funktionalitäten zurückzuführen. Vielmehr ermöglichen sie einen stabileren und vor allem schnelleren Synchronisationsvorgang.

5.1.2 Analyse im Bezug auf Sicherheit

Ziel: Analysiere die Funktionalität zum Zwecke der Auswertung in Bezug auf **Sicherheit** vom Blickwinkel des Benutzers im Kontext von Saros.

Fragen: Ist die Sicherheit zu jedem Zeitpunkt gegeben?
Werden Projekte, Dateien und Ordner vor unberechtigtem Zugriff geschützt?

Metrik: Identifiziertes Messziel ist die strikte Trennung gemeinsam genutzter Ressourcen zu lokal verfügbaren Ressourcen. Dabei wird wiederum das bedarfsgerechte und die partielle Dateisynchronisation betrachtet. Das genutzte Schema, ähnelt der vorhergehenden Messung der Richtigkeit. Die entsprechenden Personen in der horizontalen zeigen an, wer an der Sitzung teilnimmt. Somit sollten gemeinsam genutzte Dateien nur diese Personen erreichen. Erfolgte eine Übertragung partieller und ganzer Projekte, nur mit angewählten Ressourcen, wurde die Zelle grün markiert. Während des gesamten Testzyklus wird versucht die Sicherheit auszuhebeln. Somit wird beispielsweise außerhalb gemeinsamer Ressourcen editiert, oder aber Dateien und Ordner in partiell genutzte Projekte kopiert oder verschoben. Die Auswertung entspricht dem Schema aus Tabelle 5.3.

Sicherheit	Person A	Person B	Person A + B
Datei	**OK**	**OK**	**OK**
Ordner	**OK**	**OK**	**OK**
Projekt	**OK**	**OK**	**OK**

Tabelle 5.3: Auswertung zur Feststellung der **Sicherheit**

In keinem der Fälle konnten Sicherheitsmängel festgestellt werden. Dies resultiert aus einem im Quelltext verankertem Prinzip. Nur explizit angewählte Ressourcen und deren Unterressourcen (im Falle von Projekten und Ordnern) können geteilt werden. Nicht mehr geteilte Ressourcen (bspw. durch Löschung) werden aus der Menge geteilter Ressourcen entnommen. Somit ist auch ein Einschleusen einer gleichnamigen Ressource an Stelle einer gelöschten nicht möglich.

Analyse und Interpretation: Die Sicherheit kann bei beiden Erweiterungen als gewährleistet gesehen werden. Es können nur explizit freigegebene Dateien, Ordner und Projekte im Sitzungskontext vorkommen. Selbiges untermauert wiederum das STF.

5.1.3 Analyse im Bezug auf Reife

Ziel: Analysiere die Zuverlässigkeit zum Zwecke der Auswertung in Bezug auf **Reife** vom Blickwinkel des Benutzers im Kontext von Saros.

Fragen: Sind Fehlerzustände bei der Benutzung der Funktion forcierbar?
Bleibt die Funktionalität nach einem Fehlerzustand vorhanden?
Führt ein Fehlerzustand bei der partiellen Projektsynchronisation zu einem vollständigen Versagen?
Ist Saros durch die neuen Erweiterungen fehleranfälliger/instabiler als zuvor?

Metrik: Identifiziertes Messziel ist die Anzahl der Fehlzustände und deren Konsequenzen bei einer exzessiven Nutzung der neuen Funktionalitäten zu überprüfen. Dazu wird ein fehleranfälliges Szenario gewählt:
Es wird versucht Übertragungsfehler zu provozieren indem während einer Projekteinladung Dateien editiert werden.
Dies ist problematisch, da der Datenbestand des Hosts von denen der anderen Sitzungsteilnehmer abweicht. Im Normalfall wird sofort nach dem Projektaustausch eine Inkonsistenz detektiert. Dies liegt daran, dass zu einem frühen Zeitpunkt der Zustand einer Datei als gegeben angesehen wird. Da auf Basis dessen die Prüfsummen der zu teilenden Ressourcen generiert werden, kann von diesem Vorgehen nicht abgewichen werden.
Ein möglicher Lösungsansatz ist das Sperren der Editoren der Eclipse Arbeitsumgebung. So können keine Dateien während des Sitzungsstarts verändert werden. Eine andere Lösung ist die bedarfsgerechte Dateisynchronisation zu aktivieren. Diese unterbindet die Gefahr von Inkonsistenzen. Denn während eines Projektaustausches werden die editierten Dateien bevorzugt übertragen und der Sitzung hinzugefügt.

Analyse und Interpretation: Keiner der provozierten Fehlzustände führt zu schwerwiegenden Problemen. Da die Funktionalität selbst, sowie Saros stabil bleibt, ist der Punkt der Reife als erfüllt zu bewerten. Vielmehr wird ein Versagen von Saros durch die partielle Projektsynchronisation vermindert, da weniger Dateien übertragen werden.
Während der gesamten Messprozedur konnten keine Abstürze von Saros provoziert werden. Es wurden jedoch weitere Probleme gefunden, die den Sitzungsaufbau verzögern können.

5.1.4 Analyse im Bezug auf Fehlertoleranz

Ziel: Analysiere die Zuverlässigkeit zum Zwecke der Auswertung in Bezug auf **Fehlertoleranz** vom Blickwinkel des Benutzers im Kontext von Saros.

Fragen: Kann Fehlverhalten bewusst genutzt werden um Fehler zu verursachen?
Bleibt die Software auch nach Fehlern und Fehlbenutzung nutzbar?

Metrik: Identifiziertes Messziel ist die Anzahl von Fehlern bei absichtlicher Fehlbenutzung. Fehlerzustände werden an allen Schritten der partiellen Projektsynchronisation provoziert:

1. Abbruch bei der Prüfsummen-Generierung
2. Abbruch der Sitzungseinladung (Einladender und Eingeladener)
3. Abbruch bei der Übertragung von Ressourcen Informationen
4. Abbruch der Projekteinladung (Einladender und Eingeladener)
5. Abbruch der Projektsynchronisation (Einladender und Eingeladener)
6. Trennung der Verbindung während und nach Sitzungsaufbau (Einladender und Eingeladener)
7. Verlassen der Sitzung während Projekteinladung (Einladender und Eingeladener)

Die Punkte 1 bis 7 verursachen keinerlei Fehlzustände. Dies liegt vor allem an der erweiterten und im Rahmen dieser Arbeit verbesserten Fehlerbehandlung. Jede der ausgeführten Fehlerprovokationen führten zu einem Verlassen der Sitzung (falls bereits aufgebaut) und einer Benachrichtigung aller potentiellen Sitzungsteilnehmer.

Analyse und Interpretation: Keine der versuchten Fehler führt zu schwerwiegenden Problemen. Die Funktionalität selbst sowie Saros bleiben stabil und die Fehlertoleranz ist erfüllt.
Vielmehr ist der überarbeitete Synchronisationsvorgang fehlertoleranter, da er schneller auf Fehlermeldungen reagiert. Des weiteren werden diese in jedem Stadium des Sitzungsstartes, bei allen Teilnehmern korrekt verarbeitet.
Diese Erkenntnis entstammt einerseits aus dieser Messung und andererseits aus dem Review-Prozess der Saros-Arbeitsgruppe. Außerdem wurde abermals durch Zuhilfenahme vom STF die korrekte und fehlerstabile Funktion bestätigt.

5.1.5 Analyse im Bezug auf Wiederherstellbarkeit

Ziel: Analysiere die Zuverlässigkeit zum Zwecke der Auswertung in Bezug auf **Wiederherstellbarkeit** vom Blickwinkel des Benutzers im Kontext von Saros.

Fragen: Ist die Wiederherstellbarkeit in jedem Fall gewährleistet?
Werden Projekte und Dateien korrekt nach Inkonsistenzen wiederhergestellt?

Metrik: Identifiziertes Messziel ist die Anzahl erfolgreicher Wiederherstellungsversuche. Um diese messen zu können, gilt es Inkonsistenzen zu erzeugen. Dies wird für die partielle und die bedarfsgerechten Synchronisation getrennt durchgeführt. Dazu werden vorsätzlich Inkonsistenzen provoziert. Zum Beispiel indem Dateien außerhalb von Eclipse editiert werden. Diese Inkonsistenzen müssten durch den Inkonsistenzmanager erkannt und gelöst werden.
Da die partielle Projektsynchronisation nur wenig am Konsistenzmanagement geändert hat, funktioniert die Inkonsistenzdetektion wie erwartet problemlos. Darüber hinaus verlief die Konsistenzwiederherstellung zuverlässig und reproduzierbar. Da nur Konsistenzinformationen (Prüfsummen des Sitzungs-Hosts) gemeinsam genutzter Ressourcen gesendet werden, gibt es keine Probleme mit im Projekt vorhandenen unterschiedlichen Dateien.
Die Wiederherstellbarkeit ist bei der bedarfsgerechten Dateisynchronisation nicht messbar. Da beim Eingeladenen fehlende oder differierende Ressourcen auf jeden Fall übertragen werden, kann stets von einer konsistenten Ressourcenverfügbarkeit ausgegangen werden.

Analyse und Interpretation: Da die Saros interne Datei-Wiederherstellung problemlos funktioniert, gilt die Wiederherstellbarkeit als erfüllt.
Die Anpassungen der Implementierung beeinflussen die Funktionalität in keiner Weise, da lediglich die zu betrachtenden Ressourcen eingeschränkt oder erweitert werden. So lange die Liste der aktuell gemeinsam genutzten Resssourcen konsistent ist, funktioniert die Wiederherstellung problemlos.
Einzig Verbindungsabbrüche während einer Sitzung können zu unerwünschten Inkonsistenzen führen. Hierbei werden eventuell nicht alle Sitzungsteilnehmer über etwaige Ressourcenänderungen informiert. Diese Problematik lässt sich leider nicht verhindern, da die Unterbrechung des Informationsflusses immer in Inkonsistenzen resultiert.

5.1.6 Analyse im Bezug auf Analysierbarkeit

Ziel: Analysiere die Änderbarkeit zum Zwecke der Auswertung in Bezug auf **Analysierbarkeit** vom Blickwinkel des Entwicklers im Kontext von Saros.

Fragen: Wie detailliert und aussagekräftig ist die Dokumentation?
Stimmt das Code-zu-Dokumentation-Verhältnis?
Ist die Umsetzung verständlich und logisch aufgebaut?
Ist der Code lesbar?

Metrik: Identifiziertes Messziel ist die Verständlichkeit des Quelltextes. Dazu werden Kommentare zu den neuen Funktionen analysiert und entsprechend ausgewertet. In Tabelle 5.4 befindet sich die Statistik, die über gemittelte Werte aufzeigt, wie das Code-zu-Kommentarverhältnis ist. Dabei ist das Code zu Kommentarverhältnis bei dieser Arbeit bei etwa 5-zu-1. Theoretisch gibt es alle fünf Zeilen Quelltext eine Zeile Kommentar. Praktisch ist dieses Verhältnis jedoch als besser zu bewerten. Einzelne Zeilenkommentare und Modifikationen sowie Erweiterungen an vorhandenen Kommentaren wurden bei dieser statischen Codeanalyse nicht betrachtet. Vielmehr ist auch entscheidend, an welchen Stellen Kommentare notwendig sind. Dies wurde auch durch den Review-Prozess sichergestellt.

	#
hinzugekommene Zeilen	≈1700
entfernte Zeilen	≈400
davon geänderte Zeilen	≈350
neue/ersetzte Klassen	7
bearbeitete Klassen	44
hinzugefügte Kommentarzeilen	≈260
entfernte Kommentarzeilen	≈37
hinzugefügte Codezeilen	≈1440
entfernte Codezeilen	≈360

Tabelle 5.4: Statistik zur Feststellung der **Analysierbarkeit**

Analyse und Interpretation: Vor allem für die Umsetzung der partiellen Projektsynchronisation waren viele Änderungen und Erweiterungen des Synchronisations-Codes notwendig. Bei der Implementierung wurde, an geeigneten Stellen auf die Wiederver-

wendung zurückgegriffen. Dazu wurde die vorhandene Dokumentation angepasst, oder fehlende Dokumentation hinzugefügt. Neu erzeugte Methoden wurden stets mit *JavaDoc*-Kommentaren versehen und mögliche Parameter zudem ausführlich beschrieben. Rückgabetypen, falls vorhanden und nicht selbsterklärend, besitzen zudem eine eindeutige Beschreibung. Dazu wurde auf die Formatierungs- und Verlinkungsmöglichkeiten von *JavaDoc* zurückgegriffen. Sie unterstützt beispielsweise Aufzählungen und Hervorhebungen. Bei der Funktionsbeschreibung wurde stets Wert auf Verständlichkeit gelegt. Codedokumentation und logischer Aufbau orientieren sich an den Codekonventionen von Saros[21]. Daher kann von einer einheitlichen Dokumentation und Codestruktur gesprochen werden.

Des weiteren wurde die Analysierbarkeit durch Vereinfachungen von Code gesteigert. So wurde beispielsweise der Code von der *FileListDiff*[22] Klasse so angepasst, das er einerseits effizienter und andererseits übersichtlicher strukturiert und logischer aufgebaut ist. Zudem wurde jeder Schritt, zur Erzeugung des *Diff*, explizit dokumentiert.

Aufgrund dessen, das jede Implementierung in Form eines Patches den Reviewprozess durchlaufen hat, wurde einmal mehr die Lesbarkeit, Logik und Code-zu-Dokumentation-Verhältnis validiert. Die Analysierbarkeit kann in diesen Rahmen als gewährleistet angesehen werden.

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

[22] Klasse zur Feststellung der Unterschiede zweier Ressourcenlisten.

5.1.7 Analyse im Bezug auf Modifizierbarkeit

Ziel: Analysiere die Änderbarkeit zum Zwecke der Auswertung in Bezug auf **Modifizierbarkeit** vom Blickwinkel des Entwicklers im Kontext vom Saros.

Fragen: Wie leicht können die neuen Funktionalitäten modifiziert werden?

Metrik: Identifiziertes Messziel ist die „Einfachheit" wie Änderungen an der gegebenen Implementierung vorgenommen werden können. Dazu wird die durchschnittliche Methodengröße und die Modularität genauer betrachtet. Vor allem gute Modularität zeugt von einfacher Modifizierbarkeit. Übersichtliche und kompakte Methoden erfüllen eine gute Analysierbarkeit und eine einfache Modifizierbarkeit.

Analyse und Interpretation: Wie bereits bei der Untersuchung der Analysierbarkeit gezeigt, sind neue Methoden ausreichend beschrieben. Dazu kommt, dass sie weitestgehend kompakt, also relativ klein und modular gefasst sind. Mit „klein" sind Methodengrößen unter 50 Zeilen[23] zu verstehen. Die Modularität entspricht der Kapselung spezifischer Fähigkeiten. Zudem sind die Methoden wiederverwendbar und wenn möglich generisch gestaltet, damit sie komfortabel nutzbar sind. Beide Erweiterungen führten zu knapp 100 neuen Methoden, die zu 90% unter der 50 Zeilen Grenze liegen.

Ein weiterer Punkt, der die Modifizierbarkeit stark beeinflusst, ist die Nutzung des so genannten *GalleryWidgets*. (siehe (Bjö11, Seite 96 f.)) Dieses zusätzliche Plugin dient Entwicklern grafische Komponenten darzustellen, ohne Saros konfigurieren zu müssen. Oftmals ist für die Übernahme von Änderungen nicht mal der Neustart der Eclipse Instanz notwendig. Dies erlaubt neben einer schnellen Implementierung eine direkte Wiederverwendung entwickelter grafischer Oberflächen. Da alle entwickelten bzw. modifizierten Assistenten in das *GalleryWidget* mit eingebettet sind, ist die Modifizierbarkeit auch der Benutzerschnittstelle um ein Vielfaches einfacher.

[23] Entspricht der ungefähren Anzahl von Codezeilen, die auf eine Bildschirmseite mit einer vertikalen Auflösung von 800 Pixeln passt.

5.2 Anwendung und Auswertung der heuristischen Evaluation

Die heuristische Evaluation wird nach den im Abschnitt 2.2 auf Seite 24 f. entwickelten Phasen durchgeführt. Das folgende Unterkapitel definiert die Arbeitsaufgabe.
Basierend auf den Betrachtungen und gefundenen Problemen durch Björn Kahlert ((Bjö11, Seite 177 f.)), kommt es zu einer Usability-Problemsammlung.
Danach werden Auswertung und Bewertung vor und nach der Implementierung ausführlich betrachtet.

5.2.1 Arbeitsaufgabe

Die zu bewältigende Arbeitsaufgabe soll nach Sarodnick und Brau (Flo06) möglichst praxisnah und viele der zu untersuchenden Softwarebereiche betrachten. Zu diesem Zweck wird die Arbeitsaufgabe folgendermaßen definiert:

- Teilen beliebiger Ressourcen eines Projektes unter Zuhilfenahme des Projektauswahl-Assistenten.
- Erweiterung der Selektion durch Auswahl weiterer Ressourcen im Project Explorer und anschließendes Synchronisieren.
- Löschen, Verschieben und Erstellen von mindestens einem Ordner und einer Datei während der Sitzung.
- Aktivierung der bedarfsgerechten Dateisynchronisation.
- Erweiterung der gemeinsam genutzten Ressourcen durch Nutzung der aktivierten Funktionalität.
- Sitzungsabbau.
- Sitzungsaufbau mit einem Gesamtprojekt über 200 Megabyte.
- Während der Projekt-Synchronisierung: Benutzung der bedarfsgerechten Dateisynchronisation, um Dateien bevorzugt zu übertragen und nutzbar zu machen.
- Sitzungsabbau.

Die Arbeitsaufgabe ist die Grundlage für die zweite Phase der heuristischen Evaluation. Die Evaluatoren orientieren sich während der Evaluationsdurchgänge an dieser Aufgabenstellung.

5.2.2 Heuristische Evaluation vor der Implementierung

Die bei den Durchgängen gefundenen Usability-Probleme werden in ein Schema übertragen. (vgl. (Flo06)) Dies dient der einfachen Kategorisierung und Vergleichbarkeit. Das Schema umfasst die Benennung des Problems, den Fundort im System, eine Beschreibung und die daraus erwartete Auswirkung.
Die gefundenen Usability Probleme vor der Implementierung sollen zeigen, wo es im Funktionsbereich der Synchronisierung Probleme gibt. Ein Abgleich mit den Ergebnissen nach der Implementierung soll eine Einschätzung der veränderten Usability erlauben.
Die folgenden Tabellen sind nach logischer Reihenfolge sortiert.

Benennung	Funktion „*Share with...*“-Kontextmenü auch auf Unterressourcen von Projekt erwartet.
Fundort	Project Explorer
Beschreibung	Das entsprechende Kontextmenü „*Share With...*“ fehlt beim Klick auf Ressourcen die kein Projekt sind. Es wird zumindest ein deaktiviertes Menü erwartet. So wäre auch der Bezug zu diesem Menüpunkt dauerhaft gegeben.
erwartete Auswirkung	Intuitives Lernen ist eingeschränkt. Saros gibt in diesem Fall keinerlei Hinweise, warum etwas nicht dargestellt wird. Dies resultiert gegebenenfalls in Akzeptanzminderung.

Tabelle 5.5: Usability Problem - Fehlendes Kontextmenü

Benennung	Unerwartetes Verhalten, wenn geschlossenes Projekt selektiert wird.
Fundort	Auf Seite eines Einladenen bei Projektauswahl zur Synchronisation.
Beschreibung	Wenn der Einladende ein geschlossenes Projekt mit jemandem teilt, führt dies zum Sitzungsaufbau, aber nicht zur Projektsynchronisation. Weder Einladender noch Eingeladener werden darüber informiert. Erwartet würde das vorherige Öffnen des Projektes mit anschließendem Synchronisieren. Es besteht somit der Grund eines beidseitigen Sitzungsabbruchs, da erwartetes Verhalten nicht eintritt.
erwartete Auswirkung	Bei unerfahrenen Nutzern kann dies zu einer negativen Auswirkung auf die Akzeptanz führen. Aufgrund mangelnder Information kommt es zu einem Sitzungsabbruch.

Tabelle 5.6: Usability Problem - Unerwartetes Verhalten bei Projektauswahl

Benennung	Einladung von Nutzern, die offline sind, ist möglich.
Fundort	„*Select Buddy*"-Assistent und Saros-View
Beschreibung	Offline befindliche Buddies können in den entsprechenden Auswahllisten selektiert und eingeladen werden. Dies führt zu einer Fehlermeldung, die eine Nicht-Kompatibilität zu Saros vermuten lässt. Resultat ist eine automatische lokale Sitzungsbeendigung.
erwartete Auswirkung	Sitzung wird unvermittelt abgebrochen und kann zu einer negativen Auswirkung auf die Akzeptanz führen.

Tabelle 5.7: Usability Problem - Einladen von abwesenden Benutzern

Benennung	Unerwartetes Verhalten des Projekteinladungs-Assistenten.
Fundort	Auf Seite eines Eingeladenen im Projekteinladungs-Assistenten.
Beschreibung	Wenn der Eingeladende ein Projekt annimmt, so wird der Status der Datei-Synchronisation im unteren Teil des Assistenten durch einen Fortschrittsbalken wiedergegeben. Die Buttons und Eingabefelder des Assistenten werden zudem deaktiviert. Abgesehen von der untypischen Fortschrittsanzeige im Assistenten, ist diese wenig aussagekräftig. (siehe Abbildung 5.1)
erwartete Auswirkung	Es kann der Eindruck eines „Hängenbleibens“ von Saros entstehen, da der Assistent ausgegraut wird. Eventuell resultiert dies in einem Sitzungsabbruch.

Tabelle 5.8: Usability Problem - Unerwartete Darstellung im Projekteinladungs-Assistenten

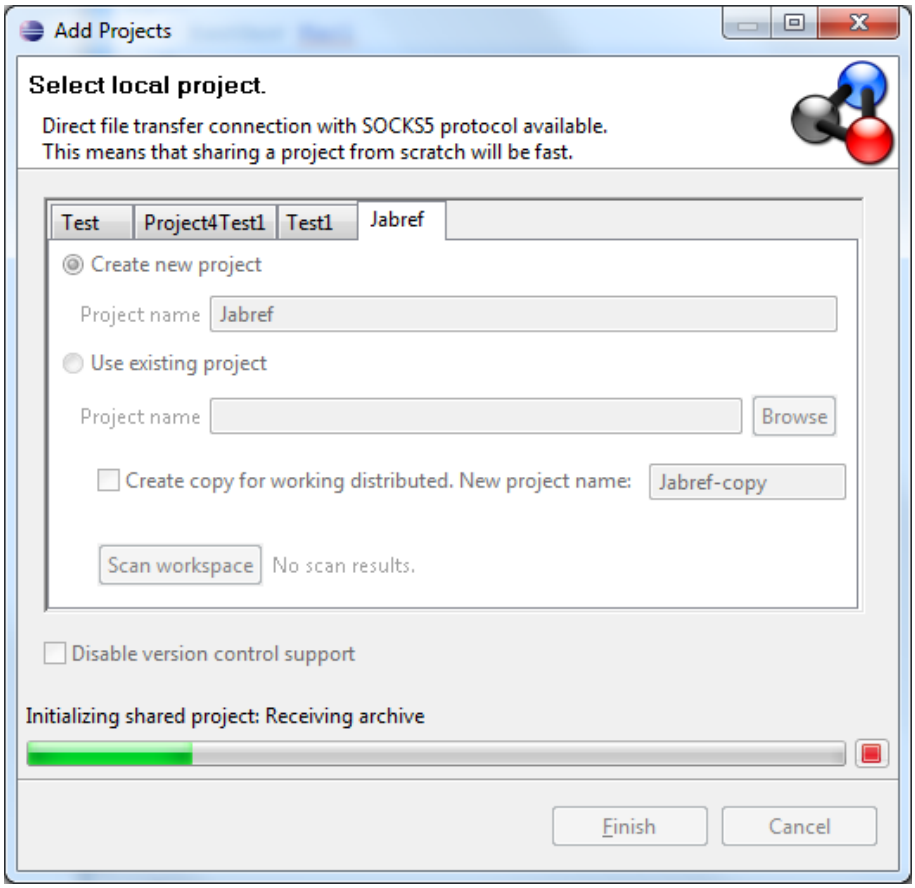


Abbildung 5.1: Deaktivierter Projekteinladungs-Assistent

Benennung	Einladender wird während Sitzungs- und Projekteinladung nicht ausreichend über den Status informiert.
Fundort	Auf Seite eines Einladenden während Sitzungs/Projektsynchronisation.
Beschreibung	Lädt der Einladende jemanden zu einer Sitzung ein, sieht er kaum was im Hintergrund passiert. Somit ist dem Sitzungshost nicht hundertprozentig ersichtlich, wann Sitzungs- und Projektsynchronisation abgeschlossen sind. Gegebene Informationen sind spärlich und schlecht ersichtlich. Sie sind nur in der Statusbar vorhanden.
erwartete Auswirkung	Bei unerfahrenen Nutzern kann dies zu einer negativen Auswirkung auf die Akzeptanz führen. Es kann vorschnell der Eindruck entstehen, dass der Synchronisationsvorgang entweder abgeschlossen ist, oder aber etwas schief gelaufen ist.

Tabelle 5.9: Usability Problem - Mangelnder Status für Einladenden

Benennung	Eingeladener wird zwischen Sitzungs- und Projekteinladungsdialog uninformiert gelassen.
Fundort	Auf Seite eines Eingeladenen während Sitzungs/Projektsynchronisation.
Beschreibung	Wenn der Einladende jemanden zu einer Sitzung einlädt, dann öffnet sich auf Eingeladenenseite sofort ein Sitzungseinladungsfenster. Wird dieses bestätigt, passiert auf Eingeladenenseite eine Zeit lang nichts mehr. Es entsteht Verwirrung, was gerade passiert.
erwartete Auswirkung	Bei unerfahrenen Nutzern kann dies zu einer negativen Auswirkung auf die Akzeptanz führen. Es kann schnell der Eindruck entstehen, dass etwas schiefgelaufen ist. Somit wird die Sitzung vorzeitig beendet. Zumindest wirkt Saros langsam.

Tabelle 5.10: Usability Problem - Uninformierter Eingeladener

Benennung	Unerwartetes Verhalten, wenn Eingeladener die Einladung zu einer Sitzung ablehnt.
Fundort	Sitzungsaufbau
Beschreibung	Lehnt der eingeladene Teilnehmer die Einladung zu einer Sitzung ab, erhält der Einladende eine Fehlermeldung anstelle einer einfachen Notiz.
erwartete Auswirkung	Nicht erwartetes Verhalten, kann in Akzeptanzminderung resultieren.

Tabelle 5.11: Usability Problem - Unerwartetes Verhalten bei Sitzungsaufbau

Benennung	Bereits gemeinsam genutztes Projekt nochmals synchronisierbar.
Fundort	„*Share Projekt*"-Assistent während Sitzung
Beschreibung	Der Sitzungs-Host kann bereits gemeinsam nutzbare Projekte nochmals über den „*Share Projekt*"-Assistenten an die Sitzungsteilnehmer synchronisieren. Dies führt zu Problemen im Sitzungsmanagement. Saros sieht die zweimalige Synchronisation desselben Projektes nicht vor, wodurch die Konsistenzkontrolle und das Ressourcenmanagement leidet.
erwartete Auswirkung	Die Sitzung kann aufgrund fehlerhafter Konsistenzkontrolle instabil werden. Zudem kann es passieren, dass das zuerst geteilte Projekt nicht mehr gemeinsam nutzbar ist. Das Problem ist nur durch Sitzungsabbau lösbar.

Tabelle 5.12: Usability Problem - Doppelte Synchronisierung

Benennung	Löschen und Verschieben von gemeinsam genutzten Ressourcen unzuverlässig.
Fundort	Project Explorer
Beschreibung	Wenn die Teilnehmer eines gemeinsam genutzten Projektes mehrere Verschiebe- und Löschoperationen hintereinander ausführen, kommt es zu unzuverlässigem Verhalten. Teilweise werden diese Aktivitäten nicht publiziert, so dass Inkonsistenzen auftreten. Diese sind nicht immer lösbar.
erwartete Auswirkung	Aufgrund nicht lösbarer Inkonsistenzen, kann nur ein Sitzungsabbruch und ein neuer Synchronisationsvorgang die Problematik entschärfen. Saros wirkt instabil und wenig verlässlich.

Tabelle 5.13: Usability Problem - Unzuverlässige Ressourcenoperationen

Benennung	Sitzungsteilnehmer, welcher nur Leseberechtigung hat, kann Klassen anlegen.
Fundort	Project Explorer
Beschreibung	Selbst unter der Beschränkung der Rechte kann ein Sitzungsteilnehmer Klassen anlegen, welche geteilt werden.
erwartete Auswirkung	Nicht erwartetes Verhalten, kann in Akzeptanzminderung resultieren.

Tabelle 5.14: Usability Problem - Klassen erzeugen ohne Rechte

Benennung	Unerwartetes Verhalten, wenn Nutzer ohne Saros eingeladen werden.
Fundort	„*Select Buddy*“-Assistent und Saros-View
Beschreibung	Wird ein Buddy ohne Saros eingeladen, wird die Sitzung sofort lokal beim Einladenden beendet. Dabei ist egal, ob noch weitere saroskompatible Buddies eingeladen waren. Erwartet würde eine Nachfrage, ob der Buddy nicht berücksichtigt werden solle. Selbiges passiert bei Buddies, die offline sind.
erwartete Auswirkung	Sitzungsabbruch und negative Auswirkung auf Akzeptanz.

Tabelle 5.15: Usability Problem - Einladen von Benutzern ohne Saros-Unterstützung

Benennung	Icon auf gemeinsam genutzten Projekten leicht übersehbar.
Fundort	Project Explorer
Beschreibung	Das Overlay-Icon kennzeichnet gemeinsam genutzter Projekte. Es ist aufgrund der geringen Größe schlecht zu erkennen.
erwartete Auswirkung	Langwieriges Suchen nach gemeinsam genutzten Ressourcen kann zeitaufwändig sein. Dies kann in Akzeptanzminderung resultieren.

Tabelle 5.16: Usability Problem - Icon schlecht erkennbar (1)

Benennung	Icon auf aktiven Ressourcen nicht in Benutzerfarbe.
Fundort	Project Explorer
Beschreibung	Das Overlay-Icon zur Kennzeichnung geöffneter Editoren ist nicht in der erwarteten Benutzerfarbe.
erwartete Auswirkung	Eine Zuordnung geöffneter Ressourcen zu (> 2) Nutzern ist nicht möglich. Mehrere aktive Remote-Editoren sind gleichzeitig möglich. Dies erhöht das Verwirrungspotential.

Tabelle 5.17: Usability Problem - Icon schlecht erkennbar (2)

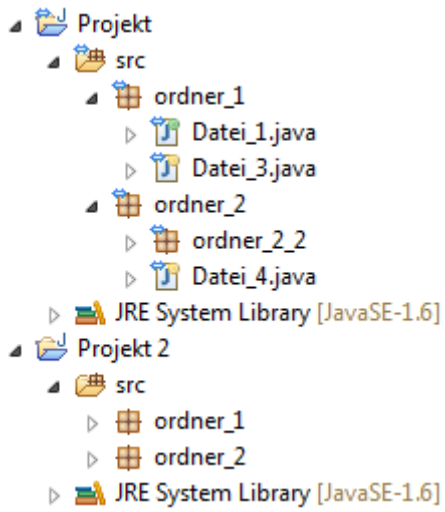


Abbildung 5.2: Datei-Annotationen

Benennung	Die Bedeutung der Datei-Annotationen sind unklar.
Fundort	Project Explorer
Beschreibung	Es ist nicht ersichtlich, was die Datei-Annotationen explizit bedeuten, und vom wem sie generiert werden. Notwendigkeit der gelben Indikatoren unklar. (siehe Abbildung 5.2 auf Seite 70)
erwartete Auswirkung	Kann verwirrend wirken und zur Akzeptanzminderung führen.

Tabelle 5.18: Usability Problem - Bedeutung der Datei-Annotationen

Benennung	Interaktionen des Akteurs im Project Explorer des Beobachters nicht sichtbar.
Fundort	Project Explorer
Beschreibung	Navigation zu Ressourcen und das Auf- und Zuklappen des Ressourcenbaumes ist nicht bei anderen Sitzungsteilnehmern nachvollziehbar.
erwartete Auswirkung	Zuordnung geteilter Ressourcen wird erschwert.

Tabelle 5.19: Usability Problem - Keine Interaktion im Project Explorer

Benennung	Verlassen der Sitzung führt zu einem Nutzer in Sitzung.
Fundort	Sitzungsabbau
Beschreibung	Wird bei einer Sitzungseinladung diese abgelehnt, oder verlassen alle Sitzungsteilnehmer die Sitzung, so ist der Host der Einzige in einer Sitzung.
erwartete Auswirkung	Dies ist nicht immer für den Host ersichtlich, da das erwartete Verhalten eine Beendigung der Sitzung wäre.

Tabelle 5.20: Usability Problem - Ein Anwender in Sitzung

5.2.3 Heuristische Evaluation nach der Implementierung

Die gefundenen Usability Probleme nach der Implementierung sollen zeigen, wo es im Funktionsbereich der Synchronisierung nach wie vor Probleme gibt. Aus diesem Grund befinden sich in diesem Abschnitt bereits bekannte Probleme (zusammengefasst in Tabelle 5.21) aber auch neu entstandene.

Benennung	Unerwartetes Verhalten, wenn Nutzern ohne Saros eingeladen werden.
Benennung	Icon auf gemeinsam genutzten Projekten/Ressourcen leicht übersehbar.
Benennung	Icon auf aktiven Ressourcen nicht in Benutzerfarbe.
Benennung	Die Bedeutung der Datei-Annotationen sind unklar.
Benennung	Interaktionen des Akteurs im Project Explorer des Beobachters nicht sichtbar.
Benennung	Verlassen der Sitzung führt zu einem Nutzer in Sitzung.

Tabelle 5.21: Usability-Probleme die vor und nach der Implementierung existieren

Benennung	Unterscheidung von kompletten zu partiell geteiltem Projekt nicht ersichtlich.
Fundort	Project Explorer
Beschreibung	Wenn die Projektbäume im Project Explorer zugeklappt sind, ist nicht ersichtlich ob ein Projekt nur teilweise oder ganz geteilt ist.
erwartete Auswirkung	Konfusion

Tabelle 5.22: Usability Problem - Darstellung partiell geteiltes Projekt

Benennung	Ressourcenauswahl des Einladenden für Eingeladenen nicht ersichtlich.
Fundort	Auf Seite eines Eingeladenen im Projekteinladungs-Assistenten.
Beschreibung	Wählt der Einladende eine Teilmenge eines Projektes zur gemeinsamen Nutzung, so wird dies dem Eingeladenen nicht ersichtlich. Er erhält lediglich einen Hinweis, dass es ein Teil des Projektes ist. Dies wird in Abbildung 5.3 dargestellt. Eine vollständige Liste der gemeinsamen Ressourcen erhält er nicht. Somit muss bis zum Ende der Synchronisierung gewartet werden, um zu erfahren welche Ressourcen tatsächlich gemeinsam genutzt werden.
erwartete Auswirkung	Informationsmangel

Tabelle 5.23: Usability Problem - Fehlende Ressourcendarstellung

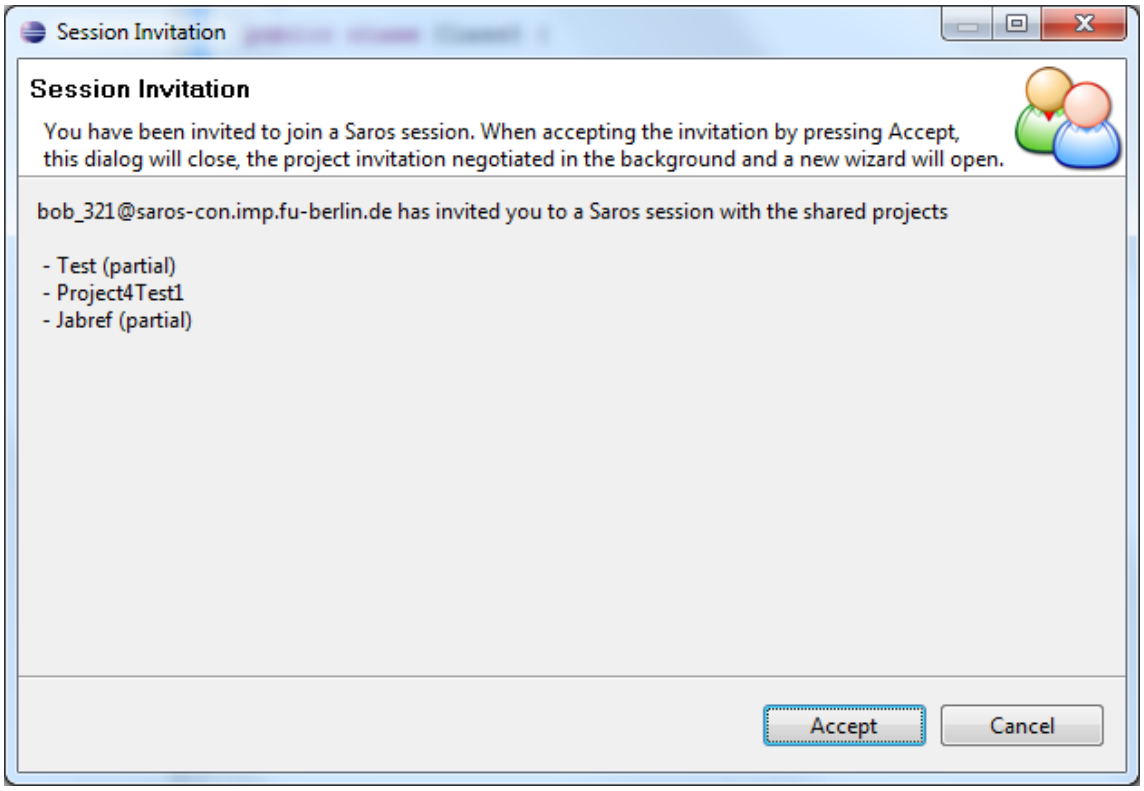


Abbildung 5.3: Session Invitation Dialog

5.2.4 Bewertung und Kategorisierung der Ergebnisse

Die vierte Phase der Heuristischen Evaluation ordnet eine Fatalität dem gefundenen Usability-Verstoß (Heuristik) zu. (vgl. Abschnitt 2.2.1 auf Seite 25 f.)

Benennung	Heuristik	Fatalität*	Lösungsvorschlag
Funktion „*Share with...*"-Kontextmenü auch auf Unterressourcen von Projekt erwartet	**3, 5 & 9**		Aktivierung der Funktionalität auf allen Ressourcenstufen.
Unerwartetes Verhalten wenn geschlossenes Projekt selektiert wird	**2, 4 & 5**		Öffnen des Projektes vor Übertragungsstart. Gegebenenfalls Dialog, der das Öffnen erfragt.
Einladung von Nutzern die offline sind, ist möglich	**5 & 6**		Verhindern, dass offline Nutzer eingeladen werden können durch Abfrage der Präsenz.
Unerwartetes Verhalten des Projekteinladungs-Assistenten	**3**		Auslagerung der Statusinformation aus dem Assistenten.
Einladender wird während Sitzungs- und Projekteinladung nicht ausreichend über den Status informiert	**3**		Deutlichere Statuswiedergabe und bessere Kontrollmöglichkeiten, sowie detailliertere Fortschrittsangaben.
Eingeladener wird zwischen Sitzungs- und Projekteinladungsdialog uninformiert gelassen.	**3 & 10**		Deutlichere Statuswiedergabe und eventuell schnellere Verarbeitung zur „Lückenschließung". Informativer Fortschrittsdialog.
Unerwartetes Verhalten wenn Eingeladener die Einladung zu einer Saros-Sitzung ablehnt	**5 & 10**		Austausch der Warnung mit Hinweis (Balloon-Tooltip).
Bereits gemeinsam genutztes Projekt nochmals synchronisierbar	**2 & 5**		Verhindern, das bereits gemeinsam genutztes Projekt ein weiteres mal ausgewählt wird.

* kosmetisch gering schwer katastrophal

Tabelle 5.24: Bewertung/Kategorisierung der Usabilityprobleme vor der Implementierung (1)

Benennung	Heuristik	Fatalität*	Lösungsvorschlag
Löschen und Verschieben von gemeinsam genutzten Ressourcen unzuverlässig	**1 & 5**		Datei- und Ordneroperationen sorgsamer verfolgen und verarbeiten. Auf die korrekte Übertragung der *AktivitätsObjekte* achten.
Sitzungsteilnehmer, welcher nur Leseberechtigung hat, kann Klassen anlegen	**2 & 5**		Rechtemanagement auf das Anlegen von Klassen anwenden.
Unerwartetes Verhalten wenn Nutzer ohne Saros eingeladen werden	**1 & 5**		Abfrage, ob Sitzung beendet werden solle, oder aber Nutzer in der Einladung ignoriert werden soll.
Icon auf gemeinsam genutzten Projekten/Ressourcen leicht übersehbar	**3 & 10**		Deutlichere oder markantere Farbgebung bzw. Wahl eines anderen Symbols.
Icon auf aktiven Ressourcen nicht in Benutzerfarbe	**3, 5 & 10**		Für alle möglichen Benutzerfarben eigenes Icon erzeugen. Der Übersicht halber eventuell das gelbe Icon (geöffnet aber im Hintergrund) entfernen.
Die Bedeutung der Datei-Annotationen sind unklar	**3 & 10**		Eindeutigere Annotationen, oder aber personenbezogene Annotationen verwenden.
Interaktionen des Akteurs im Project Explorer des Beobachters nicht sichtbar	**3, 5 & 10**		Prüfen ob eine pragmatische Darstellung von der Interaktion möglich ist. Screensharing verbessern.
Verlassen der Sitzung führt zu einem Nutzer in Session	**1**		Verlassen der Sitzung im Falle, dass es sich um die Situation einer Ersteinladung handelt.

* kosmetisch gering schwer katastrophal

Tabelle 5.25: Bewertung/Kategorisierung der Usabilityprobleme vor der Implementierung (2)

Benennung	Heuristik	Fatalität*	Lösungsvorschlag
Unterscheidung von kompletten zu partiell geteiltem Projekt nicht ersichtlich	**10**		Verwendung unterschiedlicher Overlay-Indikatoren auf Projektebene.
Ressourcenauswahl des Einladenden für Eingeladenen nicht ersichtlich	**3 & 10**		Aufzählung bei der Sitzungseinladung aller gewählter Ressourcen.

* kosmetisch gering schwer katastrophal

Tabelle 5.26: Bewertung/Kategorisierung neuer Usabilityprobleme nach der Implementierung

Tabelle 5.24 und Tabelle 5.25 stellen gefundene Usability-Probleme vor der Implementierung dar. Im unteren Teil der Tabelle 5.25 werden Usability-Probleme kategorisiert, die vor und nach der Implementierung gefunden wurden. Tabelle 5.26 stellt Usability-Probleme dar, die neu entstanden sind. Die Zuordnung der Heuristiken erlaubt die Kategorisierung von Usability-Problemen. Im Anbetracht der gefundenen Usability Probleme ist eine Vorher/Nachher Untersuchung sinnvoll. Dies wird im Abschnitt 5.3 gemacht. Herauszustellen ist die Eliminierung der beiden katastrophalen Probleme. Dies waren:

- Löschen und Verschieben von gemeinsam genutzten Ressourcen unzuverlässig
- Bereits gemeinsam genutztes Projekt nochmals synchronisierbar

Beide führten zwangsläufig zu ernsten Problemen mit Saros. Das Konsistenzmanagement war teilweise nur unzuverlässig in der Lage das Löschen und Verschieben zu korrigieren. Die mehrmalige Synchronisation von Projekten führte zu gänzlich unvorhersagbaren Verhalten.

5.3 Evaluationsergebnis

Der einmalig ausgeführte GQM-Ansatz evaluiert vor allem implementierungsabhängige Qualitätsaspekte der neuen Funktionalitäten. Dieser Ansatz ergab einen positiven Eindruck der Implementierung. Dies spiegelt sich neben den erfolgreichen Review-Prozessen auch in sarosspezifischen Tests wider. Bereits existierende Tests im Saros Test Framework untermauern diesen Eindruck. Die Implementierung der partiellen Projektsynchronisation brachte zudem eine effizientere funktionelle Umsetzung der Grundfunktionalität mit sich. Dies zeigte sich vor allem in der Stabilität der partiellen Projektübertragung. Eine statische Codeanalyse auf

Basis von *FindBugs*[24] erlaubte außerdem die faktenbasierte Evaluierung durch die GQM. *FindBugs* ermöglichte außerdem die Eliminierung unnötiger oder fehlerbehafteter Umsetzungsstrategien.

Die heuristische Evaluation lässt einen Vergleich auf Basis von Usability-Aspekten zu. Zur Validierung der bearbeiteten Usability-Probleme wurde ein zweiter Usability-Test durchgeführt. Die Durchführung war mit dem ersten Usability-Test identisch. Somit wurde ein Vergleich vor und nach der Implementierung möglich.

Anfangs wurden 16 Usability-Probleme in Verbindung mit dem Sitzungsaufbau und der Dateisynchronisation gefunden. Diese waren unterschiedlicher Natur und Fatalität. Da nicht das Gesamtsystem Saros betrachtet wurde und daher nur ausgewählte Funktionalitäten beansprucht sind, ist die Fehlerzahl moderat. Wenn eine eindeutige Zuordnung einer Heuristik zu einem Verstoß nicht möglich war, wurden mehrere zugeordnet.

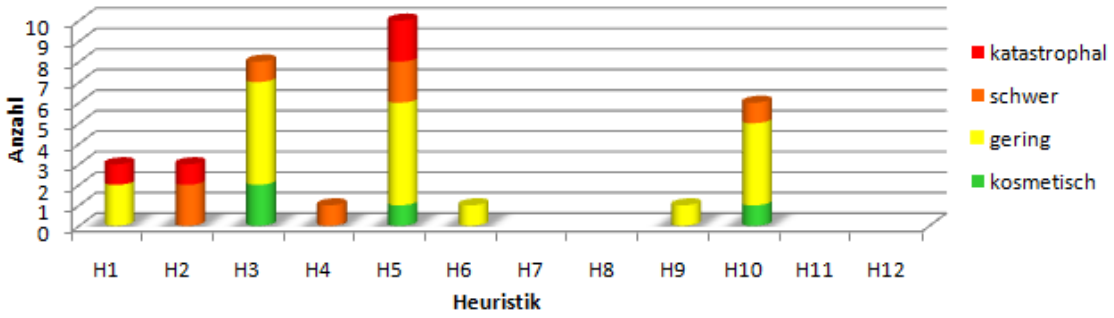


Abbildung 5.4: Verteilung der Usability-Probleme vor Implementierung

Abbildung 5.4 zeigt, dass besonders häufig gegen Heuristik 3, 5 und 10 verstoßen wurde. Dies sind:

3 Selbstbeschreibungsfähigkeit ist die einheitliche und unmittelbare Anzeige des Systemstatus. Der Benutzer sollte die Informationstiefe des Systemstatus selbst bestimmen können.

5 Erwartungskonformität: Die Informationsdarstellung sollte systemimmanent und bei plattformspezifischen Konzepten konsistent sein. Bei ähnlichen Aufgaben sollten Dialoge vergleichbar und an erwarteter Position dargestellt werden.

10 Wahrnehmungssteuerung - Das Layout sollte minimalistisch gehalten werden. Gruppierungen, Farbgestaltung und sinnvolle Informationsreduktion sollten so verwendet werden, dass die Aufmerksamkeit des Nutzers zu relevanter Information gelenkt wird.

[24] `http://findbugs.sourceforge.net/`

Anhand dieser Kategorisierung der Usability-Probleme lassen sich drei Schwerpunkte ausmachen. Die Selbstbeschreibungsfähigkeit sind meist Verstöße, die der Wissensebene zuzuordnen sind. Dem gegenüber sind Verstöße gegen die Erwartungskonfomität und Wahrnehmenungssteuerung häufig regelbasierter Natur. Dieses Wissen lässt sich bei der weiteren Entwicklung verwenden.

Zwei der gegen Heuristik 1, 2 aber vor allem 5 verstoßenden Probleme sind als katastrophal kategorisiert. Diese Verstöße gegen die Erwartungskonformität wurden bei der Implementierung priorisiert behandelt.

Nach der Implementierung wurde eine erneute heuristische Evaluation durchgeführt. Das Ergebnis ist im Diagramm 5.5 dargestellt.

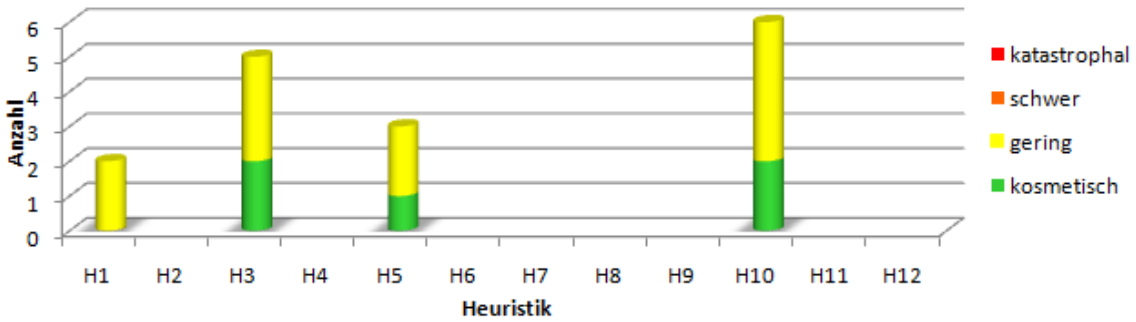


Abbildung 5.5: Verteilung der Usability-Probleme nach Implementierung

Bei der Betrachtung und dem Vergleich mit Diagramm 5.4 auf Seite 77 ist auf die unterschiedliche Skalierung der Ordinatenachse zu achten. Wurden im ersten Diagramm noch zehn Verstöße gegen die Heurisitk 5 gemessen, sind es nach der Implementierung nur noch drei. Insgesamt wurden am Ende acht Usability-Probleme ausfindig gemacht, deren Verteilung ähnlich ausgeprägt ist.

Letztlich wurden zehn der vormals 16 Usability Probleme im Laufe der Implementierung gelöst oder abgeschwächt. Dem gegenüber entstanden durch die erweiterte Funktionalität zwei neue Verstöße. Diese sind jedoch als gering bzw. kosmetisch zu werten.

Erwartet wurde nach Abschnitt 2.2 auf Seite 24 zumindest eine gleichbleibende Qualität. Es kann im Bezug auf die untersuchten und bearbeiteten Saros-Bereiche von einer Qualitätssteigerung gesprochen werden. Vor allem die Eliminierung der als katastrophal kategorisierten Probleme ist als positive Qualitätssicherung zu werten. Das Ergebnis basiert auf Auswertung durch den Entwickler selbst. Eine Evaluierung der umgesetzten Funktionalität könnte durch einen Vergleichstest mit Unbeteiligten bestätigt werden.

Im Hinblick auf das Thema und die Zielstellung dieser Masterarbeit (Abschnitt 1.1 auf Seite 2) wurde das Ziel erreicht. Es wurden qualitätsgesicherte Funktionialität umgesetzt.

6 Fazit

Im Folgenden wird diese Masterarbeit zusammengefasst. Dem schließt sich ein Ausblick auf zukünftige Erweiterungen an.

6.1 Zusammenfassung

Ziel dieser Arbeit war die Integrierung und Evaluierung partieller Projektsynchronisation und bedarfsgerechter Dateisynchronisation. Dieses Ziel wurde erreicht und beschrieben.
Diese Masterarbeit wurde in verschiedene Phasen untergliedert und ausgewertet. Zuerst erfolgte ein Einarbeitung in Saros, sowie die Betrachtung anderer DPP-Lösungen. Die erlangten Erkenntnisse waren Voraussetzung für einen Meilensteinplan, sowie die Planung der weiteren Vorgehensweise.
Die anschließende Analysephase konzentrierte sich auf die heuristische Evaluation vor der Umsetzung der Funktionalitäten. Diese Untersuchung bildet die Basis für die Usability-Untersuchung nach der Implementierung. Gefunden wurden beim ersten Durchgang 16 Probleme unterschiedlicher Natur und Fatalität.
Darauf aufbauend und anhand der Aufgabenstellung wurde die iterativ inkrementelle Umsetzung der Funktionalitäten erfolgreich durchgeführt. Der Projektstrukturplan aus Abschnitt 1.2.4 auf Seite 7 diente der Orientierung. Die entwickelte partielle Projektsynchronisation ermöglicht schnell und zuverlässig Teilprojekte gemeinsam verteilt zu editieren. Nach der erfolgreichen Erprobung und mehreren Reviews dieser Funktionalität, wurde sie in den Saros-Basiscode übernommen. Im Laufe dieser Umsetzung wurden eine Vielzahl von kleinen aber wichtigen Änderungen im Synchronisationsprozess gemacht. Eine der essentiellsten ist die zeitliche Umstrukturierung des Sitzungsaufbaus (beschrieben im Abschnitt 3.4.1 auf Seite 38 f.). Der gesamte Sitzungsaufbau und die Nutzerinteraktion ist durch diese Änderung zügiger geworden. Der Einladungsprozess konnte für den Eingeladenen wie in Abschnitt 3.6 auf Seite 44 beschrieben wesentlich verkürzt werden.
Der nächste Schritt umfasste die Umsetzung der bedarfsgerechten Dateiübertragung. Diese setzt auf der partiellen Projektsynchronisation auf und erweitert diese. Die bedarfsorientiere

Dateiübertragung ermöglicht eine schnelle Zusammenarbeit der Sitzungsteilnehmer. Sie erweitert zudem partiell geteilte Projekte bequem um weitere Dateien, falls der Bedarf besteht. Das Resultat ist ein konsistenter Weg der Ressourcensynchronisation, welcher sich vor allem durch seine Einfachheit und Flexibilität auszeichnet.
Unter der Berücksichtigung ausgewählter Usability-Aspekte wurde durch die Implementierung mehrere gefundene Probleme eliminiert. Neben einer Evaluation auf Basis der Norm ISO/IEC 9126 erfolgte ein weiterer Durchgang der heuristischen Evaluation. Gefunden wurden hier acht Probleme unterschiedlicher Natur. Die Zahl der Usability-Probleme wurde numerisch halbiert. Betrachtet man weiter die schwere der Probleme, so ist das Resultat noch besser zu bewerten. Dies ist im Abschnitt 5.3 auf Seite 76 nachzulesen.
Bezug nehmend auf die Anforderungen konnten alle geforderten Aspekte umgesetzt werden. Im Rückblick kann von einer erfolgreichen Lösung der Aufgabenstellung gesprochen werden, was durch die Integration in den Stammquellcode von Saros untermauert wird.

6.2 Ausblick

Die sorgsame Umsetzung der Funktionalitäten erzeugt zusätzlich ein überzeugendes Bild für die Usability. Damit ist der Weg für eine problemlose und konsistente Nutzung geebnet.
Die auf diese Arbeit folgenden Entwicklungen an Saros sind vielseitig. Eine Reaktivierung bzw. Stabilisierung der Streaming-Funktionalität wäre vorteilhaft. Dies könnte den Übertragungsvorgang großer Projekte ermöglichen, ohne an Speichergrenzen zu stoßen. Außerdem könnten Erweiterungen der bedarfsorientierten Dateisynchronisation Saros intuitiver gestalten. Denkbar sind beispielsweise abhängigkeitsbasierte Selektionsvorschläge oder eine automatische Ressourcensynchronisation auf Basis vorhergehender Sitzungen.
Die in dieser Arbeit gefundenen Usability-Probleme sollten im Laufe der Zeit Beachtung finden. Dabei können die Lösungsvorschläge aus Abschnitt 5.2.4 auf Seite 74 umgesetzt werden. Eine weiterführende Stabilisierung und Erweiterung des Testframeworks ist in jedem Fall hilfreich, um noch mehr Funktionalität von Saros automatisiert testen zu können.
Die Umsetzung der partiellen Projektsynchronisation und die bedarfsorientierte Dateisynchronisation bieten zukünftig ein breiteres Einsatzspektrum. Dadurch könnten mehr Nutzer angesprochen werden, da auch große Projekte leichter verwendbar sind.

Literaturverzeichnis

[Ali98] ALISTAIR COCKBURN: *Basic use case template*. http://alistair.cockburn.us/Basic+use+case+template, 1998. [Online; accessed 06.04.2011].

[Aza03] AZAD BOLOUR: *Notes on the Eclipse Plug-in Architecture*. Eclipse Corner Artikel - http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html, 2003. [Online; accessed 14.05.2011].

[Bal03] BALAJI KRISH-SAMPATH: *Understanding Decorators in Eclipse*. http://www.eclipse.org/articles/Article-Decorators/decorators.html, 2003. [Online; accessed 17.07.2011].

[Bjö11] BJÖRN KAHLERT: *Verbesserung der Out-Of-Box-Experience in Saros mittels Heuristischer Evaluation und Usability-Tests*. Masterarbeit an der FU Berlin - http://www.inf.fu-berlin.de/inst/ag-se/theses/Kahlert11-saros-out-of-box-experience.pdf, 2011. [Online; accessed 05.05.2011].

[Bun07] BUNDESZENTRALE FÜR POLITISCHE BILDUNG: *Open Source*. http://www.bpb.de/themen/32K5CW, 2007. [Online; accessed 12.07.2011].

[Chr09] CHRISTIAN EL BOUSTANI: *Die Goal-Question-Metric-Methode (GQM)*. Vorlesungsscript - http://static.se.uni-hannover.de/documents/ss2009_labor_xpe/GQM.pdf, 2009. [Online; accessed 22.05.2011].

[DAC08] DACS GOLD PRACTICES: *GOAL-QUESTION-METRIC (GQM) APPROACH*. http://www.goldpractices.com/practices/gqm/index.php, 2008. [Online; accessed 29.08.2011].

[DASH09] DEWAN, PRASUN, PUNEET AGARWAL, GAUTAM SHROFF und RAJESH HEGDE: *Distributed side-by-side programming*. In: *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, CHASE '09, Seiten 48–55, Washington, DC, USA, 2009. IEEE Computer Society.

[Dr.08] DR. ALISTAIR COCKBURN: *Alistair*. http://alistair.cockburn.us/Alistair, 2008. [Online; accessed 09.05.2011].

[Ecl10] ECLIPSE CONTRIBUTORS: *Workbench User Guide - Concepts*. http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.platform.doc.user/concepts/cnav.htm, 2010. [Online; accessed 19.05.2011].

[Flo06] FLORIAN SARODNICK, HENNING BRAU: *Methoden der Usability Evaluation - Wissenschaftliche Grundlagen und praktische Anwendungen*. 1. Auflage im Hans Huber Verlag, Bern, 2006.

[Fra07] FRAUNHOFER GESELLSCHAFT ZUR FÖRDERUNG DER ANGEWANDTEN FORSCHUNG E.V.: *Nichtfunktionale Anforderungen erheben*. http://www.re-wissen.de/Wissen/Anforderungserhebung/Praktiken/Nichtfunktionale_Anforderungen_erheben.html, 2007. [Online; accessed 21.06.2011].

[Hen11] HENDY IRAWAN, BILL MICHELL, SCOTT LEWIS ET AL.: *ECF/DocShare Plugin*. http://wiki.eclipse.org/DocShare_Plugin, 2011. [Online; accessed 10.04.2011].

[Ins08] INSTITUTE, PROJECT MANAGEMENT: *A Guide to the Project Management Body of Knowledge*. PMBOK guide. Project Management Institute, 2008.

[Lau00] LAURIE ANN WILLIAMS: *The Collaborative Software Process*. Dissertation an der Universität von Utah - http://collaboration.csc.ncsu.edu/laurie/Papers/dissertation.pdf, 2000. [Online; accessed 25.07.2011].

[Lut10] LUTZ PRECHELT: *Software Measurement*. Vorlesungsscript zu Spezielle Themen der Softwaretechnik, 2010.

[M.A09] M.A. KUPPE: *Kommunikationstalent Eclipse: Das ECF-Projekt*. Artikel bei JAXenter - http://it-republik.de/jaxenter/artikel/Kommunikationstalent-Eclipse-Das-ECF-Projekt-2403.html, 2009. [Online; accessed 10.04.2011].

[Nie94] NIELSEN, JAKOB: *Usability Engineering*. Morgan Kaufmann Publishers, San Francisco, California, 1994.

[P. 04] P. BOTELLA, X. BURGUÉS, J.P. CARVALLO, X. FRANCH, G. GRAU, J. MARCO, C. QUER: *ISO/IEC 9126 in practice: what do we need to know?* Paper - http://www.essi.upc.edu/~webgessi/publicacions/SMEF%2704-ISO-QualityModels.pdf, 2004. [Online; accessed 19.05.2011].

[Pet05] PETER EELES: *The FURPS+ System for Classifying Requirements*. http://www.ibm.com/developerworks/rational/library/4706.html, 2005. [Online; accessed 09.04.2011].

[Plu09] PLUSPEDIA: *Awareness (CSCW)*. http://de.pluspedia.org/wiki/Awareness_%28CSCW%29, 2009. [Online; accessed 16.06.2011].

[Pro09] PROGRAMMERSBASE.NET: *Java Grundlagen - Interfaces*. http://www.programmersbase.net/Content/Java/Content/Tutorial/Java/Interface.htm, 2009. [Online; accessed 19.07.2011].

[Ria06a] RIAD DJEMILI: *Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammierung*. Diplomarbeit an der FU Berlin - http://www.inf.fu-berlin.de/inst/ag-se/theses/Djemili06-eclipse-erweiterung-PP.pdf, 2006. [Online; accessed 12.07.2011].

[Ria06b] RIAS A. SHERZAD: *Factory Method Pattern in Java*. http://www.theserverside.de/factory-method-pattern-in-java/, 2006. [Online; accessed 22.06.2011].

[Seb04] SEBASTIAN STEIN: *Vorgehensmodell Wasserfallmodell und V-Modell*. http://emergenz.hpfsc.de/html/node42.html, 2004. [Online; accessed 06.04.2011].

[Som06] SOMMERVILLE, IAN: *Software Engineering: (Update) (8th Edition)*. Addison Wesley, 8 Auflage, Juni 2006.

[Ste10] STEPHAN SALINGER, CHRISTOPHER OEZBEK: *Distributed Party Programming*. Artikel auf JAXenter - http://it-republik.de/jaxenter/artikel/Distributed-Party-Programming-3036.html, 2010. [Online; accessed 25.07.2011].

[Til05] TILMAN WALTHER: *Pair Programming*. Ausarbeitung im Rahmen des Seminars Agile Softwareprozesse (FU Berlin) - http://www.tilman.de/uni/PairProgramming.pdf, 2005. [Online; accessed 25.07.2011].

[Uni10] UNIVERSITÄT DER BUNDESWEHR MÜNCHEN: *KollaborativeSoftwareentwicklung*. http://twiki.informatik.unibw-muenchen.de/Main/KollaborativeSoftwareentwicklung, 2010. [Online; accessed 27.07.2011].

[Wik11a] WIKIPEDIA: *Heuristische Evaluierung*. http://de.wikipedia.org/wiki/Heuristische_Evaluierung, 2011. [Online; accessed 11.06.2011].

[Wik11b] WIKIPEDIA: *ISO/IEC 9126*. http://de.wikipedia.org/wiki/ISO/IEC_9126, 2011. [Online; accessed 07.04.2011].

7 Anhang

A Ausführliche Beschreibung der Use Cases

Name und Identifier
Ressourcenauswahl in Saros-View
Beschreibung (description)
Der Anwender hat die Möglichkeit Projekte im Kontextmenü des Saros-Views mit einem selektierten Buddy zu teilen.
Beteiligte Akteure (actors)
Primäre Akteure: Sitzungsteilnehmer
Verwendete Anwendungsfälle (includes)
keine
Auslöser (rationale oder trigger)
Der Anwender wählt ein entsprechendes Projekt im Kontextmenü eines Buddys aus.
Vorbedingungen (preconditions)
Saros installiert. Nutzer am System angemeldet. Sitzungsteilnehmer besitzen mindestens ein Projekt im Arbeitsbereich. Es ist mindestens ein Interaktionspartner verfügbar.
Invarianten
keine
Nachbedingung / Ergebnis (postconditions)
Ausgewähltes Projekt ist in den Sitzungskontext überführt worden und gemeinsam nutzbar.
Standardablauf (normal flow)
1. Anwender selektiert Buddy im Saros-View und klickt mit der rechten Maustaste auf ihn 2. Im aufkommenden Kontextmenü wählt er „*Work together on*“ und dann ein entsprechendes Projekt
Alternative Ablaufschritte (alternative flow)
Auswahl des Projektes im Project Explorer und dann die Selektion des entsprechenden Buddys im Ressourcen Kontextmenü Auswahl des Projektes im Projektauswahl Assistenten und Selektion des Buddys im darauf folgenden Assistenen
Hinweise
In der Saros-View sind nur Projekte auswählbar.

Name und Identifier
Ressourcenauswahl im Projektauswahl-Assistenten
Beschreibung (description)
Der Projektauswahl-Assistent von Saros, soll die Selektion von Ressourcen vereinfachen. Dazu muss er Ressourcen aussagekräftig darstellen und visuelle Unterstützung bieten. Der Anwender kann beliebige Ressourcen seiner Arbeitsumgebung selektieren und in eine Sitzung überführen.
Beteiligte Akteure (actors)
Primäre Akteure: Sitzungsteilnehmer
Verwendete Anwendungsfälle (includes)
keine
Auslöser (rationale oder trigger)
Der Anwender will bequem über den Projektauswahl-Assistenten die Selektion seiner Ressourcen vornehmen, um sie in den Sitzungskontext zu überführen.
Vorbedingungen (preconditions)
Saros installiert. Nutzer am System angemeldet. Sitzungsteilnehmer besitzen mindestens ein Projekt im Arbeitsbereich. Es ist mindestens ein Interaktionspartner verfügbar.
Invarianten
keine
Nachbedingung / Ergebnis (postconditions)
Ausgewählte Ressourcen des Assistenten sind in den Sitzungskontext überführt worden und gemeinsam nutzbar.
Standardablauf (normal flow)
1. Sitzungsteinehmer geht in die Menüleiste und wählt *„Saros“–>“Share Project(s)“* 2. Sitzungsteilnehmer wählt Ressourcen aus Baumstruktur des Assistenten aus
Alternative Ablaufschritte (alternative flow)
Auswahl über Kontextmenü der Saros-View Auswahl über den Project Explorer
Hinweise
keine

Name und Identifier
Ressourcenauswahl im Project Explorer
Beschreibung (description)
Der Project Explorer bietet die Möglichkeit für versierte Anwender schnell selektierte Ressourcen mit jemandem zu teilen. Der Anwender kann beliebige Ressourcen selektieren (z.B. durch drücken der STRG Taste) und in eine Sitzung überführen. Dazu wird das Kontextmenü des Project Explorers genutzt.
Beteiligte Akteure (actors)
Primäre Akteure: Sitzungsteilnehmer
Verwendete Anwendungsfälle (includes)
keine
Auslöser (rationale oder trigger)
Der Anwender will schnell und praktikabel die von ihm selektierten Ressourcen des Project Explorers mit anderen gemeinsam nutzen. Dazu wählt er im Ressourcen-Kontextmenü den oder die gewünschten Buddy(s) aus, mit denen er die Ressourcen teilen will.
Vorbedingungen (preconditions)
Saros installiert. Nutzer am System angemeldet. Sitzungsteilnehmer besitzen mindestens ein Projekt im Arbeitsbereich. Es ist mindestens ein Interaktionspartner verfügbar.
Invarianten
keine
Nachbedingung / Ergebnis (postconditions)
Ausgewählte Ressourcen des Project Explorer sind in den Sitzungskontext überführt worden und gemeinsam nutzbar.
Standardablauf (normal flow)
1. Sitzungsteinehmer selektiert Ressourcen im Project Explorer 2. Sitzungsteilnehmer wählt im Kontextmenü einer der Ressourcen *„Share With“* und dann entsprechend einen Buddy oder *„Multiple Buddies...“*
Alternative Ablaufschritte (alternative flow)
Auswahl über Kontextmenü der Saros-View Auswahl über den Projektauswahl Assistenten
Hinweise
keine

Name und Identifier
Beliebige Projekt/Ordner/Datei-Kombinationen in ein vorhandenes Projekt synchronisieren
Beschreibung (description)
Der Saros-Nutzer hat die freie Ressourcenauswahl aus seinem Arbeitsbereich. Diese Selektion kann problemfrei in den Sitzungskontext überführt und vom Eingeladenen in ein vorhandenes Projekt synchronisiert werden.
Beteiligte Akteure (actors)
Primäre Akteure: Eingeladene Sitzungsteilnehmer Sekundärer Akteur: Sitzungsinitiator
Verwendete Anwendungsfälle (includes)
Ressourcenauswahl im Projekt-Assistenten Ressourcenauswahl im Projekt-Explorer
Auslöser (rationale oder trigger)
Der Sitzungsinitiator wählt Ressourcen eines Projektes aus und will sie mit Sitzungsteilnehmern teilen.
Vorbedingungen (preconditions)
Saros installiert. Nutzer am System angemeldet. Sitzungsteilnehmer besitzen Projekt bereits im Arbeitsbereich. Sitzungsteilnehmer werden zu Sitzung eingeladen.
Invarianten
im Fehlerfall: Status loggen und Fehlerausgabe, Sitzung beenden
Nachbedingung / Ergebnis (postconditions)
Ausgewählte Ressourcen des Sitzungsinitiators sind „nahtlos“ in das Projekt der Sitzungsteilnehmer überführt worden. Das Projekt kann nun gemeinsam genutzte und sitzungsfremde Ressourcen beinhalten.
Standardablauf (normal flow)
1. Sitzungsinitiator wählt Ressourcen + Buddies aus und lädt diese zu einer Sitzung ein 2. Sitzungsteilnehmer erhalten Sitzungseinladung, bestätigen diese und bekommen Projekteinladungs-Assistenten 3. Sitzungsteilnehmer wählen bereits vorhandenes Projekt aus
Alternative Ablaufschritte (alternative flow)
keine
Hinweise
keine

Name und Identifier
Beliebige Projekt/Ordner/Datei-Kombinationen in ein nicht vorhandenes Projekt synchronisieren
Beschreibung (description)
Der Saros Nutzer hat die freie Ressourcenauswahl aus seinem Arbeitsbereich. Diese Selektion kann problemfrei in den Sitzungskontext überführt und vom Eingeladenen in ein neues Projekt synchronisiert werden.
Beteiligte Akteure (actors)
Primäre Akteure: Eingeladene Sitzungsteilnehmer Sekundärer Akteur: Sitzungsinitiator
Verwendete Anwendungsfälle (includes)
Ressourcenauswahl im Projektauswahl Assistenten Ressourcenauswahl im Project Explorer
Auslöser (rationale oder trigger)
Der Sitzungsinitiator wählt Ressourcen eines Projektes aus und will sie mit Sitzungsteilnehmern teilen.
Vorbedingungen (preconditions)
Saros installiert. Nutzer am System angemeldet. Sitzungsteilnehmer werden zu Sitzung eingeladen.
Invarianten
im Fehlerfall: Status loggen und Fehlerausgabe, Sitzung beenden
Nachbedingung / Ergebnis (postconditions)
Ausgewählte Ressourcen des Sitzungsinitiators sind als separate Projekt im Arbeitsbereich der Sitzungsteilnehmer vorhanden. Das Projekt kann nur gemeinsam genutzte Ressourcen beinhalten.
Standardablauf (normal flow)
1. Sitzungsinitiator wählt Ressourcen + Buddies aus und lädt diese zu einer Sitzung ein 2. Sitzungsteilnehmer erhalten Sitzungseinladung, bestätigen diese und bekommen Projekteinladungs-Assistenten 3. Sitzungsteilnehmer erzeugen neues Projekt
Alternative Ablaufschritte (alternative flow)
keine
Hinweise
keine

Name und Identifier
Beliebige Projekt/Ordner/Datei-Kombinationen in eine laufende Sitzung in vorhandenes Projekt synchronisieren
Beschreibung (description)
Jeder Sitzungsteilnehmer hat die freie Ressourcenauswahl aus seinem (ungeteiltem) Arbeitsbereich auch während einer Sitzung. Diese Selektion kann problemfrei in den Sitzungskontext übernommen und von allen anderen Sitzungsteilnehmern in ein vorhandenes Projekt übernommen werden.
Beteiligte Akteure (actors)
Primäre Akteure: Sitzungsteilnehmer
Verwendete Anwendungsfälle (includes)
Ressourcenauswahl im Projekt-Assistenten Ressourcenauswahl im Projekt-Explorer
Auslöser (rationale oder trigger)
Ein beliebiger Sitzungsteilnehmer wählt Ressourcen aus die er mit den anderen Sitzungsteilnehmern teilen will.
Vorbedingungen (preconditions)
Saros installiert. Nutzer am System angemeldet. Sitzungsteilnehmer besitzen Projekt bereits im Arbeitsbereich. Sitzungsteilnehmer sind in einer gemeinsamen Sitzung.
Invarianten
im Fehlerfall: Status loggen und Fehlerausgabe, Sitzung für Teilenden beenden
Nachbedingung / Ergebnis (postconditions)
Ausgewählte Ressourcen sind „nahtlos“ in das Projekt der Sitzungsteilnehmer überführt worden. Das Projekt kann nun gemeinsam genutzte und sitzungsfremde Ressourcen beinhalten.
Standardablauf (normal flow)
1. Sitzungsteilnehmer wählt Ressourcen aus und fügt diese der Sitzung hinzu 2. Sitzungsteilnehmer erhalten Projekteinladungs-Assistenten 3. Sitzungsteilnehmer wählen bereits vorhandenes Projekt aus
Alternative Ablaufschritte (alternative flow)
keine
Hinweise
Das vorhandene Projekt kann bereits gemeinsam genutzte Ressourcen enthalten oder aber komplett sitzungsfremd sein.

Name und Identifier
Beliebige Projekt/Ordner/Datei-Kombinationen in eine laufende Sitzung in nicht vorhandenes Projekt synchronisieren
Beschreibung (description)
Jeder Sitzungsteilnehmer hat die freie Ressourcenauswahl aus seinem (ungeteiltem) Arbeitsbereich auch während einer Sitzung. Diese Selektion kann problemfrei in den Sitzungskontext übernommen und von allen anderen Sitzungsteilnehmern in ein neues Projekt übernommen werden.
Beteiligte Akteure (actors)
Primäre Akteure: Eingeladene Sitzungsteilnehmer Sekundärer Akteur: Sitzungsinitiator
Verwendete Anwendungsfälle (includes)
Ressourcenauswahl im Projekt-Assistenten Ressourcenauswahl im Projekt-Explorer
Auslöser (rationale oder trigger)
Ein beliebiger Sitzungsteilnehmer wählt Ressourcen aus die er mit den anderen Sitzungsteilnehmern teilen will.
Vorbedingungen (preconditions)
Saros installiert. Nutzer am System angemeldet. Sitzungsteilnehmer sind in einer gemeinsamen Sitzung.
Invarianten
im Fehlerfall: Status loggen und Fehlerausgabe, Sitzung für Teilenden beenden
Nachbedingung / Ergebnis (postconditions)
Ressourcen sind als separates Projekt im Arbeitsbereich der Sitzungsteilnehmer vorhanden. Das Projekt kann nur gemeinsam genutzte Ressourcen beinhalten.
Standardablauf (normal flow)
1. Sitzungsteilnehmer wählt Ressourcen aus und fügt diese der Sitzung zu 2. Sitzungsteilnehmer erhalten Projekteinladungs-Assistenten 3. Sitzungsteilnehmer erzeugen ein neues Projekt
Alternative Ablaufschritte (alternative flow)
keine
Hinweise
keine

Name und Identifier
Bedarfsgerechte Dateisynchronisation während Projektsynchronisation
Beschreibung (description)
Möchte der Anwender während einer langwierigen Projektsynchronisation bereits gemeinsam an einzelnen Dateien arbeitn, so kann er auch während der Synchronisation Dateien priorisiert übertragen. Diese Möglichkeit ergebt sich wenn der Anwender während einer von ihm ausgehenden Projektsynchronisation ausgewählte Ressourcen editiert werden.
Beteiligte Akteure (actors)
Primärer Akteur: Ressourcenhost Sekundäre Akteure: Sitzungsteilnehmer
Verwendete Anwendungsfälle (includes)
Ressourcenauswahl im Projektauswahl Assistenten Ressourcenauswahl im Project Explorer Projektauswahl im Saros-View
Auslöser (rationale oder trigger)
Ressourcenhost beginnt während der laufenden Projektsynchronisation zu synchronisierende Dateien zu editieren.
Vorbedingungen (preconditions)
Saros installiert. Nutzer am System angemeldet. Es ist mindestens ein Interaktionspartner verfügbar.
Invarianten
keine
Nachbedingung / Ergebnis (postconditions)
Priorisiert übertragene Dateien sind vor den synchronisierten Dateien im Arbeitsbereich der Sitzungsteilnehmer vorhanden und gemeinsam nutzbar.
Standardablauf (normal flow)
1. Selektion von einem oder mehreren Projekten zur Synchronisation 2. Selektion eines oder mehrerer Buddies mit denen gemeinsam gearbeitet werden soll 3. Ressourcenhost fängt während des Übertragungsprozesses bereits an zu arbeiten
Alternative Ablaufschritte (alternative flow)
keine
Hinweise
Funktioniert nur auf vollständig zu übertragenden Projekten, d.h. keine partiellen Projekte.

Name und Identifier
Bedarfsgerechte Dateisynchronisation in partiell geteiltes Projekt
Beschreibung (description)
Der Ressourcenhost kann durch simples Editieren einer nicht gemeinsam genutzten Datei eines partiell geteilten Projektes dieses automatisch in den Sitzungskontext überführen.
Beteiligte Akteure (actors)
Primärer Akteur: Ressourcenhost Sekundäre Akteure: Sitzungsteilnehmer
Verwendete Anwendungsfälle (includes)
Ressourcenauswahl im Projektauswahl Assistenten Ressourcenauswahl im Project Explorer
Auslöser (rationale oder trigger)
Ressourcenhost editiert nicht geteilte Datei eines partiell geteilten Projektes.
Vorbedingungen (preconditions)
Saros installiert. Nutzer am System angemeldet. Sitzungsteilnehmer besitzen mindestens ein partiell geteiltes Projekt im Arbeitsbereich. Es ist mindestens ein Interaktionspartner verfügbar.
Invarianten
keine
Nachbedingung / Ergebnis (postconditions)
Eine vorher nicht gemeinsam genutzte Ressource ist nun dem partiell gemeinsam genutzten Projekt hinzugefügt worden und gemeinsam nutzbar.
Standardablauf (normal flow)
1. Übertragung eines oder mehrerer partieller Projekte an einen oder mehrere Buddies 2. Nach der Übertragung beginnt der Ressourcenhost in einer nicht geteilten Datei des partiell geteilten Projektes zu editieren 3. Ressourcenhost lässt die automatische Synchronisation zu im aufkommenden Abfragedialog
Alternative Ablaufschritte (alternative flow)
keine
Hinweise
Gilt nur für partiell geteilte Projekte.

B Modifikationen der *plugin.xml* Datei

Dieses Kapitel zeigt die Modifikationen der *plugin.xml* Datei und deren Auswirkungen. Codesegment 7.1 zeigt eine *Extension* der Eclipse Oberfläche um weitere *Decorator*. Dazu wird der Punkt der Erweiterung angegeben und als *Decorator* definiert. Die restliche Beschreibung umfasst den *Decorator* selbst. (vgl. (Bal03))

```
<extension
       point="org.eclipse.ui.decorators">
 <decorator
   adaptable="true"
   class="de.fu_berlin.inf.dpp.ui.decorators.SharedProjectDecorator"
   id="de.fu_berlin.inf.dpp.ui.decorators.SharedProjectDecorator"
   label="Shared Project Decorator"
   lightweight="true"
   state="true">
    <enablement>
       <objectClass name="org.eclipse.core.resources.IResource"/>
    </enablement>
 </decorator>
  ...
```

Codesegment 7.1: Aktivierung der *Decorator* auf allen Ressourcen

adaptable: Zeigt ob Klassen die auf entsprechende Ressourcen adaptierbar sind auch dekoriert werden sollen.

class: Klasse die den *Decorator* implementiert. Sie ist verantwortlich für die Kennzeichnung der Ressourcen mit eigenen Annotationen.

id: Definiert die ID des *Decorators*. Diese ID muss global einzigartig sein.

label: Definiert die Beschriftung des *Decorators* für die *Label Decorations Preference Page* der Eclipse Arbeitsumgebung.

lightweight: Definiert ob der *Decorator* „leichtgewichtig“ ist. Dabei handelt es sich um die Lösung von Problemen mit dem Bildmanagement die in Verbindung mit den *Decorators* stehen. *Lightweigt Decorators* verarbeitet das dekorieren in einem Hintergrund-Thread und blockiert somit nicht den UI-Thread.

state: Beschreibt den Standardzustand des *Decorators*. Diese kann aktiviert oder deaktiviert sein.

objectClass: Klasse der Ressourcen welche es zu dekorieren gilt.

Codesegment 7.2 beschreibt die Umsetzung einer Definition in Saros. Wie an dem *<extension>* Tag zu erkennen ist handelt es sich wiederum um eine Plugin-Erweiterung. Eine Definition ermöglicht beispielsweise eine dynamische Interaktion mit dem Plugin. In diesem Fall handelt es sich um die Überprüfung, ob es sich bei einem selektierten Element um eine Ressource handelt. Anhand dessen kann beispielsweise ein Menü angezeigt oder deaktiviert werden.

```
<extension
id="de.fu_berlin.inf.dpp.ui.definitions.Saros"
point="org.eclipse.core.expressions.definitions">
 <definition id="de.fu_berlin.inf.dpp.ui.definitions.isConnected">
    <with variable="de.fu_berlin.inf.dpp.Saros">
       <test property="de.fu_berlin.inf.dpp.isConnected">
       </test>
    </with>
 </definition>
 ...
```

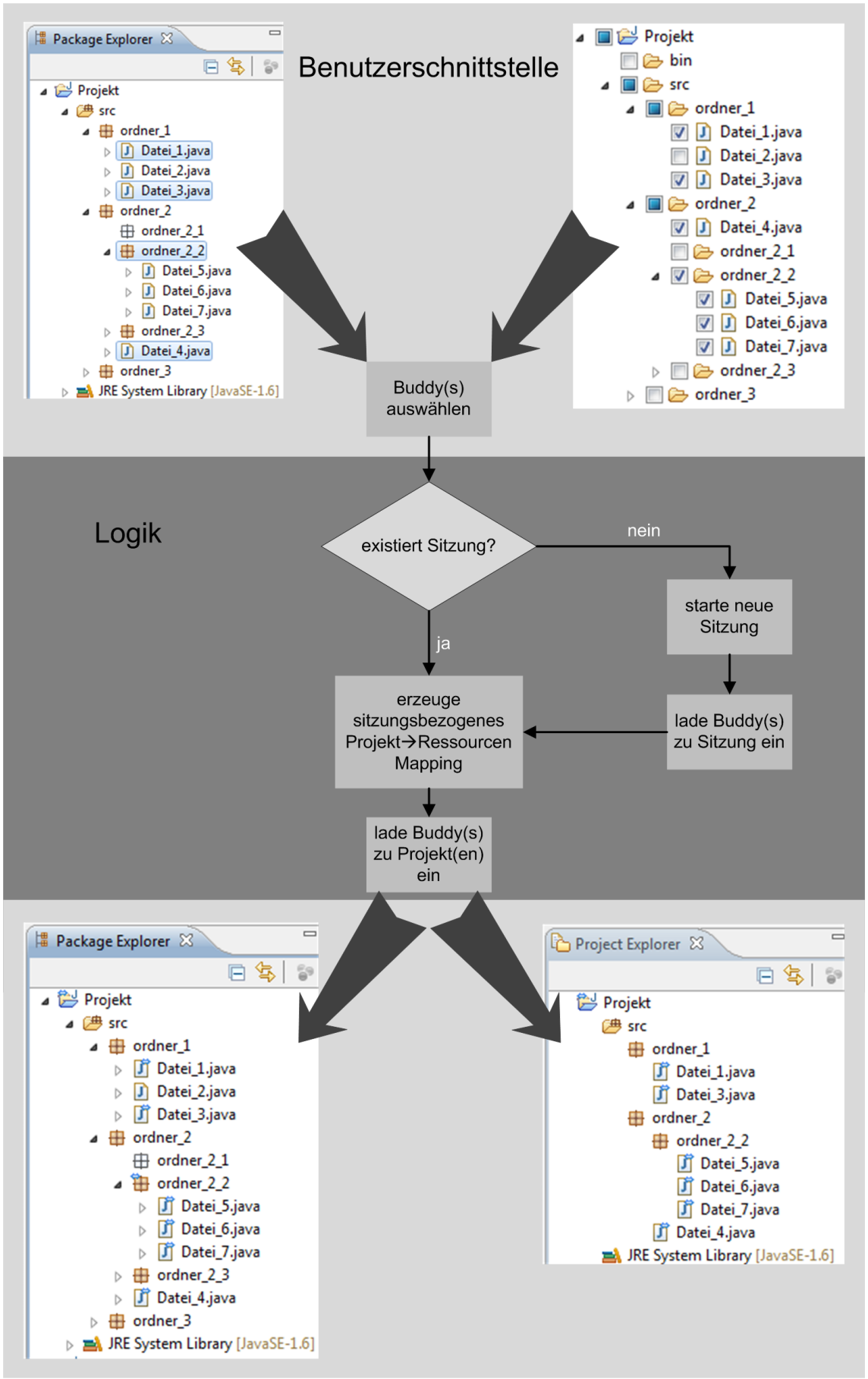
Codesegment 7.2: Definition in der *plugin.xml* Datei

Codesegment 7.3 zeigt zwei weitere neue Definitionen im Rahmen dieser Arbeit. In der oberen wird eine Adaption auf das *IResource* Objekt der Eclipse Umgebung gemacht. Das heißt in der Anwendung dieser Definition wird das gewählte Objekt versucht auf *IResource* abzugleichen. Bei Erfolg würde dies ein „*true*“ zurückliefern, andernfalls ein „*false*“.

```
<definition id="de.fu_berlin.inf.dpp.ui.definitions.isResource">
 <adapt type="org.eclipse.core.resources.IResource">
 </adapt>
</definition>
<definition
      id="de.fu_berlin.inf.dpp.ui.definitions.resourcesSelected">
 <with variable="selection">
  <iterate
        ifEmpty="false"
        operator="and">
   <reference
      definitionId="de.fu_berlin.inf.dpp.ui.definitions.isResource">
   </reference>
  </iterate>
 </with>
</definition>
```

Codesegment 7.3: Weitere Definitionen in der *plugin.xml* Datei

C Struktur der partiellen Projektsynchronisation



D AktivitätsObjekte

ChangeColorActivityDataObject	AktivitätsObjekt zur Farbänderung
ChecksumActivityDataObject	AktivitätsObjekt zur Prüfsummenübermittlung
ChecksumErrorActivityDataObject	AktivitätsObjekt zur Inkonsistenzauflösung
EditorActivityDataObject	AktivitätsObjekt für (Speichern/Aktivieren/Schließen) von Editorfenstern
FileActivityDataObject	AktivitätsObjekt zur Übertragung von Dateiaktivitäten
FolderActivityDataObject	AktivitätsObjekt zur Übertragung von Ordneraktivitäten
JupiterActivityDataObject	AktivitätsObjekt des Jupiter Algorithmus
PermissionActivityDataObject	AktivitätsObjekt der Rechtevergabe
PingPongActivityDataObject	AktivitätsObjekt zur Ermittlung von Übertragungszeiten
ProgressActivityDataObject	AktivitätsObjekt zur Kommunikation mit Sitzungsteilnehmern
ProjectsAddedActivityDataObject	AktivitätsObjekt zur Übernahme eines Projektes in die Sitzung
ProjectExchangeInfoDataObject	AktivitätsObjekt welches Projektinformationen zum Sitzungsaufbau enthält
StopActivityDataObject	AktivitätsObjekt zur Signalisierung von Start/Stop der Aktivitätsgenerierung
TextEditActivityDataObject	AktivitätsObjekt zur Übertragung von Textänderungen
TextSelectionActivityDataObject	AktivitätsObjekt zur Übertragung von Selektionen im Text
VCSActivityDataObject	AktivitätsObjekt für den VCS Support
ViewportActivityDataObject	AktivitätsObjekt für den Viewport des Editors

Tabelle 7.1: *AktivitätsObjekte* des Aktivitätsmanagements des Saros-Plugins