**Hochschule für Angewandte Wissenschaften Hamburg**

***Hamburg University of Applied Sciences***

# Bachelorarbeit

Marcus Kraßmann

Verteilte Paarprogrammierung mit Saros - Einführung und Integration in bestehende Entwicklungsprozesse

*Fakultät Technik und Informatik*
*Department Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

Marcus Kraßmann

# Verteilte Paarprogrammierung mit Saros - Einführung und Integration in bestehende Entwicklungsprozesse

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Bettina Buth
Zweitgutachter: Prof. Dr. Olaf Zukunft

Abgegeben am 3. August 2011

**Marcus Kraßmann**

**Thema der Bachelorarbeit**
Verteilte Paarprogrammierung mit Saros - Einführung und Integration in bestehende Entwicklungsprozesse
**Stichworte**
Software, Eclipse, Saros, verteilte Paarprogrammierung, Entwicklung, agil
**Kurzzusammenfassung**
Im Kontext standortunabhängiger Teams und agiler Softwareentwicklung stellt sich die Frage, inwiefern die Technik der Paarprogrammierung sinnvoll von räumlich voneinander getrennten Entwicklern eingesetzt werden kann. Im Rahmen dieser Arbeit wird das Eclipse-Plugin Saros, welches verteilte Paarprogrammierung ermöglicht, in die Infrastruktur und Entwicklungsprozesse der Firma Vattenfall Europe Information Services GmbH integriert. Für die Einführung ins Entwicklerteam wird ein Konzept erstellt. Der Erfolg der Integration wird anhand vordefinierter Kriterien bewertet.

**Marcus Kraßmann**

**Title of the paper**
Distributed Pair Programming with Saros - Introduction and integration in existing development processes
**Keywords**
Software, Eclipse, Saros, Distributed Pair Programming, Development, agile
**Abstract**
In the context of location independent teams and agile software development, the question of how to sensibly use the pair programming technique for developers in different locations arises. Within the scope of this thesis the Eclipse plugin Saros which facilitates distributed pair programming will be integrated into the infrastructure and development processes of the company Vattenfall Europe Information Services GmbH. There will be a concept for the introduction into the development team developed. The result of the integration will be assessed based on established criteria.

# Inhaltsverzeichnis

# 1 Einleitung

Überall auf der Welt wird Software entwickelt. Entwickler erstellen auf Basis von Anforderungen Programmcode, der am Ende eine hoffentlich lauffähige Software ergibt. Arbeiten zwei oder mehr Entwickler an derselben Software, entstehen verschiedene Risiken:

- Die Entwickler haben unterschiedliche Zielvorstellungen und entwickeln aneinander vorbei.
- Einzelne Entwickler werden zu Experten ihres Codes. Sie kennen nur den eigenen Programmcode, kennen aber den Code der anderen Entwickler nicht.
- Entwickler wenden völlig unterschiedliche Programmiertechniken an. Der Code ist nicht homogen, so dass ein Einarbeiten neuer Entwickler zusätzlich erschwert wird.
- Fremderzeugter Programmcode lässt sich in der eigenen Umgebung aufgrund verschiedener Entwicklungsumgebungen nicht starten.

Auch wenn das Entwicklerteam aus nur einem Mitglied besteht, existieren alle zuvor genannten Risiken, und zwar sobald der Entwickler durch jemand anderen ausgetauscht werden muss. In diesem Fall ist der gesamte Programmcode von den genannten Problemen betroffen. Bei wachsender Anzahl produzierter Softwareanwendungen werden Unternehmen mehr und mehr von der Anwesenheit jedes einzelnen Mitarbeiters abhängig. Urlaubsregelungen werden komplizierter. Niemand möchte den Support für die Anwendungen abwesender Kollegen übernehmen - nicht aus Angst vor dem Programmierstil des Autors, sondern aus dem Wissen, dass man im Supportfall mit hoher Wahrscheinlichkeit nichts bewirken kann, so lange der Autor nicht wieder da ist. Die innerbetrieblichen Vorgänge werden mit der Zeit immer starrer; die Flexibilität sinkt mit jedem neuen Ein-Mann-Projekt.

> *Bus factor*
>
> „[...] a measurement of the concentration of information in a single person, or very few people. The bus factor is the total number of key developers who would need to be incapacitated, (as by getting hit by a bus) to send the project into such disarray that it would not be able to proceed.” (Wikipedia, 2011a)

## 1.1 Motivation

Der europäische Stromkonzern Vattenfall Europe AG ist ein expandierendes Unternehmen, welches insbesondere in den letzten Jahren stark gewachsen ist. 2002 übernahm Vattenfall die Hamburgische Electricitäts-Werke AG [sic] (Vattenfall, 2011). 2009 erfolgte die Übernahme der niederländischen Firma N.V. Nuon Energy. Die erfolgte Expansion hat auch Einfluss auf die Vattenfall Europe Information Services GmbH (VEIS). Dort existieren mittlerweile zentrale IT-Standorte in Schweden, Deutschland, Polen und den Niederlanden. Der dadurch resultierende Wechsel weg von rein zentraler hin zu über Ländergrenzen hinweg verteilter Softwareentwicklung stellt die konzerninternen IT-Prozesse vor neue Herausforderungen. An verschiedenen Standorten arbeiten nun Expertengruppen vergleichbarer Problemdomänen, deren technologisches Know-how sich jedoch teilweise deutlich voneinander unterscheidet. Diese Situation sowie die räumliche Distanz zwischen den Entwicklergruppen hat verschiedene Auswirkungen. Einzelne Gruppen verwenden die ihnen vertrauten Technologien, um Lösungen zu erstellen. Entwickler anderer Teams verstehen den entstandenen Programmcode dann oft nicht oder nur unvollständig. Arbeiten Teams verschiedener Standorte gemeinsam an Projekten, wird auf Quellcodeebene oftmals ersichtlich, dass vergleichbare Problemstellungen in den verschiedenen Gruppen unterschiedlich gelöst werden. Der resultierende Quellcode unterscheidet sich je nach Entwicklerteam stark. Die bisher vom Softwareentwicklungsteam angewendeten Entwicklungsprozesse wurden bisher nicht auf eine derartige Zusammenarbeit hin optimiert. Das gilt ebenso für die Prozesse der neu hinzugekommenen Teams.

Aus diesem Grund äußerte das Management den Wunsch, die Entwicklungsprozesse der verschiedenen Teams nach und nach zu vereinheitlichen. Das vorrangige Ziel ist, dass Softwareentwickler aus verschiedenen Ländern mit möglichst geringem Aufwand effektiv zusammenarbeiten können.

Diese Arbeit untersucht, ob und wie mit Hilfe der von Studenten der Freien Universität Berlin entwickelten Softwarelösung *Saros* die zuvor genannten Probleme gelöst werden können. Saros ermöglicht es Softwareentwicklern, an verschiedenen Standorten gleichzeitig gemeinsam an derselben Codebasis live zu arbeiten und dabei die Aktionen der anderen Teilnehmer unmittelbar mitverfolgen zu können. Der Schwerpunkt der Untersuchung liegt auf der Fragestellung, ob und wie sich verteilte Paarprogrammierung sowie verteilte Code-Reviews durch den Einsatz von Saros sinnvoll in bestehende Softwareentwicklungsprozesse einbinden lassen.

## 1.2 Gliederung der Arbeit

Diese Arbeit ist im Wesentlichen in fünf Teile aufgespalten, deren Inhalt an dieser Stelle kurz genannt werden sollen.

Kapitel 2 Grundlagen widmet sich der Erläuterung von Begriffen und Zusammenhängen, die für das Verständnis dieser Arbeit vorausgesetzt werden. Neben der Einführung verschiedener Vorgehensmodelle (Abschnitt 2.1) wird der Leser nach der Einführung in die Begriffe Integrierte Entwicklungsumgebung (Abschnitt 2.2), Paarprogrammierung (Abschnitt 2.3) und Verteilte Softwareentwicklung (Abschnitt 2.4) in das Thema Unterstützung für räumlich verteilte Paarprogrammierung (Abschnitt 2.5) einführt.

Kapitel 3 dient dazu, die Problemstellung genauer zu analysieren. Dafür werden zuerst die Rahmenbedingungen bei der VEIS genannt (Abschnitt 3.1). Anschließend werden die angestrebten Ziele definiert (Abschnitt 3.2), die mit Hilfe von Saros erreicht werden sollen. Aus den Zielen werden dann Anforderungen erstellt (Abschnitt 3.3). Drei möglichen Anwendungsszenarien (Abschnitt 3.4) und eine Bestimmung der Realisierungsreihenfolge (Abschnitt 3.5) beenden das Kapitel.

Kapitel 4 befasst sich mit der technischen Integration von Saros in die bestehende technische Infrastruktur. Zunächst werden die relevanten Systeme identifiziert (Abschnitt 4.1). Anschließend wird eine allgemein gehaltene konzeptionelle Planung durchgeführt (Abschnitt 4.2). Die Realisierung (Abschnitt 4.3) nennt dann die konkret durchzuführende Schritte und dokumentiert die Ergebnisse der praktischen Umsetzung. Ein Fazit (Abschnitt 4.4) fasst das Ergebnis der technischen Realisierung zusammen.

Kapitel 5 erarbeitet ein Konzept, wie den betroffenen Entwicklern der Sinn von verteilter Paarprogrammierung und die Bedienung von Saros verständlich gemacht werden. Nach einer kurzen Analyse der Ausgangssituation (Abschnitt 5.1) und der Nennung der treibenden Kraft hinter der Prozessänderung (Abschnitt 5.2) wird ein Konzept für eine Einführungsveranstaltung vorgelegt (Abschnitt 5.3). Im Fazit (Abschnitt 5.4) wird das Kapitel kurz zusammengefasst.

Kapitel 6 bewertet die erarbeitete Lösung anhand der in der Analyse aufgestellten Anforderungen (Abschnitt 6.1). Das Fazit (Abschnitt 6.2) fasst die Bewertung zusammen.

In Kapitel 7 wird diese Arbeit mit einer Zusammenfassung und einem Fazit (Abschnitt 7.1) sowie einem Ausblick auf mögliche weiterführende Untersuchungen und Arbeiten (Abschnitt 7.2) abgeschlossen.

# 2 Grundlagen

In diesem Kapitel werden die für diese Arbeit relevanten Begriffe erläutert.

## 2.1 Vorgehensmodelle

Auf dem Weg von der Produktidee bis zur fertigen Software gibt es eine Vielzahl an Tätigkeiten, die durchgeführt werden müssen. Angefangen bei der Sammlung von Wünschen und Anforderungen wird das weitere Vorgehen geplant, Ideen priorisiert, Programmiertätigkeiten durchgeführt, Rückmeldung eingeholt und das fertige Produkt umfassend getestet und an den Kunden ausgeliefert. Diese Schritte können auf verschiedene Weise zueinander in Bezug gesetzt werden. Die Anordnung der für die Entwicklung eines Produkts notwendigen Schritte in eine bestimmte Reihenfolge wird Vorgehensmodell genannt. Im Bereich der Softwareentwicklung haben sich über die Jahrzehnte hinweg verschiedene Vorgehensmodelle herauskristallisiert, welche einen besonderen Stellenwert in der Softwarebranche eingenommen haben.

Für eine Einordnung werden im Folgenden drei ausgewählte Vorgehensmodelle vorgestellt. Begonnen wird mit einem der berüchtigsten Modelle: Dem *Wasserfallmodell* (Unterabschnitt 2.1.1). Das *V-Modell* (Unterabschnitt 2.1.2) zeigt eine Erweiterung des Wasserfallmodells, welche sich durch ein hohes Maß an Modellspezifikation auszeichnet. Zum Schluss wird dann das derzeit sehr populäre iterative Modell *Scrum* (Unterabschnitt 2.1.3) präsentiert. Für den Einsatz von Paarprogrammierung (Abschnitt 2.3) verspricht der Einsatz eines iterativen Vorgehensmodells den größten Mehrwert.

### 2.1.1 Wasserfallmodell

Das wohl bekannteste und gleichzeitig älteste Vorgehensmodell ist das Wasserfallmodell. Es bietet ein einfach zu verstehendes Vorgehen, anhand dessen der Softwareentwicklungsprozess strukturiert werden kann. Es gibt wie in Abbildung 2.1 dargestellt fünf disjunkte Phasen, welche eine nach der anderen vollständig durchlaufen werden. Jede Phase endet mit einer expliziten Abnahme sowie einem oder mehreren Dokumenten, welche die Abnahme dokumentieren. Damit endet die aktuelle Phase und die nächste beginnt. Ein späterer Rücksprung in eine frühere Phase ist im klassischen Wasserfallmodell nicht möglich.

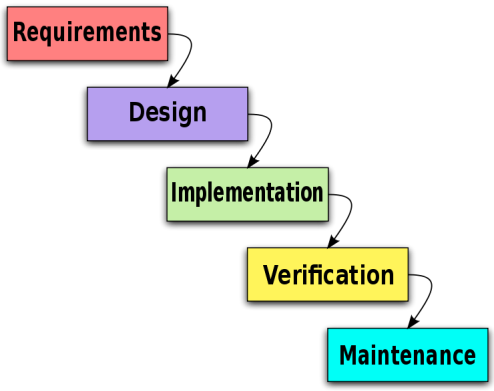


Abbildung 2.1: Wasserfallmodell aus Wikipedia (2011b)

#### 2.1.1.1 Phasen

Initial findet die erste grundlegende **Planung** des Systems statt. Auf Basis eines ersten, noch groben Kundenkonzepts wird die grundsätzliche Mach- bzw. Umsetzbarkeit untersucht. Ist diese gewährleistet, wird vom Kunden ein Lastenheft eingefordert. Auf dessen Basis erstellen die Softwarearchitekten ihrerseits ein Pflichtenheft, welches im weiteren Projektverlauf den Status eines Vertrags zwischen Kunde und Entwickler hat. Die dort festgeschriebenen Punkte definieren unter anderem die Abnahmekriterien für das zu erstellende System.

Nun wird auf Basis der vorliegenden Dokumente das **Design** des geplanten Systems entworfen. Dazu wird es durch Softwarearchitekten in mehrere kleinere Module zerteilt. Durch diese Strukturierung werden die anfangs noch unübersichtlichen Aufgaben in beherrschbare kleinere Einheiten unterteilt, welche anschließend verschiedenen Teams zugewiesen werden. Diese führen dann abschließend eine weitere Zerlegung ihrer Module in Komponenten durch. Jede Komponente sollte dabei idealerweise von einer Einzelperson bearbeitet werden können.

Ist die Designphase abgeschlossen, beginnt jeder Programmierer mit der **Implementierung** seiner Komponenten. Eine Komponente gilt als implementiert, sobald sie den Anforderungen des Pflichtenhefts genügt. Aus diesem Grund muss ein gutes Pflichtenheft das Zielsystem umfassend beschreiben, so dass die Entwickler es verwenden können, um den Fertigstellungsgrad der eigenen Komponenten beurteilen zu können.

Ist das System fertig implementiert, erfolgt die **Verifikation** des Systems anhand mehrerer Tests. Zuerst müssen die entwickelten Komponenten getestet werden. Dies sollte bereits in der Implementierungsphase geschehen sein. Anschließend erfolgt die Integration der verschiedenen Komponenten und Module, um nach und nach immer größer werdende funktionierende Teilsysteme zu erhalten. Ist die Integration abgeschlossen, wird das Gesamtsystem im vorletzten Schritt einem vollständigen Systemtest unterzogen. Wurde auch dieser erfolg-

reich bestanden, findet seitens des Kunden ein abschließender Akzeptanztest statt.

Die letzte Phase ist der **Betrieb** der Software, beginnend mit deren Auslieferung. Ab diesem Zeitpunkt kann das System aus Sicht des Kunden produktiv verwendet werden.

#### 2.1.1.2 Bewertung

Das Vorgehen bei Anwendung des Wasserfallmodells ist sehr methodisch, gut strukturiert und entspricht dem allgemeinen Verständnis von Planung und Umsetzung. Durch die detaillierte schriftliche Fixierung von Kunden- und Entwicklerzusagen ist mit Beginn der Designphase, also bereits zu einem recht frühen Stadium, für den Kunden klar, dass (und zu einem gewissen Teil auch wie) seine derzeitigen Anforderungen umgesetzt werden. Da erst die Anforderungen vollständig dokumentiert werden, eignet sich das Wasserfallmodell beispielsweise gut für Projekte, bei denen eine öffentliche Ausschreibung erforderlich ist.

Es gibt allerdings auch eine Reihe von Schwierigkeiten, welche beim Wasserfall strukturbedingt auftreten können. So setzen die initiale Planung und der daraus resultierende Umsetzungsvertrag voraus, dass vor Beginn der Designphase bereits sämtliche Anforderungen an das geplante System vollständig bekannt sind. Nachträgliche Änderungen der Anforderungen sind nicht innerhalb des Entwicklungsprozesses möglich, sondern müssen im Rahmen eines Folgeauftrags angegangen werden. Das kann dazu führen, dass die Software nicht das tut, was der Benutzer von ihr erwartet (Sommerville, 2007). Eine weitere Schwierigkeit ist das Erstellen einer lückenlosen, widerspruchsfreien Architektur, in der die Kommunikation aller miteinander interagierenden Komponenten einwandfrei spezifiziert ist. Werden hier Ungenauigkeiten nicht erkannt, werden im Laufe der Implementierung möglicherweise zueinander inkompatible Komponenten entwickelt und der erfolgreiche Projektabschluss ist gefährdet. Nicht zuletzt bedarf es seitens der Architekten eines immensen Expertenwissens, welches eine vollständige Vorabplanung ermöglicht.

Zusammengefasst ist das Wasserfallmodell ein strukturiertes Vorgehensmodell, welches für Projekte geeignet sein kann, die entweder eine überschaubare Architektur haben oder für die eine vollständige Vorabspezifikation sinnvoll und auch möglich ist.

### 2.1.2 V-Modell

Das V-Modell ist der Entwicklungsstandard für IT-Systeme des Bundes. Der Name lässt sich auf die Anordnung der einzelnen Phasen zurückführen und liegt aktuell in der Version XT 1.3 vor. Basierend auf dem V-Modell 97 wird es vom Bundesamt für Informationsmanagement und Informationstechnik der Bundeswehr (IT-AmtBw) A5 und dem Bundesministerium des Innern (BMI)-KBSt weiterentwickelt. Abbildung 2.2 veranschaulicht das V-Modell. Im

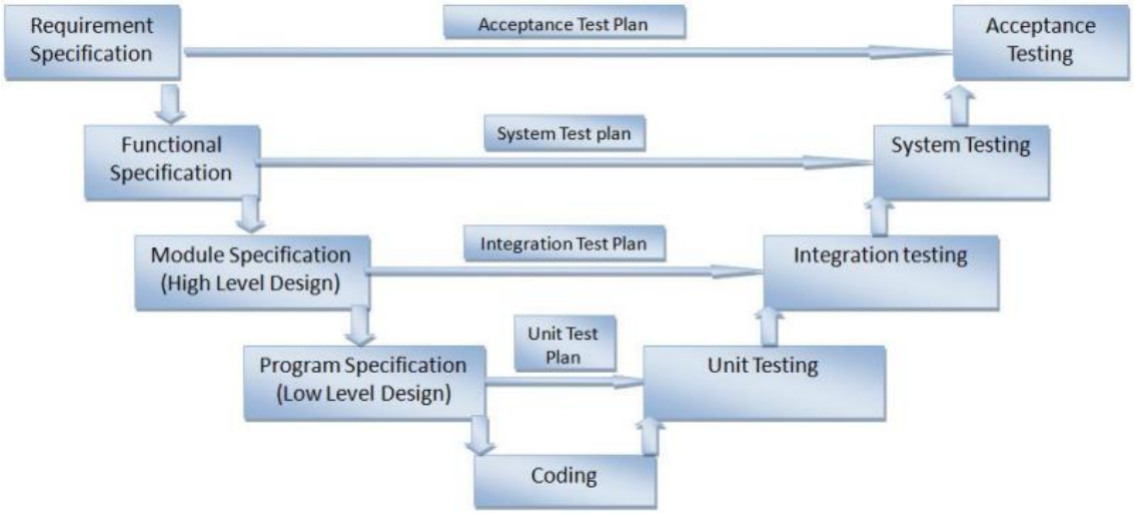


Abbildung 2.2: V-Modell(agile42, 2009)

Gegensatz zum Wasserfallmodell, bei dem die Phase der Verifikation eine der Implementierung nachgelagerte Phase ist, sieht das V-Modell bereits während der Planungsphasen die Erstellung entsprechender Testpläne vor. Dadurch ist bereits vor Beginn der Implementierungsphase definiert, welche konkreten Abnahmetests die Software durchlaufen muss.

#### 2.1.2.1 Phasen

Das V-Modell beinhaltet Verifikations- und Validierungsphasen. Während der Verifikationsphasen werden wobei zu jeder Verifikationsphase genau eine Validierungsphase gehört. Die Implementierung erfolgt nach Abschluss der Verifikationsphasen. Im Anschluss wird die Software in den Validierungsphasen gemäß der erzeugten Testdokumente getestet und bei Bestehen freigegeben.

**Verifikation** In der ersten Phase werden die Anforderungen, die der Kunde an das geplante System hat, erfasst. Im nächsten Schritt wird der funktionale Umfang spezifiziert. Anschließend folgt in zwei Phasen äquivalent zum Wasserfallmodell der Entwurf des Systemdesigns. Mit Beginn der Programmierung endet die Phase der Verifikation, und die Phase der Validierung beginnt.

**Validierung** Die Systemvalidierung findet in umgekehrter Richtung wie die Verifikation statt. Als erstes werden die kleineren Softwarekomponenten in Form von Unittests gegen die Spezifikation getestet. Anschließend erfolgen Integrationstests auf Basis der High-Level-Design-Dokumente, die das korrekte Zusammenspiel der einzelnen Module gewährleisten. Sobald

diese vollständig korrekt ablaufen, findet ein Systemtest statt, welcher sicherstellt, dass die festgelegten funktionalen Anforderungen korrekt umgesetzt sind. Abschließend testet der Kunde das System auf die Vollständigkeit der initial genannten Anforderungen. Mit Bestehen der Akzeptanztests endet der Entwicklungsprozess des Produkts. Eine Weiterentwicklung ist dann im Rahmen von neu zu definierenden Change Requests möglich.

#### 2.1.2.2 Bewertung

Das V-Modell unterstützt durch die eineindeutige Verknüpfung der Verifikations- und Validierungsphasen das Erstellen eines Testfallkatalogs auf Basis der Planungsdokumente für jede einzelne Phase besonders gut. BMI (2009) bietet eine umfassende Dokumentation an, wie das V-Modell XT anzuwenden ist. Entgegen dem recht einfachen Grundschema definiert das Modell die Rollen Projektmanagement, Qualitätssicherung, Konfigurationsmanagement sowie Problem- und Änderungsmanagement. Für jede Phase ist exakt spezifiziert, welche formalen Kriterien erfüllt sein müssen, um dem Modell zu genügen. Daher ist das V-Modell XT ein sehr formales Modell, welches sich aufgrund der zugrunde liegenden Zielsetzung besonders für große bis sehr große Softwareprojekte eignet, insbesondere solche, bei denen eine vorherige öffentliche Ausschreibung erforderlich ist. Für kleinere und mittlere Projekte ist dieses Vorgehensmodell jedoch zu aufwändig. Außerdem bietet auch das V-Modell keine echte Möglichkeit an, die zu entwickelnde Software iterativ zu entwickeln. Projekte, bei denen die Produktvision noch unscharf ist, benötigen daher einen anderen Ansatz wie z.B. Scrum.

### 2.1.3 Scrum

Es existieren Projekte, in denen die Produktanforderungen nicht im Voraus in Form eines vollständigen Lasten- bzw. Pflichtenheftes erfasst und dokumentiert werden können. Die gewünschte Grundfunktionalität ist zwar bekannt, aber zu den Details kann oder möchte sich der Kunde wegen bestehender Unsicherheiten nicht im Vorfeld festlegen. Ein Phasenmodell scheidet daher aus. Für solche Projekte bietet sich ein Vorgehensmodell an, bei dem das Produkt inkrementell entwickelt wird. Das bedeutet, dass das Produkt gerade soweit gebaut wird, dass es vom Kunden für erste Funktionstests verwendet werden kann. Dabei entstehen dann neue, detailliertere Wünsche, die dann in einem Folgezyklus umgesetzt werden. Auf diese Weise entwickelt sich das Produkt Schritt für Schritt zusammen mit den Wünschen des Kunden. Das derzeit bekannteste und populärste Vorgehensmodell für die Umsetzung iterativer Softwareentwicklung ist *Scrum*.

Scrum definiert die Rollen Product Owner, Team sowie Scrum Master. Wie genau Scrum funktioniert und welche Aufgaben die Rollen übernehmen, fasst ScrumAlliance (2011) wie folgt zusammen:

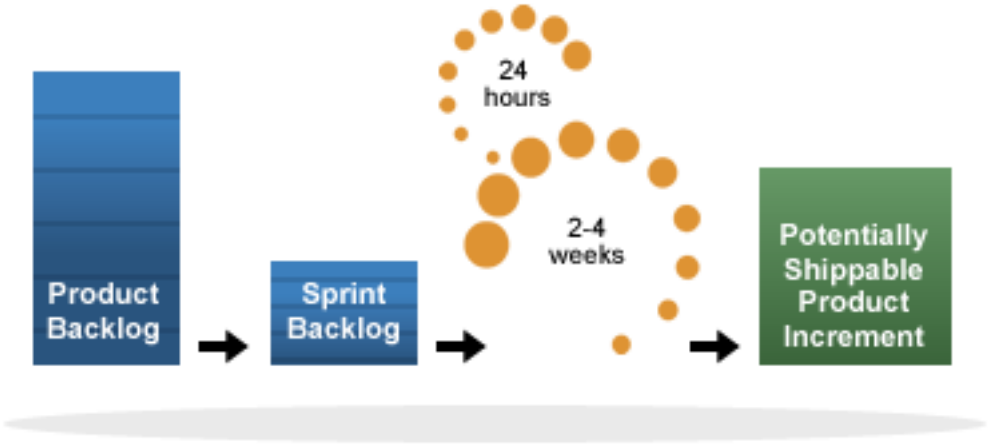


Abbildung 2.3: Der Scrumprozess nach ScrumAlliance (2011)

- Der *Product Owner* erzeugt (meistens in Form von Karteikarten) eine priorisierte Wunschliste. Dieses *Product Backlog* ist eine Sammlung aller dem Kunden bekannten Anforderungen, die er an sein Produkt hat. Es ersetzt das Lastenheft und ist zu einem gewissen Grad jederzeit dynamisch änderbar.
- Während der sogenannten *Sprintplanung* verschiebt das *Entwicklerteam* einige der hochpriorisierten Anforderungen in das *Sprint Backlog* und entscheidet, wie diese umgesetzt werden.
- Im nun folgenden *Sprint* versucht das Team, die herausgesuchten Anforderungen in einem zuvor festgelegten Zeitraum von zwei bis vier Wochen technisch zu realisieren. Während eines Sprints gibt es täglich ein Stand-up Meeting namens *Daily Scrum*, wo jeder Entwickler kurz sagt, was er am vergangenen Tag getan hat, woran er den aktuellen Tag arbeiten wird und ob es Probleme gibt, die ihn bei seiner Arbeit behindern.
- Während der gesamten Zeit sorgt der *Scrum Master* dafür, dass das Team störungsfrei arbeiten kann und das Ziel nicht aus den Augen verliert.
- Am Ende eines Sprints sollte ein neues, potenziell auslieferbares Produktinkrement existieren, welches die zu Beginn des Sprints ausgewählten Anforderungen erfüllt.
- Der Sprint endet mit einem *Sprint Review* und *Retrospektive*.
- Anschließend beginnt der nächste Sprint mit der nächsten Sprintplanung.

Im Gegensatz zum Phasenmodell erfolgt die Vorausplanung bei Scrum zunächst einmal nur für die Dauer eines Sprints. Es muss also vor einem Sprint keine vollständige Anforderungsliste geben - es müssen nur genügend Anforderungen bekannt sein, um diese innerhalb einer Iteration umsetzen zu können. Scrum bietet also einen strukturierten Weg, iterativ sein Wunschprodukt entwickeln zu lassen - wenn genügend Finanzierungsmittel zur Verfügung stehen. Genau hier haben iterative Vorgehensmodelle ihre Schwachstelle: Vorherige

Kostenschätzungen sowie die Dauer der Gesamtentwicklung sind aufgrund der fehlenden spezifizierten Anforderungen nur in begrenztem Umfang durchführbar. Für dieses Problem gibt es aber verschiedene Ansätze. So stellt beispielsweise Coldewey (2011) ein Modell vor, welches in iterativen Modellen längerfristige Zukunftsprognosen erlaubt.

Insgesamt zeichnen sich iterative Vorgehensmodelle durch eine hohe Nützlichkeit der erzeugten Produkte aus. Scrum mit seinem schlanken Regelwerk erlaubt es den Kunden, während der laufenden Entwicklung Einfluss auf die zu implementierenden Features zu nehmen. Dadurch werden Risiken möglicher Fehlkonzeptionen minimiert und die Chance, am Ende das zu erhalten, was man möchte, steigt. Nachteilig anzumerken ist die fehlende Planungsgrundlage vor Projektbeginn. Hier haben Phasenmodelle eine bessere Position. Allerdings bekommt der Kunde dort das, was er spezifiziert. Das kann am Ende aber etwas anderes sein als das, was er später tatsächlich möchte.

## 2.2 Integrierte Entwicklungsumgebung

Fundamentaler Teil der Softwareentwicklung ist das Programmieren. Je nach Programmiersprache unterscheidet sich Syntax und Semantik des Quellcodes. Allen Sprachen gemeinsam ist aber, dass der geschriebene Quellcode in der Regel menschenlesbar ist[1]. Der Programmierer schreibt dabei den Quellcode der zu entwickelnde Anwendung mit Hilfe einer geeigneten Software nieder. Im einfachsten Fall ist dies ein normaler Texteditor, welcher den Programmierer mit einfachen Funktionen wie dem Kopieren und Einfügen von Textblöcken unterstützt. Mit einem solchen Editor kann man durchaus Programme mit einer überschaubaren Größe schreiben. Mit wachsendem Umfang und beginnender Modularisierung wird es jedoch zunehmend schwieriger, den Programmcode zu warten, da fortgeschrittene Funktionen wie das Umbenennen von Funktionen über Dateigrenzen hinweg fehlen. Daher gibt es spezielle Editoren, die auf die besonderen Bedürfnisse von Programmierern ausgelegt sind. Solche Editoren werden mit dem Begriff Integrierte Entwicklungsumgebung (Integrated Development Environment, IDE) bezeichnet.

Um einen Überblick darüber zu erhalten, was eine IDE dem Entwickler an Vorteilen bietet, erfolgt hier eine unvollständige Auflistung nützlicher Zusatzfunktionen.

**Syntaxhervorhebung** bereitet den im Editor angezeigten Quellcode optisch auf. Dadurch wird es für den Programmierer einfacher, die Struktur sowie die einzelnen Anweisungen und Terme des Programmcodes zu erkennen. Die meisten IDEs verwenden bei der Syntaxhervorhebung nicht nur die der Sprache zugrunde liegende Grammatik, sondern beziehen auch semantische Aspekte mit ein. Dadurch können beispielsweise Aufrufe globaler Variablen anders dargestellt werden als Aufrufe lokaler Variablen. Ab-

[1] Ausnahmen sind die Programmiersprachen „Whitespace” und „Brainfuck”.

bildung 2.4 zeigt Syntaxhervorhebung am praktischen Beispiel einer einfachen Java-Klasse.

**Refactorings** sind Änderungen der internen Struktur von Programmen, um sie verständlicher und wartungsfreundlicher zu machen, ohne dass sich das beobachtbare Verhalten der veränderten Codefragmente ändert (Fowler, 1999). Ein Refactoring ist ein immer wiederkehrender Prozess beim Programmieren. Einige mögliche Maßnahmen sind das Umbenennen von Symbolnamen, das Extrahieren von Codefragmenten in eigene Module sowie das Verschieben von Symbolen in neue Module. Moderne IDEs bringen Unterstützung für die gebräuchlichsten Refactorings mit, denn ohne eine solche Unterstützung sind Umstrukturierungen aufwändig und fehleranfällig. Änderungen öffentlicher Symbolnamen müssen in allen das Symbol referenzierenden Modulen ebenfalls angepasst werden. Werden Module übersehen, so funktionieren diese Teile nicht mehr ordnungsgemäß. Ein globales Suchen und Ersetzen des Symbolnamens wiederum ändert möglicherweise Symbolnamen in Modulen, die nichts mit dem zu ändernden Symbol zu tun haben. Die Verwendung von IDE-Funktionen für Refactorings sorgt dafür, dass beide Probleme vermieden werden.

**Debugger** unterstützen bei auftretenden Fehlern, die Ursache im Quelltext aufzuspüren. IDEs bieten dazu die Möglichkeit, auf die Prozesssteuerung in der Ausführungsumgebung des Programms Einfluss zu nehmen. Dazu wird im Editor ein sogenannter Haltepunkt für eine Programmanweisung gesetzt. Die Ausführung des Programms wird pausiert, sobald diese Anweisung ausgeführt werden soll. Der Entwickler kann dann den internen Zustand des Programms analysieren und die Programmausführung Zeile für Zeile fortsetzen.

Für alle genannten Funktionalitäten gilt, dass sie in der Regel nur für eine spezielle Programmiersprache gelten, da jede Sprache eine eigene Grammatik sowie verschiedene spezifische Eigenschaften hat. Besonders verbreitete IDEs halten für diese Zwecke bereits Schnittstellen vor, über die für verschiedene Sprachen die benötigten Funktionen programmiert werden können. Dadurch sind Aussehen und Bedienung für verschiedene Programmiersprachen weitestgehend gleich.

### 2.2.1 Eclipse

Eine im Javaumfeld sehr verbreitete IDE ist die quelloffene Software *Eclipse*[2] (engl. „Sonnenfinsternis"). Ins Leben gerufen wurde das Eclipse-Programm von der Firma IBM mit dem Ziel, die zu dieser Zeit zueinander inkompatiblen monolithischen Entwicklungsumgebungen durch eine neue, flexibel erweiterbare Entwicklungsplattform abzulösen. Allerdings war der erste Prototyp unbekannt, so dass die meisten Java-Entwickler bewährte IDEs wie „Visual

[2] URL: http://www.eclipse.org/

Age for Java" oder Borlands „JBuilder" den Vorzug gaben. Daher wurde im November 2001 das Eclipse-Projekt von einer Gruppe bestehend aus IBM und acht weiteren Firmen, welche zusammen das initiale Eclipse-Konsortium gründeten, unter einer Open Source-Lizenz veröffentlicht mit dem Ziel, den Bekanntheitsgrad und die Akzeptanz der IDE zu erhöhen (Cernosek, 2005). Der Plan des Konsortiums ist aufgegangen. Ursprünglich zur Unterstützung der Entwicklung von Java-Programmen entworfen, existieren mittlerweile verschiedenste Erweiterungen, die die Basisfunktionalitäten von Eclipse in viele verschiedene Richtungen erweitern. Tatsächlich ist Eclipse mittlerweile zu einer vollständigen Rich Client Plattform gereift, was bedeutet, dass jegliche Funktionalität der Plattform durch Plugins bereit gestellt wird. Im April 2011 befanden sich laut `http://marketplace.eclipse.org/favorites/top` unter den zehn beliebtesten Erweiterungen die folgenden Plugins:

1. Das Plugin *Subclipse* bietet eine ausgereifte, eigenständig funktionierende Subversion-Anbindung innerhalb der IDE. Geänderte Dateien und Zeilen sind direkt aus den Editoren ersichtlich.
2. Mit *FindBugs* existiert für Java ein Werkzeug für statische Code-Analyse, welches über 200 Fehlermuster durch Bytecodeanalyse aufspüren und gefundene Treffer übersichtlich in der IDE darstellen kann.
3. Das Plugin *PyDev* erweitert Eclipse um Sprachunterstützung für die Programmiersprache Python. Unterstützte Features sind beispielsweise Codevervollständigung, Debugging sowie Unterstützung für Unittests.

Diese drei Plugins stehen exemplarisch für die Vielzahl von Möglichkeiten, um die Eclipse erweitert werden kann.

Eclipse sowie seine Erweiterungen sind vollständig in Java geschrieben. Die Oberfläche basiert auf dem Standard Widget Toolkit (SWT), welches ein Teil des Eclipse-Projekts ist. Für die visuelle Darstellung ist Eclipse neben dem Menü als Navigationselement in drei fundamentale Komponenten unterteilt, welche man als Softwareentwickler nutzen kann:

**Editoren** sind eingebettete Fenster, welche das Anzeigen und Bearbeiten von verschiedensten Inhalten erlauben. Üblicherweise verwenden Java-Entwickler den Texteditor für Java-Quellcode. Dieser bietet umfangreiche Unterstützung für Code-Vervollständigung, Syntax-Hervorhebung und vieles mehr. Es existieren auch Editoren zum Erstellen grafischer Benutzeroberflächen.

**Sichten** sind in die Eclipse-GUI eingebettete Fenster, um bestimmte Informationen innerhalb eigener Fenstern in einer für den Entwickler geeignet Art und Weise anzuzeigen. So zeigt die „Project Explorer"-Ansicht in einer Baumstruktur alle editierbaren Elemente eines Projekts an und markiert auf Wunsch die zurzeit in einem Editor geöffnete Datei. Die „Problems"-Ansicht hingehen fasst alle gefundenen Probleme der statischen Codeanalyse in einer einfachen Tabellenstruktur zusammen.

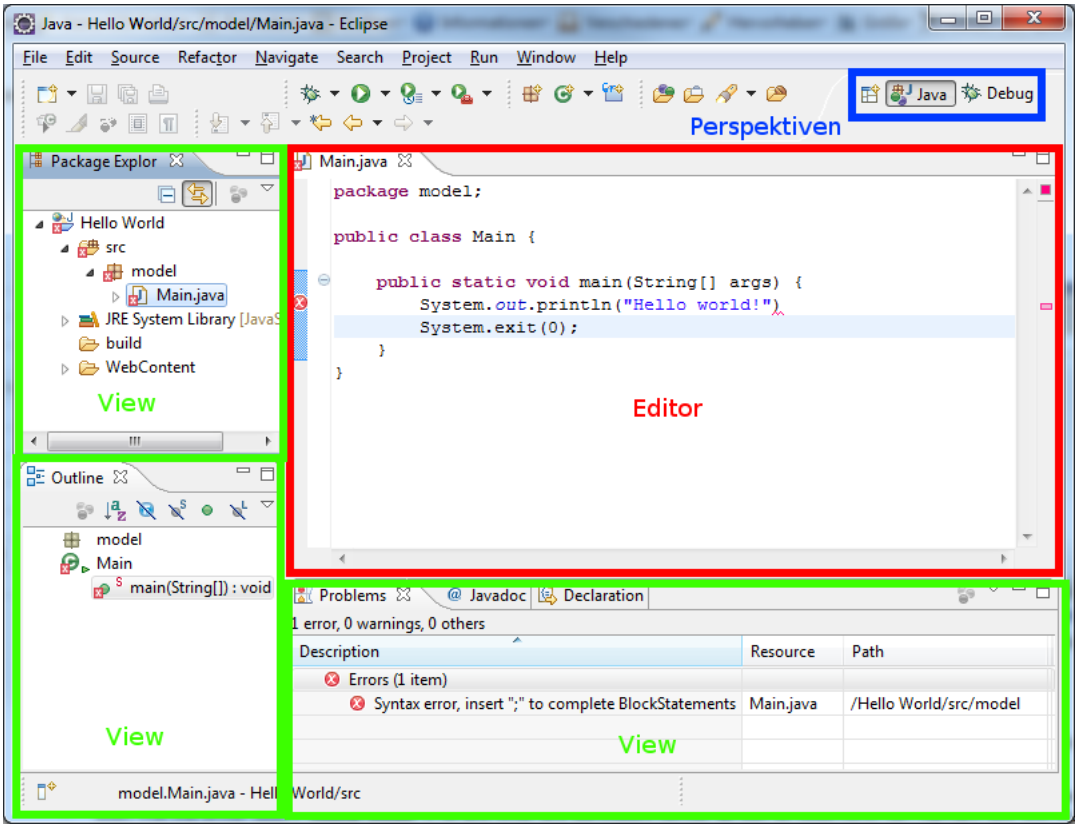


Abbildung 2.4: Verschiedene Eclipse-Komponenten

**Perspektiven** ermöglichen dem Entwickler, einzelne Eclipse-Komponenten wie Editoren in spezieller Weise anzuordnen. So existieren für verschiedene Anwendungsfälle bereits Perspektiven für Java EE-Entwicklung, Debugging sowie für die Code-Synchronisierung mit externen Repositories. Eigene Fenster-Anordnungen lassen sich als eigene Perspektive speichern.

Alle diese Komponenten zusammen sind Bestandteil des sogenannten *Workspace*. Dieser ist ein Container für alle geöffneten Projekte sowie die projektübergreifenden Benutzereinstellungen, die ein Entwickler hinterlegt hat. Durch dieses Konzept ist es möglich, mit derselben Eclipse-Installation mehrere Entwicklungsumgebungen zu erstellen, welche beispielsweise unterschiedliche Projekte, Formatierungsrichtlinien und Plugin-Einstellungen haben können. Dadurch erhält man die Möglichkeit, seine Projekte zu gruppieren, als Freiberufler beispielsweise nach unterschiedlichen Auftraggebern.

### 2.2.2 Alternativen

Eine weitere bekannte quelloffene Java-IDE ist *Netbeans*[3] aus dem Hause Oracle. Sie wird sowohl einzeln aus auch gebündelt mit der jeweils aktuellsten Java-Version zum Download angeboten. Für die visuelle Darstellung wird im Gegensatz zu Eclipse die Swing-Bibliothek verwendet. Der Aufbau der Oberfläche ist ähnlich zu der von Eclipse. Auch für Netbeans gibt es einen umfangreichen Plugin-Mechanismus, über den sich die IDE um neue Funktionalitäten erweitern lässt.

Die Firma JetBrains bietet mit *IntelliJ IDEA*[4] schließlich eine Java-IDE an, welche es sowohl in einer kostenlos verfügbaren, quelloffenen Community-Edition als auch in einer Ultimate-Edition mit kommerziellen Lizenz gibt. Beide Versionen unterscheiden sich vor allem durch den gebotenen Funktionsumfang sowie das dazugehörige Preismodell.

## 2.3 Paarprogrammierung

Paarprogrammierung ist eine Methode, bei der sich zwei Softwareentwickler zeitgleich denselben Computer teilen. Der Entwickler, der aktiv Tastatur und Maus bedient, wird als *Driver* bezeichnet, während die Rolle des passiven Entwicklers *Navigator* genannt wird. Während der Driver programmiert und sich auf das lokale Problem konzentriert, denkt der Navigator eher strategisch beispielsweise darüber nach, ob der gewählte Ansatz funktionieren kann oder ob das gesamte System vereinfacht werden kann. Dabei sichtet, analysiert und bewertet er laufend den zuletzt geschriebenen Programmcode. Erkannte Möglichkeiten zur

[3] URL: http://www.netbeans.org/
[4] URL: http://www.jetbrains.com/idea/

Verbesserung des Softwaredesigns werden unmittelbar miteinander besprochen. Dadurch entscheidet sich das Paar frühzeitig für den erfolgversprechendsten bekannten Lösungsweg (Williams, 2010). In dieser Form beschrieben ist Paarprogrammierung erstmals in Beck (1999). Bei der Methode *eXtreme Programming* (XP) ist Paarprogrammierung ein wesentlicher Bestandteil.

Bei der klassischen Programmierung löst jeder Programmierer im Idealfall ein ihm zugewiesenes Teilproblem einer definierten Komponente des zu entwickelnden Softwaresystems. In diesem Szenario ist jeder Entwickler selbst vollständig dafür verantwortlich, dass der von ihm produzierte Programmcode verschiedenen Aspekten wie Korrektheit, Vollständigkeit, Performance, Sicherheit und Lesbarkeit entspricht. Erschwerend kommt hinzu, dass der Programmcode für gewöhnlich unter einem gewissen Maß an Zeitdruck entsteht. Diese Kombination fördert das Entstehen von Defekten im erzeugten Code, zum Teil unbewusst durch Übersehen von relevanten Ausnahmesituationen, zum Teil aber auch bewusst durch Verzicht auf umfangreiche Tests. Paarprogrammierung begegnet diesem Problem, indem es die Verantwortung eines einzelnen Programmierers auf zwei Köpfe verteilt. Der Driver kann sich darauf verlassen, dass unberücksichtigte Details mit hoher Wahrscheinlichkeit vom laufend mitdenkenden Navigator bemerkt und unmittelbar angesprochen werden. Bei einem geistigen „Blackout" kann der Partner die Rolle des Drivers übernehmen, so dass längere unproduktive Denkpausen reduziert werden. In Hinblick auf mögliche architektonische Entscheidungen können verschiedene Lösungswege diskutiert werden, um den geeignetsten Weg ermitteln zu können, anstelle auf eine Bauchentscheidung angewiesen zu sein. Außerdem wird durch das Paaren von Entwicklern das Wissen über den entstehenden Code auf zwei Personen übertragen. Wenn jede vorhandene Codestelle mehr als nur einer Person bekannt ist, wird damit das Risiko für das Team reduziert, dass durch den Ausfall einer einzelnen Person bestimmte Codestellen nicht mehr oder nur unter hohem Aufwand angepasst werden können (Williams, 2010).

Ein kontrovers behandelter Aspekt bei der Anwendung von Paarprogrammierung sind die dadurch entstehenden Zusatzkosten. Bei erster Betrachtung könnte man davon ausgehen, dass sich durch die Verdopplung der Entwicklerzahl die Produktivität halbiert werden würde, da von zwei Entwicklern jeweils nur einer aktiv Code erzeugt. In Williams (2010) werden aber verschiedene Studien benannt, welche alle in etwa das gleiche Ergebnis hatten: Teams, die Paarprogrammierung einsetzen, sind vergleichbar produktiv wie Teams, in denen ausschließlich Soloprogrammierung eingesetzt wird. Anderslautende Studien, wonach Paarprogrammierung die Produktivität senkt, verwenden laut Williams (2010) die Formel „Codezeilen pro Personenmonat" (Lines of Code, LoC) als Maß für die Produktivität. Da in Paaren aber häufiger Chancen für Codevereinfachung erkannt werden als bei solo arbeitenden Entwicklern, ist diese Formel mit Vorsicht zu genießen.

Durchgehend positive Ergebnisse erzielte Paarprogrammierung bei der Betrachtung der Produktqualität in Bezug auf die Anzahl vorhandener Defekte. Paarprogrammierung führt zu ei-

ner veränderten Teammoral, bei der sich die Partner gegenseitig zu höherer Leistung als bei Soloprogrammierung anspornen. Dadurch befinden sich die Entwickler beim Programmieren kontinuierlich in einem Zustand erhöhter Konzentration, was sich ebenso positiv wie das kontinuierliche Code-Review des Navigators auf die Anzahl eingebauter Defekte auswirkt (Cockburn und Williams, 2001).

Paarprogrammierung eignet sich besonders gut für Softwareprojekte, bei denen eine agile Vorgehensweise wie z.B. eXtreme Programming verwendet wird. Das liegt in erster Linie daran, dass bei der agilen Softwareentwicklung im Vorfeld keine konkreten Vorgaben zur gewünschten Struktur des Programmcodes wie z.B. UML-Klassendiagramme existieren. Stattdessen wird die Struktur des Programmcodes iterativ bei der Umsetzung bestehender funktionaler Anforderungen entwickelt. Da ein einzelner Programmierer bei agilen Vorgehensmodellen demnach neben der Rolle des Programmierers auch die Rolle des Architekten einnimmt, kann das Bilden von Entwicklerpaaren ein guter Weg sein, die Last der Verantwortung für den erstellten Quellcode auf mehrere Schultern zu verteilen. Aber auch Projekte, welche ein klassisches Vorgehensmodell wählen, können von den positiven Aspekten der Paarbildung profitieren.

## 2.4 Verteilte Softwareentwicklung

Idealerweise befinden sich alle Entwickler eines Softwareprojekts in räumlicher Nähe zueinander. Das ist erstrebenswert, da Entwickler als Teil des Gesamtprojekts immer wieder mit Codefragmenten anderer Entwickler in Kontakt kommen. Entstehen dabei Fragen oder Änderungswünsche, lassen sich diese zeitnah und effektiv mit den Autoren kommunizieren. In der industriellen Praxis trifft man aber immer wieder auf die Situation, dass verschiedene Entwickler räumlich voneinander getrennt arbeiten, so dass persönliche Treffen nicht oder nur unter erheblichem Kosten- und Zeitaufwand möglich sind. Bei verteilter Paarprogrammierung (Distributed Pair Programming, DPP) wird gemeinsam Programmcode zeitgleich von verschiedenen Arbeitsplätzen aus bearbeitet (Schümmer und Schümmer, 2000). Während diese Konstellation bei klassischen Vorgehensmodellen weniger problematisch ist, sind bei agilen Modellen Kommunikationsprobleme zwischen den Entwicklern ein echtes Problem. Es besteht dann die Gefahr, dass an den verschiedenen Standorten Insellösungen entstehen, welche zwar lokal funktionieren, sich aber nicht oder nur unter hohem Aufwand in die Komponenten der anderen Standorte integrieren lassen. Eine wirkungsvolle Gegenmaßnahme können regelmäßige Meetings sein, welche entweder real oder virtuell stattfinden. Für letzteres eignen sich insbesondere Videokonferenzen, da neben den gesprochenen Worten auch eine visuelle Unterstützung in Form von Bildschirmpräsentationen oder gemeinsam genutzten elektronischen Whiteboards möglich ist. Für die tägliche Kommunikation sind das Telefon bzw. eine *Voice over Internet Protocol* (VoIP)-Lösung, eine gemeinsame Chatsoftware sowie spezialisierte Softwarelösungen Möglichkeiten, den Informationsaustausch zu

unterstützen.

Paarprogrammierung im eigentlichen Sinne stellt keine besonderen Ansprüche an Hard- und Software, da nur der Driver die vorhandenen Eingabegeräte verwendet. Möchte der Navigator eingreifen, übernimmt er Tastatur und Maus und nimmt somit die Rolle des Drivers an. Anders gestaltet sich die Sache, wenn sich die Entwickler an verschiedenen Standorten befinden. Da in diesem Szenario jeder Entwickler seine eigene Peripherie besitzt, bietet verteilte Paarprogrammierung erweiterte Möglichkeiten, eine gemeinsame Programmiersitzung zu gestalten. Ein Navigator könnte seinen Rechner dazu nutzen, den gerade aktuellen Entwicklungsstand zu kompilieren und lokal zu debuggen, ohne den Arbeitsfluss des Drivers zu stören. Ebenso könnten Programmteile analysiert werden, welche der Driver derzeit nicht betrachtet. Verteilte Paarprogrammierung bietet also neue Möglichkeiten, um die Rolle des Navigators noch vielfältiger zu gestalten.

Damit mehrere Rechner gepaart werden können, ist die Verwendung spezieller Software notwendig. Dazu gibt es zwei verschiedene Ansätze: *Bildschirmfreigabe* sowie *Collaboration Awareness*. Beide Ansätze werden im Folgenden kurz erklärt.

### 2.4.1 Bildschirmfreigabe

Bei Verwendung der Bildschirmfreigabe erlaubt ein Benutzer einem anderen, seinen freigegebenen Desktop auf dem entfernten Rechner zu sehen und bei entsprechend vorhandener Berechtigung auch zu steuern. Die Desktop-Ansicht wird dabei über ein Netzwerk vom Driver zu den Navigatoren übertragen. Die Benutzer teilen sich konzeptionell einen Arbeitsplatz. Entwickelt wurde das Konzept der Bildschirmfreigabe für die Fernadministration, bei der entfernte Rechner vom eigenen Rechner aus bedient werden können. Diese Art der Datenfreigabe kommt der ursprünglichen Definition von Paarprogrammierung am nächsten. Mit Hilfe spezieller Software wie der Microsoft Remotedesktopfreigabe oder der Open-Source-Lösung *Virtual Network Computing* (VNC) kann eine solche Verbindung hergestellt werden. Eine Echtzeitsynchronisierung der Projektdaten bei Paarprogrammierung zwischen den beteiligten Entwicklern ist nicht nötig. Nachteilig ist die hohe Datenmenge, die für die Übertragung des sich ändernden Desktops nötig ist und eine entsprechend gut ausgebaute Netzwerkverbindung zwischen den beteiligten Personen voraus setzt. Außerdem bleibt das erweiterte Potenzial, welches der zusätzliche Computer des Navigators bietet, ungenutzt.

### 2.4.2 Collaboration Awareness

Der Ansatz der Collaboration Awareness versucht die Nachteile der Bildschirmfreigabe zu umgehen. Anstelle den Bildschirminhalt zu übertragen, verwenden die beteiligten Benutzer dieselbe Software, welche kollaboratives Arbeiten unterstützt. Die Software (ein geeigneter

Texteditor oder eine IDE) muss technisch in der Lage sein, die Eingaben des Drivers zeitnah über ein Netzwerk an die Navigatoren zu senden. Auf deren Seite müssen wiederum die Aktionen des Drivers abgebildet werden. Denkbar ist eine sofortige Aktualisierung der geänderten Inhalte. Eine andere Möglichkeit ist, über eingehende Änderungen lediglich zu informieren, diese aber in einem Zwischenpuffer zu sammeln und erst durch manuellen Start der Synchronisation in die eigene Umgebung zu integrieren. Ein weiterer Aspekt ist, welche Aktionen überhaupt übertragen werden. Von Textänderungen über Tastaturanschläge bis hin zur Bedienung der Computermaus ist vieles möglich, aber nicht immer sinnvoll. Entscheidend ist, dass die Übertragung der Änderungen schnell erfolgt und die übertragenen Informationen für die Benutzer in irgendeiner Form relevant aufbereitet werden können.

## 2.5 Unterstützung für räumlich verteilte Paarprogrammierung

Der Fokus dieser Ausarbeitung liegt auf der Einführung des Eclipse-Plugins Saros, welches den Ansatz der Collaboration Awareness verfolgt. Es folgt eine Einführung in diese Erweiterung. Anschließend werden zwei alternative Produkte genannt, die Collaboration Awareness unterstützen.

### 2.5.1 Saros

Die Eclipse-Erweiterung *Saros*[5] (ein besonderer Zyklus zwischen Sonnenfinsternissen (Espenak, 2009)) erlaubt Entwicklern an verschiedenen Standorten, ein oder mehrere Eclipse-Projekte untereinander zu synchronisieren, um anschließend daran parallel miteinander arbeiten zu können. Das Projekt entstand im Rahmen einer Diplomarbeit an der Freien Universität Berlin (Djemili, 2006) und wird aktiv von einer Entwicklergruppe der Hochschule weiterentwickelt. Vom 1. Januar 2011 bis zum 1. Juli 2011 wurden sechs neue Releases veröffentlicht.

#### Kernfunktionalität

Zu Beginn einer DPP-Sitzung lädt der Driver alle Teilnehmer ein, gemeinsam an einem oder mehreren Projekten zu arbeiten. Durch Akzeptieren der Einladung werden die Projekte in den Workspace der Teilnehmer übertragen. Als Navigator hat man dabei mehrere Optionen. Zum einen kann man ein neues Projekt erstellen und die Projektdaten einmalig vom Driver zum Navigator transportieren lassen. Zum anderen kann ein Navigator auch ein lokal

[5] URL: http://www.saros-project.org/

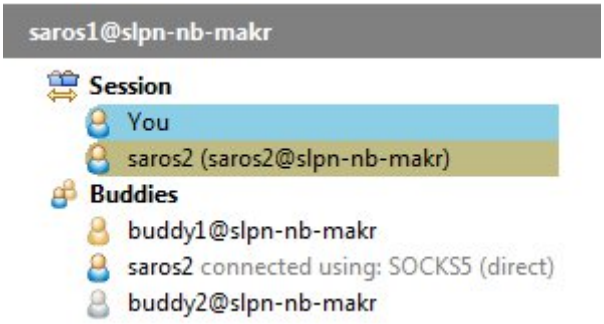


Abbildung 2.5: Roster-Ansicht in Saros

vorhandenes Projekt auswählen, welches für die gemeinsame Sitzung verwendet wird. Das kann sinnvoll sein, wenn es sich um ein großes Projekt handelt, da dadurch die zu übertragende Datenmenge aufs Minimum reduziert wird. Nach erfolgreicher Synchronisierung ist die Verbindung zwischen Driver und Navigator erfolgreich aufgebaut. Von nun an kann werden alle wesentlichen Aktionen des Drivers an die angeschlossenen Teilnehmer übertragen. Dazu gehören Textänderungen, Bereichsmarkierungen im Programmcode sowie der Wechsel zwischen Dateien. Damit die Navigatoren diese Änderungen mitbekommen, müssen sie die Dateien, die gerade bearbeitet werden, ebenfalls geöffnet haben. Damit das automatisch geschieht, gibt es für Navigatoren die Möglichkeit, allen Aktionen des Drivers automatisch zu folgen. In diesem *Follow-Modus* führt das Öffnen einer Datei durch den Driver dazu, dass diese Datei ebenfalls im Editor des folgenden Navigators geöffnet wird. Gleiches gilt bei Änderungen des aktiven Editor-Fensters. Dieser Modus entspricht am ehesten der Paarprogrammierung, da alle Beteiligten stets dasselbe zu sehen bekommen. Wird im Editor eine andere Datei geöffnet, führt das zur sofortigen Beendigung des Follow-Modus.

Für den Informationsaustausch zwischen den Teilnehmern verwendet Saros generell das *Extensible Messaging and Presence Protocol* (XMPP), welches im RFC3920 (2004) definiert ist. Dieses Protokoll wird überwiegend für die Übertragung von Sofortnachrichten verwendet. Damit mehrere Personen an einer gemeinsamen DPP-Sitzung teilnehmen können, müssen sich Driver und Navigator gegenseitig authentifiziert haben. RFC3921 (2004) definiert diesen *Subscription* genannten Prozess. Alle hinzugefügten Benutzer werden innerhalb eines XMPP-Clients in einem *Roster*[6] angezeigt. Saros beinhaltet einen Roster-View, über den ein Anwender seine Kontakte sowie deren Status sehen kann. Abbildung 2.5 zeigt, wie die Kontakte dargestellt werden: Der Benutzer „saros2" benutzt Saros (gelb-blaues Icon) und ist Teilnehmer der aktuellen Sitzung (Kategorie „Session"). „buddy1" ist online, nutzt aber kein Saros (gelbes Icon). „buddy2" ist offline (graues Icon). XMPP ist ein serverbasiertes Protokoll, was zur Folge hat, dass alle versendeten Nachrichten über die XMPP-Server der kommunizierenden Teilnehmer verschickt werden. Da bei der Initialisierung Saros-Sitzung

[6] Liste aller Kontakte, für die eine Subscription beantragt wurde

oftmals große Datenmengen übertragen werden, wird für diesen Prozess versucht, eine Direktverbindung zwischen den Clients aufzubauen, um die Projektdaten möglichst effizient zu synchronisieren. Falls eine Direktverbindung nicht möglich ist, werden die Projektdaten über XMPP verschickt.

Nach dem erfolgreichen Aufbau einer Saros-Sitzung besitzen sowohl der Driver als auch die Navigatoren das Recht, das freigegebene Projekt aktiv zu ändern. Dieses Recht kann durch den Driver jederzeit entzogen werden, so dass die betroffenen Navigatoren nur noch lesend aktiv sein können. Im Schreibmodus hingegen können alle freigegebenen Dateien geöffnet werden. Das funktioniert auch, wenn Driver und Navigator dieselbe Datei verändern. Textblöcke im Editor, welche ein anderer Teilnehmer in der laufenden Sitzung einfügt, ändert oder mit der Maus markiert, werden in den Editoren der anderen Teilnehmer farblich hervorgehoben, wobei jedem Entwickler eine eigene Farbe zugewiesen ist. Eigene Änderungen werden grundsätzlich nicht farblich hinterlegt. Abbildung 2.6 zeigt die Editoren von zwei gepaarten Entwicklern. Das erste Bild zeigt den Editor des ersten Entwicklers, in dem die Änderungen des entfernten Teilnehmers blau hinterlegt sind. Das zweite Bild zeigt dieselbe Ansicht aus der Perspektive des entfernten Entwicklers, bei dem die nicht selbst erfolgten Änderungen braun hinterlegt sind. Ebenfalls Teil des Roster-Views ist ein eingebauter Gruppenchat. Über diesen können die Teilnehmer während einer Saros-Sitzung miteinander Textnachrichten austauschen.

#### Weitere Features

Voraussetzung für echte Paarprogrammierung ist, dass die gepaarten Entwickler fortlaufend miteinander kommunizieren. Saros beinhaltet zwar eine Funktion, um Textnachrichten austauschen zu können. Diese Form der zwischenmenschlichen Kommunikation ist allerdings umständlich und zeitraubend, da ein Softwareentwickler die Tastatur bereits für das Erstellen der Software verwendet. Man kann entweder mit den Partner kommunizieren oder am Programmcode arbeiten, aber nicht beides zugleich. Um dieses Problem zu umgehen erlaubt Saros eine VoIP-Direktverbindung zwischen genau zwei Personen. Für Telefon- und Videokonferenzen unterstützt Saros die VoIP-Software Skype. Haben Driver und Navigator ihren Skype-Benutzernamen in Saros hinterlegt, kann durch Rechtsklick des Benutzers im Roster ein Skype-Telefonat gestartet werden. Im Gegensatz zur ersten Lösung unterstützt Skype Konferenzen mit mehreren Teilnehmern. Die Möglichkeit einer bidirektionalen Videoübertragung wertet das Verfahren zusätzlich auf.

Manche Sachverhalte lassen sich jedoch allein durch die Mittel der natürlichen Sprache schlecht erklären. Flipchart und Stift sind bei der Paarprogrammierung ein gängiges Werkzeug, um komplexe Sachverhalte zu veranschaulichen. Saros bietet als Äquivalent ein elektronisches Whiteboard, auf dem man mit Hilfe der Maus einfache Skizzen erzeugen und sie

Main.java RunUtil.java

```
1  package app;
2
3  import java.util.logging.Level;
4  import java.util.logging.Logger;
5
6  import model.RunUtil;
7
8  public class Main {
9
10     private static final Logger LOGGER = Logger.getLogger(Main.class.getName());
11
12     public static void main(String... args) {
13         LOGGER.log(Level.INFO, "Hello world!");
14         RunUtil.times(5, new Runnable() {
15             private int i = 1;
16             public void run() {
17                 System.out.println("Line #" + i++);
18             }
19         });
20     }
21
22 }
23
```

Main.java RunUtil.java

```
package app;

import java.util.logging.Level;
import java.util.logging.Logger;

import model.RunUtil;

public class Main {

    private static final Logger LOGGER = Logger.getLogger(Main.class.getName());

    public static void main(String... args) {
        LOGGER.log(Level.INFO, "Hello world!");
        RunUtil.times(5, new Runnable() {
            private int i = 1;
            public void run() {
                System.out.println("Line #" + i++);
            }
        });
    }

}
```

Abbildung 2.6: Textansichten der Editoren von Entwickler 1 und Entwickler 2

live mit den Sitzungsteilnehmern teilen kann. Ebenso wie im Texteditor können alle Teilnehmer gleichzeitig schreibend auf das Whiteboard zugreifen.

Ein wichtiges Merkmal bei Saros ist, dass auf jedem Client der geteilte Programmcode lokal vorliegt. Dadurch kann jeder Teilnehmer das Projekt lokal testen, starten und debuggen. Für den Fall, dass sich bei einem der Teilnehmer das Projekt bei dessen Ausführung anders verhält als bei den anderen Teilnehmern, helfen die zuvor genannten Features nur bedingt weiter. Basierend auf der Echtzeitkompressionssoftware Xuggler erlaubt Saros eine echte Bildschirmfreigabe (vgl. Abschnitt 2.4.1).

### 2.5.2 Alternativen

Neben Saros gibt es weitere Softwarelösungen im Bereich Collaboration Awareness. Für Eclipse existiert das Plugin *XPairtise*[7], welches vergleichbar mit Saros Unterstützung für verteilte Paarprogrammierung bietet. Der Texteditor *Gobby*[8] ist ein eigenständiges Programm, dessen Installationsdatei gerade einmal 7 MB groß ist. Damit kann es eine Alternative zur Installation einer vollständigen IDE sein.

#### XPairtise

Die Eclipse-Erweiterung XPairtise entstand im Wintersemester 2006/07 an der FernUniversität Hagen im Rahmen des Fachpraktikums „CSCW - Entwicklung einer kooperativen Anwendung". Es bietet gleichzeitiges Bearbeiten von Texten, Projektsynchronisierung, verteilte Programm- und Testausführung, Benutzermanagement, einen Textchat sowie ein gemeinsames Whiteboard. Die grundsätzliche Bedienung unterscheidet sich nur unwesentlich von Saros. Es gibt verschiedene Views, in denen der Roster, der Chat und das Whiteboard dargestellt werden. Technologisch setzt XPairtise für die Kommunikation zwischen den Teilnehmern allerdings einen ActiveMQ-Server voraus. Die Kommunikation wird über Java Message Service (JMS) realisiert.

#### Gobby

Eine IDE bietet sehr viele Funktionen, wodurch eine hohe Komplexität unvermeidlich ist. Um eine IDE effektiv nutzen zu können, bedarf es einer gewissen Einarbeitungszeit. Mit dem kollaborativen Texteditor Gobby existiert eine leichtgewichtige Alternative. Die am 28. März 2011 veröffentlichte Version 0.4.94 beinhaltet alle wichtigen Features, die ein Texteditor, den man parallel nutzen möchte, benötigt:

[7] XPairtise-URL: http://xpairtise.sourceforge.net/
[8] Gobby-URL: http://gobby.0x539.de/

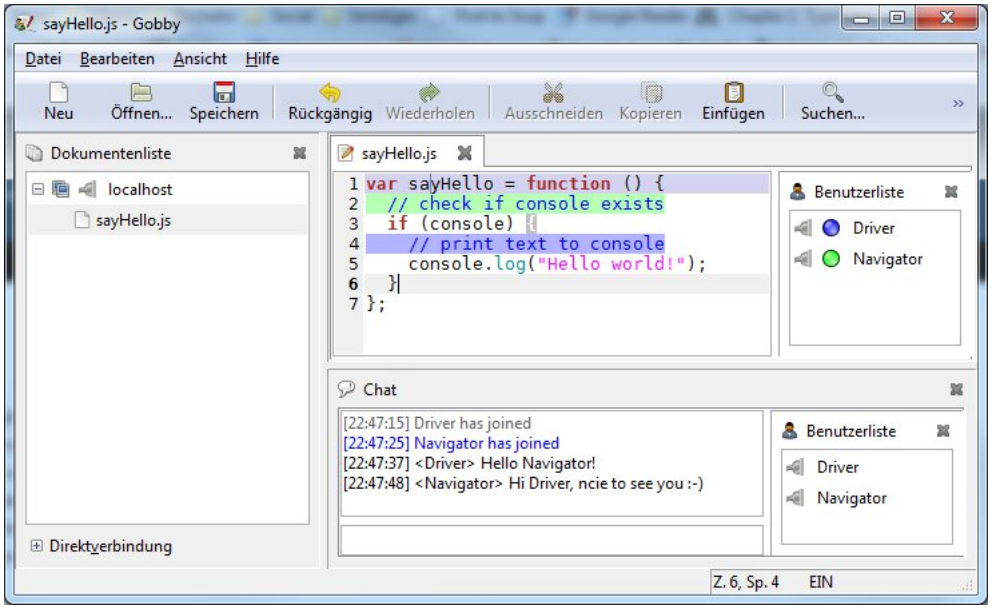


Abbildung 2.7: Laufende Gobby-Sitzung zwischen zwei Teilnehmern

- Echtzeitzusammenarbeit auf Dateiebene
- Anzeige der teilnehmenden Benutzer einer DPP-Sitzung
- Anzeige geänderter Textbereiche in der Farbe des verantwortlichen Teilnehmers
- Anzeige der Cursorposition sowie markierter Textbereiche der anderen Teilnehmer
- Echtzeitkommunikation via Textchat

Zusätzlich beherrscht der Editor Syntaxhervorhebung für über 50 Sprachdialekte, u.a. für Java, LaTeX und Prolog. Abbildung 2.7 zeigt den Inhalt einer in Gobby geöffneten JavaScript-Datei, welche von den Teilnehmern „Driver” und „Navigator” bearbeitet wird. Neben den geänderten Zeilen wird der Cursor des entfernten Teilnehmers in Zeile 1 auf dem Wort „sayHello” dargestellt. Für die Synchronisierung verbinden sich die Teilnehmer mit einem speziellen Server (wird zusammen mit Gobby ausgeliefert).

# 3 Analyse

Im Folgenden werden die Rahmenbedingungen (Abschnitt 3.1) bei der VEIS vertieft. Anschließend werden sowohl allgemeine wie auch VEIS-spezifische Ziele definiert (Abschnitt 3.2), die durch die Einführung von Saros erreicht werden sollen. Aus den Zielen werden dann Anforderungen aus Sicht des Managements ermittelt (Abschnitt 3.3). Nach weiteren allgemeinen Überlegungen zu möglichen Anwendungsszenarien (Abschnitt 3.4) wird die Reihenfolge (Abschnitt 3.5) für die Planung und Realisierung formuliert.

## 3.1 Rahmenbedingungen

Das Web-Application-Entwicklerteam der VEIS ist für Stand-Alone-Applikationen verantwortlich, die sowohl für Vattenfall-interne als auch externe Kunden angefertigt werden. Das Team besteht im Wesentlichen aus acht Softwareentwicklern in Hamburg. Zusätzlich existieren derzeit zwei Entwickler in Polen sowie einige Entwickler in den Niederlanden. Diese können im Bedarfsfall einzelne Projektteams verstärken, führen aber unabhängig davon auch eigene Softwareprojekte durch.

Neben dem Web-Application-Team gibt es das Team *Web Content Management* (WCM). Dieses ist verantwortlich für alle Web-Entwicklungen, die in irgendeiner Weise mit dem offiziellen Internetauftritt von Vattenfall zu tun haben. Das sind neben dem verwendeten Content-Management-System auch in den Internetauftritt eingebettete Web-Anwendungen. Diese werden durch WCM-Entwickler sowohl erstellt als auch betreut.

### 3.1.1 Infrastruktur

Jeder Entwickler hat Zugriff auf einen eigenen Computer, entweder in Form einer Arbeitsstation oder in Form eines Notebooks. Dieser Computer dient als zentraler Arbeitsplatz und wird üblicherweise nicht mit anderen Mitarbeitern geteilt. Konzernweit ist die Ausstattung dieser Arbeitsplätze im Kern identisch. Es existiert ein zentrales Identity-Management-System, an welches die Arbeitsplätze angebunden sind. Mit Ausnahme von Windows 7, welches sich innerbetrieblich in einer Testphase befindet, sind andere Betriebssysteme für die Arbeitsstationen der Mitarbeiter nicht gestattet. Alle Arbeitsplätze sind mit vergleichbarer Hardware ausgestattet.

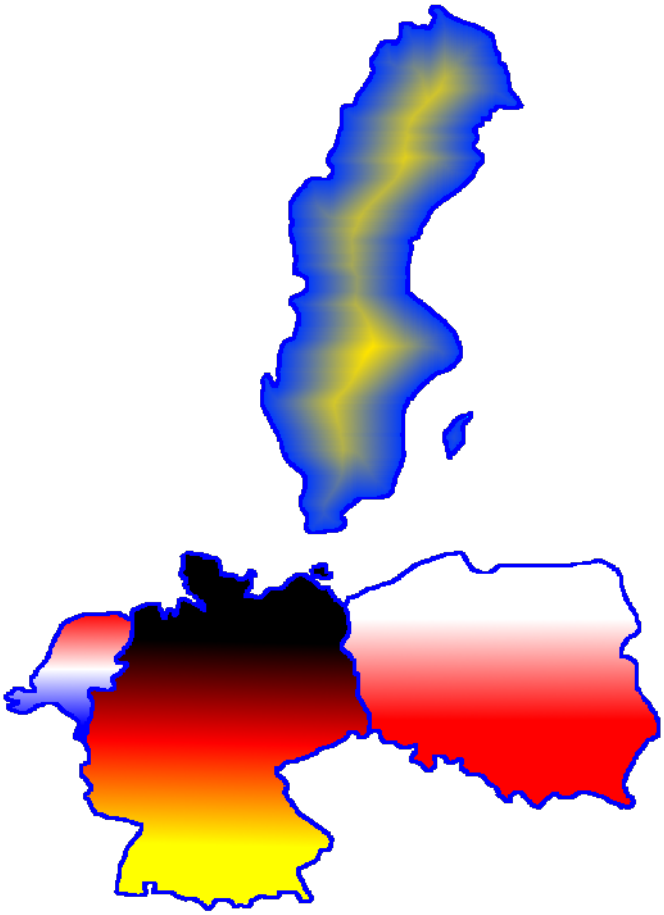

Abbildung 3.1: Nationen der Teammitglieder (Schweden, Niederlande, Polen und Deutschland)

### Standardsoftware

Auf den Arbeitsstationen stehen stets folgende vorinstallierte Anwendungen der Firma Microsoft zur Verfügung:

**Windows XP** ist das auf den Arbeitsplatzcomputern vorinstallierte Betriebssystem. Es ist an das konzernweite Active Directory angebunden. Jeder Benutzer ist in der Lage, sich an jeder Workstation mit seinem Domänenbenutzer anzumelden. Die Entwicklertools werden allerdings aufgrund der begrenzten Größe des Benutzerprofils auf einem lokalen Laufwerk installiert, so dass ein Anmelden an einer anderen Workstation lediglich den Zugriff auf dort vorinstallierte Standardsoftware garantiert. Neu aufgesetzte Arbeitsplatzcomputer können im Rahmen eines Testlaufs wahlweise auch mit Windows 7 ausgeliefert werden.

**Office** ist die vom Konzern vorgegebene Bürosoftware. Die folgenden Komponenten werden hauptsächlich genutzt:

- **Word** wird zum Erstellen von Textdokumenten verwendet. Es existieren Vorlagen, welche das Aussehen von Benutzer- und Systemdokumentation vereinheitlichen.
- **Excel** wird insbesondere von den Kunden für Beispielberechnungen, Tabellen sowie ganz allgemein zur Verbesserung des Verständnisses fachlicher Anforderungen verwendet.
- **PowerPoint** ist die Lösung zur elektronischen Erstellung und Vorführung von Präsentationen. Es stehen im Konzern Vorlagen bereit, welche dem Corporate Design Vattenfalls entsprechen.
- **Outlook** wird für den regulären elektronischen Schriftverkehr (E-Mail) inner- und außerhalb der Firma verwendet. Hierüber werden bevorzugt fachlich relevante Inhalte verschickt. Außerdem werden interne Meetings sowie generell alle personenbezogenen Termine vollständig über Outlook verwaltet.

**NetMeeting** ist ebenfalls auf allen Arbeitsplätzen installiert, allerdings nicht über das Startmenü verlinkt. Ein Start ist aber durch den direkten Aufruf der dazugehörigen ausführbaren Programmdatei möglich. Mit Hilfe von NetMeeting ist eine wie in 2.4.1 beschriebene Bildschirmfreigabe möglich.

### Spezialsoftware

Zusätzlich gibt es weitere Anwendungen, die von den Softwareentwicklern verwendet werden. Diese werden von den einzelnen Entwicklern in einem initialen Arbeitsschritt selbst nachinstalliert.

**Java** ist sowohl eine Programmiersprache als auch eine Umgebung, in der in Java geschriebene Programme ausgeführt werden können. Aktuell für die Entwicklung und Ausführung verwendet ist die Version 6. Java ist die Grundlage nahezu aller bei der VEIS entwickelten Webanwendungen. Das Entwicklerteam entwickelt ausschließlich Individualsoftware in Form von Enterprise-Webanwendungen für die Java-Plattform. Diese werden auf dem JEE5[1]-konformen Applikationsserver *Oracle WebLogic 10.3* ausgeführt.

**Eclipse 3.6** ist die Java-IDE, mit deren Hilfe die geforderten webbasierten Softwarelösungen umgesetzt werden. Es handelt sich um eine speziell konfigurierte Installation, welche von allen Teammitgliedern heruntergeladen und lokal verwendet wird. Somit ist gewährleistet, dass Eclipse-Projekte auf verschiedenen Workstations verwendet werden können, ohne dass Inkompatibilitäten aufgrund fehlender oder abweichender Eclipse-Plugins auftreten. Diese innerhalb des Teams „Corporate Eclipse"-genannte Installation ist für das Team von zentraler Bedeutung.

**Spark** ist ein Programm, welches die Kontaktaufnahme zu anderen Benutzern ermöglicht. Es erfordert, dass die Benutzer XMPP-Anmeldedaten besitzen. Innerhalb des Vattenfallnetzes können alle Kollegen miteinander mit Hilfe des Spark-Clients Sofortnachrichten austauschen. Diese Form der Kommunikation wird häufig zur informellen Kontaktaufnahme verwendet, insbesondere für Verfügbarkeitsanfragen sowie kurze Rückfragen. In Situationen, wo ein Teilnehmer nicht frei telefonieren kann, ist die Verwendung von Spark oftmals ein akzeptabler Ersatz.

**TortoiseSVN** ist eine Erweiterung für den Windows-Explorer, welche sowohl lesenden als auch schreibenden Zugriff auf ein Subversion-Repository ermöglicht. Subversion versioniert sowohl Dateien als auch Verzeichnisse und fungiert gleichzeitig als Backup. Bei jedem Übertragen eines Datei- oder Verzeichnisstands ins Repository wird eine neue sogenannte Revision erzeugt, welche den übertragenen Stand eindeutig identifiziert. Die wichtigsten Aktionen im Zusammenhang mit Subversion besitzen eine eigene Terminologie: Für das Versionieren lokaler Daten verwendet man im Sprachgebrauch die Begriffe „Einchecken" bzw. „Committen". Beim erstmaligen Übertragen bereits versionierter Daten auf den eigenen Computer spricht man vom „Auschecken".

#### Serversoftware

Abschließend werden die wichtigsten serverseitigen Dienste genannt, die den Entwicklern direkt bzw. indirekt zur Verfügung stehen.

**Active Directory** wird für die zentrale Benutzer- und Rechteverwaltung verwendet. Der Active Directory-Server wird innerhalb des Konzerns für verschiedene Single Sign-on-

[1] Java Enterprise Edition 5

Dienste auf Basis von NT LAN Manager und Kerberos verwendet. Beispiele für Dienste, die vom Active Directory Gebrauch machen sind die Windows XP-Benutzeranmeldung, Outlook sowie der Instant Messaging (IM)-Client Spark.

**Openfire** ist ein Serverdienst, welcher die Kommunikation über XMPP ermöglicht. Der Openfire-Server ist so konfiguriert, dass darüber versendete Nachrichten innerhalb des Firmennetzwerks bleiben. Der XMPP-Server ist ausschließlich aus dem konzerneigenen Intranet erreichbar. Kommunikation mit XMPP-Clients außerhalb des Firmennetzwerks ist nicht möglich und zum Zeitpunkt der Erstellung dieser Arbeit auch nicht erwünscht. Der Openfire-Server ist so konfiguriert, dass die Mitarbeiter in XMPP-Clients ihre im Active Directory hinterlegten Logindaten zur Benutzeranmeldung verwenden können.

**Subversion** ist ein serverbasiertes Versionskontrollsystem. Entwickler können ihren Quellcode in ein zentrales Code-Repository übertragen (einchecken) und anschließend wieder von dort auf den eigenen Rechner übertragen (auschecken). Sämtlicher Quelltext, der bei Vattenfall im Laufe der Softwareentwicklung erzeugt wird, wird in einem zentralen Subversion-Repository versioniert. Dadurch ist sichergestellt, dass der Ausfall eines Arbeitsplatzrechners nicht zum Verlust des Quellcodes führt. Außerdem erleichtert Subversion das Zusammenführen von Änderungen.

**Jenkins** ist ein System für kontinuierliche Integration (Continuous Integration, CI). In Jenkins kann man für Softwareprojekte einen Pfad zu einem Versionskontrollsystem sowie einen Befehl angeben. Werden Änderungen in das Projekt-Repository eingecheckt, überträgt Jenkins die Änderungen in ein lokales Verzeichnis und führt anschließend den Befehl (in der Regel ein sogenanntes Build-Script[2]) aus. Läuft der Befehl erfolgreich bis zum Ende durch, gilt das Projekt als erfolgreich gebaut. Schlägt der Build-Prozess fehl, können verschiedene Folgeaktionen gestartet werden. Üblich ist das Versenden einer E-Mail, die auf das Problem hinweist.

Die Arbeitsstationen aller Mitarbeiter haben ausschließlich Zugriff auf Computersysteme innerhalb des Vattenfall-Konzernnetzwerks. Das würde bedeuten, dass der Aufruf externer Internetseiten nicht möglich wäre. Aus diesem Grund sind die Internetbrowser der Arbeitsplatzcomputer so konfiguriert, dass sie für HTTP- und HTTPS-Anfragen einen dafür bereitgestellten Proxyserver verwenden. Über diesen sind Zugriffe auf den größten Teil des Internets möglich. Der Zugriff auf bestimmte potenziell schädliche Seiten wird durch den Server blockiert.

Die Mitarbeiter in Hamburg befinden sich größtenteils in durch mobile Trennwände in Sektionen aufgeteilten Büros. Pro Sektion sind bis zu drei Arbeitsplätze einrichtbar. Auf Wunsch kann jeder Entwickler ein eigenes Telefon erhalten. Für Meetings gibt es einen eigens dafür vorgesehenen Raum, in den sich Teams mit bis zu sechs Mitgliedern für Diskussionen

[2] Anweisungen, um aus Quellcode auslieferbare Artefakte zu erzeugen

zurückziehen können, um die Kollegen nicht zu stören. Snack- und Getränkeautomaten befinden sich in anderen Gebäudebereichen.

In Amsterdam befindet sich das Team in einem Großraumbüro. Nicht jeder Arbeitsplatz verfügt über ein Telefon. In dem Büro ist es üblicherweise recht laut, so dass dort Telefonieren nur erschwert möglich ist. Eigens dafür vorgesehene Räume sind oftmals belegt und müssen von Angestellten gebucht werden. Innerhalb des Büros existiert ein Kaffeeautomat. Es ist üblich, die Pausen direkt am Arbeitsplatz zu verbringen und sich dort mit den Kollegen auszutauschen.

### 3.1.2 Projektmanagement

Das WebApplication-Team der VEIS arbeitet in einzelnen Projektteams, was heißt dass jedem Projekt feste Entwickler zugewiesen werden. Diese erstellen dann nicht nur die Anwendung, sondern kümmern sich nach Fertigstellung des Produkts auch um den Folgesupport sowie spätere Weiterentwicklungen.

Bei entsprechender Auftragslage werden für einzelne Projekte bedarfsweise Kollegen aus Polen und den Niederlanden angefordert. Je nach Umfang und Komplexität des Projekts kommt es vor, dass die Kollegen für ein Kick-off-Meeting sowie eine anschließende Planungsphase für einen Zeitraum von ein bis zwei Wochen nach Deutschland reisen müssen. In dieser Zeit werden die wichtigsten bekannten Informationen weitergegeben und mögliche Probleme und Unklarheiten diskutiert. Am Ende reisen die ausländischen Kollegen wieder ab und führen die weitere Entwicklung bei ihrem Heimatstandort fort. Aufgrund der verhältnismäßig hohen Reise- und Übernachtungskosten sowie der langen Anreisezeit insbesondere für die polnischen Kollegen wird ein weiteres Treffen wenn möglich durch das Management vermieden.

## 3.2 Ziele

Die Einführung einer Softwarelösung, welche verteilte Paarprogrammierung ermöglicht, soll keinem Selbstzweck dienen. Vielmehr verspricht man sich dadurch eine qualitative Verbesserung des Softwareentwicklungsprozesses. Zum einen sind das allgemeingültige Ziele, die unabhängig vom Unternehmen erreicht werden sollen. Zum anderen gibt es aufgrund der bestehenden Teamstruktur Vattenfalls sich daraus ableitende Ziele, die das Management erreichen möchte.

### 3.2.1 Agile Werte und Prinzipien

Die folgenden Werte und Prinzipien stehen für die sogenannte leichtgewichtige Entwicklung. Die Ideen dahinter stammen aus zwei verschiedenen Richtungen: Das *Agile Manifest* wurde im Jahr 2001 durch eine Gruppe von 17 Leuten, die im Bereich Softwareentwicklung tätig waren, gemeinsam entwickelt (Beck u. a., 2001). Es definiert sowohl Werte als auch Prinzipien, welche den Softwareentwicklungsprozess als solchen verbessern sollen.

#### Agile Werte

Im Agilen Manifest werden acht Grundwerte definiert, welche innerhalb der agilen Softwareentwicklung gelten. Für jeden Wert gilt, dass die Werte der rechten Seite wichtig sind, die der linken Seite sind aber noch wichtiger. Die zwei Wertepaare, die einen deutlichen Bezug zu Paarprogrammierung haben, sind:

**Individuen und Interaktionen mehr als Prozesse und Werkzeuge** Gute Software entsteht nicht allein durch festgelegte Prozesse und Werkzeuge, sondern durch engagierte, motivierte Entwickler mit Teamgeist. Saros als Werkzeug allein löst keine Kommunikationsprobleme zwischen den Entwicklern. Es ist aber ein Mittel, um Kommunikation über Raumgrenzen hinweg zu vereinfachen.

**Funktionierende Software mehr als umfassende Dokumentation** Dokumentation, die während des Entwicklungsprozesses erstellt wird, ist ein wichtiges Hilfsmittel, um ein Softwareprodukt später einfacher warten zu können. Ohne funktionierende Software aber ist der Wert der erzeugten Dokumentation fraglich. Ein Mittel, um die Wahrscheinlichkeit zu erhöhen, dass die Software am Ende funktioniert (also das tut, was von ihr verlangt wurde), ist die Anwendung von Paarprogrammierung, wie auf dieser Seite verdeutlicht wird.

Die anderen vier Werte lauten „Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung” sowie „Reagieren auf Veränderung mehr als das Befolgen eines Plans”.

#### Agile Prinzipien

Die agilen Werte geben die grundsätzliche Richtung vor, welche agile Softwareentwicklung einschlägt. Um diese recht allgemeinen Aussagen mit Leben zu füllen, verabschiedeten die am Agilen Manifest beteiligten Entwickler zusätzlich eine Liste von zwölf Prinzipien.

**Funktionierende Software ist das wichtigste Fortschrittsmaß.** Funktioniert die Software nicht, kann sie in der Regel nicht sinnvoll eingesetzt werden. Aber auch wenn die Software startet, heißt das nicht, dass sie auch funktioniert. Erst wenn die Software exakt das tut, was der Auftraggeber von ihr erwartet, kann man von funktionierender Software sprechen. Unsaubere Anforderungen können von Individuen fehlerhaft interpretiert werden, was zu unerwünschtem Verhalten in der Software führt. Durch das Vier-Augen-Prinzip der Paarprogrammierung fallen mehrdeutige Formulierungen eher auf, da der Partner eine Formulierung anders verstehen könnte und das anmerkt. Aber auch Peer Code Reviews können in der Entwicklungsphase helfen, Fehler in der Programmierung zu erkennen.

**Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.** Die Anwendung von Paarprogrammierung ist ein wichtiger agiler Prozess. Durch die Verteilung des Fachwissens auf mindestens ein zweites Paar Schultern wird dafür gesorgt, dass auch bei leichten Personalfluktuationen die Weiterentwicklung nicht wesentlich bedroht ist.

**Ständiges Augenmerk auf technische Exzellenz und und [sic] gutes Design fördert Agilität.** Agilität lebt von hochwertig codierter Software. Es ist nur logisch, dass gut ausgebildete hochmotivierte Entwickler diesen Anspruch eher erfüllen als mittelmäßig erfahrene desinteressierte Entwickler. Durch Paarprogrammierung und Peer Code Reviews bestehen gute Chancen, dass Designschwächen und Wissenslücken in Bezug auf technische Standards erkannt und behoben werden. Der Austausch im Review sowie das Reflektieren der gemeinsam getroffenen Entscheidungen in Bezug auf Design und verwendete Technologien bieten die Möglichkeit, das Wissen aller Beteiligten zu erweitern.

**Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteam zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.** Klassische Vorgehensmodelle (vgl. Abschnitt 2.1) legen einen besonders großen Wert auf eine umfassende schriftliche Dokumentation von Anforderungen, Verpflichtungen und der für die Software erstellten Systemarchitektur. Diese Dokumente sind immer vertraglich geforderte technische Dokumente, die fast ausschließlich von Softwareentwicklern erstellt werden, die die Erstellung der Dokumentation mehr als notwendiges, zeitaufwändiges Übel denn als sinnvolle fordernde Tätigkeit sehen. Dementsprechend ist am Ende häufig auch die Qualität des Dokuments: Entweder wurde es schlampig erstellt und es fehlen viele Details, oder es ist ein mehr als vollständiges Dokument, in dem sowohl alle wichtigen als auch unwichtigen Aspekte der Software erfasst sind - die schiere Fülle an wertlosen Informationen macht ein solches Dokument unbrauchbar.

Aus diesen Gründen wurde nach alternativen Wegen gesucht, um zumindest innerhalb des Entwicklungsteams Informationen effizienter austauschen zu können. Im agilen Umfeld hat

sich die direkte Kommunikation zwischen den Beteiligten als der beste Weg für den Informationsaustausch bewährt. Durch Paarprogrammierung wird der Informationsaustausch direkt mit dem Softwareentwicklungsprozess verknüpft. Informationen werden unmittelbar dort übermittelt, an denen sie relevant sind.

### 3.2.2 Schlanke Softwareentwicklung

Eine der agilen Softwareentwicklung ähnliche Praktik ist die schlanke Entwicklung („lean development"). Sie basiert auf der Idee der schlanken Produktion („lean production"), welche den Wertschöpfungsprozess bei der industriellen Produkterzeugung optimieren soll. Demnach wird alles, was nicht Wert für den Endkunden erzeugt, als Ballast angesehen, der idealerweise beseitigt werden soll. Wertvoll ist dabei alles, wofür der Kunde bereit ist Geld zu zahlen. Ende des letzten Jahrtausends erlangte Lean production große Bekanntheit mit dem Erscheinen von Womack und Jones (1997) sowie durch den konsequenten Einsatz bei Toyota in Form des *Toyota Production System* (TPS) (Holweg, 2006). Insbesondere letzterer ist ein Ergebnis vorheriger Verschlankungsmaßnahmen, bei denen die Produktion soweit optimiert wurde, dass nahezu keine überflüssigen (im Sinne von optimierbarer) Prozesse mehr vorhanden sind. Da im Bereich der Softwareentwicklung kein fertiges Produkt in Fließbandarbeit erzeugt wird, setzen die Lean-Optimierungen beim Entwicklungsprozess an.

Ebenso wie das Agile Manifest gibt es auch für die schlanke Entwicklung verschiedene Prinzipien, die das Umsetzen dieser Praktik erleichtern sollen. Da der Entwicklungsprozess in der Automobilindustrie aber anders abläuft als in der Softwareentwicklung, existieren für beide Bereiche unterschiedliche Prinzipien. Für die Softwareentwicklung gelten nach Poppendieck und Poppendieck (2003) folgende Prinzipien:

1. Entferne unnötigen Ballast
2. Verstärke kontinuierliches Lernen
3. Entscheide so spät wie möglich
4. Liefere so schnell wie möglich aus
5. Stärke das Team
6. Bau Integrität ein
7. Sieh das Ganze

Alle diese Prinzipien haben als gemeinsames Ziel, bestehende Werte unter Verringerung der damit verbundenen Arbeit zu erhalten. Der Einsatz von Paarprogrammierung und Code-Reviews begünstigt die Anwendung dieser Prinzipien, indem sich die Teammitglieder gegenseitig daran erinnern sie einzuhalten.

### 3.2.3 Vattenfall

Vattenfall setzt derzeit für die Softwareentwicklung aus Kostengründen Nearshoring ein. Im Rahmen der Nuon-Übernahme werden Entwickler zukünftig zusätzlich mit niederländischen Kollegen zusammenarbeiten, oftmals jeder an seinem Standort. Ein Ziel ist, die Kommunikation zwischen den beteiligten Entwicklern toolgestützt zu verbessern.

Ein weiteres Problem ist die Handhabung von Anwendungen, deren Entwicklung bereits mehrere Jahre her ist. Oftmals verhindern Architektur sowie Struktur des Quellcodes der Anwendungen, dass die Anwendung zu angemessenen Kosten weiterentwickelt werden kann. Aus diesem Grund entscheidet das Management oftmals, dass die Anwendung neu erstellt werden soll mit dem Ziel der Wartbarkeit unter Einhaltung der aktuellen Architektur- und Coding-Standards. Wie genau diese Standards aussehen ist jedoch nicht klar definiert. So reicht es für manche Entwickler, wenn die Anwendung am Ende läuft, während andere Entwickler durch Anwendung erweiterter Praktiken wie testgetriebene Entwicklung die Qualität des resultierenden Quellcodes auf einem hohen Level halten möchten.

Der Schwerpunkt der vom Management gewünschten Verbesserungen liegt auf einer Verbesserung in Bezug auf die gemeinsame Softwareentwicklung. Entwickler an verschiedenen Orten sollen über einen standardisierten Prozess ein gemeinsam auf denselben Code zugreifen und gemeinsam ein Peer Code Review durchführen können. Dieses soll ähnlich effektiv sein wie in dem Fall, dass beide Entwickler das Review an einem gemeinsamen Rechner durchführen.

## 3.3 Anforderungen

Aus den bisher genannten Anwendungsszenarien und Zielen sollen nun Anforderungen aus Sicht des Managements formuliert werden. Da die Technik der Paarprogrammierung insbesondere durch Beck (1999) bzw. Beck (2005) größere Popularität erhielt, werden im Folgenden die Anforderungen im Stil von Scrum in einem Product Backlog dokumentiert (vgl. Unterabschnitt 2.1.3), und zwar in Form von sogenannten *User Stories*. User Stories bestehen für gewöhnlich aus genau einem Satz, der wie folgt aufgebaut ist:

> „Als <Rolle> möchte ich <Wunsch/Ziel> (damit <Nutzen>)"

<Rolle> beschreibt den Rollennamen des Benutzers, der das System benutzt.

<Wunsch/Ziel> beschreibt das Feature, welches die Benutzerrolle anwenden können soll.

<Nutzen> beschreibt den tatsächlichen Mehrwert, der sich durch das gewünschte Feature ergibt. Die Angabe des Nutzens ist optional, aber durchaus empfehlenswert, da es den Kunden dazu zwingt, den Mehrwert des Features zu hinterfragen.

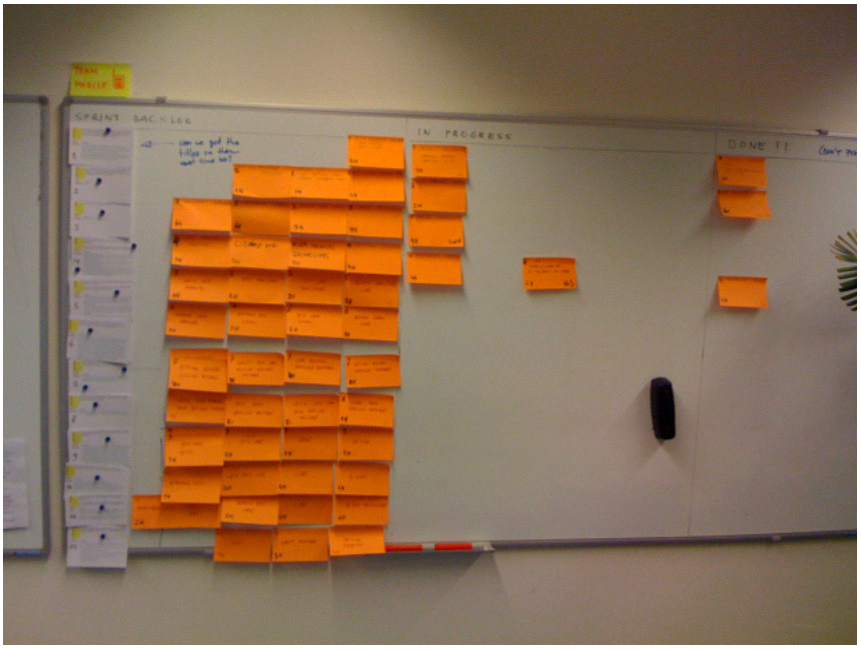


Abbildung 3.2: Product Backlog mit User Stories (Eickmann, 2009)

Eine Beispiel-Story könnte so aussehen:

> „Als Leser dieser Abschlussarbeit möchte ich die Anforderungen des Managements an die technische Integration kennen, damit ich spätere Entscheidungen bei der Planung und Realisierung besser nachvollziehen kann."

Die Story definiert als Rolle den Leser der Abschlussarbeit, als Ziel die Dokumentation der Anforderungen sowie als Nutzen die bessere Nachvollziehbarkeit späterer Textteile. Gewöhnlich werden für die einzelnen Stories vor deren Bearbeitung funktionale Abnahmekriterien definiert. Diese in Form einer Checkliste notierbaren Kriterien werden als *Definition of Done* bezeichnet. Sind alle Kriterien erfüllt, gilt die User Story als erfolgreich bearbeitet. Sie wird dann aus dem Spring Backlog entfernt und zu den erledigten Stories gelegt. Im weiteren Verlauf wird auf die Definition of Done verzichtet.

Das Management hat tatsächlich nur eine sehr allgemein gehaltene Anforderung definiert: „Entwickler sollen mit Hilfe von Eclipse verteilte Paarprogrammierung anwenden können". Eine solch allgemeine Anforderung wird im Umfeld agiler Softwareentwicklung als *Epic* bezeichnet. Ein Epic ist zu groß oder zu vage formuliert, um ihn wie durch Scrum vorgegeben in einem Sprintzyklus abarbeiten zu können. Aus dem Epic lassen sich aber verschiedene User Stories ableiten, mit denen der iterative Entwicklungsprozess erfolgreich umgesetzt und verifiziert werden kann.

Es folgen die abgeleiteten User Stories:

**Als Manager möchte ich, dass eine DPP-Sitzung ähnlich komfortabel durchgeführt werden kann wie eine normale PP-Sitzung, damit beide Prozesse eine vergleichbare Qualität haben und wir Reise- und Übernachtungskosten sparen können.** Driver und Navigator nutzen bei verteilter Paarprogrammierung die technischen Hilfsmittel Saros und eine Anwendung, die idealerweise eine Videokonferenz, mindestens aber VoIP erlaubt. Der Sinn liegt darin, die fehlende körperliche Anwesenheit der beteiligten so gut es geht zu kompensieren. Funktioniert die Lösung hinreichend gut, kann dies Kosten einsparen. Aber ist diese Lösung tatsächlich gleichwertig zur Paarprogrammierung vor einem gemeinsamen Computer, oder ergeben sich hier doch spürbare Nachteile? Falls ja, welche Probleme machen sich bemerkbar?

**Als Manager möchte ich, dass die Entwickler mehr miteinander kommunizieren, damit sie ihr Wissen miteinander teilen.** Die Anwendung von Paarprogrammierung setzt eine intensive Form der Kommunikation zwischen Driver und Navigator voraus. Für einen effektiven Einsatz ist es daher sinnvoll, wenn sich die Teilnehmer einer DPP-Sitzung bemühen, die Kommunikation miteinander zu optimieren. Wird der Einsatz verteilter Paarprogrammierung tatsächlich eine Verbesserung der Kommunikation zwischen den Beteiligten bewirken?

**Als Manager möchte ich, dass die Entwickler gegenseitige Code-Reviews durchführen, damit sich die Qualität der Software verbessert.** Die Bereitstellung einer DPP-Softwarelösung führt nicht zwangsläufig zu deren intensiver Nutzung, sofern diese nicht durch das Projektmanagement verbindlich angeordnet wird. Code-Reviews als Anordnung bieten die Möglichkeit, eine DPP-Sitzung zu starten. Interessant wäre zu wissen, ob die Entwickler im Rahmen von Code-Reviews tatsächlich Fehler im untersuchten Code finden und ob dadurch wirklich die Qualität der ausgelieferten Software steigt.

**Als Administrator möchte ich die bestehende Saros-Installation im Corporate Eclipse gegen eine neuere Version austauschen können, damit die Entwickler von neuen Features sowie Fehlerbereinigungen profitieren können.** Die Besonderheit an Saros ist dessen über Jahre hinweg kontinuierliche Weiterentwicklung. Seit Anfang 2010 sind auf der SourceForge-Projektseite quasi monatlich neue Releases veröffentlicht worden. Das schafft Vertrauen in das Produkt. Andere DPP-Lösungen wie XPairtise werden seit Jahren nicht mehr aktiv weiterentwickelt. Daher ist es ein nachvollziehbarer Wunsch, die installierte Saros-Version gegen ein neueres Release austauschen zu können. Ob das auch wirklich möglich ist, wird sich zeigen.

**Als Entwickler möchte ich eine kurze Einführung in das Thema „Verteilte Paarprogrammierung", damit ich mich darauf einstellen kann, was mich in Zukunft erwarten**

**wird.** Die Installation und Bereitstellung von Saros ist nur der technische Teil der Prozesseinführung. Für eine erfolgreiche Gesamtintegration müssen die Entwickler, die später Paarprogrammierung anwenden sollen, in diese Programmiertechnik eingewiesen werden. Die Frage ist, ob der Weg über eine Schulung erfolgversprechend ist oder ob Paarprogrammierung direkt durch die Anwendung von Paarprogrammierung vermittelt werden sollte. Vielleicht ist es auch sinnvoll, erst nur wenige Leute in die Technik einzuweisen und anschließend darauf zu vertrauen, dass diese durch Paarbildung mit nicht eingewiesenen Leuten ihr Wissen weitergeben.

**Als Entwickler möchte ich gezeigt bekommen, wie man Saros korrekt für eine DPP-Sitzung verwendet, damit ich verteilte Paarprogrammierung bestmöglich anwenden kann.** Hier verhält es sich so wie bei der vorherigen User Story mit dem Unterschied, dass hier den Entwicklern die Bedienung von Saros vermittelt werden soll. Für das Vorgehen ergeben sich analog die gleichen Fragen wie zuvor.

**Als Entwickler möchte ich, dass der Verbindungsaufbau einer Saros-Sitzung maximal 60 Sekunden dauert, damit ich nicht zu viel Zeit mit dem Warten auf den Beginn der DPP-Sitzung verliere.** Natürlich sollen die Entwickler keine unnötige Zeit beim initialen Verbindungsaufbau verlieren. Daher wird als Toleranzgrenze recht willkürlich eine Minute gewählt. Es wird sich zeigen, ob dieser Wert realistisch ist.

## 3.4 Anwendungsszenarien

Für die weitere Betrachtung werden verschiedene Szenarien definiert, für die eine Einführung verteilter Paarprogrammierung analysiert und bewertet wird.

### Alle Entwickler am gleichen Standort

In diesem Szenario wird davon ausgegangen, dass alle Entwickler sich zumindest im gleichen Gebäude, nicht aber unbedingt im gleichen Büro befinden. Es ist also für die verschiedenen Entwickler möglich, sich innerhalb von wenigen Minuten persönlich zu treffen. Dieses Szenario findet man in Unternehmen mit genau einer Softwareentwicklungsabteilung an einem geographischen Standort vor, bei denen die Entwickler parallel an mehreren Softwareprojekten mit jeweils unterschiedlichen Teammitgliedern arbeiten.

### Alle Entwickler an verschiedenen Standorten

Dieses Szenario beschreibt den Grenzfall, dass alle Entwickler an verschiedenen Standorten arbeiten, so dass zwischen allen Entwickler eine starke räumlich Distanz herrscht. Diese Konstellation kann leicht bei Open-Source-Projekten entstehen, die in der Anfangsphase von einem einzelnen Entwickler gestartet wurden. Mit fortschreitendem Wachstum und einer steigenden Anzahl von Features beginnen oftmals Dritte, sich für ein solches Projekt zu interessieren, meistens weil sie eigene Wünsche umsetzen oder weil sie sich aktiv an der Weiterentwicklung beteiligen möchten, um die Richtung, die die Weiterentwicklung einschlägt, mit beeinflussen zu können. In solchen Fällen lernen sich die Entwickler über das Internet kennen, so dass die Wahrscheinlichkeit, dass die Entwickler eine starke räumlich Distanz zueinander haben, sehr groß ist.

### Entwicklergruppen an verschiedenen Standorten

Größere Unternehmen mit einer eigenen IT-Entwicklungsabteilung setzen häufig an verschiedenen Standorten Softwareentwickler ein. Das ist dann besonders vorteilhaft, wenn sich die Kunden der zu entwickelnden Softwarelösungen direkt am jeweiligen Standort befinden. Dadurch können offene Fragen bei unklaren Anforderungen effektiv geklärt werden. Manchmal müssen an solchen Projekten aber Mitarbeiter unterschiedlicher Standorte miteinander arbeiten. Mögliche Gründe sind personelle Engpässe an einem Standort, oder es gibt für verschiedene fachliche Bereiche verschiedene Fachexperten an unterschiedlichen Standorten, so dass eine Aufteilung der Entwickler sinnvoll sein kann. Das führt dann dazu, dass an den verschiedenen Standorten eigene Entwicklergruppen an der Realisierung beteiligt sind.

## 3.5 Reihenfolge

Die Einführung einer neuen Technologie in bereits bestehende Prozesse ist immer eine Herausforderung. Insbesondere nicht explizit durch die Betroffenen eingeforderte Änderungen können zu heftigen Widerständen führen. Aber auch die technische Integration kann eine heikle Sache sein. Trotz sorgfältiger Planung der durchzuführenden Schritte und Berücksichtigung anzunehmender Problemquellen können bei der Durchführung unberücksichtigte Problem auftauchen, die im schlimmsten Fall nicht gelöst werden können - die Integration ist gescheitert.

Es gibt also zwei kritische Problemquellen bei einer Neueinführung: Den Faktor Technik sowie den Faktor Mensch. Die Frage ist, in welcher Reihenfolge man an einer Lösung arbeitet.

Aus mehreren Gründen bietet es sich an, zuerst die technische Realisierung durchzuführen und erst anschließend den Zielgruppe die funktionierende Lösung nahe zu bringen:

Testsystem Existiert bereits eine Testinstallation, kann diese für Demonstrations- und Schulungszwecke verwendet werden.

Kosten Gibt es unüberbrückbare technische Hindernisse, sind keine unnötigen Kosten durch Schulungen und dadurch verbundene Arbeitsausfälle der Mitarbeiter entstanden.

Psyche Scheitert die Einführung eines technik-gestützten Prozesses an eben dieser Technik, wird das bei den Zielpersonen einen schlechten Eindruck hinterlassen. Das kann dazu führen, dass bei einem späteren Versuch aufgrund des gescheiterten Erstversuchs ein übertriebenes Misstrauen sowie eine schlechtere Grundhaltung gegenüber dem Werkzeug und dem Prozess entsteht.

Daher wird in den folgenden Kapiteln zuerst auf die Einbindung von Saros in die vorhandene Infrastruktur eingegangen. Im Anschluss wird ein Konzept erstellt, wie die Softwareentwickler an den neuen Prozess und das neue Werkzeug herangeführt werden können.

# 4 Technische Integration

Dieses Kapitel beschreibt, wie eine erfolgreiche technische Integration von verteilter Paarprogrammierung in eine bestehende Infrastruktur aussehen kann. Im Abschnitt 3.5 wurde bereits eine grundsätzliche Vorgehensweise geplant. Abschnitt 4.1 zählt die von der technischen Integration betroffenen Systeme auf. In Abschnitt 4.2 wird ein konkreter Plan erarbeitet, mit dessen Hilfe die Wahrscheinlichkeit einer erfolgreichen Integration möglichst hoch sein wird. Danach folgt in Abschnitt 4.3 ein Konzept sowie ein Erfahrungsbericht, wie die Umsetzung des Konzepts funktioniert hat. Dabei wird insbesondere darauf eingegangen, welche Komplikationen sich bei der Integration von Saros ergeben haben. Am Ende wird in Abschnitt 4.4 ein kurzes Fazit gezogen.

## 4.1 Relevante Systeme

Zunächst wird ausgehend von den Systemanforderungen von Saros eine Liste der bei der VEIS betroffenen Hard- und Softwaresysteme erstellt, die im Zuge der Saros-Integration relevant sind. Diese Systeme werden im Folgenden kurz aufgezählt und beschrieben.

### Corporate Eclipse

Im Unterabschnitt 3.1.1 auf Seite 24 wurde bereits auf die vorhandene technische Infrastruktur eingegangen, u.a. auch auf das gemeinsam genutzte Eclipse, welchem Saros hinzugefügt werden soll. Die Eclipse-Installation wird bei Vattenfall in einem Subversion-Repository versioniert und von dort durch die Benutzer mit Hilfe des SVN-Clients TortoiseSVN auf den lokalen Rechner übertragen (Details hierzu befinden sich im Unterabschnitt A.3). Spätere Eclipse-Aktualisierungen können damit sehr einfach durch Synchronisierung mit dem serverseitigen Repository durchgeführt werden.

### XMPP-Server

Für die Kommunikation zwischen den verschiedenen DPP-Teilnehmern verwendet Saros das XMPP-Protokoll. Vattenfall betreibt einen aus dem internen Firmennetzwerk erreichbaren XMPP-Server für die Realisierung von Instant Messaging innerhalb des Konzerns. Dieser

Server greift für die Benutzerverwaltung auf einen Teil der konzernweiten Mitarbeiterstammdaten zu. Dadurch kann der Server nahezu wartungsfrei verwendet werden.

### TortoiseSVN

Ebenso wie Änderungen des Corporate Eclipse mit TortoiseSVN aus dem Repository geholt werden können, so werden damit die administrativen Aktualisierungen der gemeinsamen Eclipse-Installation auch wieder ins Subversion-Repository übertragen und so den restlichen Entwicklern zur Verfügung gestellt. Um diesen Vorgang ausführen zu können muss der entsprechende Benutzer über Schreibzugriff auf das entsprechende Repository verfügen. Dieses Recht haben nur einige wenige Entwickler, die für die Wartung der Corporate Eclipse-Installation zuständig sind.

## 4.2 Planung

Nachdem die relevanten Softwarekomponenten identifiziert sind, kann mit der Planung der durchzuführenden Tätigkeiten begonnen werden. Ein wesentliches Kriterium ist, dass nach der Integration alle bereits existierenden Systeme und die vorinstallierten Eclipse-Plugins weiterhin funktionieren. Zusätzlich gilt es, die folgenden technischen Besonderheiten zu berücksichtigen:

- Die zu Beginn der Planung verwendete Eclipse-Version ist das Release 3.5. Ein neues Corporate Release 3.6 wurde aber bereits zum größten Teil zusammengestellt. Daher erscheint es sinnvoll, Saros in die neue Eclipse-Installation zu integrieren und somit zeitgleich mit dem Releasewechsel für die Entwickler verfügbar zu machen.
- Die Eclipse-Installation ist so konfiguriert, dass es zwei verschiedene Verzeichnisse für Erweiterungen gibt. Das zentrale Erweiterungsverzeichnis beinhaltet die durch den Eclipse-Administrator installierten Erweiterungen, die für jedes Teammitglied vorhanden sein sollen. Werden Plugins lokal vom jeweiligen Entwickler hinzugefügt, wird das gemeinsame Plugin-Verzeichnis nicht geändert. Stattdessen gibt es ein zusätzliches lokales Erweiterungsverzeichnis. In diesem werden eigens installierte Erweiterungen abgelegt, die unabhängig vom gemeinsamen Standard lokal der Eclipse-Installation hinzugefügt werden. Auch wenn die lokale Installation von Plugins nur in besonderen Ausnahmen erfolgt, sollte Saros mit solch lokal installierten Plugins nicht auf unerwartete Weise interferieren.
- Die einzelnen technischen Arbeitsplätze sind nicht direkt mit dem Internet verbunden. Um auf Ressourcen außerhalb des Firmennetzwerks zugreifen zu können, muss ein vorhandener HTTP-Proxyserver verwendet werden. Die Installation von Saros sollte

idealerweise unter Verwendung der zentralen Konfiguration auf einem gewöhnlichen Arbeitsplatzcomputer durchführbar sein.

- Der vorhandene XMPP-Server wird von den Teammitgliedern bereits aktiv für Instant Messaging verwendet. Jeder Entwickler hat auf seiner Workstation den IM-Client *Spark* laufen. Eine bestehende Saros-Sitzung darf die normale Spark-Nutzung nicht negativ beeinflussen.
- Der für XMPP-Dienste verwendete Openfire-Server ist so konfiguriert, dass er bestimmte XMPP-Programme blockiert. Während die Nutzung mit dem Spark-Messenger möglich ist, verweigert der Server beispielsweise die Nutzung durch den IM-Client *Pidgin*. Da der Server nicht unmittelbar dem Team untersteht, muss ein vorheriger Betriebstest mit den Verantwortlichen des Servers koordiniert werden.

Basierend auf diesen Informationen lässt sich ein mehrstufiger Integrationsplan erstellen. Dieser sorgt für eine strukturierte Durchführung der technischen Integration, ohne zu sehr auf Implementierungsdetails einzugehen. Der Plan sieht folgende Schritte vor:

### Lokale Installation

Dieses ist der erste Schritt des gesamten technischen Integrationsprozesses. Saros wird zunächst lokal in das gemeinsame Eclipse integriert. Das Ziel ist, dass nach einem Neustart die Saros-Benutzeroberfläche zu sehen und eine Ersteinrichtung möglich ist.

### Sicherung der modifizierten Installation

Weitere Einstellungen und Tests könnten die lauffähige Saros-Installation unbrauchbar machen. Daher soll das um Saros erweiterte Eclipse in einem eigenen Repository gesichert werden. Das Ziel ist, dass auf einfache Weise wieder eine lauffähige Eclipse-Installation erzeugt werden kann, falls in einer späteren Phase etwas schief geht.

### Lokale Sandbox-Tests

Die ersten Funktionstests sollen in einer eigenen Arbeitsplatzumgebung durchgeführt werden, ohne dass bestehende Serverdienste angesprochen werden. Dazu wird die Saros-Eclipse-Installation auf Dateisystemebene dupliziert, so dass zwei Eclipse-Instanzen parallel gestartet werden können. Außerdem wird lokal ein XMPP-Server installiert und gestartet. Dieser wird den vorhandenen Server für die lokalen Tests ersetzen. Das Ziel ist, dass man die Saros-Installation funktional testen kann, ohne andere Dienste zu gefährden.

### Integrationstests

Abschließend muss sich zeigen, inwiefern die Saros-Installation unter realen Bedingungen funktioniert. Da es keinen XMPP-Testserver gibt, wird für den Test der fürs Instant Messaging verwendeten XMPP-Server verwendet. Das Ziel ist, dass zwei Benutzer an verschiedenen Computern über den Konzern-XMPP-Server erfolgreich eine DPP-Sitzung aufbauen und verwenden können.

Sobald alle Schritte erfolgreich verlaufen sind, ist die technische Integration nahezu abgeschlossen. Als letzter Schritt fehlt dann noch die Veröffentlichung der Eclipse-Installation für alle Benutzer, was aber erst mit der offiziellen Umstellung von Eclipse 3.5 auf Version 3.6 stattfinden wird.

## 4.3 Realisierung

Basierend auf der zuvor gemachten Planung wird jetzt die technische Realisierung dokumentiert. Dazu werden zuerst die durchzuführenden Schritte genannt, die für den Erfolg des jeweiligen Durchführungsschritts relevant sind. Anschließend werden die guten und schlechten Aspekte genannt.

### 4.3.1 Lokale Installation

#### Planung

Fundamental für den weiteren Verlauf der Saros-Integration ist, dass die Installation in die neue Eclipse-Installation andere Plugins nicht einschränkt. Die durchzuführenden Schritte, um Erweiterungen so einzuspielen, dass die Änderung an alle Teammitglieder verteilt wird, ist im unternehmensinternen Wiki dokumentiert. Das dahinterstehende Konzept kann in A.4 nachgelesen werden.

1. Als erstes wird die im Aufbau befindliche Eclipse-3.6-Installation aus dem Versionskontrollsystem ausgecheckt.
2. Über den internen Plugin-Mechanismus wird in Eclipse die Erweiterung Saros installiert. Die dazugehörige Internetseite bietet zwei mögliche Wege an:

   Update-URL Die Installation kann durch Hinzufügen einer bestimmten URL, welche auf der Saros-Website dokumentiert ist, in Eclipse erfolgen. Dazu wird das Plugin durch Eclipse aus dem Internet heruntergeladen und dann installiert (siehe Abbildung 4.1).

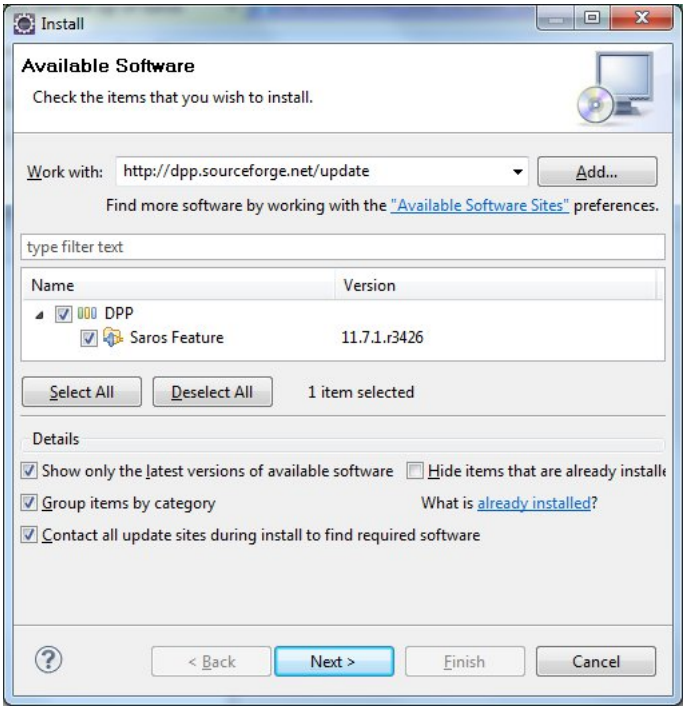


Abbildung 4.1: Pluginauswahl nach Eingabe der Saros-Update-URL

Dropin Laut der Saros-Website kann alternativ auch ein speziell dafür vorgesehenes Installationsarchiv heruntergeladen und in das Verzeichnis „dropins" unterhalb des Eclipse-Installationsverzeichnisses extrahiert werden. Beim nächsten Eclipse-Start wird dann das Plugin automatisch installiert.

Für die Installation soll der Weg über die Update-URL gewählt werden. Dieser Weg erlaubt es, zu einem späteren Zeitpunkt auf einfache Weise die Eclipse-Installation inklusive aller installierten Plugins mit Hilfe der Update-URL zu aktualisieren. Schlägt dieser Weg fehl, wird eine Installation als Dropin versucht.

### Durchführung

Zuerst wurde die noch nicht freigegebene Eclipse 3.6-Installation auf die lokale Festplatte ausgecheckt. Diese beinhaltet bereits eine Version der Datei eclipse.ini, bei der neue Plugins direkt in die Installationsumgebung integriert werden.

Anschließend sollte das zum Zeitpunkt der Installation aktuelle Saros-Release 11.5.6.r3294 mit Hilfe der Saros-Update-URL installiert werden. Der Ablauf für das Hinzufügen des Eclipse-Plugins ist sehr gut dokumentiert und eigentlich unkompliziert. In diesem Fall scheiterte die Installation jedoch in der Phase, in der Eclipse die Plugin-Dateien aus dem Internet herunter

lädt. Die Installation brach stets mit mehreren „Repository timed out”-Fehlermeldungen ab. Da die Update-URL über einen Browser gut erreichbar war, schien das Problem auf Eclipse begrenzt zu sein. Aus diesem Grund sollte der zweite Weg durchgeführt werden: Installation als Dropin. Aber auch dieser Weg scheiterte: Für das o.g. Release existierte schlichtweg kein Dropin-Archiv.

Um das aktuellste Saros-Release innerhalb des Vattenfall-Netzwerks zu installieren, musste also die Problemursache gefunden werden. Nach etwas Recherche stellte sich heraus, dass der Vattenfall-Proxyserver zu viel Zeit für die Beantwortung der Saros-HTTP-Downloadverbindungen brauchte und es so zu den Timeout-Meldungen kam. Das war eine neue Beobachtung, da andere Plugins zu früheren Zeitpunkten über eine Update-URL installiert werden konnten. Da nicht ausgeschlossen werden konnte, dass das Problem mit dem zu installierenden Saros-Release zusammenhängen könnte, wurde ein Installationsversuch mit dem Vorgängerrelease 11.3.25.r3201, für das ein Dropin-Archiv existierte, gestartet. Nach dem abschließenden Neustart begrüßte Eclipse den Anwender mit dem Saros-Einrichtungsassistent sowie der Sicht „Saros”. Die Installation war erfolgreich verlaufen.

Am 21. Mai 2011 wurde der Eclipse 3.6-Releasekandidat (die Installation ohne Saros) dann auf einem Rechner kopiert, der direkten Zugang zum Internet hatte. Hier verlief die Installation des Saros-Releases 11.5.6.r3294 über den Update-URL-Mechanismus auf Anhieb erfolgreich. Auf dem ans interne Firmennetzwerk abgeschlossenen Computer wollte die Installation aber nach wie vor nicht gelingen. Daher wurde recherchiert, ob der HTTP-Download-Mechanismus von Eclipse so konfiguriert werden kann, dass die Wartezeit auf HTTP-Antworten erhöht werden kann. In der Tat konnte das Installationsproblem durch Anpassen der Konfigurationsdatei eclipse.ini gelöst werden:

```
-Dorg.eclipse.ecf.provider.filetransfer.retrieve.closeTimeout=5000
-Dorg.eclipse.ecf.provider.filetransfer.retrieve.readTimeout=5000
```

Das Setzen dieser Eigenschaften sorgt dafür, dass die Wartezeit für HTTP-Antwortzeiten auf 5 Sekunden erhöht wird. Ein Blick in den Quellcode der Javaklasse „AbstractRetrieveFileTransfer” im Java-Package „org.eclipse.ecf.provider.filetransfer.retrieve” verrät, dass der Standardwert für beide Eigenschaften „1000” ist, was einer Sekunde Wartezeit entspricht. Für den HTTP-Proxy Vattenfalls scheinen diese Standardwerte zu gering gewesen zu sein. Mit der Änderung auf die neuen Werte konnte Saros schlussendlich auch auf einem Arbeitsplatzcomputer erfolgreich im damals aktuellsten Release installiert werden.

### 4.3.2 Sicherung der modifizierten Installation

#### Planung

Nach der erfolgreichen lokalen Installation soll die um Saros erweiterte Eclipse-Installation in einem separaten Zweig des Versionskontrollsystems namens *EclipseSaros* versioniert wer-

den. Der Subversion-Client TortoiseSVN bietet das nützliche Feature, eine lokal geänderte Version eines bestehenden Versionierungsszweigs direkt in einem neuen Zweig ablegen zu können. Damit bleibt der bereits vorhandene Administrationszweig von den Änderungen unberührt.

#### Durchführung

Da das Repository, in dem sich die Eclipse-Installation befindet, für die meisten Mitarbeiter schreibgeschützt ist, musste zuerst der Subversion-Benutzer des Saros-Verantwortlichen Schreibberechtigung erhalten. Danach funktionierte das Übertragen in einen neuen Subversion-Zweig auf Anhieb problemlos.

### 4.3.3 Lokale Sandbox-Tests

#### Planung

Für einen ersten Funktionstest soll auf die Einbindung externer Dienste vollständig verzichtet werden. Stattdessen wird auf dem Arbeitsplatzcomputer die für den Saros-Betrieb notwendige Infrastruktur bereitgestellt und darauf basierend ein erfolgreicher Aufbau einer DPP-Sitzung getestet. Die durchzuführenden Schritte sind:

1. Für einen lokalen Funktionstest muss auf dem Testsystem neben einer Eclipse-Installation auch ein XMPP-Server in Betrieb sein. Ein solcher muss also zuerst lokal installiert werden.
2. Es müssen zwei XMPP-Benutzer angelegt werden, mit denen dann der Testlauf durchgeführt werden kann.
3. Es werden zwei Eclipse-Instanzen gestartet. Da die Saros-Einstellungen in den Workspace-Profilen hinterlegt werden, muss für jede Instanz ein eigener Workspace verwenden werden. In den Instanzen wird jeweils einer der beiden Benutzer für den Betrieb von Saros hinterlegt.
4. In einer Instanz wird ein neues Projekt erstellt. Anschließend lädt dieser Benutzer den anderen zu einer gemeinsamen DPP-Sitzung ein. Der Eingeladene akzeptiert die Anfrage und nimmt somit an der Sitzung teil.
5. Beide Benutzer agieren jeweils einmal als Driver und einmal als folgender Navigator.

Wenn all diese Schritte erfolgreich verlaufen, ist der Weg frei für den einen abschließenden Integrationstest.

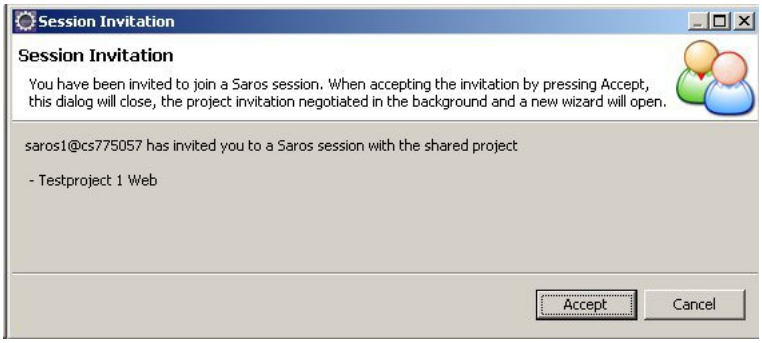


Abbildung 4.2: Einladung zu einer DPP-Sitzung

### Durchführung

Nach der Saros-Installation war die letzte benötigte Komponente für die Verwendung von Saros ein laufender XMPP-Server. Um dem realen System möglichst nahe zu kommen, wurde auf der Arbeitsstation ein Openfire[1]-Server mit der Versionsnummer 3.6.4 installiert. Dieser wird von Vattenfall für das Instant Messaging eingesetzt, ist allerdings aufwändiger konfiguriert. Für den lokalen Test wurde eine minimale Installation durchgeführt, bei der sich neue Benutzer selbstständig am Server registrieren können (das ist auch die Standardeinstellung von Openfire). Die Installation selbst verlief ohne Probleme. Allerdings war nach deren Abschluss, in dessen Verlauf explizit ein Administrator-Konto angelegt werden muss, keine Anmeldung an der Admin-Konsole möglich. Erst ein Neustart des Openfire-Servers erlaubte das Verwenden des Administratorzugangs für eine erfolgreiche Anmeldung, was auf einen Fehler im verwendeten Openfire-Release zurück zu führen ist[2].

Anschließend sollten für die folgenden Tests zwei Benutzer „Saros1" und „Saros2" angelegt werden. Da der Saros-Einrichtungsassistent die Möglichkeit bietet, auf einem dafür ausgelegten XMPP-Server direkt ein neues Benutzerkonto anzulegen, wurde diese Funktion gleich mit getestet. Die Einrichtung von sowohl den Benutzern als auch dem zu verwendenden XMPP-Server funktionierte tadellos. Ein anschließender Test mit dem Instant Messenger Spark zeigte, dass die Benutzer erfolgreich angelegt worden waren: Beide Benutzer konnten sich auch über Spark mit dem lokalen XMPP-Server verbinden. Damit die beiden Benutzer allerdings eine gemeinsame Saros-Sitzung aufbauen konnten, mussten sie sich gegenseitig in ihrem jeweiligen Roster hinzufügen. Da das über den in Saros integrierten Roster möglich ist, wurde diese Funktion ebenfalls getestet - und mit Erfolg angewendet. Nun sind beide Benutzer „Buddies" und können direkt miteinander XMPP-Nachrichten austauschen.

Abschießend sollte die Kernfunktionalität getestet werden. Zu diesem Zweck wurde in die Eclipse-Instanz des Benutzers Saros1 ein lokal existierendes Java-Projekt importiert. An-

[1] Openfire-Website: `http://www.igniterealtime.org/projects/openfire/`
[2] Version 3.7.0 beseitigt das Problem: `http://issues.igniterealtime.org/browse/OF-217`

schließend wurde der Benutzer Saros2 über den Roster eingeladen, an einer gemeinsamen DPP-Sitzung teilzunehmen. Im zweiten Eclipse erschien eine Dialogbox mit einem Einladungstext und der Möglichkeit, der Sitzung beizutreten (siehe Abbildung 4.2). Nach der Annahme der Einladung konnte der Benutzer auswählen, wie er das Projekt in seinen Workspace importieren möchte. Im einfachsten Fall werden alle Projektdateien über das Netzwerk direkt von einem auf den anderen Computer übertragen. Möglich ist auch, ein bereits vorhandenes Projekt auszuwählen, was sich anbietet, wenn beide Benutzer denselben Projektstand haben (z.B. direkt nach den Auschecken des Projekts aus einem Versionskontrollsystem). Im Test wurde das Projekt erfolgreich zum zweiten Benutzer kopiert, womit der Verbindungsaufbau abgeschlossen war. Beide Benutzer konnten nun am selben Projekt gleichzeitig arbeiten sowie wahlweise dem Partner „folgen". Alle Teilnehmer einer Saros-Sitzung haben standardmäßig Schreibrechte und können somit als Driver fungieren. Wer Driver ist und wer Navigator ist also in erster Linie eine Frage der Kommunikation. Derjenige, der eine Session initiiert, hat allerdings die Möglichkeit, anderen Teilnehmern die Schreibrechte wahlweise zu entziehen. Dadurch eignet sich Saros auch potenziell für Demonstrations- und Schulungszwecke, bei denen man Änderungen am Projekt durch andere Teilnehmer explizit unterbinden möchte.

Ein abschließender Vorabtest sollte zeigen, inwiefern die gleichzeitige Anmeldung eines Benutzer im Instant-Messenger Spark sowie in Saros zu unerwünschten Seiteneffekten führen könnte. Für XMPP ist in RFC3920 (2004) für einen solchen Fall das sogenannte „Resource Binding" definiert. Verschiedene XMPP-Clients dürfen demnach parallel mit denselben Benutzerdaten betrieben werden. Der Test zeigte, dass beide in Saros angemeldete Benutzer zeitgleich auch in Spark angemeldet sein und dort miteinander Sofortnachrichten austauschen konnten. Was dabei auffiel war, dass sich der Anwesenheitszustand eines Benutzers ändert, wenn das Eclipse-Fenster nicht mehr den Anwendungsfokus besitzt. Dann setzt Saros den Benutzerstatus auf „Abwesend" mit dem Hinweistext „Eclipse window is in the background". Da das im Prinzip ständig passiert (Nutzung eines Browsers, Arbeiten mit E-Mails), kann das für die Benutzer des Instant Messengers irritierend sein. Besonders kritisch daran ist, dass alle autorisierten Kontakte genau nachvollziehen können, ob der Anwender Eclipse geöffnet hat und ob er es gerade aktiv benutzt oder ob er ein anderes Programm verwendet. Somit verliert der Anwender einen Teil der eigenen Privatsphäre. Dieses Verhalten sollte daher optional sein und durch den Benutzer deaktiviert werden können.

Zusammenfassend hat der lokale Test gezeigt, dass Saros im zukünftigen Corporate Eclipse funktionieren kann. Endgültige Gewissheit werden die abschließenden Integrationstests zeigen.

### 4.3.4 Integrationstests

Nachdem erfolgreichen Abschluss der lokalen Tests sollen Integrationstests zeigen, ob das ins neue Corporate Eclipse integrierte Saros auch mit der vorhandenen technischen Infrastruktur funktioniert.

#### Planung

Für den abschließenden Integrationstest werden zwei Probanden das neue Corporate Eclipse auf ihrem Arbeitsplatzcomputer aus dem Versionskontrollsystem auschecken, das Eclipse starten, ihre XMPP-Daten in Saros hinterlegen und abschließend eine Verbindung mit dem fürs Instant Messaging verwendeten XMPP-Server aufbauen. Die Probanden prüfen, ob DPP-Sitzungen funktionieren. Der Saros-Administrator überwacht die Tests und prüft zusätzlich, ob alle genutzten Systeme weiterhin normal funktionieren.

Für den Funktionstest sollen zwei Szenarien betrachtet werden: Zuerst führen zwei Benutzer am selben Standort einen Integrationstest durch. Im Falle eines Problems kann der Saros-Administrator sofort einschreiten. Abschließend erfolgt ein Test, bei dem sich beide Probanden an unterschiedlichen Firmenstandorten befinden. Ergeben sich in diesem Szenario ebenfalls keine Komplikationen, gelten die Integrationstests als erfolgreich abgeschlossen.

#### Durchführung

Nachdem sich Saros mit einem lokal laufenden Openfire-Server verbinden konnte, blieb noch die Frage offen, ob der Betrieb auch mit dem Firmen-XMPP-Server funktionieren würde. Aufgrund der vorhandenen Restriktionen war das im Vorfeld nicht klar (siehe Abschnitt 3.1.1 Infrastruktur -> Serversoftware -> Openfire), und leider konnten die XMPP-Server-Administratoren aus organisatorischen Gründen nicht in den Test mit einbezogen werden. Auf Anfrage stimmten Sie aber einem ausführlichen Integrationstest zu. Voraussetzung war, dass parallel andere Kollegen mit dem Spark-Client am Server angemeldet waren und beobachteten, ob beim Verbindungsaufbau von Saros sowie im weiteren Betrieb Unregelmäßigkeiten auftreten würden. Die Tests begannen nun mit dem ersten Szenario.

**Gleicher Standort** Zwei Teammitglieder, die ihre Arbeitsplätze direkt nebeneinander hatten, meldeten sich zunächst mit ihren Spark-Clients am XMPP-Server an und kontrollierten, dass das Instant Messaging reibungslos funktionierte. Anschließend wurde von beiden Benutzern der Saros-Einrichtungsassistent gestartet. Jeder hinterlegte dort die auch im Spark-Client verwendeten Zugangsdaten. Anschließend wurde ein Verbindungsaufbau initiiert. Es traten sowohl in Saros als auch in dem Spark-Clients des Testers sowie der überwachenden

Teammitglieder keine unerwarteten Überraschungen auf. Auf beiden Systemen waren die Benutzer nun sowohl in Spark als auch in Saros angemeldet.

Der nächste Schritt beinhaltete den Aufbau einer gemeinsamen DPP-Sitzung. Für diesen Zweck importierten beide Benutzer ein bereits vorhandenes reales Java Enterprise-Projekt in ihren Workspace. Ein Entwickler lud anschließend den anderen Benutzer zu einer Saros-Sitzung ein. Auch dieser Vorgang funktionierte problemlos. Die gemessene Zeit für die Initialisierung sowie die Synchronisierung des Projekts betrug 20 Sekunden. Nachdem die Verbindung aber vollständig aufgebaut war, konnten beide Probanden flüssig und störungsfrei mit Saros arbeiten und im Paar programmieren.

**Verschiedene Standorte** Für das zweite Szenario wurde eine Saros-Sitzung zwischen Teammitgliedern in Hamburg (Deutschland) und Amsterdam (Niederlande) aufgebaut. Der Verbindungsaufbau funktionierte ebenso wie im ersten Szenario, benötigte allerdings erheblich mehr Zeit: 3 Minuten 15 Sekunden dauerte der Abgleich der Projektdaten. Diese Zeit konnte in einem zweiten Verbindungsaufbau auf deutlich unter eine Minute reduziert werden, indem beide Entwickler bereits denselben Projektstand besaßen und die Saros-Option „Use existing project" verwendeten. Hier bietet sich generell an, dass beide Entwickler das Softwareprojekt vor einem Verbindungsaufbau über ein Versionskontrollsystem synchronisieren. Diese handhaben solche Abgleiche anscheinend deutlich effizienter als Saros. Nachdem die Sitzung erfolgreich gestartet war, verlief auch diese Sitzung insgesamt gefühlt flüssig und ohne spürbare Verzögerungen. Technische Messungen der Übertragungsverzögerung wurden aufgrund des hohen technischen Aufwands nicht durchgeführt. Der subjektive Eindruck zur Übertragungsgeschwindigkeit der DPP-Daten wurde von Management als hinreichendes Kriterium akzeptiert.

## 4.4 Fazit

In diesem Kapitel wurde ein Weg gezeigt, wie eine Integration von Saros in bestehende IT-Infrastruktur aussehen kann. Am Beispiel der Firma Vattenfall Europe Information Services wurde ein zuvor ausgearbeitetes Konzept umgesetzt. Es zeigte sich, dass bei der Installation von Saros trotz vorheriger Planung innerhalb der Unternehmensumgebung Schwierigkeiten auftraten, mit denen vorher nicht gerechnet wurde und deren Lösung nicht trivial war. Nachdem Saros erfolgreich installiert war, verliefen die weiteren Tests erfolgreich. Durch die Anwendung des ausgearbeiteten Testkonzepts besteht nur noch ein geringes Restrisiko, dass bei der endgültigen Freigabe der neuen Eclipse-Installation infrastrukturbedingte Probleme auftreten.

# 5 Einführung ins Team

Dieses Kapitel erarbeitet ein Konzept, wie Saros sowie die dahinter stehenden Konzepte zu Leben erweckt werden können. Abschnitt 5.1 geht zunächst auf die bestehende Situation bei der Vattenfall Europe Information Services ein. Abschnitt 5.2 nennt zwei Möglichkeiten, wie Prozessänderungen in ein Team Einzug erhalten können. In Abschnitt 5.3 wird dann ein Weg aufgezeigt, wie eine Informationsveranstaltung für die betroffenen Mitarbeiter vorbereitet und durchgeführt werden kann. Am Ende werden die Ergebnisse in einem kurzen Fazit zusammengefasst.

## 5.1 Ausgangssituation

Der erfolgreiche Abschluss der technischen Integration in die IT-Landschaft der VEIS war der erste Schritt, um verteilte Paarprogrammierung in den Software-Entwicklungsprozess des Unternehmens zu integrieren. Die Existenz einer technischen Lösung allein bewirkt aber nichts, so lange es niemanden gibt der sie aktiv einsetzt. Deshalb müssen im nächsten Schritt die Entwickler, die Saros zukünftig verwenden sollen, umfassend über das neue Werkzeug informiert werden. Da das Werkzeug Saros sowie das damit einhergehende Programmierparadigma den Teammitgliedern bisher nicht bekannt sind, müssen die Entwickler zunächst eine Einführung in die Thematik erhalten.

Des Weiteren erscheint es unrealistisch, Paarprogrammierung von Beginn an im gesamten Team dauerhaft anwenden zu können, denn das Team hat in der Vergangenheit bisher keinerlei Erfahrungen mit dieser Arbeitstechnik gesammelt. Allerdings unterstützen sich die Teammitglieder regelmäßig gegenseitig bei Problemen, deren Lösung Einsicht in den Quellcode des Fragenden erfordert. In dieser Situation setzen sich schon jetzt der Fragende sowie der Helfende gemeinsam vor einen Rechner, um die Problemursache gemeinsam zu lokalisieren und zu beheben. In dieser besonderen Situation wird Paarbildung bereits gelebt. Beide Entwickler nutzen dieselbe Workstation und kommunizieren über denselben Code. Welcher der Beteiligten die Rolle des Drivers bzw. Navigators annimmt ist dabei von Fall zu Fall unterschiedlich. Jeder der Beteiligten nutzt dabei die vorhandene Peripherie wie Tastatur und Maus, zeigt mögliche Passagen im Quellcode und spricht seine Gedanken dazu aus. Im Gegensatz zu echter Paarprogrammierung, bei der dauerhaft zwei Entwickler miteinander

am Projekt arbeiten, trennt sich hier das Paar jedoch nach erfolgreicher oder erfolgloser Lösung des Ursprungsproblems und jeder arbeitet wieder alleine für sich. Trotzdem ist dieses Verfahren ein guter Ansatzpunkt für eine erfolgreiche Saros-Integration ins Team: Der bereits funktionierenden Mechanismus der Paarbildung bei der gemeinsamen Lösung von Problemen hat das Potenzial, durch geeignete technische Unterstützung soweit vorangetrieben zu werden, dass auch bei vorliegender räumlicher Distanz ein gemeinsames Unterstützen auf Code-Ebene in Echtzeit möglich ist.

Basierend auf diesen Informationen wird nun ein Konzept zur Einführung von Saros auf Teamebene erstellt.

## 5.2 Treibende Kraft

Grundsätzlich gibt es zwei Möglichkeiten, wer die Integration von Saros in das Entwicklerteam forciert: Entweder möchten ein oder mehrere Mitglieder des Teams den Prozesswechsel initiieren, oder es wird eine Anweisung durch das Management erlassen, die den Prozesswechsel diktiert. Beide Wege haben ihre Vor- und Nachteile, auf die im Folgenden eingegangen wird.

Im ersten Fall erkennen ein oder mehrere Entwickler eine Gelegenheit, einen bestehenden Prozess durch den Einsatz einer neuen Technologie zu optimieren. Sie evaluieren diese zunächst für sich im Kleinen, später dann möglicherweise in einem laufenden Projekt. Bewährt sich die vorgenommene Änderung, werden einige oder alle Teammitglieder informell über den Nutzen der neuen Technologie und in die gemachten Erfahrungen eingeweiht, so dass sich diese nun ihrerseits damit auseinander setzen. Auf diesem Weg werden Änderungen in eine Gruppe integriert. Am Ende besteht beim Team der Wunsch nach Veränderung. Dieser Wunsch wird dann dem Management mitgeteilt mit dem Hinweis, dass die Prozessänderung bereits erfolgreich im Stillen getestet wurde. Die Einführung erfolgt also von „unten” nach „oben”. Vorteilhaft an diesem Weg ist, dass die Teammitglieder in einer stillen Evaluierungsphase für sich selbst entscheiden können, ob die vorgenommenen Änderungen eher gut oder eher schlecht sind. Unabhängig davon ist die Wahrscheinlichkeit geringer, dass allein der Gedanke an einen Prozesswechsel zu Ablehnung führt, wie es bei einer diktierten Änderung manchmal der Fall sein kann. Ein gravierender Nachteil dieser Vorgehensweise besteht in der Möglichkeit, dass das Management die Prozessänderung trotz positiver Erfahrungen der Teammitglieder ablehnt.

Der andere Weg funktioniert vollständig anders. Hier sieht das Management ein existierendes Problem, für welches es eine technische Lösung herbeiführen möchte. Ein Mitarbeiter wird damit beauftragt, mögliche Kandidaten zu evaluieren und anschließend eine für den Teilbereich des Managers geeignete Lösung zu präsentieren. Steht die technische Integration, werden die Mitarbeiter über die geplante Einführung der neuen Technologie unterrichtet.

Die Einführung erfolgt von „oben” nach „unten”. Der große Vorteil dieser Vorgehensweise ist, dass die gewünschten Änderungen ins Team integriert werden, ohne dass diese von den Teammitglieder ohne guten Grund verweigern können. Daraus ergibt sich unmittelbar der größte Nachteil: Akzeptiert das Team die gewünschte Lösung nicht, wird es sie mit hoher Wahrscheinlichkeit nur halbherzig einsetzen. Dabei ist es unerheblich, ob die Änderung auch für die Teammitglieder vorteilhaft ist oder nicht. So lange das Team nicht davon überzeugt ist, wird die Zufriedenheit der Mitarbeiter sinken.

Für den Rest dieses Kapitels wird ausschließlich das zweite Szenario betrachtet: Das Management möchte eine Prozessänderung im Team etablieren. Damit die Einführung ins Team nicht von Beginn an zu scheitern droht, sollte die Etablierung in mehreren Phasen durchgeführt werden:

1. Die Mitarbeiter werden umfassend über die neue Technologie sowie das zugrunde liegende Paradigma informiert.
2. Es erfolgt eine einführende Schulung, damit die Mitarbeiter lernen, die Prozesse korrekt durchzuführen.
3. Die Mitarbeiter wenden das Gelernte selbstständig an. Die ersten Gehversuche werden dabei von einem erfahrenen Coach begleitet, so dass bei ungewollten Abweichungen oder unerwarteten Problemen geholfen werden kann.

In den folgenden Abschnitten werden diese Phasen konzeptionell ausgeführt. Darauf basierend können dann konkrete Informationsmaßnahmen eingeleitet und durchgeführt werden. Diese wurden im Rahmen dieser Ausarbeitung jedoch nicht durchgeführt, so dass zu dem nun folgenden Konzept keine praktischen Erfahrungen existieren.

## 5.3 Einführungsveranstaltung

Der erste Schritt besteht darin, in Form einer Präsentation die Mitarbeiter über die Einführung von Saros zu informieren. Der Sinn dieser Veranstaltung ist, dass den Teammitgliedern zum einen deutlich gemacht wird, warum Saros eingeführt wird und welchen Mehrwert es bringt. Zum anderen sollten alle Teilnehmer der Informationsveranstaltung eine positive Einstellung der auf sie zu kommenden Änderungen erhalten, da nur so die Einführung von Saros ein echter Erfolg werden kann. Nach Ryan (2010) existieren vier grundlegenden Basistechniken, um technische Veränderungen erfolgreich in bestehende Teamprozesse integrieren zu können:

1. Kompetenz erlangen
2. Die Nachricht überliefern
3. Die Technik demonstrieren

4. Vertrauen erzeugen

Da die Einführung von Saros neben der technischen auch Änderungen auf sozialer Ebene verursachen, muss die Informationsrunde zum einen die grundsätzlichen Vorteile von Paarprogrammierung und Code-Reviews verdeutlichen. Zum anderen muss dargestellt werden, warum dieser Prozess auch über Standortgrenzen hinweg durchgeführt von Bedeutung ist. Letzteres ist zwar lediglich eine konsequente Umsetzung der zugrunde liegenden Theorie, allerdings ist genau dieser Aspekt der Grund für die gewollte Einführung von Saros. Daher muss darauf geachtet werden, dass die zugrunde liegende Theorie den Teilnehmern klar und nachvollziehbar präsentiert wird. Sollte bereits die Vermittlung der Grundlagen scheitern, wird der zweite Teil, die technische Einführung, bereits vor Beginn deren Präsentation scheitern.

### 5.3.1 Kompetenz erlangen

Kompetenz ist ein weitreichender Begriff. Im vorliegenden Kontext bedeutet „Kompetenz erlangen", dass sowohl die theoretischen Konzepte verteilter Paarprogrammierung als auch die Bedienung und Funktionsweise von Saros derart gut verinnerlicht sind, dass kritische Rückfragen idealerweise bereits während der Präsentation qualifiziert beantwortet werden können. Ein gewisses Maß an Teilkompetenz wurde bereits mit der Grundlagenrecherche im Kapitel 2 sowie der Durchführung der technischen Integration in Kapitel 4 erlangt. Insbesondere der im vorausgegangenen Kapitel durchgeführte Integrationstest zeigte, dass Saros in der Unternehmens-IT grundsätzlich funktioniert. Es wurde jedoch bisher keine tief gehenden Fachkenntnisse bezüglich der korrekten, effektiven Bedienung des Plugins erworben. Um echte fachliche Kompetenz zu erlangen, ist es daher notwendig, sich tiefer in Saros einzuarbeiten. Um die theoretischen Details zu verinnerlichen, sollte zum einen die vorhandene Dokumentation auf dem offiziellen Internetauftritt des Saros-Projekts eingehend studiert werden. Dort gibt es neben der Bedienungsanleitung auch darüber hinaus gehende Hintergrundinformationen, beispielsweise zur internen Architektur des Plugins sowie die im konkret verwendeten Technologien. Zum anderen sollte der Vortragende auch über ein gewisses Maß an praktischen Erfahrungen besitzen. Dazu können die auf der Saros-Website beschriebenen Vorgänge auf dem lokalen Testsystem mit zwei parallel laufenden Eclipse-Instanzen nachvollzogen werden. Hierbei sich ergebene Fragestellungen, Sonderfälle sowie Dokumentationslücken werden in einem Projekt-Wiki notiert und im Anschluss an die Lernphase separat betrachtet. Wichtig ist, dass man nicht einfach auf gut Glück los klickt und hofft, dabei die richtigen Erfahrungen zu erhalten. Stattdessen sollte der bisherige Entwicklungsprozess mindestens ein Mal vollständig unter Verwendung von Saros durchgespielt werden. Dadurch ergeben sich mit Sicherheit Fragen, die in der Benutzerdokumentation nicht beantwortet werden, wie beispielsweise diese:

- Wenn zwei Benutzer eine Datei ändern und einer auf Speichern drückt, wird die Datei

dann auch beim anderen Entwickler automatisch gespeichert?

- Wenn der geteilte Programmcode unter Versionskontrolle steht, wer checkt dann den Code ein? Was passiert dann beim anderen Entwickler?

Eine vollständige Liste kann an dieser Stelle nicht präsentiert werden. Dafür sind die Unterschiede in den möglichen Einsatzszenarien zu groß und die Prozesse zu verschieden. Intensives Auseinandersetzen mit Saros in der zu erwartenden Zielumgebung ist aber ein wichtiger Schritt, mögliche Fallstricke frühzeitig auf zu decken. Durch diese strukturierte Vorgehensweise werden schließlich nicht nur die Kenntnisse des treibenden Projektverantwortlichen gefestigt. Bei der genaueren Betrachtung von Theorie und Praxis können und sollen auch kritisierbare Aspekte notiert, hinterfragt und gewichtet werden. Es ist notwendig, mögliche Gründe, die eine Einführung von Saros erschweren können, bereits im Voraus zu kennen. Gibt es akzeptable Wege, mit solchen Problemen umzugehen, so sollten diese im Voraus bekannt sein. Existiert kein solcher Weg und ist das Problem kritisch, muss der erwartete Mehrwert, den der Einsatz des neuen Werkzeugs bringen soll, gegen die zu erwartenden Problemen abgewägt werden.

### 5.3.2 Die Nachricht überliefern

Nachdem das eigene Wissen um das Produkt sowie die zugrunde liegenden Konzepte bestmöglich erweitert wurde, wird mit der Erstellung der eigentlichen Präsentation begonnen. Dabei wird ein besonderes Augenmerk darauf gelegt, dass die Teilnehmer nicht nur die zugrunde liegenden Konzepte verstehen, sondern auch den daraus für sie resultierenden Nutzen, der durch den Einsatz von Saros möglich ist. Für die Präsentation ist entscheidend, die wesentlichen positiven Aspekte zu vermitteln, welche die Verwendung von Saros mit sich bringt. Dabei gilt es zu vermeiden, kritikwürdige Aspekte zu vertiefen, sofern diese keine schwerwiegenden Gründe gegen eine Einführung von Saros sind. Die Präsentation soll in erster Linie den Nutzen von Saros hervorheben, nicht mögliche unkritische Probleme. Den bisherigen Status Quo ausführlich zu erläutern ist zwar verlockend, aber aus dem vorher genannten Grund eher keine gute Idee. Möglicherweise wurden zum Zeitpunkt der Informationsveranstaltung qualitätssichernde Maßnahmen wie Code Reviews oder Qualitätssicherung als separate Testphase vor der Auslieferung an den Kunden wegen fehlender zeitlicher und finanzieller Ressourcen noch nicht durchgeführt. Also muss sich die Präsentation auf die zu erwartenden Vorteile konzentrieren. Williams (2010) hat verschiedene Studien analysiert und folgende Vorteile von Paarprogrammierung zusammengefasst:

1. Verteilung der Verantwortung auf mehrere Schultern
2. Mehr Abwechslung durch regelmäßigen Rollenwechsel
3. Weniger Defekte durch ständiges Code-Review
4. Besserer Code durch frühzeitig durchgeführte Refactorings

5. Zusätzlicher Fokus auf der Architektur
6. Weniger Frustration bei der Programmierung

In der klassischen Programmierung werden häufig einzelne Module von einem einzelnen Entwickler modelliert und programmiert. Dadurch ist ein Programmierer nicht nur für seinen erzeugten Quellcode verantwortlich, sondern er muss auch Sorge tragen, dass sein Modul die Schnittstellen anderer Module korrekt einbindet. Zusätzlich soll die eigene Schnittstelle von anderen Modulen sinnvoll einsetzbar sein, so dass man sich über die eigenen Probleme auch Gedanken über andere Nutzer des eigenen Codes machen muss. Paarprogrammierung kann diese Mehrfachverantwortung auf mehrere Rollen verteilen. Der Driver beschränkt sein primäres Denken auf das Lösen akuter Probleme. Gleichzeitig denkt der Navigator in größeren Dimensionen und überlegt sich, wie die entstehende Architektur verbessert werden kann. Seine Gedanken dazu teilt er dem Driver mit, oder er nimmt in einem eigenen Refactoring-Zyklus selbst die Rolle des Drivers ein. Dadurch teilen sich beide Programmierer die Verantwortung für den entstanden Code. Ein weiterer Aspekt ist, dass der Navigator durchs Zusehen des Drivers laufend den Code reviewt. Dadurch erhöht sich die Wahrscheinlichkeit, dass während der Programmierung erzeugte Bugs und unsauber codierte Passagen umgehend bemerkt und korrigiert werden, bevor sie in den gemeinsamen Codestand integriert werden.

Ein weiterer Aspekt ist, dass die Paarbildung das allgemeine IT-Wissen einzelner auf das ganze Paar überträgt. Kennt sich jemand besonders gut mit einzelnen Technologien aus und bindet diese während der Entwicklung ein, dann lernt der Navigator automatisch mit dazu. Das funktioniert vor allem deshalb gut, weil als Navigator unklare Sachverhalte stets hinterfragt werden sollen, bis beide Entwickler die vorgeschlagene Lösung qualifiziert beurteilen können. Paarprogrammierung verteilt also nicht nur projektbezogenes Wissen, sondern auch allgemein IT-Wissen zwischen den Paaren. Das ist ein besonderer Grund, warum sich Paare regelmäßig neubilden sollten: Nur dann erreicht der Wissenstransfer das gesamte Team.

Der letzte hier genannte große Vorteil ist, dass einem Paarprogrammierung das Gefühl gibt, am Ende des Tages wirklich produktiv gewesen zu sein. Tatsächlich arbeitet man im Paar meistens deutlich fokussierter als allein. Die Disziplin bei der Programmierung steigt, wodurch kleinere „Sünden" ja direkt vom Navigator bemerkt werden. Der Driver achtet dadurch mehr als sonst auf das Einhalten von Coding-Standards und integriert den eigenen Code häufiger ins gemeinsame Code-Repository.

Damit hat man stimmige Argumente, um die Nützlichkeit von Paarprogrammierung zu unterstreichen. Ryan (2010) weist an dieser Stelle aber noch eindrücklich darauf hin, dass bei allem vorhandenen Enthusiasmus aus der sachlichen Präsentation keine Werbeveranstaltung wird. Bei allen positiven Argumenten muss die eigene Glaubwürdigkeit gewahrt bleiben, ansonsten werden die Teilnehmer misstrauisch gegenüber der vermittelten Botschaft. Das gilt es zu vermeiden.

### 5.3.3 Die Technik demonstrieren

Die theoretischen Konzepte und Vorteile verteilter Paarprogrammierung zu nennen ist wichtig, um das Verständnis und Interesse der Mitarbeiter für den Einsatz der neuen Konzepte „Code Review" und „Paarprogrammierung" zu fördern. Wird allerdings Saros ebenfalls ausschließlich theoretisch behandelt, wird sich den Teilnehmern dessen praktischer Nutzen nur schwer erschließen. Deshalb bietet sich im Rahmen der Präsentation eine Live-Vorführung an.

Mit einer Live-Demonstration erhalten die Teilnehmer einen direkten Eindruck davon, wie die Software aussieht, was sie kann und wie man sie einsetzen kann. Der Vortragende kann sich die interessantesten Features aussuchen und diese dazu verwenden, um die eigene Begeisterung auf die Zuschauern zu übertragen. Durch dieses Maß an gesteuerter Transparenz kann das Vertrauen der Zuschauer weiter gestärkt werden. Man kann zusätzlich auch das Plenum im Anschluss an die eigene Vorführung dazu auffordern, eigene Fragen direkt als Teil der Vorführung beantworten zu lassen. Wenn das problemlos funktioniert, werden auch Teilnehmer, die dem Produkt Saros anfangs noch kritisch gegenüberstehen, mit großer Wahrscheinlichkeit ihre Vorbehalte ablegen.

Eine Live-Vorführung birgt aber immer auch Risiken. Das größte Risiko ist, dass etwas im Vorführungsaufbau nicht wie geplant funktioniert, z.B. wenn der XMPP-Server nicht startet bzw. nicht über das Netzwerk erreichbar ist. Das hinterlässt beim Publikum immer einen schlechten Eindruck nach dem Motto „Wenn es schon nicht in der Informationsrunde funktioniert, wie soll es erst im echten Einsatz werden?". Eine andere Gefahr lauert, wenn man im Vorfeld kritische Probleme mit der Software bewusst außen vor genannt hat und man anbietet, die Fragen der Teilnehmer anschaulich innerhalb der Vorführung zu beantworten. Gibt es einen Zuschauer, der diese kritische Schwachstelle kennt, wird er sie mit hoher Wahrscheinlichkeit auch ansprechen. Das Beantworten von Fragen sollte wirklich nur dann angeboten werden, wenn man sich wirklich keiner kritischen Schwachstelle bewusst ist. Trotz allem kann es passieren dass ein Zuschauer auf ein kritisches Problem hinweist, von dem man nichts wusste. Das ist zwar schlecht für die Präsentation, sollte aber als Chance gesehen werden, sich noch rechtzeitig vor der endgültigen Einführung mit diesem speziellen Punkt auseinandersetzen zu können und dann je nach Ergebnis Konsequenzen daraus zu ziehen.

Wie kann nun eine Live-Vorführung aussehen? Für eine Saros-Demonstration können äquivalent zum in Unterabschnitt 4.3.3 beschriebenen Aufbau zwei Eclipse-Instanzen mit verschiedenen Workspaces gestartet und auf dem Bildschirm nebeneinander platziert werden. Durch diesen Aufbau sehen die Zuschauer stets beide Seiten der Saros-Sitzung. Nichts wird versteckt. Je nachdem, ob XMPP bereits im Unternehmen verwendet wird, können nun über den Saros-Einrichtungsassistenten entweder neue Benutzer angelegt oder bestehende Benutzerdaten hinterlegt werden. Anschließend wird Saros in beiden Instanzen gestartet und die Benutzer fügen sich gegenseitig zum Roster hinzu, sofern das nicht bereits im Vorfeld

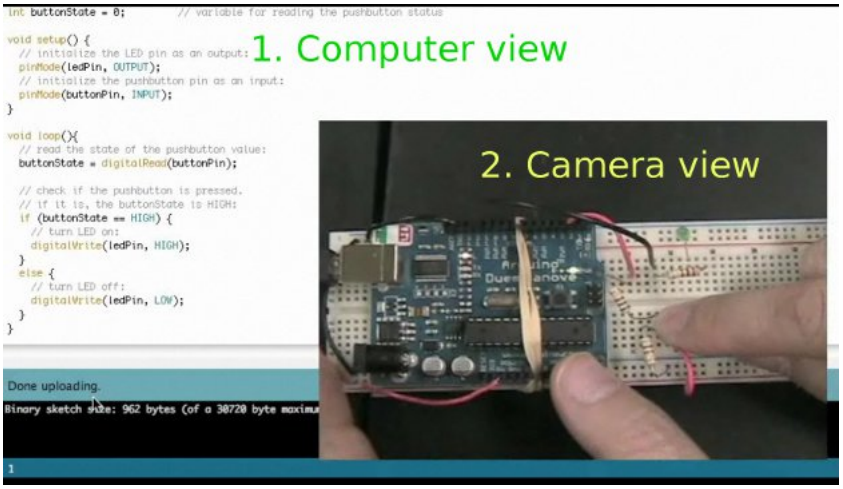


Abbildung 5.1: Splitscreen-Präsentation, 1: Bildschirmdarstellung, 2: Kamerabild (Falconphysic, 2010)

geschehen ist. Der weitere Verlauf sollte davon abhängig gemacht werden, mit welcher Art von Eclipse-Projekten die Zuschauer üblicherweise arbeiten. Sinnvoll ist, sich ein entsprechendes, im Voraus erstelltes Projekt zu importieren oder aus einem VCS auszuchecken und dann einen kurzen Arbeitszyklus durchzugehen. Dazu gehört das gemeinsame Editieren einer Datei, das Folgen des Drivers beim Navigieren durch verschiedene Dateien, das Ausführen automatisiert ablaufender Tests, ein kurzer Debug-Durchlauf sowie (falls ein VCS verfügbar ist) die abschließende Integration der Änderungen in ein Code-Repository.

Eine andere Möglichkeit besteht darin, sich für die Vorführung einen Partner zu holen, der auf einem eigenen Rechner jeweils einmal die Rolle des Navigators und des Drivers einnimmt, so dass man auf dem für die Präsentation verwendeten Beamer beide Rolle jeweils einmal kennenlernt. Dieses Beispiel entspricht eher dem realen Einsatzszenario, erfordert aber auch eine bereits funktionierende Infrastruktur und eine vorhandene Netzwerkverbindung. Daher steigt das Risiko technischer Komplikationen. Allerdings besticht dieses Szenario durch die Möglichkeit, einigen Zuschauern die Möglichkeit zu geben, den Partner abzulösen und selbst Teil der DPP-Sitzung zu werden. Verfügt der Präsentationsraum über eine hinreichend stabile technische Infrastruktur, kann diese Form der Präsentation sehr vielversprechend sein.

Ein letzter Punkt, den es zu berücksichtigen gilt, ist die Frage, ob sich alle Zuschauer vor Ort befinden oder ob die Präsentation bereits über das Netzwerk an verschiedene Standorte übertragen wird. Moderne Präsentationswerkzeuge erlauben wie in Abbildung 5.1 dargestellt ein Live-Online-Streaming der eigenen Präsentation, bei dem gesteuert werden kann, ob die entfernten Zuschauer den Vortragenden, dessen Bildschirmpräsentation oder einen Splitscreen mit beiden Elementen zu sehen bekommen.

### 5.3.4 Erzeuge Vertrauen

Sowohl die Präsentation als auch die Demonstration werden bei den Teilnehmern zu Fragen führen. Einige davon werden rationaler Art sein, während andere aufgrund von irrationalen Ängsten entstehen. Die Handhabung sowie die Antworten spielen eine entscheidende Rolle, ob das einzuführende Produkt von den Mitarbeitern akzeptiert wird oder nicht. Aber auch die Inhalte der Präsentation können Vertrauen ebenso erzeugen, wie sie es zerstören können. Beispielsweise wird es ein schlechtes Bild auf die Präsentation werfen, wenn offensichtliche Probleme unerwähnt bleiben. Auf der anderen Seite wird ein detailliertes Aufzählen selbst eher unwichtiger Kritikpunkte zu einem Misstrauen der Lösung an sich führen. Der Weg zu Vertrauen gleicht also ein Balanceakt, bei dem sich Nennen und Verschweigen möglicher Kritikpunkte die Waage halten müssen. Idealerweise ist dem Vortragenden bewusst, warum bestimmte negative Informationen verschwiegen werden, sei es weil sie anderweitig kompensiert werden können oder weil sie die Nützlichkeit des Produkts nicht wesentlich einschränken. Entscheidend ist, dass man sich möglicher Schwächen sowie Kritik im Vorfeld bewusst ist. Im Falle von Paarprogrammierung kann mit an Sicherheit grenzender Wahrscheinlichkeit davon ausgegangen werden, dass in der Fragerunde jemand die Kostensicht betrachtet und unterstellt, die Anwendung von Paarprogrammierung verdopple die Projektkosten. Ohne valide Gegenbeispiele wird sich solche Kritik nicht entkräften lassen.

## 5.4 Fazit

In diesem Kapitel wurde sich mit der Frage auseinandergesetzt, wie eine durch das Management durchgesetzte technische Neuerung im Allgemeinen sowie die Einführung von Saros im Speziellen im Rahmen einer Einführungsveranstaltung für Mitarbeiter eingeführt werden kann. Es wurde sowohl auf mögliche Chancen als auch Risiken einer solchen Veranstaltungen eingegangen. Außerdem wurden relevante inhaltliche Fragestellungen sowie Argumente für den Einsatz (verteilter) Paarprogrammierung aufgeführt, welche im Rahmen einer Präsentation eingesetzt werden können. Zusätzlich wurden verschiedene Tipps zur Art und Weise gegeben, wie die Präsentation aufgebaut und durchgeführt werden kann.

# 6 Bewertung der Lösung

Nachdem die technische Integration erfolgt ist und für die Teameinführung ein Konzept vorliegt, sollen in diesem Kapitel die Ergebnisse der Kapitel 4 und 5 ausgewertet werden. Zu diesem Zweck werden die Anforderungen aus Abschnitt 3.3 für die Anwendungsszenarien aus Abschnitt 3.4 auf Basis der erzielten Resultaten bewertet. Am Ende dieses Kapitels werden die wichtigsten Ergebnisse nochmals zusammengefasst.

## 6.1 Auswertung der Anforderungen

Die Bewertung erfolgt auf Basis der Anforderungen aus der Analyse.

**Als Manager möchte ich, dass eine DPP-Sitzung ähnlich komfortabel durchgeführt werden kann wie eine normale PP-Sitzung, damit beide Prozesse eine vergleichbare Qualität haben und wir Reise- und Übernachtungskosten sparen können.** Für die produktive Anwendung von Paarprogrammierung in Projekten, bei denen sich alle Entwickler am gleichen Firmenstandort befinden, ist der Einsatz von Saros nur selten sinnvoll. Der Vorteil, für eine Paarprogrammierungssitzung den eigenen Arbeitsplatz nicht verlassen zu müssen wiegt die fehlende visuelle Wahrnehmung von Gestik und Mimik des Programmierpartners nicht auf. Obwohl sich die Bedienbarkeit von Saros bereits auf einem hohen Niveau befindet, ist es bei der Klärung komplexer Sachverhalte nach wie vor schneller und effektiver, für deren Skizzierung Zettel und Stift oder ein Whiteboard zu verwenden. Insgesamt komfortabler ist es daher, eine reale Paarbildung durchzuführen.

Unter Berücksichtigung des Kostenaspekts ist der Einsatz für verteilte Teams aber durchaus lohnenswert. Einige Studien zu verteilter Paarprogrammierung wurden von der University of North Carolina at Chapel Hill sowie der North Carolina State University mit Studenten durchgeführt. Beide kamen zum Ergebnis, dass eine Paarbildung über das Internet ein großartiges Potenzial biete und sich im Vergleich zu verteilten Teams ohne Paarbildung als nutzbringend erweise(Williams, 2010). Ein weiteres Ergebnis war allerdings auch, dass die Probanden die klassische Paarprogrammierung der verteilten vorziehen. Beide Wege seien aber grundsätzlich als positiv eingestuft worden. Dazu muss erwähnt werden, dass in den Studien DPP über Bildschirmfreigabe angewendet wurde. Der von Saros verwendete Ansatz der Collaboration Awareness hebt die Qualität verteilter Paarprogrammierung auf ein neues Level.

Leider konnten keine Studien gefunden werden, in denen kleine Entwicklergruppen an verschiedenen Standorten Paarprogrammierung auch, aber nicht nur standortübergreifend anwenden möchten. Aus den Ergebnissen für die anderen beiden Anwendungsszenarien lässt sich aber das ideale Vorgehen ableiten: Befindet sich das Paar am gleichen Standort, versammeln sich beide Teilnehmer an einem Rechner. Sind die Standorte der Entwickler verschieden, ist Saros das Mittel der Wahl.

**Als Manager möchte ich, dass die Entwickler mehr miteinander kommunizieren, damit sie ihr Wissen miteinander teilen.** Die Frage, ob die Entwickler durch Anwenden von (verteilter) Paarprogrammierung mehr miteinander kommunizieren hängt auch davon ab, wie viel die Entwickler vorher miteinander kommuniziert haben. Je weniger vor der Einführung miteinander gesprochen wurde, desto größer wird der Erfolg in Bezug auf Kommunikation sein - wenn die Entwickler die Praktiken der Paarprogrammierung berücksichtigen. Ein Mehr an Kommunikation wird durch Paarprogrammierung durchaus erzwungen, wenn man sich permanent mit den Tätigkeiten des Teampartners auseinandersetzt. Dazu bedarf es aber keiner technischen Unterstützung, wenn sich die Entwickler in kurzer Zeit persönlich treffen können.

Während bei der direkten Paarprogrammierung der Navigator gezwungen ist, dem Driver bei der Entwicklung zuzusehen, kann bei verteilter Paarprogrammierung jederzeit der Follow-Modus deaktiviert werden, so dass sich der Navigator unabhängig vom Driver im Projektcode bewegen kann. Außerdem bekommt der Driver nicht mit, wenn der Navigator in der gemeinsamen Saros-Sitzung nicht aktiv die Eingaben des Drivers verfolgt. Eine wirksame Möglichkeit, diesem Problem zu begegnen sind regelmäßige Rückfragen des Drivers an den Navigator. Kommunikation bleibt auch bei verteilter Paarprogrammierung das wichtigste Bindeglied zwischen den Teilnehmern.

Generell hängt es von den persönlichen Vorlieben ab, ob jemandem das Programmieren nach dem Vier-Augen-Prinzip gefällt oder nicht. Der überwiegende Teil durchgeführter Studien belegt aber, dass die Mehrheit der Entwickler Paarprogrammierung als Bereicherung des Arbeitsablaufs wahrnimmt - wenn sie erst einmal mit der neuen Technik vertraut sind. Die Eingewöhnungsphase braucht allerdings etwas Zeit, die man zu Beginn aufbringen muss.

**Als Manager möchte ich, dass die Entwickler gegenseitige Code-Reviews durchführen, damit sich die Qualität der Software verbessert.** Die Schlussfolgerungen des vorherigen Absatzes gelten auch für Code-Reviews. Code-Reviews sind eine Frage der Gewöhnung. Allerdings gibt es verschiedene Arten von Code-Reviews, auf die Cohen (2006) im Detail eingeht. Wenn man keine davon kennt, kann der Prozess schnell jeglichen Nutzen verlieren. Eine einfach umzusetzende Form ist ein Code-Review, bei dem ein Entwickler aus einem anderen Team den eigenen Code auf Probleme überprüft. Dass es jemand anders

als man selbst ist ist dabei entscheidend, denn man selbst ist gewöhnlich so sehr mit den Programmcode verbunden, dass man Fehler kaum wahrnimmt (Cohen, 2006). Ansonsten gilt wie bereits genannt: Klassische Paarbildung am selben Standort sollte gegenüber der verteilten Paarbildung die bevorzugte Wahl sein.

**Als Administrator möchte ich die bestehende Saros-Installation im Corporate Eclipse gegen eine neuere Version austauschen können, damit die Entwickler von neuen Features sowie Fehlerbereinigungen profitieren können.** Wie im Analyseteil erwähnt schreitet die Entwicklung von Saros kontinuierlich voran. Im Schnitt monatlich erscheinen neue Releases, welche Fehler bereinigen und neue Funktionalitäten anbieten. Gleichzeitig empfehlen die Saros-Entwickler aber, möglichst niemals eine DPP-Sitzung aufzubauen, wenn die Teilnehmer verschiedene Saros-Versionen verwenden. Für dieses Problem existiert innerhalb der VEIS bereits eine halbautomatische Lösung. Das Corporate Eclipse wird einmal gepflegt und dann an alle Entwickler der Aufruf geschickt, ihre lokale Eclipse-Installation durch ein Update mit dem Stand im Subversion-Repository abzugleichen. Gut daran ist, dass die Änderung nur einmal durchgeführt werden muss und danach allen anderen zur Verfügung gestellt werden kann. Nicht so gut ist jedoch, dass die Entwickler selbst in der Verantwortung stehen, ihre lokale Installation zu aktualisieren. Auf allen Arbeitsplätzen befindet sich zwar ein Programm, welches Software aus dem Netzwerk automatisiert nachinstallieren kann, allerdings kann es nur von den Administratoren der Arbeitsplatzcomputer verwendet werden. Hier gibt es zumindest noch weiteres Optimierungspotenzial.

**Als Entwickler möchte ich eine kurze Einführung in das Thema „Verteilte Paarprogrammierung", damit ich mich darauf einstellen kann, was mich in Zukunft erwarten wird.** Mit Kapitel 5 Einführung ins Team wird ein möglicher Weg beschrieben, wie diese Anforderung erfüllt werden kann. Indem sich das Vorgehen stark an Ryan (2010) orientiert, wird mit dem vorliegenden Konzept ein erprobtes Verfahren angewendet, um die Themen „Verteilte Paarprogrammierung" und „Saros" auf strukturierte Weise den zukünftigen Anwendern zu präsentieren und demonstrieren. Die Hinweise auf mögliche Risiken erlauben es den für eine Einführung verantwortlichen Personen, qualifizierte Entscheidungen darüber zu treffen, welche Elemente bei der Einführung eingebaut werden und auf welche verzichtet wird.

**Als Entwickler möchte ich gezeigt bekommen, wie man Paarprogrammierung und Saros korrekt verwendet, damit ich verteilte Paarprogrammierung bestmöglich anwenden kann.** Die grundlegende Bedienung wird in der Live-Demonstration gezeigt. Darüber hinaus bietet Saros aber auch die erweiterten Funktionen Whiteboard, Chat und Bildschirmfreigabe. Diese Features werden im Einführungskonzept nicht vorgestellt, um nicht vom wesentlichen Kern des Plugins abzulenken. Anwendern den vollständigen Funktionsumfang

von Saros zu vermitteln ist ein eigenes Thema, welches in dieser Arbeit nicht weiter ausgeführt wird.

**Als Entwickler möchte ich, dass der Verbindungsaufbau einer Saros-Sitzung maximal 60 Sekunden dauert, damit ich nicht zu viel Zeit mit dem Warten auf den Beginn der DPP-Sitzung verliere.** Im Laufe der Integrationstests nach der erfolgreichen Saros-Installation wurden bereits Zeitmessungen für den Verbindungsaufbau durchgeführt. Mit 3 Minuten 15 Sekunden gab es exakt einen Fall, bei dem das 60-Sekunden-Limit verletzt wurde. Diese Überschreitung war das Resultat eines Verbindungsaufbaus zwischen Hamburg und Amsterdam, bei dem ein kleines vollständiges JEE-Maven-Projekt zwischen den Entwicklern ausgetauscht wurde. War das Projekt bereits auf beiden Arbeitsplätzen synchron, verringerte sich die Zeit für den Verbindungsaufbau auf deutlich unter eine Minute. Ein Auschecken aus dem Code-Repository vor Beginn einer Saros-Sitzung ist daher empfehlenswert.

## 6.2 Fazit

Mit Ausnahme der zwei letztgenannten Anforderungen konnten in den Kapiteln 4 (Technische Integration) und 5 (Einführung ins Team) die Wünsche, welche aus dem Epic „Entwickler sollen mit Hilfe von Eclipse verteilte Paarprogrammierung anwenden können” abgeleitet wurden, durchgehend erfüllt werden. Der Integrationsplan sowie das Einführungskonzept decken zusammen die wichtigsten Anforderungen ab. Für die Anforderung einer maximalen Zeit für den Verbindungsaufbau gibt es einen funktionierenden Workaround, durch den die Zeit deutlich verringert werden kann. Das Fazit ist, dass Saros erfolgreich in bestehende Entwicklungsprozesse eingeführt bzw. integriert werden kann.

# 7 Schluss

In diesem Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst und bewertet (Abschnitt 7.1) werden. Der Ausblick (Abschnitt 7.2) zeigt Aufgaben und Themen für mögliche weiterführende Untersuchungen und Arbeiten auf, deren Bearbeitung im Rahmen dieser Bachelorarbeit nicht möglich war.

## 7.1 Zusammenfassung und Fazit

Im Rahmen dieser Arbeit wurde untersucht, wie die Technik der verteilten Paarprogrammierung in die vorhandenen Entwicklungsprozesse von Softwarefirmen eingeführt werden kann. Hierzu wurde zunächst die vorhandene Infrastruktur der VEIS katalogisiert. Anschließend wurden basierend auf allgemeinen sowie konkreten Zielen Anforderungen an eine erfolgreiche Einführung und Integration sowie das weitere Vorgehen hergeleitet. Vor Beginn der Realisierung wurde die Reihenfolge der durchzuführenden Schritte festgelegt (Kapitel 3).

Darauf aufbauend wurde in Kapitel 4 die Systeme identifiziert, die von der Saros-Integration unmittelbar betroffen sein würden. Für die technische Integration wurde dann die Installations- und Testplanung durchgeführt. Im Anschluss wurde die Umsetzung der Planung sowie die dabei entstandenen Problemfälle dokumentiert. Am Ende stand eine funktionsfähige Eclipse-Saros-Installation zur Verfügung.

Ein Konzept, welche Aspekte auf welchem Weg den Softwareentwicklern gezeigt und erklärt werden sollten wurde in Kapitel 5 ausgearbeitet. Eine Bildschirmpräsentation vermittelt die Theorie zur Paarprogrammierung, während für die Einführung in Saros eine Live-Vorführung empfohlen wurde. Das Kapitel ist sowohl mit Hintergrundinformationen zu Paarprogrammierung als auch mit Tipps für den allgemeinen Aufbau der Präsentation angereichert.

Anhand der zu Beginn aufgestellten Anforderungsliste konnte in Kapitel 6 der Erfolg der wichtigsten Anforderungen innerhalb der Realisierung festgestellt werden. Das Überschreiten eines Zeitlimits ließ sich durch eine Änderung des Workflows umgehen. Ein Schulungskonzept konnte im Rahmen dieser Arbeit nicht mehr erstellt werden.

Insgesamt kann festgestellt werden, dass sich verteilte Paarprogrammierung mit Saros gut in Unternehmen integrieren lässt, sofern

1. alle Softwareentwickler einen XMPP-Account besitzen bzw. erstellen können, der von den anderen Entwicklern erreicht werden kann,
2. XMPP-Verbindungen nicht im Firmennetzwerk blockiert werden, und
3. für Programmierarbeiten eine Eclipse-Version verwendet wird, die kompatibel zu einem aktuellen Saros-Release ist.

## 7.2 Ausblick

Aufbauend auf dieser Arbeit könnte das Konzept zur Einführung ins Team ebenso wie die technische Integration in der Praxis durchgeführt und die Beobachtungen sowie das Ergebnis dokumentiert werden. Ebenso müssen die Anwender im Umgang mit der neuen Arbeitstechnik geschult werden. Denkbare Wege sind klassische Mitarbeiterschulungen, bei denen allen Softwareentwicklern die Grundlagen für einen erfolgreichen Einstieg in die Paarprogrammierung vermittelt werden. Interessant ist die Möglichkeit, dieses Wissen lediglich einigen wenigen Leuten zu vermitteln und dann darauf zu bauen, dass diese ihr Wissen durch Anwendung der Praktik Paarprogrammierung weitergeben. Welche dieser Methoden funktioniert besser, und kann mit Hilfe von Paarprogrammierung Paarprogrammierung für dessen Einführung angewendet werden?

Weiterhin stellt sich die Frage, ob das Werkzeug zu einem späteren Zeitpunkt tatsächlich aktiv von den Softwareentwicklern genutzt wird. Nach mehreren Einsätzen werden bei den Benutzern möglicherweise weitere Feature-Wünsche geweckt, die Saros in einer späteren Version anbieten könnte. Ebenso spannend ist die Frage, ob die Einführung und Integration von Saros in das Softwareentwicklungsteam zu spürbaren Veränderungen beim Softwareentwicklungsprozess geführt hat und wie diese genau aussehen. Sind die in den bisher veröffentlichten Studien genannten Vorteile aus 5.3.2 tatsächlich eingetreten? Wie hat Paarprogrammierung mit Saros die tatsächliche Qualität der entwickelten Produkte beeinflusst? Wie hat sich das Qualitäts- und Wissensniveau der Softwareentwickler zueinander verändert? Und hat der Einsatz verteilter Paarprogrammierung zu einem verbesserten standortübergreifenden Teamgefühl geführt?

Ein weiterer Ansatzpunkt für weitere Forschungen ist die Auswahl der Anforderungen. Im Kontext dieser Arbeit wurden an die Einführung und Integration größtenteils weiche funktionale Anforderungen gestellt, bei denen subjektive Erfahrungen für eine erfolgreiche Abnahme ausreichend waren. Hier gibt es noch die Möglichkeit, nicht-funktionale Anforderungen wie die maximal erlaubte Zeit vom Anstoßen einer Aktion bis zu deren Anzeige beim Navigator in die Liste der Anforderungen aufzunehmen.

# A Hintergrundinformationen zum Corporate Eclipse

In diesem Anhang werden weitere Informationen zum Corporate Eclipse geliefert, auf die innerhalb der Ausarbeitung verzichtet wurden.

## A.1 Ursprung

Die gemeinsame Eclipse-Installation entstand als Lösungsstrategie aus einer Reihe von Problemen, mit denen die Entwickler aufgrund der Nutzung verschiedener Versionen und Ausprägungen von Eclipse zu kämpfen hatten. Das Entwicklerteam war in den letzten Jahren immer weiter gewachsen, teilweise sogar über Ländergrenzen hinweg. Damit sowohl neue Entwickler als auch Supporter den Quellcode exakt so zu sehen bekommen wie die Entwickler kam der Gedanke auf, eine einheitliche Programmierplattform zu schaffen, die die wichtigsten Bedürfnisse der Entwickler abdeckt.

Erste Versuche erforderten, dass die Entwickler sich ein eigenes Eclipse auf dem Rechner installierten und dann anschließend viele schriftlich fixierte Anweisungen ausführten. Dieser Weg funktionierte grundsätzlich auch, war aber aufgrund der Masse vorzunehmender Konfigurationseinstellungen fehleranfällig. Daher wurde versucht, die Anzahl manuell durchzuführender Schritte bei der Einrichtung des Systems auf das notwendigste Minimum zu reduzieren. Das war die Geburtsstunde des ersten Corporate Eclipse.

## A.2 Richtlinien

Das Corporate Eclipse enthält alle nötigen Einstellungen und Erweiterungen, die den firmeninternen Richtlinien entsprechen. Diese Richtlinien entstammen zwei Quellen: Zum einen bestehen sie aus im Laufe der Zeit getroffenen Vereinbarungen der Entwickler, um das gemeinsame Arbeiten an Projekten zu erleichtern. Die wichtigsten Einstellungen hierbei sind Regeln für automatische Codeformatierung und Codebereinigungen. Ohne diese Regeln würden Dateien mit Quellcode auf verschiedenen Entwicklungssystemen unter Umständen

unterschiedlich formatiert werden. Die Folge wäre, dass bei Verwendung des Versionskontrollsystems mehr inhaltliche Änderungen einer Datei angezeigt werden würden, als tatsächlich durch die Arbeit des Entwicklers betroffen wären. Zum anderen sind durch konkrete Anforderungen der Qualitätssicherungsabteilung an den produzierten Quellcode Erweiterungen mit eingeflossen, welche die Entwickler dabei unterstützen, den Auflagen der Qualitätssicherung einfacher zu genügen. Die wichtigsten einheitlich definierten Einstellungen sind die Regeln für Code-Formatierung und automatische Suche nach potenziellen Programmierfehlern, Anforderungen an die Dokumentation öffentlicher Schnittstellen sowie Pfade zu externen Programmen, die aus Eclipse heraus aufgerufen werden können.

## A.3 Installation

Im Zuge der Erstinstallation führt ein Entwickler einen Checkout der gesamten Eclipse-Installation auf seine lokale Workstation durch. Anschließend kann Eclipse direkt gestartet werden, allerdings sind dann noch nicht alle gewünschten teamübergreifenden Einstellungen gesetzt. Die Ursache liegt in der Art und Weise begründet, an welchen Stellen im Dateisystem Eclipse benutzerdefinierte Einstellungen sichert. Einige Einstellungen werden in einem Konfigurationsverzeichnis unterhalb des Eclipse-Installationsverzeichnisses gespeichert. Beispiele dafür sind der zuletzt vom Benutzer verwendete Workspace sowie globale Eigenschaften installierter Plugins; das Subclipse-Plugin merkt sich, ob anonymisierte Daten an die Entwickler des Plugins geschickt werden dürfen. Nahezu alle vom Eclipse-Benutzer änderbaren Einstellungen lassen sich allerdings nur innerhalb von Workspaces als Standardwerte sowie innerhalb einzelner Eclipse-Projekte hinterlegen. Daher muss jeder Benutzer nach Anlage eines neuen Workspaces einige Konfigurationsdateien importieren, damit die Teamrichtlinien auch auf zukünftige Projekte angewendet werden.

## A.4 Aktualisierung

Die normale Eclipse-Installation ist so eingerichtet, dass eigene Erweiterungen in ein separates Installationsverzeichnis installiert werden. Zusätzlich ist die globale Konfiguration schreibgeschützt, d.h. dort hinterlegte Plugins können nicht entfernt werden. Um diese Aktionen durchführen zu können, müssen in der Datei eclipse.ini die folgenden Zeilen auskommentiert werden:

```
-Dosgi.configuration.cascaded=true
-Dosgi.configuration.area=user-configuration
-Dosgi.sharedConfiguration.area=configuration
-Dosgi.sharedConfiguration.area.readOnly=true
```

Beim nächsten Start von Eclipse wird die benutzerspezifische Konfiguration ignoriert und stattdessen direkt die globale Konfiguration beschreibbar verwendet. Alle nun durchgeführten Änderungen an der globalen Eclipse-Konfiguration führen dazu, dass bei einem Übertragen ins Subversion die Änderungen für alle anderen Entwickler sichbar werden, sobald diese ihre lokale Installation mit dem Stand im Subversion abgleichen.

# Literaturverzeichnis


[RFC3920 2004] : *Request for Comments: 3920 - Extensible Messaging and Presence Protocol (XMPP): Core*. 10 2004. – URL http://xmpp.org/rfcs/rfc3920.html

[RFC3921 2004] : *Request for Comments: 3921 - Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence*. 10 2004. – URL http://xmpp.org/rfcs/rfc3921.html

[agile42 2009] AGILE42: Agile testing agile42 (Veranst.), Presented at Agile testing coaching, 2009, S. 5

[Beck 1999] BECK, Kent: *Extreme programming explained : embrace change*. First edition. Addison-Wesley, 1999

[Beck 2005] BECK, Kent: *Extreme programming explained : embrace change*. Second edition. Addison-Wesley, 2005

[Beck u. a. 2001] BECK, Kent ; BEEDLE, Mike ; BENNEKUM, Arie v. ; COCKBURN, Alistair ; CUNNINGHAM, Ward ; FOWLER, Martin ; GRENNING, James ; HIGHSMITH, Jim ; HUNT, Andrew ; JEFFRIES, Ron ; KERN, Jon ; MARICK, Brian ; MARTIN, Robert C. ; MELLOR, Steve ; SCHWABER, Ken ; SUTHERLAND, Jeff ; THOMAS, Dave: *Manifest für Agile Softwareentwicklung*. 2001. – URL http://agilemanifesto.org/iso/de/

[BMI 2009] BMI: *Das V-Modell XT*. 11 2009. – URL http://www.cio.bund.de/DE/IT-Methoden/V-Modell_XT/v-modell_xt_node.html

[Cernosek 2005] CERNOSEK, Gary: *A brief history of Eclipse*. November 2005. – URL http://www.ibm.com/developerworks/rational/library/nov05/cernosek/index.html

[Cockburn und Williams 2001] COCKBURN, Alistair ; WILLIAMS, Laurie: *The costs and benefits of pair programming*. S. 223–243. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2001. – URL http://portal.acm.org/citation.cfm?id=377517.377531. – ISBN 0-201-71040-4

[Cohen 2006] COHEN, Jason a.: *Best Kept Secrets of Peer Code Review*. SmartBeer Software, 2006

[Coldewey 2011] COLDEWEY, Jens: Vorhersage des Unvorhersehbaren: Langfristige Planung in agilen Vorhaben. In: *OBJEKTspektrum* it-agile GmbH (Veranst.), URL http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2011/01/coldewey_OS_01_11.pdf, 1 2011, S. 22–27

[Djemili 2006] DJEMILI, Riad: *Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammierung*, FU Berlin, Diplomarbeit, 08 2006

[Eickmann 2009]

[Espenak 2009] ESPENAK, Fred: *Eclipses and the Saros*. 08 2009. – URL http://eclipse.gsfc.nasa.gov/SEsaros/SEsaros.html

[Falconphysic 2010] FALCONPHYSIC: *Debugging with Serial Communication*. Podcast. 4 2010. – URL http://blip.tv/falconphysics-podcast/debugging-with-serial-communication-3474005. – Author name unknown, Falconphysic is a nickname

[Fowler 1999] FOWLER, Martin: *Refactoring: Improving the design of existing code*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1999. – ISBN 0-201-48567-2

[Holweg 2006] HOLWEG, Matthias: The genealogy of lean production. In: *Journal of Operations Management* 25 (2006), 05, S. 420–437. – URL http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6VB7-4JX379W-1-5&_cdi=5919&_user=572227&_orig=search&_coverDate=03%2F31%2F2007&_sk=999749997&view=c&wchp=dGLbVtz-zSkzV&md5=da1dbd36e8c1a651aa706e0c37b45771&ie=/sdarticle.pdf

[Poppendieck und Poppendieck 2003] POPPENDIECK, Mary ; POPPENDIECK, Tom: *Lean Software Development: An Agile Toolkit*. Addison-Wesley, 05 2003. – 240 S

[Ryan 2010] RYAN, Terrence ; CARTER, Jacquelyn (Hrsg.): *Driving Technical Change*. Pragmatic Bookshelf, 11 2010. – 200 S. – ISBN 978-1-934356-60-9

[Schümmer und Schümmer 2000] SCHÜMMER, Till ; SCHÜMMER, Jan: Support for Distributed Teams in eXtreme Programming. In: *Proceedings of eXtreme Programming and Flexible Processes Software Engineering - XP2000*, Addison Wesley, 2000, S. 355–377

[ScrumAlliance 2011] SCRUMALLIANCE: *What is Scrum?* 07 2011. – URL http://www.scrumalliance.org/pages/what_is_scrum. – Zugriffsdatum: 27.07.2011

[Sommerville 2007] SOMMERVILLE, Ian: *Software Engineering*. 8. Harlow, England : Addison-Wesley, 2007. – ISBN 978-0-13-703515-1

[Vattenfall 2011] VATTENFALL: *Europäischer Energieversorger mit Tradition - Meilensteine der Vattenfall Europe AG*. 07 2011. – URL `http://www.vattenfall.de/de/historie.htm`. – Zugriffsdatum: 01.08.2011

[Wikipedia 2011a] WIKIPEDIA: *Bus factor*. 2011. – URL `http://en.wikipedia.org/wiki/Bus_factor`. – Zugriffsdatum: 27.07.2011

[Wikipedia 2011b] WIKIPEDIA: *Waterfall model*. 06 2011. – URL `http://en.wikipedia.org/wiki/Waterfall_model`. – Zugriffsdatum: 27.07.2011

[Williams 2010] WILLIAMS, Laurie: Pair Programming. In: ORAM, Andy (Hrsg.) ; WILSON, Greg (Hrsg.): *Making Software - What Really Works, and Why We Believe It*. O'Reilly, 2010, S. 311 – 328

[Womack und Jones 1997] WOMACK, James P. ; JONES, Daniel T.: *Auf dem Weg zum perfekten Unternehmen (Lean Thinking)*. Campus Verlag, 1997

# Abbildungsverzeichnis

*Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.*

Hamburg, 3. August 2011 Marcus Kraßmann