Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Benjamin Kirstgen

Automatisches Prozessmonitoring bei der Softwareentwicklung zur Unterstützung der Kollaboration

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Benjamin Kirstgen

Automatisches Prozessmonitoring bei der Softwareentwicklung zur Unterstützung der Kollaboration

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Professor : Prof. Dr. rer. nat. Kai von Luck
Zweitgutachter : Prof. Dr. rer. nat. Gunter Klemke

Abgegeben am 30. Dezember 2011

Benjamin Kirstgen

Titel der Ausarbeitung

Automatisches Prozessmonitoring bei der Softwareentwicklung zur Unterstützung der Kollaboration

Stichworte

Teamarbeit, Softwareentwicklung, CSCW, Wissensmanagement, Enterprise 2.0, kontext-sensitive Anwendungen, Awareness, Diskurs

Kurzzusammenfassung

Diese Masterarbeit beschäftigt sich mit Möglichkeiten der Unterstützung eines Softwareentwicklers bei der Arbeit in Teams und Projekten. Dabei liegt das Hauptaugenmerk auf den Chancen und Risiken des sich entwicklenden Forschungsgebiets *Enterprise 2.0*. Dieses hebt im besonderen Maße den Nutzen von Kollaborationssystemen für den Einzelanwender hervor. In dieser Arbeit wird ein Werkzeug zur Unterstützung des Entwicklers bei der Teamarbeit entworfen und ein Prototyp realisiert, das zum Einen den Aufbau des aktuellen Arbeitskontexts unterstützt und zum Anderen die Awareness über die Aktivitäten der Kollegen fördert.

Benjamin Kirstgen

Title of the paper

Automated process monitoring in software development to support collaboration

Keywords

teamwork, software development, CSCW, knowledge management, Enterprise 2.0, context aware applications, awareness, discourse

Abstract

This master thesis examines various options of supporting individual software developers in collaborative projects. The focus lies on opportunities and challenges of the evolving research field of *Enterprise 2.0*, which is characterized by emphasizing the benefits of collaboration systems for the single user. Within the scope of this thesis, the author has designed a tool to assist a developer in teamwork and implemented a corresponding prototype to simplify setting up the context of a current task as well as increase awareness of colleagues' activities.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Erstellung dieser Arbeit und während meines Studiums unterstützt haben.

Hierzu zählt das gesamte Team des Projekt Home Office 2.0, in dessen Rahmen das hier behandelte Discourse-Tool entstanden ist und auf dem die vorliegende Ausarbeitung fußt. Im besonderen Maße gilt mein Dank dabei Karsten Panier und Matthias Holsten. Die intensive und lehrreiche Zusammenarbeit hat mich während des gesamten Masterstudiums stets bereichert.

Für die hervorragende Betreuung beim Verfassen dieser Arbeit spreche ich meinem Erstprüfer Prof. Dr. rer. nat. Kai von Luck vielen herzlichen Dank aus. Der kontinuierliche fachliche Ausstausch und die anregenden Denkanstöße waren mir während des gesamten Prozesses sehr behilflich. Ich danke zudem Prof. Dr. rer. nat. Gunter Klemke, der sich bereit erklärt hat, die Zweitkorrektur dieser Arbeit zu übernehmen.

Auch aus meinem persönlichen Umfeld habe ich Unterstüzung erfahren, für die ich besonders Rebekka Kirstgen und Hans-Walter Schmitt meinen Dank aussprechen möchte. Nicht zuletzt bin ich meinen Eltern sowie meiner Freundin Dorothee Schmitt überaus dankbar, die mich nicht nur bei dieser Arbeit, sondern während meines gesamten Studiums bedingungslos und intensiv unterstützt haben.

Inhaltsverzeichnis

1. Einleitung **10**
- 1.1. Motivation . . . 10
- 1.2. Zielsetzung . . . 11
- 1.3. Aufbau der Arbeit . . . 12

2. Grundlagen **14**
- 2.1. Computer-Supported Cooperative Work . . . 14
  - 2.1.1. CSCW-Forschungsgebiet . . . 14
  - 2.1.2. Groupware . . . 16
  - 2.1.3. Awareness-Unterstützung . . . 17
- 2.2. Wissensmanagement . . . 19
- 2.3. Kontext . . . 21
- 2.4. Agile Entwicklungsmethoden . . . 22

3. Analyse **26**
- 3.1. Aktuelle Herausforderungen bei der Softwareentwicklung . . . 26
  - 3.1.1. Distributed Development . . . 27
  - 3.1.2. Cognitive Overload . . . 29
  - 3.1.3. Schatten-IT . . . 31
- 3.2. Enterprise 2.0 . . . 34
- 3.3. Szenario . . . 36
  - 3.3.1. Old Software Never Dies . . . 36
  - 3.3.2. Awareness-Unterstützung . . . 38
- 3.4. Related Work . . . 40
  - 3.4.1. ACTIVE Projekt . . . 41
  - 3.4.2. Mylyn . . . 42
  - 3.4.3. Mining Software Repositories . . . 43
- 3.5. Fazit Analyse . . . 45

4. Design & Realisierung **48**
4.1. Vision . . . 48
4.2. Projekt *Home Office 2.0* . . . 49
4.2.1. Architektur . . . 50
4.2.2. Datenmodell . . . 52
4.2.3. Einordnung Discourse in LuPanKu-Projekt . . . 54
4.3. Architektur Discourse . . . 54
4.3.1. Service-orientierte Architekturen . . . 57
4.4. Framework . . . 58
4.4.1. OSGi - Open Services Gateway initiative . . . 59
4.4.2. Equinox . . . 62
4.5. Kommunikation . . . 62
4.5.1. Distributed OSGi . . . 63
4.5.2. Eclipse Communication Framework . . . 63
4.5.3. LuPanKu Remote Services . . . 64
4.6. Integration in Entwicklungsumgebung . . . 66
4.6.1. Eclipse-Entwicklungsumgebung . . . 66
4.6.2. Eclipse-PlugIn . . . 69
4.6.3. Discourse-PlugIn . . . 71
4.7. LuPanKu-Analyzer . . . 79
4.7.1. Discourse-Analyzer . . . 79
4.8. Fazit & Evaluation . . . 84

5. Fazit und Ausblick **88**
5.1. Zusammenfassung & Fazit . . . 88
5.2. Ausblick . . . 91

Literaturverzeichnis **93**

A. Klassendiagramme **102**
A.1. Klassendiagramm Discourse Eclipse-PlugIn . . . 102
A.2. Klassendiagramm Discourse-Analyzer . . . 103

Abkürzungsverzeichnis

ACM Association for Computing Machinery

CSCW Computer Supported Cooperative Work bzw. Computer Supported Collaborative Work

ECF Eclipse Communication Framework

GUI Graphical User Interface

HAW Hochschule für angewandte Wissenschaften

ICSE International Conference on Software Engineering

IDE Integrated Development Environment (dt.: integrierte Entwicklungsumgebung)

IT Informationstechnik

JVM Java Virtual Machine

MSR Mining Software Repositories

OSGi Open Services Gateway initiative

SOA Service-orientierte Architektur

SWT Standart Widget Toolkit

Abbildungsverzeichnis

2.1. CSCW-Bezugsrahmen 16
2.2. Awareness-Informationsverarbeitung 18
2.3. Ebenen der agilen Softwareentwicklung 23
2.4. Scrum Sprint 24

3.1. Cognitive Overload 30
3.2. Szenario Old Software Never Dies 36
3.3. Szenario Awareness Unterstützung 39

4.1. Schematische Darstellung der Systemarchitektur des LuPanKu-Systems 51
4.2. UML-Klassendiagramm des verwendeten Datenmodells 53
4.3. Einordnung des Teilprojekts Discourse in das LuPanKu-Projekt 55
4.4. Komponenten Discourse 56
4.5. OSGi Framework 60
4.6. Eclipse Workbench Window Struktur 68
4.7. Beispiel einer Eclipse-View 71
4.8. Komponenten Discourse-PlugIn 71
4.9. Titelleiste & Werkzeugleiste des DiscourseView 72
4.10. Sequenz Diagramm Kommunikation 73
4.11. Anzeige Fortschritt Discourse-Call in der Statusleiste 75
4.12. Discourse View in Eclipse-IDE mit Discourse-Informationen 76
4.13. vereinfachtes Modell der Klassen des Discourse-PlugIns 77
4.14. Discourse-Analyzer Komponente auf dem LuPanKu-Server 80
4.15. vereinfachtes Modell der Klassen des Discourse-Analyzers 83

A.1. Klassendiagramm Paket org.aysada.lupanku.discourse.ui 102
A.2. Klassendiagramm Paket org.aysada.lupanku.discourseanalyzer 103

Tabellenverzeichnis

3.1. Unterschiede Schatten-IT und Unternehmens-IT 33
3.2. Forschungsarbeiten im Bereich von Mining Software Repositories (MSR) 44

4.1. Awareness States 74
4.2. Discourse View im Hintergrund und minimiert 75

Quellcodeverzeichnis

4.1. IClientListener Schnittstelle 65
4.2. ICallBackService Schnittstelle 65
4.3. IAnalyzer Schnittstelle 79

1. Einleitung

1.1. Motivation

Durch die heutige globalisierte und schnelllebige Wissensgesellschaft befinden sich die Arbeitsformen und die damit verbundenen Anforderungen an einen Wissensarbeiter in einem stetigen Wandel. Um die Wissensarbeiter effizienter zu gestalten, ist sowohl der Mitarbeiter selbst als auch das Unternehmen gefordert, die vorherrschenden Arbeitsmethoden kontinuierlich anzupassen. In diesem Zusammenhang werden in Unternehmen vermehrt Neue Medien, so genannte *Web 2.0-Anwendungen*, eingesetzt, die in der Internet-Community bereits länger Anwendung finden. Insbesondere die Verwendung von *Social Software* im Unternehmenskontext ist in den letzten Jahren stark angestiegen. Welche Chancen und Risken damit verbunden sind und wie diese genutzt bzw. behandelt werden können, wird daher in dem noch jungen Forschungsgebiet *Enterprise 2.0* untersucht.

In diesem Rahmen wurde an der Hochschule für angewandte Wissenschaften (HAW) Hamburg das Projekt *Home Office 2.0* initiiert. Ziel des Projektes ist die Förderung der **Computer-Supported Cooperative Work** (CSCW) und die Unterstützung des **Wissensmanagements** in Unternehmen. Der Fokus liegt dabei speziell auf der Arbeit von Softwareentwicklern in (verteilten) Teams.

In Unternehmen sind häufig Inkompatibilitäten zwischen der Arbeitsweise der Entwickler, der Unternehmensphilosophie und den durch das Unternehmen zur Verfügung gestellten Werkzeugen (Toolboxen, z.B. Wikis, Versionsverwaltungen, Aufgabenverwaltungen) erkennbar. Die Bereitstellung von Toolboxen kann kein bestimmtes Vorgehen des Entwicklers erzwingen, weshalb die Effizienz solcher Werkzeuge meist stark von der Akzeptanz der Mitarbeiter abhängt. Dies stellt die Entwicklung von kollaborationsun-

terstützenden[1] Systemen vor eine große Herausforderung, da neue Entwicklungsmethoden häufig eine enge Zusammenarbeit und somit die Verwendung solcher Werkzeuge fordern. Aus diesem Grund ist die Vision des Projektes *Home Office 2.0* ein System zu entwerfen, welches so in bestehende Arbeitsstrukturen eines Unternehmens integriert werden kann, dass die Verwendung vorhandener Tools gefördert wird, ohne mit den individuellen Arbeitsweisen der Entwickler zu kollidieren. Darüber hinaus soll es die Wissensarbeit unterstützen, indem es Wissen, das mit Hilfe der Kollaborationswerkzeuge erstellt wurde, einen Mehrwert durch zusätzliche Analysen abgewinnt.

1.2. Zielsetzung

Die vorliegende Arbeit gliedert sich, als Teilprojekt *Discourse* in das Projekt *Home Office 2.0* ein, das sich der Unterstützung von Softwareentwicklern bei der Behandlung einer konkreten Aufgabe widmet. Die Idee dabei ist, dem Entwickler einen Diskurs zu der zu bearbeitenden Ressource[2] zur Verfügung zu stellen. Unter einem Diskurs wird in diesem Zusammenhang der bisherige Entwicklungsprozess einer Ressource verstanden sowie die Diskussionen, die in diesem Rahmen zwischen Mitarbeitern stattgefunden haben. Demnach dokumentiert ein solcher Diskurs nicht nur konkrete Arbeitsaufgaben und -ergebnisse, sondern auch den Prozess der Wissensgenerierung, der zu diesen geführt hat. Ziel ist es, dem Entwickler zu helfen, den Kontext eines aktuellen Arbeitsauftrags an einer Datei aufzubauen, sodass er schneller mit dessen effektiver Bearbeitung beginnen kann.

In Entwicklerteams mit so genannten weak oder collectiv Code ownership (Fowler, 2006) sind jedoch nicht nur vergangene, sondern auch aktuelle Entwicklungen einer Ressource von Interesse. Denn nur wenn ein Bewusstsein über relevante gegenwärtige Aktivitäten anderer Mitarbeiter besteht, können mögliche Überschneidungen wahrgenommen, Synergien genutzt oder Konflikte vermieden werden. Daher kann die Unterstützung der Teamarbeit durch Förderung eines Bewusstseins (*Awareness*) über Teamkollegen und deren Tätigkeiten als ein weiteres Ziel für diese Arbeit identifiziert werden.

[1] Im Kontext dieser Arbeit bezieht sich der Begriff Kollaboration auf den wirtschaftswissenschaftlichen Sinn der engen Kooperation und nicht auf die historischen Bedeutung der Zusammenarbeit mit dem Feind

[2] In dieser Arbeit wird eine Ressource häufig im Sinne der Informatik als Synonym für eine Datei verwendet

Im Discourse-Projekt sollen sowohl das Design als auch der erste Prototyp entwickelt werden, das den genannten Zielen begegnen kann. Anhand umfangreicher Analysen relevanter Forschungsstände zielt die vorliegende Arbeit auf eine wissenschaftliche Fundierung dieses Entwurfs und dient der Präsentation sowie der Diskussion der Ergebnisse.

1.3. Aufbau der Arbeit

Nachdem die Einleitung Motivation und Ziele der Arbeit vorgestellt und so an das Thema herangeführt hat, gilt es, die hierfür relevanten Grundlagen zu formulieren. Kapitel 2 erläutert daher den theoretischen Bezugsrahmen und die für das Verständnis der Arbeit wesentlichen Begrifflichkeiten. Dazu gehört:

- das Forschungsgebiet der *Computer-Supported Cooperative Work* (2.1), mit dem sich das *Home Office 2.0*-Projekt beschäftigt.
- der durch den Wandel zur Wissensgesellschaft immer bedeutsamere Bereich des *Wissensmanagements* (2.2).
- der mit unterschiedlichen Bedeutungen versehene und daher zu definierende Begriff *Kontext* (2.3).
- die Thematik der *agilen Entwicklungsmethoden*, die in der aktuellen Softwareentwicklung zunehmend an Bedeutung gewinnt und zusätzliche Anforderungen an die Arbeit im Team stellt (2.4).

Anschließend wird in Kapitel 3 der Sachverhalt zur Entwicklung des Discourse-Tools analysiert. Dies wird aus drei Perspektiven vorgenommen:

Die **theoretische** Sicht (3.1 & 3.2) betrachtet gegenwärtige Herausforderungen in der Softwareentwicklung, die für diese Arbeit relevant sind, und das Forschungsgebiet *Enterprise 2.0*, welches sich mit den beschriebenen Problemen auseinandersetzt. Aus dem **fiktiv-praktischen** Blickwinkel (3.3) werden zwei Szenarien beschrieben, die die Funktionalität des Discourse-Prototypen erläutern und eingrenzen. Die dritte Perspektive beleuchtet aus einer **real-praktischen** Sicht (3.4) verschiedene aktuelle Forschungsgebiete und -arbeiten, die sich mit einer ähnlichen Thematik auseinandersetzen. Zum

Abschluss der Analyse werden im Fazit (3.5) die durch die Analyse deutlich gewordenen Anforderungen an das Discourse-Tool formuliert.

Auf Grundlage dieser Erkenntnisse werden in Kapitel 4 zum einen das Design für das Discourse-Tool erarbeitet und zum anderen die Schritte zur Realisierung des Prototypen beschrieben. Dazu erfolgt zunächst die Darstellung des visionären Designs für das Tool (4.1). Um den Rahmen zu verstehen, in dem dieses entwickelt wird, skizziert Kapitel 4.2 das Projekt *Home Office 2.0* hinsichtlich dessen Ziele, Architektur und Datenmodells. In diesen Kontext gilt es anschließend das Discourse-Projekt einzuordnen (4.2.3). Kapitel 4.3 identifiziert die Komponenten des Discourse-Tools und erarbeitet dessen Architektur. Die folgenden Abschnitte (4.4 bis 4.7) befassen sich detailliert mit diesen einzelnen Komponenten und deren Realisierung. Dabei wird so vorgegangen, dass die Komponenten zunächst allgemeingültig, ohne konkrete Realisierungsansätze, betrachtet werden und dann zunehmend detaillierter auf die eingesetzten Techniken und die Implementierung eingegangen wird, also von grob- zu feingranular. Dadurch können in weiterführenden Arbeiten im *Home Office 2.0*-Projekt, in denen andere Implementierungsansätze verwendet werden, dennoch die allgemeingültigen Betrachtungen mit einbezogen werden. Im Fazit des Design-Kapitels (4.8) werden die Architektur und der entwickelte Prototyp unter verschiedenen Gesichtspunkten evaluiert.

Die Arbeit schließt in Kapitel 5 mit einer Zusammenfassung der Kapitel und den daraus hervorgegangenen Ergebnissen ab. Als letztes wird in Kapitel 5.2 ein spekulativer Ausblick über die weitere Entwicklung der Thematik der vorliegenden Arbeit aus Sicht des Autors formuliert.

Um die Lesbarkeit zu verbessern, wurde in dieser Arbeit von einer Geschlechtertrennung abgesehen. Im Folgenden sind bei allen geschlechtsbezogenen Begriffen, wie Mitarbeiter, Benutzer oder Entwickler immer sowohl weibliche als auch männliche gemeint.

2. Grundlagen

Dieses Kapitel beschreibt alle für das Verständnis der Arbeit wichtigen Grundlagen und Begriffe und führt in die Thematik ein. Dazu wird das Forschungsgebiet Computer Supported Cooperative Work bzw. Computer Supported Collaborative Work (**CSCW**, 2.1) beleuchtet, das sich mit der Kollaboration in Teams mit Hilfe von Computertechnologien beschäftigt. Da für die Arbeit speziell die Förderung des **Wissensmanagements** von Interesse ist, wird dieses in Kapitel 2.2 betrachtet. Der Begriff **Kontext** hat in verschiedenen Forschungsdisziplinen unterschiedliche Bedeutungen; daher wird er in Abschnitt 2.3 für die weitere Verwendung in dieser Arbeit definiert. Die **agilen Entwicklungsmethoden** (2.4) gewinnen in der Softwareentwicklung zunehmend an Bedeutung und stellen zusätzliche Anforderungen an die Teamarbeit im Rahmen von CSCW.

Neben der Schaffung eines grundlegenden Verständnisses für das Thema der Arbeit und der häufig verwendeten Begriffe zielt dieses Kapitel darauf ab, einige grundlegende Anforderungen für die Entwicklung des Discourse-Tools hervorzuheben.

2.1. Computer-Supported Cooperative Work

2.1.1. CSCW-Forschungsgebiet

Das interdisziplinäre Forschungsgebiet CSCW beschäftigt sich mit Möglichkeiten der Unterstützung kooperativen Arbeitens zwischen Personen mittels Computertechnologie (Ellis u. a., 1991, S.39).

> „...CSCW looks at how groups work and seeks to discover how technology (especially computers) can help them work.“ (Ellis u. a., 1991, S.39)

Das Ziel der CSCW-Forschung ist, das Zusammenarbeiten von Menschen durch den Einsatz von Informations- und Kommunikationstechniken effizienter und flexibler zu gestalten. Im Ausgangs- und Mittelpunkt der Forschung steht folglich immer der Mensch und wie er mit anderen kollaboriert. Aus diesem Grund sind auch andere Forschungsdisziplinen in diesem Zusammenhang relevant, wie etwa Organisations- und Führungslehre, Psychologie, Soziologie, Anthropologie oder die Wirtschaftsinformatik (Back und Seufert, 2000, S.6).

Bereits seit Anfang der 1980er Jahre wird im Bereich der CSCW geforscht, wobei der Begriff CSCW erstmals von Irene Greif und Paul Cahsman als Titel für einen Workshop verwendet wurde. Dieser diente dem Austausch zwischen Forschern verschiedener Disziplinen von Informationen und Arbeitsergebnissen zur computergestützten Teamarbeit (Koch, 2009). 1986 richtete die *Association for Computing Machinery* (ACM) in Austin (USA) die erste CSCW-Konferenz aus, die seither alle zwei Jahre stattfindet (Back und Seufert, 2000, S.6).

Die im Jahr 1995 in Deutschland gegründete Fachgruppe CSCW der Gesellschaft für Informatik befasst sich mit

> „[..] der Anwendung von Telekommunikationstechnologien zur Unterstützung von Gruppen- und Teamarbeit also der Mensch-Computer-Mensch-Interaktion."

Da bei der Teamarbeit jedoch die Mensch-Mensch-Interaktion im Vordergrund steht, gilt es die Unterstützung durch Computer möglichst transparent zu gestalten, um so ein Gefühl möglichst direkter Interaktion zwischen den Mitarbeitern zu erzeugen (Koeckritz, 2006, S.4).

Die CSCW-Forschung ist einem ständigen Wandel unterworfen. Neue Möglichkeiten der Informations- und Kommunikationstechnologie sowie das sich ändernde Arbeitsumfeld (z.B. Globalisierung) erfordern eine kontinuierliche Anpassung sowohl der Forschungsrichtung als auch der CSCW-Systeme an die gegebenen gesellschaftlichen und technischen Bedingungen. Deswegen sollte sich auch die Entwicklung von CSCW-Systemen in einem stetigen Wandel befinden. Es gilt nicht nur das Verständnis über die Teamarbeit zu berücksichtigen, sondern auch vorhandene Entwicklung zu bewerten und die entsprechenden Konsequenzen daraus zu ziehen (siehe Abbildung 2.1).

Unter CSCW-Applikationen werden Systeme verstanden, die ein computergestütztes kooperatives Arbeiten ermöglichen und dabei Problematiken, wie räumliche oder zeit-

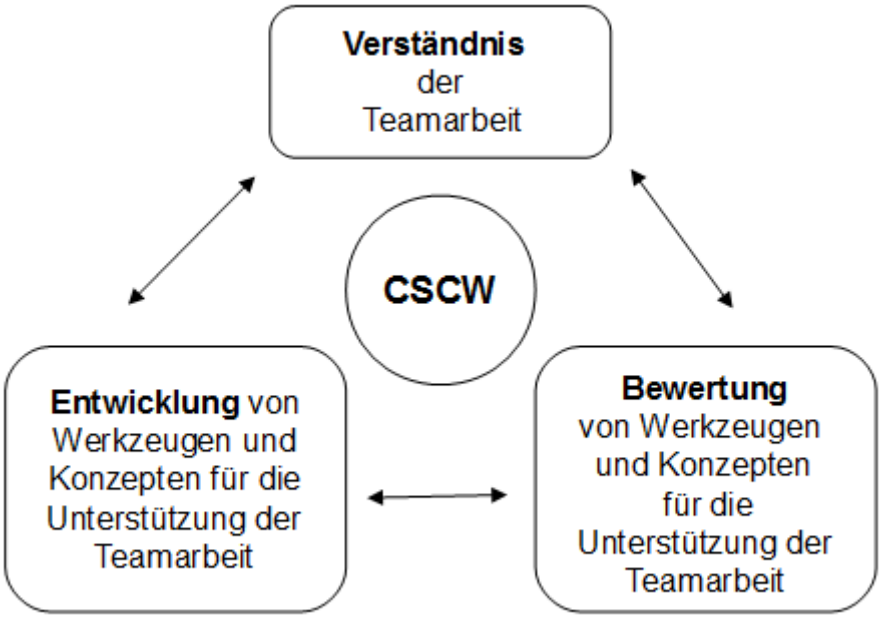


Abbildung 2.1.: CSCW-Bezugsrahmen (Quelle: Back und Seufert (2000, S.6))

liche Barrieren, überwinden. Diese Systeme werden auch als *Groupware* bezeichnet (Back und Seufert, 2000, S.7) (siehe Kapitel 2.1.2).

2.1.2. Groupware

Groupware kann als Anwendung der in der CSCW-Forschung erlangten Erkenntnisse verstanden werden. Dabei umfasst der Begriff nicht nur die entwickelte Software, sondern auch Hardware und Services zur Unterstützung der Kollaboration (Gross und Koch, 2007, S.6). In den Anfängen der CSCW-Forschung wurden zunächst einzelne Applikationen entwickelt, die jeweils Teilgebiete der Teamarbeit unterstützten. Doch im Laufe der Zeit wurden so genannte *CSCW-Suites* entworfen, die die Technologien mehrerer Einzelapplikationen in ein gemeinschaftliches Framework einbetten (Back und Seufert, 2000, S.5).

Im Gegensatz zur traditionellen Software zur Unterstützung mehrerer Benutzer versucht Groupware explizit, die Isolation der Benutzer voneinander zu reduzieren. Um kooperative Arbeit mit effizienter Kommunikation zu fördern, soll ein Gewahr- und Bewusstsein über die Kollegen und deren Aktivitäten erzeugt werden (Gross und Koch, 2007, S.7). Das Ziel, die so genannte *Awareness* zu schaffen, kennzeichnet diese zentrale Eigenschaft von Groupware, wie Lynch u. a. (1990) beschreibt:

> „Groupware is distinguished from normal software by the basic assumption it makes: groupware makes the user aware that he is part of a group, while most other software seeks to hide and protect users from each other.“ (Lynch u. a., 1990, S.160)

Die wohl größte Herausforderung bei der Entwicklung effizienter Groupware ist daher, die komplexe Situation der Abhängigkeiten zwischen sozialem und technischem System zu erfassen und zu berücksichtigen (Gross und Koch, 2007, S. 9).

2.1.3. Awareness-Unterstützung

Wie beschrieben ist die Schaffung einer Awareness über andere Nutzer der Groupware eine Charakteristik von CSCW-Systemen und wesentlich für die Gewährleistung einer effizienten Kommunikation in der Gruppenarbeit. Hierfür muss die Groupware den Benutzern Informationen übereinander zur Verfügung stellen. So kann beispielsweise einem Kollegen vor einem Kommunikationsversuch angezeigt werden, ob der Ansprechpartner anwesend und zu einem Gespräch bereit ist.

Die Herausforderung der Groupware ist dabei das Erfassen, Verarbeiten und Präsentieren der Awareness-Informationen sowie der gleichzeitige Schutz der Privatspäre. Es handelt sich um Informationen über den Benutzer und seine Arbeitsumgebung sowie über gemeinsam genutzte Ressourcen (Gross und Koch, 2007, S.59).

Der Aufbau der Awareness-unterstützenden Anwendungen besteht meist aus Sensoren und Indikatoren, die sich auf den Rechnern der Benutzer befinden, und einem zentralen Ereignis-Server (siehe Abbildung 2.2). Die Sensoren registrieren die Awareness-Informationen und senden sie an den Ereignis-Server. Dieser leitet sie an die Indikatoren weiter, die für die Darstellung der Informationen verantwortlich sind. Die meisten Awareness-Anwendungen sind bidirektional, das heißt, dass sie sowohl die Sensoren zur Sammlung von Informationen als auch die Indikatoren zur Darstellung empfangener

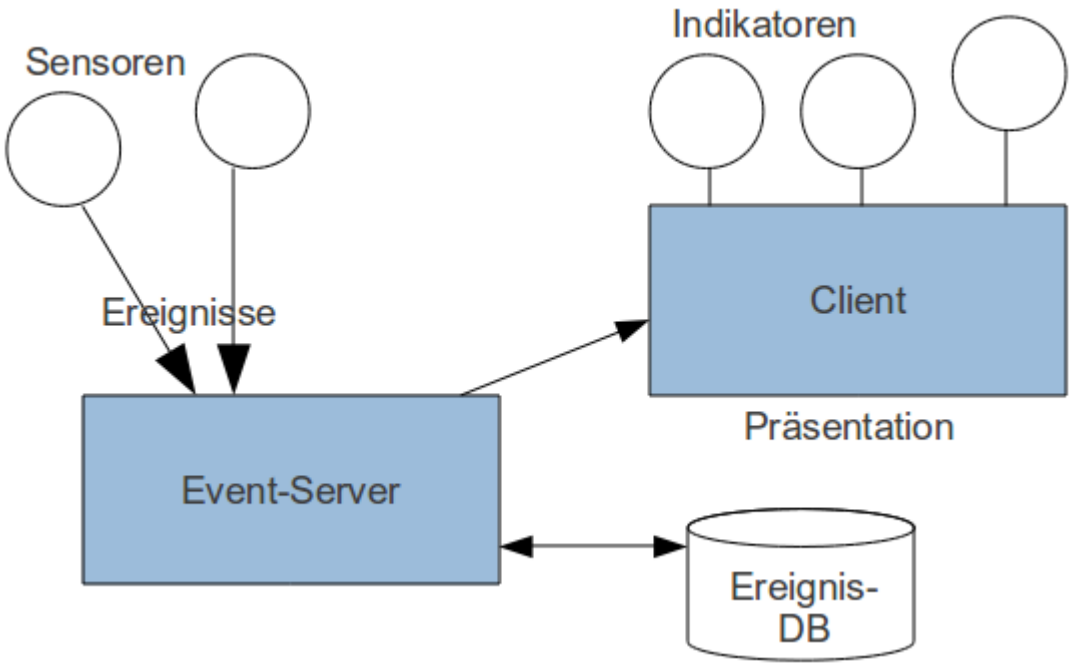


Abbildung 2.2.: Awareness-Informationsverarbeitung nach (Gross und Koch, 2007, S.60)

beinhalten. Dabei wird entweder nach dem Push- oder dem Pull-Prinzip[1] vorgegangen. Der Server ist zudem für das Speichern von Ereignissen und folglich für das Erstellen einer Historie verantwortlich. Eine Erweiterung dieses Grundmodells ist die Ergänzung einer Awareness-Pipeline zum Ereignis-Server, die die Awareness-Informationen nach Zugriffsrechten und Interessen der Benutzer filtert (Gross und Koch, 2007, S.61).

Ein populäres Beispiel für Awareness-unterstützende Anwendungen sind Instant-Messaging (IM)-Systeme. Diese stellen den Benutzern Informationen über die Präsenz und Verfügbarkeit anderer Anwender der Software zur Verfügung.

[1] Push: Informationen werden in regelmäßigen Abständen vom Server an die Klienten gesandt; Pull: die Klienten vordern die Informationen nach Bedarf an

Peripheral Awareness

Eine Herausforderung bei der Gestaltung von Awareness-Systemen ist die Darstellung der Awareness-Informationen, zu deren Aufnahme der Anwender nicht von seiner Primäraufgabe abgelenkt werden sollte. Gleichzeitig dürfen allerdings wichtige Meldungen in den Awareness-Informationen nicht übersehen werden (Cadiz u. a., 2001).

Das Thema *Peripheral Awarness* behandelt die unterbewusste Aufnahme von Informationen. Da der Aufwand zur Aufnahme von Awareness-Informationen so gering wie möglich gehalten werden soll (Gross und Koch, 2007, S. 62), versuchen einige Awareness-Anwendungen die Verwendung der unbewussten Wahrnehmung zu fördern (Gennaro, 2008). Dabei nimmt der Anwender die Awareness-Informationen unterbewusst auf, ohne dass dabei seine aktuelle Aufgabe unterbrochen wird.

Ein Beispiel für Peripheral Awareness, außerhalb von Software, ist das Wissen über das aktuelle Wetter. Wenn man sich in einem Zimmer mit Fenster befindet, ist einem das aktuelle Wetter jederzeit bekannt, auch wenn man nicht bewusst aus dem Fenster gesehen hat (Cadiz u. a., 2001).

2.2. Wissensmanagement

Wissensarbeit gewinnt in der heutigen Gesellschaft zunehmend an Bedeutung. So hat eine Studie von Hall (2007) gezeigt, dass bereits über 30% der deutschen Erwerbstätigen in so genannten wissensintensiven Berufen beschäftigt sind. Dazu gehören beispielsweise Ingenieure, Wissenschaftler, Lehrer, Bänker, Ärzte, Juristen oder auch Softwareentwickler (Hall, 2007, S. 10). Die Prognosen deuten an, dass sich dieser Anteil im Wandel von einer Industrie- zu einer Wissensgesellschaft noch deutlich erhöhen wird (North und Güldenberg, 2008, S. 9).

Der Begriff des Wissensarbeiters wurde 1959 von P. Drucker geprägt. Er bezeichnete damit Arbeiter, deren Tätigkeiten die Erstellung und Verarbeitung von Wissen sind (Drucker, 1972). Diese Definition wurde von verschiedenen Forschern weiterentwickelt. So definiert North und Güldenberg (2008) die **Wissensarbeit** als

> „[..] eine auf kognitiven Fähigkeiten basierende Tätigkeit mit immateriellem Arbeitsergebnis, deren Wertschöpfung in der Verarbeitung von Informationen, der Kreativität und daraus folgend der Generierung und Kommunikation von Wissen begründet ist.“ (North und Güldenberg, 2008, S. 22)

Die zunehmende Bedeutung der Wissensarbeit fordert die Unternehmen, das im eigenen Betrieb vorhandene Wissen effizient zu verwalten. **Wissensmanagement** wird dadurch zu einem zentralen Aspekt der Produktivität in einem Unternehmen (Willke, 1998, S. 163). So schreibt Drucker bereits 1999: „The most valuable asset of a 21st-century institution (whether business or nonbusiness) will be its knowledge workers and their productivity (Drucker, 1999, S. 1). “

Wissensarbeiter entwickeln ihr Wissen kontinuierlich weiter und erzeugen daraus wiederum neues Wissen, oft in Zusammenarbeit mit anderen Wissensarbeitern. Die Steigerung ihrer Produktivität stellt sowohl Anforderungen an das Unternehmen als auch an die Arbeiter selbst. Zwei zentrale Ansprüche an den Wissensarbeiter sind dabei Teamkompetenz und Wissensmanagement, während das Unternehmen die Aufgabe hat, für beides einen unterstützenden Rahmen bereitzustellen.

- Die Teamkompetenz bezieht sich auf Kommunikationsfähigkeiten und Sozialkompetenzen des Mitarbeiters. Sie ist Voraussetzung für das gemeinschaftliche Schaffen und den Austausch von Wissen. Dieser kann vom Unternehmen durch die Bereitstellung von verschiedenen Kommunikationstechnologien gefördert werden (Willke, 1998, S. 168).
- Zur Speicherung seines Wissens verfügt der Mensch nicht nur über sein Gedächtnis. Er verwendet dafür zusätzliche verschiedene Hilfsmittel, wie Bücher oder Dokumente, um sein Wissen festzuhalten und zu einem späteren Zeitpunkt wieder darauf zuzugreifen. Andy Hunt definierte in diesem Zusammenhang den Begriff des *Exocortex*, der das so gespeicherte Wissen beschreibt (Hunt, 2008, S. 66). Um effizient arbeiten zu können, benötigt der Mitarbeiter einen schnellen und flexiblen Zugriff auf den eigenen sowie den Exocortex anderer. Daher fällt es in den Aufgabenbereich des Unternehmens, eine Infrastruktur zum Wissensmanagement zur Verfügung zu stellen, die eine entsprechende Verwaltung der Exocortexe ermöglicht.

Zur Unterstützung der Zusammenarbeit und des Wissensmanagement haben sich einige Technologien des *Web 2.0* etabliert, wie beispielsweise Wikis, Blogs oder Instant-

Messaging Systeme. Sie fördern sowohl die Generierung von explizitem (oder auch *formuliertem*) Wissen als auch die Kommunikation und sind im Unternehmenskontext den *Enterprise 2.0*-Systemen zuzuordnen (siehe Kapitel 3.2). Das Forschungsgebiet *Enterprise 2.0* untersucht außerdem Möglichkeiten der Aufbereitung und Analyse des vorhandenen gespeicherten Wissens, um zusätzliche Informationen über das implizite (oder *stillschweigende*) Wissen der Mitarbeiter zu erhalten. Dies können etwa Auskünfte über den Entstehungsverlauf von Wissen oder soziale Verbindungen im Unternehmen sein (Willke, 1998, S. 164).

Das Wissensmanagement erleichtert demnach nicht nur das Generieren und Speichern von explizitem Wissen, sondern bietet auch Möglichkeiten, an das implizite Wissen der Mitarbeiter zu gelangen.

2.3. Kontext

Der Begriff *Kontext* besitzt in verschiedenen Forschungsdisziplinen unterschiedliche Definitionen. Der Ursprung des Begriffs kommt aus der Linguistik, wobei der Kontext als Teil der Sprache angesehen wird (Winograd, 2001, S. 3). In dieser Arbeit wird der Kontext der Mensch-Computer-Interaktion betrachtet und gemäß Dey (2001) wie folgt definiert:

> „Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. “ (Dey, 2001, S. 4)

Kontextinformationen beschreiben demnach die Situation, in der sich eine Entität befindet. Bei einer Entität handelt es sich in diesem Zusammenhang um eine Person, einen Ort oder ein Objekt, das an der Interaktion eines Nutzers mit einer Anwendung beteiligt ist.

Passend zu der vorliegenden Arbeit wurde vom Autor ein Beispiel für die Definition entwickelt:

Anwendung – eine Entwicklungsumgebung zur Bearbeitung einer Datei

Entität – die zu bearbeitende Ressource

Kontext – Informationen wie beispielsweise: Zu welchem Projekt gehört die Datei? Ist sie an eine Versionsverwaltung gebunden? Wenn ja, welche Version? Wann wurde sie zuletzt bearbeitet? Vorherige Bearbeiter?

Die vorhandenen Kontextinformationen können von der Anwendung dazu genutzt werden, die Interaktion mit dem Menschen an die Gegebenheiten anzupassen. So kann im obigen Beispiel die Entwicklungsumgebung dem Anwender zusätzliche Bearbeitungsmöglichkeiten oder unterschiedliche Informationsdarstellungen je nach Kontext der Datei anbieten.

Anwendungen, die sich dem Kontext der Entitäten angleichen und dadurch dem Benutzer einen Mehrwert liefern, werden als kontext-sensitiv (**context aware**) bezeichnet. Die Zusammenstellung der Kontextinformationen ist dabei von dem Verwendungszweck der Anwendung abhängig (Schilit u. a., 1994, S. 1).

2.4. Agile Entwicklungsmethoden

Die in der *agilen Softwareentwicklung* eingesetzten Entwicklungsmethoden zeichnen sich durch höhere Flexibilität und Offenheit aus als jene der durchgeplanten und linearen Entwicklung mit klassischen Methoden, wie dem Wasserfallmodell[2]. Im Gegensatz zu diesen setzt die agile Softwareentwicklung nicht auf eine detaillierte Planung im Vorfeld, sondern auf einen iterativen Projektablauf mit kurzen sich wiederholenden Planungs- und Entwicklungsphasen, in die der Kunde enger eingebunden werden kann. Zudem ist vorgesehen, stets alle am Produkt beteiligten Ebenen des Unternehmens - Management, Team und Programmierung - zu involvieren (siehe Abbildung 2.3). Dies ermöglicht eine flexiblere Reaktion auf sich ändernde Anforderungen (it agile, 2011).

2001 verfassten verschiedene Entwickler aus Wirtschaft und Forschung das *Manifest für agile Softwareentwicklung* (Beck u. a., 2001), welches vier Werte und zwölf Prinzipien formuliert, die die besonderen Eigenschaften der agilen Softwareentwicklung beschreiben. Bei den Werten des Manifests handelt es sich um:

- **Individuen und Interaktionen** mehr als Prozesse und Werkzeuge
- **Funktionierende Software** mehr als umfassende Dokumentation

[2] http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

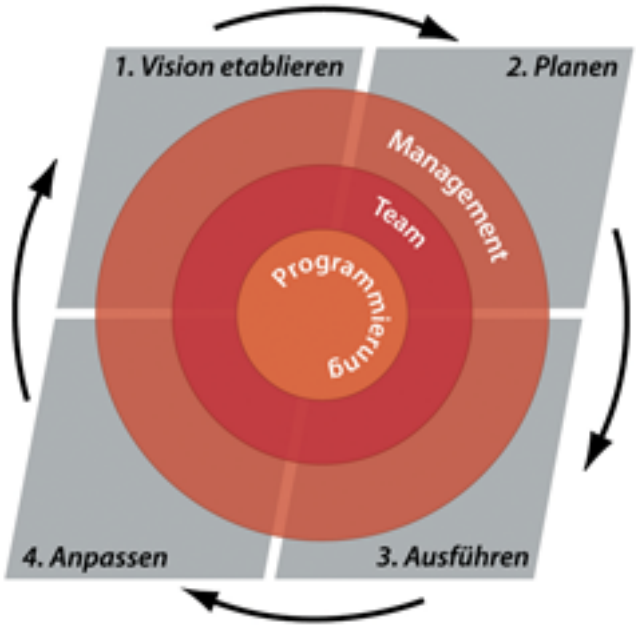


Abbildung 2.3.: Ebenen der agilen Softwareentwicklung (it agile, 2011)

- **Zusammenarbeit mit dem Kunden** mehr als Vertragsverhandlung
- **Reagieren auf Veränderung** mehr als das Befolgen eines Plans

(Beck u. a., 2001)

Beispielhaft für agile Entwicklungsmethoden wird im kommenden Abschnitt eine der populärsten betrachtet. Dabei handelt es sich um *Scrum*.

Scrum

Bei der agilen Entwicklungsmethode *Scrum* steht das sich selbst organisierende **Team** ohne Projektleiter im Mittelpunkt. Es wird lediglich ein **Scrum-Master** bestimmt, der die Scrum-Regeln festlegt und für ihre Einhaltung sorgt. Er ist verantwortlich für den reibungslosen Projektablauf und kümmert sich um etwaige Störungen. Zudem stellt er die

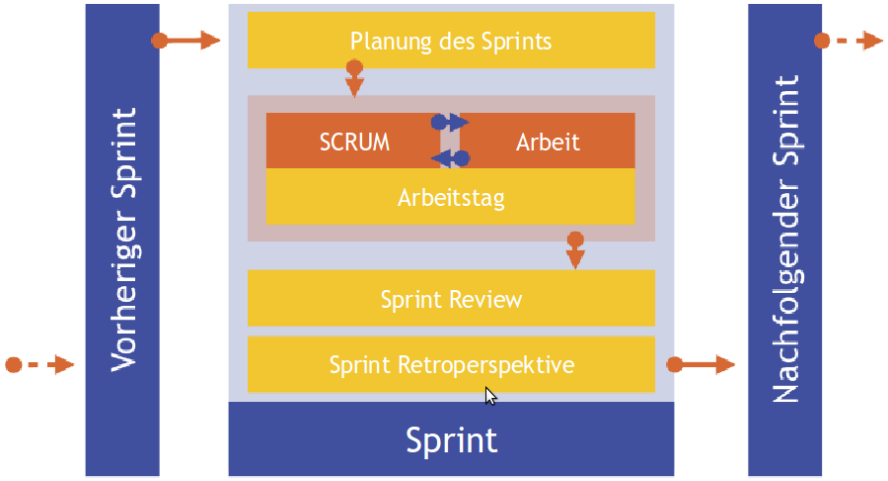


Abbildung 2.4.: Scrum Sprint (Andresen, 2011, S. 51)

Verbindung zwischen dem Entwicklerteam und dem **Product-Owner** dar. Dieser wiederum steht im Kontakt zum Kunden und ist für die klare Definition von Anforderungen zuständig (it agile, 2011).

Ein Scrum-Projektverlauf gliedert sich in so genannte *Sprints* (siehe Abbildung 2.4), die unterschiedliche Zeiträume umfassen können, z.B. eine, zwei oder vier Wochen. Währendessen kann das Entwicklerteam auf ein definiertes Ziel, einen Prototypen mit festgelegten Funktionalitäten hinarbeiten, ohne dabei durch neue Anforderungen gestört zu werden. Eine essentielle Rolle spielen Besprechungen, da Scrum auf klare Absprachen und eine ständige Anpassung zur kontinuierlichen Verbesserung des Projektverlaufs setzt. Diese werden sowohl täglich (manchmal auch nur kurz im Stehen) als auch zu Beginn und Ende eines jeden Sprints abgehalten. Inhaltlich dienen sie der Erörterung des Projektstands und des Verlaufs des Entwicklungsprozesses sowie der Diskussion möglicher Fehler oder Korrekturen. Scrum sieht wenig Hierarchie im Team vor, um einen offenen Austausch und den Respekt untereinander zu fördern. Ergeben sich während Sprints Änderungen bezüglich der Anforderungen an das Produkt, fließen diese in die Planung des nächsten mit ein (Andresen, 2011, S. 52). Durch diese Einteilung in Sprints ist zum einen gewährleistet, dass die Entwickler immer störungsfrei auf ein fest definiertes Ziel hinarbeiten können, zum anderen, dass der Kunde durch die entwickelten Prototypen einen guten Einblick in den aktuellen Entwicklungsstand

hat.

Die Vorteile von Scrum sind die fest definierten Rollen (Entwicklerteam, Scrum-Master und Product-Owner) und die einfache, schnell erlernbare Struktur, die zwar einen klaren Ablauf definiert, zugleich aber auch eine flexibele Reaktionen auf Änderungen ermöglicht (it agile, 2011).

3. Analyse

In diesem Kapitel wird der Sachverhalt zur Entwicklung einer Software betrachtet, die der Unterstützung von Teamarbeit in der Softwareentwicklung dient. Der Fokus der vorliegenden Arbeit liegt auf Möglichkeiten des Umgangs mit Problematiken bei räumlich verteilten Entwicklungsteams, der kognitiven Überlastung der Mitarbeiter und der Schatten-IT. Diese werden zunächst im Kapitel 3.1 erläutert. Diesen und anderen Herausforderungen widmet sich das junge Forschungsgebiet der *Enterprise 2.0*-Systeme (Kapitel 3.2), das daher für die vorliegende Arbeit von großer Relevanz ist. Durch diese beiden Kapitel wird sowohl die Notwendigkeit der Forschung im Bereich der Kollaboration in Softwareunternehmen aufgezeigt, als auch einige grundlegende Anforderungen bei der Entwicklung solcher Systeme analysiert. Kapitel 3.3 entwirft daraufhin zwei Szenarien aus der kooperativen Softwareentwicklung, die das Discourse-Tool unterstützen soll. Anschließend werden Forschungsprojekte und -arbeiten vorgestellt, die für dessen Entwicklung sowie die vorliegende Arbeit von Interesse sind (Kapitel 3.4). Zum Abschluss werden die Ergebnisse der Analyse in Kapitel 3.5 zusammengefasst und die wesentlichen Anforderungen formuliert, die sich daraus für die Entwicklung des Discourse-Tools ergeben.

3.1. Aktuelle Herausforderungen bei der Softwareentwicklung

Die Arbeitsformen der Wissensarbeiter haben sich durch neue technologische Möglichkeiten und die globalisierte Wissensgesellschaft in den letzten Jahren stark verändert. Daraus erwachsen für den Arbeiter einerseits neue Chancen zur Kollaboration, andererseits entstehen jedoch auch neue Anforderungen. Mitglieder in Software-Projekten sind in der heutigen internationalisierten Arbeitswelt oft nicht mehr alle an ein und demselben Entwicklungsstandort beschäftigt, sondern agieren räumlich getrennt

(**Distributed Development**). Dies erschwert die Kollaboration und Kommunikation der Beteiligten und stellt zusätzliche Anforderungen an die Mitarbeiter sowie die verwendete Technologie. Ferner haben Unternehmen heute meist weniger das Problem unvollständiger oder fehlender Informationen, sondern vielmehr das Problem einer Informationsflut, aus der sich der Mitarbeiter selbstständig die für ihn relevanten herausfiltern muss (**Cognitive Overload**). Hinzu kommt die Schwierigkeit, dass immer mehr Mitarbeiter ihre eigene Informationstechnologie in den Betrieb einbringen, die außerhalb der unternehmensweiten IT steht und nicht kontrolliert werden kann (**Schatten-IT**). Diese Problematiken kennzeichnen wesentliche aktuelle Herausforderungen der Softwareentwicklung und werden im Folgenden genauer betrachtet.

3.1.1. Distributed Development

In der heutigen Zeit bilden Projekte in Softwareunternehmen, deren Teammitglieder auf verschiedene Standorte verteilt sind, zur Norm. Als zentrale Ursachen können unter anderem folgende ausgemacht werden:

- die weltweite Verteilung und Verfügbarkeit von Expertise
- die Kostensenkungen der Unternehmen durch die Auslagerung von Arbeit in günstiger produzierende Länder
- unternehmerisches Interesse an der Erschließung und Abdeckung fremder Märkte
- sowie Fusionen mit und Akquisitionen von anderen Unternehmen

(Herbsleb, 2007).

Nicht nur die räumliche Distanz, sondern auch kulturelle, sprachliche, politische und zeitliche Differenzen erschweren dabei den Projektmitgliedern Kommunikation und Informationsaustausch, was somit den gesamten kollaborativen Arbeitsprozess behindert (Eckstein, 2009, S. 1). Zudem steigt die Notwendigkeit regelmäßiger Reisen der Mitarbeiter, woraus wiederum signifikante zeitliche und finanzielle Kosten für das Unternehmen entstehen (Helms, 1991). Vor allem aber durch die fehlende direkte Kommunikation entstehen neue Herausforderungen, mit denen das Projektteam umgehen muss. Im besonderen Maße gilt dies für die agile Softwareentwicklung (siehe Kapitel 2.4),

der die direkte Kommunikation besonders wichtig ist und als eins der zwölf Prinzipien identifiziert wurde:

> „The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.“ (Beck u. a., 2001, Principles)

Eine Studie von Herbsleb und Mockus (2003) zeigte, dass die Arbeit an einer ähnlichen Aufgabe in einem verteilten Team zweieinhalb Mal länger dauert als in einem, das gemeinschaftlich am gleichen Standort arbeitet.

Karsten Panier (2011), der ebenfalls am Projekt *Home Office 2.0* partizpiert, hat einige wesentliche Herausforderungen des *Distributed Development* identifiziert:

- Das **Vertrauen** zwischen den Teammitgliedern ist die Basis für eine ehrliche und effiziente Kollaboration. Die Mitarbeiter müssen das Gefühl haben, dass sie ihre Kollegen offen auf Probleme und Fehler ansprechen können. Außerdem ist Vertrauen die Voraussetzung für die Akzeptanz der Arbeitsergebnisse der Mitarbeiter. Es stellt sicher, dass die entwickelten Lösungen der Kollegen angenommen und nicht um deren Ergebnisse herum gearbeitet wird (Ambler, 2002). Das Aufbauen von Vertrauen ist ein langwieriger Prozess und bedarf des persönlichen Kontakts der Mitarbeiter. Durch eine offene und zuverlässige Kommunikation kann dies gefördert werden (Handy, 1995).
- Bei der verteilten Entwicklung arbeiten die Beteiligten jeweils für sich an den einzelnen Aufgabenpaketen. Die wohl größte Herausforderung für das Unternehmen ist an dieser Stelle das Schaffen einer Umgebung, in der sich die Projektmitglieder als Teil eines Teams fühlen und effizient kollaborieren. Folglich ist die **Teambildung** von großer Relevanz. Auch hierfür ist der persönliche Kontakt die beste Voraussetzung (Ambler, 2002). Verteilten Teams schlägt Ambler (2002) daher vor, zu Beginn eines Projekts über einen gewissen Zeitraum an einem Standort zusammenzuarbeiten. So kann sich eine entsprechende Gemeinschaftbilden und das Fundament für eine gute Kollaboration geschaffen werden (McKinney und Whiteside, 2006).
- Die **Kommunikation** stellt die Grundlage des Vertrauens dar. Eine Schwierigkeit bei verteilten Teams ist, dass non-verbale Ausdrücke durch traditionelle Kommunikationsmittel, wie Telefon oder Emails, nicht übertragen werden; dies führt leicht zu Missverständnissen zwischen den Gesprächspartnern (von Thun, 1981). Eine

weitere Herausforderung ist die Inizierung eines Kontakts, wobei zwischen der synchronen und der asynchronen Kommunikation unterschieden werden kann. Wenn erstere initiiert wird, etwa ein Telefonanruf, kann der Kollege möglicherweise bei einer wichtigen Tätigkeit unterbrochen werden. Wird die asynchrone Kommunikation gewählt wird, beispielsweise eine Email, dauert die Beantwortung einer möglicherweise wichtigen Frage unter Umständen zu lange.

Für eine effiziente Kommunikation im Team ist dabei nicht nur die Qualität der Inhalte der Gespräche relevant, sondern auch die Erkenntnis, wann ein Gespräch geführt werden sollte. Da bei lokalen Arbeitsgruppen eine direkte Kommunikation stattfindet (auch im Flur oder in der Pause) und so die Gedanken und Ideen der Kollegen präsenter sind, ist es leichter zu erkennen, zu welchem Zeitpunkt ein Gespräch geführt werden muss.

- In einem verteilten Team ohne direkte Kommunikation besteht eine größere Gefahr der **redundanten Arbeit**. Da einige Anforderungen und Schwierigkeiten eines Softwareprojekts erst während der Entwicklung deutlich werden, müssen einzelne Komponenten immer wieder angepasst werden. Wenn die Realisierung eines Problems an mehreren Standort parallel geschieht, kann es passieren, dass lokal und redundant an einer Lösung gearbeitet wird. Das Bestehen eines Vertrauensverhältnisses im Team sowie der offene Austausch zwischen den Mitgliedern wirkt dieser Problematik entgegen. Aus diesem Grund definieren agile Methoden tägliche Absprachen (siehe Kapitel 2.4), um sich über akute Problem zu verständigen.

Die aufgeführten Herausforderungen und Probleme verdeutlichen die Relevanz der Unterstützung der Wissensarbeiter durch die Bereitstellung effizienter Kommunikations- und Kollaborationswerkzeuge seitens der Unternehmen.

3.1.2. Cognitive Overload

Ein Softwareentwickler hat heute bei seiner Arbeit nicht mehr das Problem fehlender Informationen, sondern vielmehr das Problem einer Informationsflut sowohl im Unternehmen als auch im Internet. Für den Wissensarbeiter ist es essentiell, Wissen und Information mit seiner aktuellen Aufgabe in Verbindung zu bringen. Er muss sich den

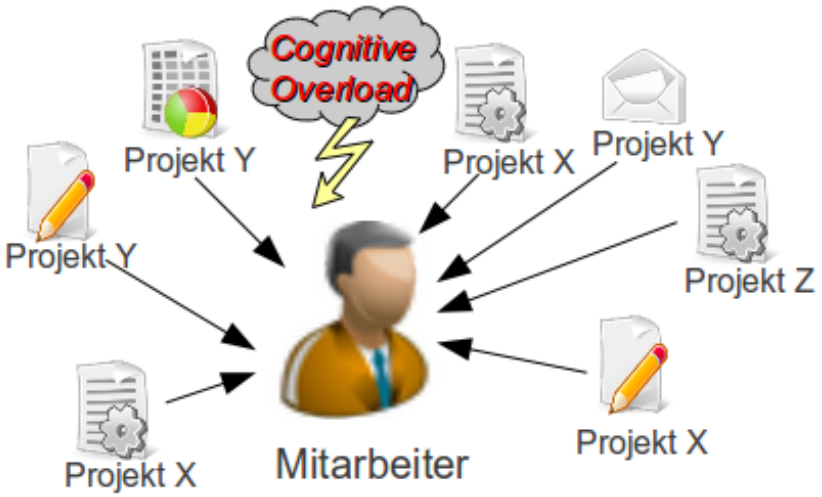


Abbildung 3.1.: Cognitive Overload

Kontext der Aufgabe aufbauen, um aus den Informationen Wissen generieren zu können. Die Herausforderung der Mitarbeiter heute besteht folglich eher in der Filterung aller Informationen, um die für die Aufgabe relevanten zu ermitteln (Kirsh, 2000, S. 21). Diese Problematik wird als *Cognitive Overload* (kognitive Überlastung) bezeichnet (siehe Abbildung 3.1) und kann zu Stresssituationen führen, die wiederum die Produktivität senken (Kirsh, 2000, S. 51). Allerdings existieren einige Faktoren in der heutigen Arbeitswelt, die die kognitive Überlastung fördern:

Mehrere Projekte – Ein Entwickler ist meistens in mehreren Projekten gleichzeitig aktiv und muss an einem Tag häufiger zwischen diesen wechseln.

Verschiedene Rollen – In den verschiedenen Projekten nimmt ein Mitarbeiter oft auch unterschiedliche Rollen ein. So kann er bspw. in einem Projekt als Entwickler und in einem anderen als Projektleiter tätig sein. Dabei sind je nach Rolle auch unterschiedliche Informationen eines Projektes von Relevanz.

Pflege alter Software – Auch wenn Software, im Gegensatz zur Hardware, keinen

> Verschleiß aufweist, muss sie dennoch häufig in der Produktion gewartet werden. So führen etwa neue Umgebungsfaktoren dazu, dass Software angepasst werden muss. Ein bekanntes Beispiel hierfür war die Jahrtausendumstellung.

Durch die genannten Faktoren wird der Wechsel zwischen verschiedenen Projekten und Arbeitsaufgaben an einem Arbeitstag häufig notwendig. Der Wechsel kostet den Entwickler viel Zeit, da er sich jedes Mal den Kontext der aktuellen Arbeitsaufgabe aufbauen und die relevanten Informationen ermitteln muss (Kersten, 2007, S. 1).

Kirsh (2000) sieht den *Cognitive Overload* als

> „[..] seems an inevitable consequence of the complexity of our information intense environments.“(Kirsh, 2000, S. 23)

Aktuell werden verschiedene Lösungen entwickelt, um dieser Problematik entgegen zu wirken. In der Regel geht es meist darum, den Entwickler darin zu unterstützen, den Kontext der aktuellen Arbeitsaufgabe aufzubauen, indem die präsentierten Informationen anhand der aktuellen Situation vorgefiltert werden. Ein Beispiel dafür ist Mylyn (Kapitel 3.4.2).

3.1.3. Schatten-IT

Schatten-IT ist IT im Unternehmen, die nicht von der IT-Abteilung, sondern von den Mitarbeitern selbst initiiert und eingeführt wird (Raden, 2005). Sie ist abgekoppelt von der Unternehmens-IT-Struktur und verwendet ihre eigenen Systeme und Regeln. Daher ist sie nur sehr schwer bis gar nicht zu kontrollieren (Bayan, 2004). Beispiele für Schatten-IT sind ein privater USB-Stick, der verwendet wird, um Daten vom Arbeitsrechner mit nach Hause zu nehmen oder eine IM(Instant Messaging)-Applikationen, die der Mitarbeiter im Unternehmen zur privaten Kommunikation oder im Rahmen der Arbeit verwendet.

Die aktuelle Situation, die zur Problematik der Schatten-IT führt, kann in zwei Hauptbereiche unterteilt werden. Zum einen die neuen technologischen Möglichkeiten und zum anderen die soziale Situation in Unternehmen.

- Durch die Möglichkeiten des Internets und die dort frei verfügbare und zugreifbare Software können Entwickler ihre Werkzeuge selbst wählen, herunterladen und verwenden. Mitarbeiter bevorzugen Software, die sie gewohnt sind und mit der sie gerne arbeiten. Beispielsweise verwenden sehr viele Entwickler IM und nur die wenigsten Unternehmen stellen eigene IM-Systeme zur Verfügung. Dies führt zu dem Problem, dass immer mehr Daten, die sich auf die Arbeit beziehen, außerhalb der Unternehmens-IT produziert werden und die Akzeptanz der Unternehmens-IT sinkt (Worthen, 2007).
- Die soziale Situation bezieht sich auf das Arbeitsgefühl der Mitarbeiter im Unternehmen. So entsteht Schatten-IT vor allem dann, wenn Mitarbeiter davon überzeugt sind, dass sie ihre Arbeit effizienter durchführen können, wenn sie nicht an die Unternehmens-IT gebunden sind. Diese Arbeiter, oder auch ganze Teams, nehmen die Gestaltung der IT-Struktur in die eigenen Hände. Daher zeigt Schatten-IT häufig eine Differenz zwischen den Vorstellungen der Mitarbeiter und der IT-Abteilung (Spafford, 2004). Mitarbeiter wollen Werkzeuge, die ihrer Arbeit vereinfachen und auf ihre individuellen Bedürfnisse eingehen. Die IT-Abteilungen dagegen wollen Software, die sicher, kontrollierbar und skalierbar ist. Sie konzentrieren sich auf das Datenmanagement aus der Unternehmenssicht. Daher kommt im Unternehmen die Verwaltungsmöglichkeit eines Werkzeugs häufig vor der User-Experience des Mitarbeiters (Raden, 2005).

Schatten-IT kann zu verschiedenen Problemen im Unternehmen führen:

- Gefährdung der Infrastruktur des Unternehmens; bspw. können durch den Download von ungeprüfter Software Löcher in Firewalls geöffnet werden.
- Offenlegen vertraulicher Daten durch unvorsichtige Nutzung von Laptops, Handhelds und USB-Sticks.
- Verstoß gegen gesetzliche Bestimmungen durch falsche Handhabung vertraulicher Daten.
- Entstehen inkonsistenter Daten, wenn die Daten nicht zentral und gemeinschaftlich verwendet, sondern immer wieder kopiert und lokal modifiziert werden.
- Zeit- und Geldverschwendung, durch versteckte Kosten für die Pflege und Aktualisierung von Daten, die nicht in die IT-Struktur eingebunden sind (Raden, 2005).

Schatten-IT	Unternehmens-IT
Übernimmt sehr selten Kernaufgaben im Unternehmen (z.B. Netzwerk, Sicherheit). Sie deckt vornehmlich Lücken ab, die die IT-Abteilung offen lässt, wie etwa Kommunikations- und Kollaborationswerkzeuge (Raden, 2005)	Konzentriert sich auf Kernaufgaben.
Entsteht per Zufall und nach Bedarf vom Anwender initiiert (Sherman, 2004)	Ist von Grund auf geplant und wird organisiert durch die IT-Abteilung eingeführt.
Meist sehr unstrukturiert, unzusammenhängend und schwer zu verwalten. (Worthen, 2007)	Strukturiert, organisiert und dadurch leicht zu verwalten und zu kontrollieren.

Tabelle 3.1.: Unterschiede Schatten-IT und Unternehmens-IT

Eine Studie von Elitsa Shumarova und Paul A. Swatman hat ergeben, dass vor allem Kommunikations- und Kollaborationswerkzeuge als Schatten-IT im Unternehmen auftreten (Shumarova und Swatman, 2008). Sie schufen dabei den Begriff der Schatten-CIT (*collaborative information technologies*). Dazu gehören offene, web-basierte, soziale Kollaborationswerkzeuge, wie Blogs, Wikis, Foren und RSS-Feeds. Vom Unternehmen angebotene Werkzeuge finden meist nur wenig Akzeptanz und häufig werden nur firmen-interne Email-Server und Teleconferencing-Systeme genutzt (Shumarova und Swatman, 2008). Es besteht folglich eine große Diskrepanz zwischen der steigenden Benutzerakzeptanz von externen sozialen Kollaborationssystemen und der geringen Benutzerakzeptanz von offiziellen Kollaborationswerkzeugen der Unternehmen.

Matthew Glotzbach, Product Management Director bei Google, äußerte sich in diesem Zusammenhang bereits 2006 kritisch:

> „Enterprise applications do not deliver enough value to the end user. (...) Enterprise technologies are built by the experts for the experts. As enterprise technology has evolved historically, it has become less user friendly. And that is a big problem because another evolution is happening at the same time, and that is the merging of the work life and the personal life.“(Shumarova und Swatman, 2008)

Wie bereits erwähnt, zeigt Schatten-IT unter anderem an, inwieweit die Unternehmens-IT den Service-Vorstellungen des Mitarbeiters entspricht. In vielen Fällen bedarf es einer engeren Absprache zwischen den Mitarbeitern und der IT-Abteilung, sodass diese deren Bedürfnisse kennen und auf sie eingehen können (Spafford, 2004). Eine angebrachte Reaktion eines Unternehmens auf Regelverstöße sollten nicht Bestrafungen sein, sondern viel mehr Überlegungen, warum die Regel gebrochen wurde und was daraus gelernt werden kann (Marquis, 2006).

3.2. Enterprise 2.0

Das Forschungsgebiet *Enterprise 2.0* untersucht die Möglichkeiten von *Social Software*, um den Informationsaustausch und die Kollaboration in Unternehmen zu unterstützen. Unter *Social Software* werden dabei Web 2.0-Technologien, wie Wikis, Weblogs, Instant Messaging (IM), Social Tagging und Präsenz-Awareness-Systeme verstanden, die eingesetzt werden um, eine flexible Zusammenarbeit sowie den freien Wissensaustausch in Unternehmen zu verbessern. Zusätzlich soll die Bildung sozialer Netze auch über Standortgrenzen hinaus gefördert und so das Wir-Gefühl im Unternehmen gestärkt werden (Koch, 2008, S. 2). Dies unterstützt die Herausforderungen des *Distributed Development* (siehe Kapitel 3.1.1).

Der Begriff *Enterprise 2.0* wurde im Jahre 2006 von McAfee, als Synonym für *Social Software*-Systeme geprägt, die die Ergebnisse und Praktiken der Wissensarbeiter offenlegen und so den Kollegen zugänglich machen (McAfee, 2006, S. 23). Im Unterschied zu traditionellen Kollaborationswerkzeugen, verfolgen *Enterprise 2.0*-Systeme das Ziel, nicht nur dokumentiertes, sondern auch implizites Wissen und „Best Practices“ sichtbar zu machen (Koch, 2008, S. 3). Zusätzlich sind bei der Entwicklung dieser Systeme die Prozessunterstützung und die UserExperience wichtiger als Struktur und Technik, die hinter den Systemen steht. Dadurch wird die Akzeptanz durch die Mitarbeiter gesteigert (Shumarova und Swatman, 2008, S. 384).

Web 2.0-Technologien haben schon seit geraumer Zeit Einzug in die Kollaboration in Softwareunternehmen erhalten. Allerdings werden in diesem Zusammenhang meist Insellösungen verwendet, die in keiner Verbindung zueinander stehen. *Enterprise 2.0*-Systeme verfolgen das Ziel, diese Insellösungen zu verbinden und so zusätzliche Informationen und damit implizites Wissen sichtbar zu machen (McAfee, 2006, S. 23).

Dieses kann beispielsweise den Aufbau des Kontextes einer Arbeitsaufgabe unterstützen (siehe Kapitel 3.1.2).

Die Einführung der *Enterprise 2.0*-Systeme wirkt dem Auftreten von Schatten-IT (siehe Kap. 3.1.3) entgegen, bei der wertvolles generiertes Wissen möglicherweise nur dem Verfasser zugänglich ist. So kann beispielsweise die Installation eines unternehmensweiten Blogs den Informationsfluss zwischen Mitarbeitern vereinfachen oder die Verwendung eines firmeninternen IM-Systems die Möglichkeit, Erkenntnisse der Kommunikation anderen Mitarbeitern zugänglich zu machen, bieten.

Ein weiterer wichtiger Indikator für die Relevanz von *Enterprise 2.0*-Systemen ist die kommende Generation der „Digital Natives“. Dabei handelt es sich um junge Menschen, die mit den neuen Web 2.0-Technologien und der Verwendung von digitalen sozialen Netzwerken aufgewachsen sind. Diese kommende Arbeitergeneration ist die soziale Interaktion und die Kollaboration über diese Technologien gewohnt (Gasser, 2009, S. 35ff.). Daher ist es für Unternehmen relevant, diese Möglichkeiten auch auf Unternehmensebene zur Verfügung zu stellen und so die Effizienz dieser Mitarbeiter zu steigern.

McAfee betont, dass die Einführung von *Enterprise 2.0* in ein Unternehmen nicht nur aus der Installation der entsprechenden Systeme besteht. Er identifizierte folgende Anforderungen:

- Das Schaffen einer offenen Unternehmenskultur, in der Mitarbeiter für neue Technologien empfänglich sind.
- Die Bereitstellung einer unternehmensweiten Plattform, auf der die Mitarbeiter auch über Standortgrenzen hinaus kollaborieren können.
- Ein informeller Rollout, der die Systeme den Mitarbeitern nicht aufzwingt, sondern nur einzelne Gruppen zur Nutzung ermutigt und so das Interesse für die Systeme von selbst verbreitet wird.
- Die Unterstützung der Geschäftsführung; diese muss geschlossen hinter der Einführung stehen und die Mitarbeiter zur Nutzung der Systeme ermutigen.

(McAfee, 2006, S. 26)

Da sich die vorliegende Arbeit ebenfalls mit der Bewältigung der oben genannten Herausforderungen beschäftigt, fällt auch sie in den Bereich der *Enterprise 2.0*-Forschung.

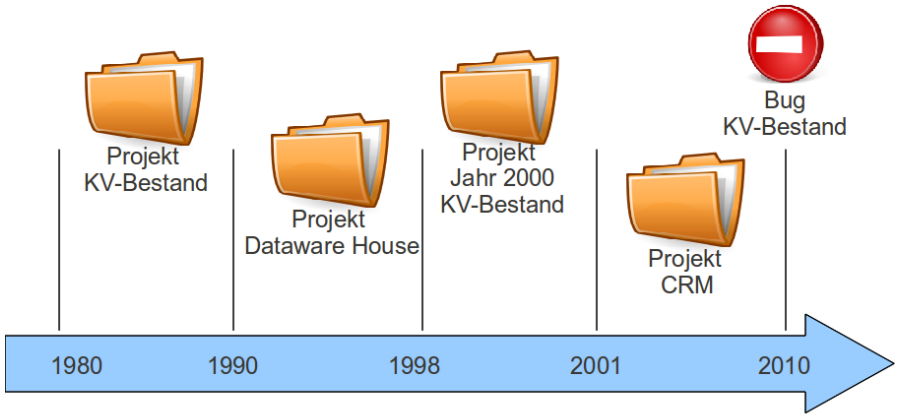


Abbildung 3.2.: Szenario Old Software Never Dies

3.3. Szenario

In diesem Kapitel werden zwei Szenarien beschrieben, die durch das Discourse-Tool unterstützt werden sollen. Eine grundlegende Annahme für die Szenarien ist, dass es sich bei den beschriebenen Unternehmen um Softwarehäuser mit zehn oder mehr Mitarbeitern handelt und verschiedene Werkzeuge zur Kollaboration, wie Versionsverwaltung, Wikis und Aufgabenverwaltungssyteme, verwendet werden. Bei den Mitarbeitern wiederum handelt es sich um Softwareentwickler.

3.3.1. Old Software Never Dies

Das Szenario **Old Software Never Dies** (siehe Abbildung 3.2) behandelt die Arbeit eines Mitarbeiters an einer Datei eines bereits „abgeschlossenen“ Projektes.

Auch wenn Software keinem physikalischen Verfall, wie Hardware unterliegt, muss sie sich dennoch Änderungen im Laufe der Zeit anpassen. Dazu gehören sowohl neue Anforderungen an die Software als auch Änderungen im Umfeld der Software. Ein bekanntes Beispiel dafür ist das „Jahr-2000-Problem[1]“.

[1] http://www.at-mix.de/y2k_problem.htm

In dem dargestellten Szenario wird in dem vermeintlich abgeschlossenen Projekt KV-Bestand aus den 1980er Jahren im Jahr 2010 ein Fehler entdeckt. Das Projekt war 1990 geschlossen und bereits zur Jahrtausendwende wieder eröffnet worden, um es für das Jahr-2000-Problem vorzubereiten. Die Mitarbeiter, die damals für das Projekt verantwortlich waren, sind jedoch 2010 in anderen Projekten involviert oder nicht mehr beim Unternehmen angestellt. Als Folge erhält ein Mitarbeiter, der erst seit einigen Jahren im Unternehmen tätig ist, die Aufgabe, den Fehler in den Klassen des Projekts KV-Bestand zu finden und zu beheben. In diesem Zusammenhang muss er sich aus der Informationsflut im Unternehmen die für die Aufgabe relevanten Informationen herausfiltern (*Cognitive Overload*, siehe Kapitel 3.1.2). Diese Problematik gilt nicht nur für Mitarbeiter, denen das Projekt bisher unbekannt war, sondern auch für jene, die vor zehn Jahren an dem Projekt beteiligt waren. Es ist auch für sie zeitaufwendig, sich in zurückliegende Vorgänge erneut einzuarbeiten.

Das Problem, das sich für den Mitarbeiter stellt, ist die Filterung der im Unternehmen vorhandenen Informationen im Hinblick auf die Aufgabe, um sich den Kontext für diese aufzubauen. Eine für den Entwickler hilfreiche Variante der Unterstützung sind Diskurse über die Entwicklungen bezüglich des Projekts, mit dem die Aufgabe verbunden ist. Dabei lassen sich verschiedene Typen von unterstützenden Diskursen identifizieren:

Projektdiskurse – Der Projektdiskurs beinhaltet Informationen über das gesamte Projekt, beispielsweise Diskussion zur Analyse und Design eines Projekts, Entwicklungen an der Architektur, die Teammitglieder, die identifizierten und delegierten Aufgaben, Änderungen des Designs während der Entwicklung, Termine und Milestones.

Aufgabendiskurse – Der Aufgabendiskurs dreht sich um Entwicklungen und Informationen im Bezug auf eine Aufgabe im Projekt. Dazu gehören der verantwortliche Mitarbeiter, Wikiseiten, die im Rahmen der Aufgabe erstellt wurden, und Ressourcen, die zur Bewältigung der Aufgabe bearbeitet werden müssen.

Dateidiskurse – Der Dateidiskurs bezieht sich auf die Entwicklung an einer bestimmten Ressource. In diesem Zusammenhang sind verschiedene Informationen interessant. Zum Beispiel die ehemaligen Entwickler an der Datei, die Veränderungen der Da-

tei im Laufe der Zeit, Wikiseiten im Zusammenhang mit der Datei oder Arbeitsaufgaben, in deren Rahmen die Ressource bearbeitet wurde.

In dieser Arbeit wird der Aufbau eines Dateidiskurses betrachtet, der einem Entwickler einen schnellen Überblick über die bisherige Entwicklung an der zu bearbeiteten Ressource verschaffen kann. Der Diskurs einer Ressource beinhaltet dabei nicht nur die Historie der Datei, sondern auch zusätzliche Informationen, die während des Entwicklungsprozesses erstellt wurden. Diese sind davon abhängig, in welchem Rahmen und im Zusammenhang mit welchen Werkzeugen (Wikis, Aufgabenverwaltungssystemen, etc.) an der Ressource gearbeitet wurde.

Der Fehler im Projekt KV-Bestand im obigen Szenario lässt sich auf einige Ressourcen eingrenzen. Der beauftragte Mitarbeiter checkt die entsprechenden Dateien von der Versionsverwaltung aus. Anschließend kann er anhand des Tools Discourse passende Informationen zu diesen abfragen. Dadurch erfährt er zum Beispiel, welche anderen Mitarbeiter die Ressource bearbeitet haben, um mögliche Ansprechpartner bei Fragen zu finden. Oder Informationen über Arbeitsaufgaben, die eine Aussage darüber treffen können im Zusammenhang mit welchen Aufgaben die Datei bearbeitet wurde. Oder Wikiseiten, die im Rahmen der Entwicklung an der Ressource erstellt wurden. Diese gesammelten analysierten Informationen werden für den Mitarbeiter bequem in der Entwicklungsumgebung zugänglich gemacht.

Zusammenfassend unterstützt das Tool Discourse das Einarbeiten in eine Arbeitsaufgabe, indem relevante Informationen zu aktuellen Ressourcen in einem Diskurs in der Entwicklungsumgebung dargestellt werden. So kann sich der Mitarbeiter den Kontext der Arbeitsaufgabe schneller aufbauen und folglich schneller mit der effektiven Fehlerbehebung beginnen.

3.3.2. Awareness-Unterstützung

Das Szenario „Awareness-Unterstützung“ behandelt die Schaffung eines Bewusstseins (*Awareness*, siehe Kapitel 2.1.3) über die aktuelle Entwicklung an einer Ressource.

Mitarbeiter „Bob“ erhält den Arbeitsauftrag, eine Methode zu einer bestehenden Klasse X hinzuzufügen (siehe Abbildung 3.3). Allerdings gehört die Klasse zu einem noch nicht abgeschlossenen Projekt und wird noch aktuell von anderen Mitarbeitern verwendet

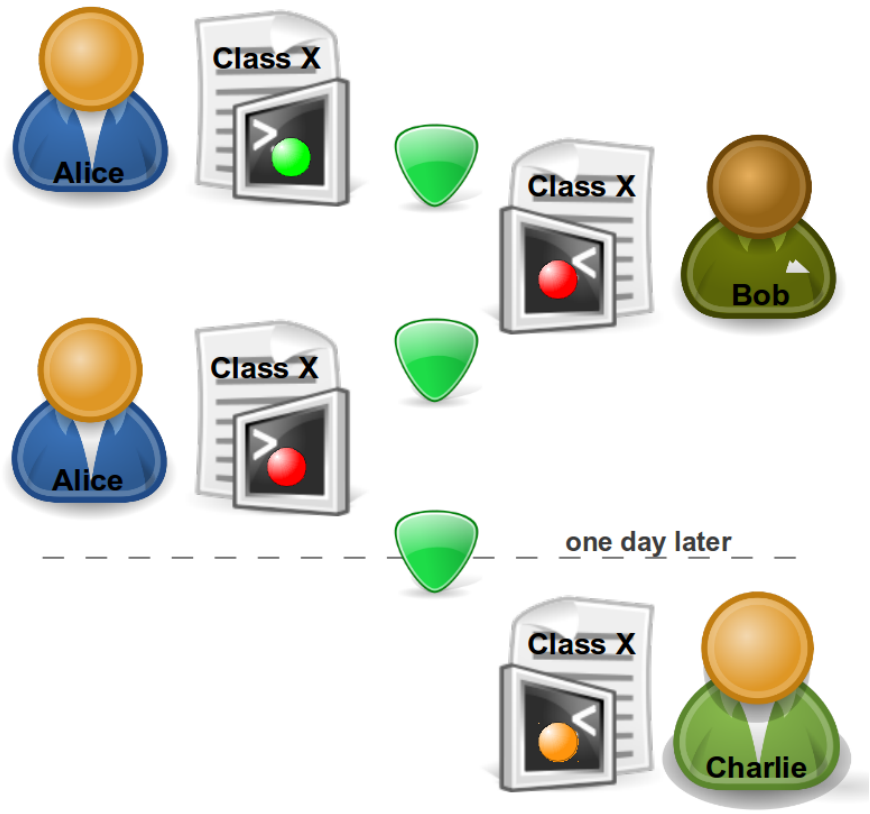


Abbildung 3.3.: Szenario Awareness Unterstützung

und modifiziert. Daher ist es möglich, dass andere Mitarbeiter (z.B. „Alice“) gleichzeitig oder zeitnah an der Ressource arbeiten. Es ist an dieser Stelle sowohl für „Bob“ interessant, darauf aufmerksam gemacht zu werden, dass bereits an der Klasse gearbeitet wird, als auch für „Alice“, dass ein anderer Mitarbeiter beginnt, an der Klasse zu arbeiten. Dadurch können mögliche Konflikte und Überschneidungen frühzeitig angezeigt und möglicherweise umgangen werden. Zusätzlich ist es für Mitarbeiter „Charlie“, der zwei Tage nach der letzten Aktualisierung die Klasse wieder betrachtet, informativ, dass vor kurzem an der Klasse gearbeitet wurde. Dies gibt ihm einen Hinweis, dass in der nächsten Zeit möglicherweise noch andere Mitarbeiter auf sie zugreifen könnten.

Das Szenario kann durch ein „Peripheral Awareness“-System (siehe Kapitel 2.1.3) unterstützt werden. Dieses soll den beiden Entwicklern „Alice“ und „Bob“ signalisieren, dass noch jemand anderes an der Ressource arbeitet. So kann beispielsweise durch eine Ampel-Darstellung angezeigt werden, wie intensiv die Arbeiten an einer Klasse zu einem bestimmten Zeitpunkt sind.

Rot – signalisiert, dass andere Mitarbeiter aktuell an der Ressource arbeiten und es beim Übertragen in die Versionsverwaltung zu Konflikten kommen kann oder sogar möglicherweise redundant gearbeitet wird.

Gelb – zeigt an, dass die Ressource in der letzten Zeit (vielleicht vor ein bis zwei Tagen) bearbeitet wurde und möglicherweise in nächster Zeit wieder von einem anderen Mitarbeiter geöffnet wird.

Grün – signalisiert, dass die Datei das letzte Mal vor längerer Zeit bearbeitet wurde und keine Konflikte bzw. redundante Arbeit zu befürchten sind.

Durch die Signalisierung kann die *Awareness* über die Entwicklung an der Datei und die aktuellen Tätigkeiten der Kollegen gestärkt werden.

3.4. Related Work

In diesem Kapitel werden Forschungen und Projekte betrachtet, die sich ebenfalls mit der Unterstützung von Softwareentwicklern und ihrer Kollaboration beschäftigen. Ein bekanntes und erfolgreiches Forschungsprojekt ist **ACTIVE**.

Das **Mylyn**-Projekt befasst sich mit der Unterstützung eines Softwareentwicklers durch Hilfe beim Aufbau des Kontexts zu einer Arbeitsaufgabe.

Zusätzlich wird schon seit einigen Jahren daran geforscht, wie die Daten in Software Repositories durch Analysen dazu verwendet werden können zusätzliche Kenntnis über Projekte und Unternehmensstrukturen zu gewinnen (**Mining Software Repositories**).

3.4.1. ACTIVE Projekt

ACTIVE ist ein Konsortium aus verschiedenen europäischen Unternehmen, das sich zum Ziel gesetzt hat, das ungeteilte Wissen der Mitarbeiter („versteckte Intelligenz“ des Unternehmens) in zugreifbares und anwendbares Wissen umzuwandeln, um die Kollaboration und Problemlösung zu unterstützen (ACTIVE Consortium, 2011a).

ACTIVE macht dabei eine deutliche Unterscheidung zwischen formellem und informellem Wissen. Bei informellem Wissen handelt es sich um Wissen, das der Mitarbeiter nur für sich selbst kreiert, ohne es offiziell zu teilen. Das formelle Wissen dagegen wird gezielt mit den anderen Mitarbeitern geteilt (z.B. Wiki-Seite).

Das Konsortium verfolgt drei Forschungsschwerpunkte:

- Die Unterstützung des Wissensaustauschs im Unternehmen, sowohl des formellen als auch des informellen Wissens.
- Das Teilen und Wiederverwenden von informellen Wissensprozessen
- Das Verstehen des Benutzerkontextes

(ACTIVE Consortium, 2011a)

Die Vision des Konsortiums, ist die Produktivität von Wissensarbeitern in Teams zu verbessern durch die Bereitstellung der passenden Informationen zu einer Aufgabe, wobei auch das versteckte Wissen offengelegt und anderen zugreifbar gemacht wird (ACTIVE Consortium, 2011a, Vision); damit fällt die Forschung von ACTIVE in das Forschungsgebiet *Enterprise 2.0* (siehe Kapitel 3.2). Um zu Erkenntnissen zu gelangen, beobachtet das ACTIVE Projekt, wie Wissensarbeiter ihr Wissen verwenden und erweitern um ein bestimmtes Ziel zu erreichen (Perez u. a., 2009, S. 1).

Bei ACTIVE geht es allerdings nicht nur um die Forschung, sondern auch um die Umsetzung der Ergebnisse. Dafür wird der Ansatz einer kontextsensitiven Arbeitsumgebung verfolgt (Schilit u. a., 1994). In diesem Zusammenhang wurde die Arbeitsumgebung „ACTIVE knowledge workspace software (AKWS)“ entwickelt und die Ergebnisse der Forschung implementiert. Die AKWS soll die vorhandenen Applikationen zur Kollaboration nicht ersetzen, sondern wird in den Arbeitsprozess des Benutzers eingebunden, indem es mit den alltäglichen Applikationen verbunden wird. Die Arbeitsumgebung erstellt dann einen Arbeitskontext, der mit anderen Teammitgliedern geteilt werden kann. Der Kontext wird sowohl automatisch durch das häufige Verwenden von Ressourcen, wie Dokumente, Emails oder Webseiten, als auch durch manuelles Hinzufügen von Informationen erstellt. Zusätzlich können Kommentare und Tags zu den Ressourcen hinzugefügt werden. Auch informelle Wissensprozesse können in der Arbeitsumgebung aufgezeichnet werden und dem Kontext hinzugefügt werden (z.B. durch welche Schritte ein bestimmtes Problem gelöst werden kann). Durch das Teilen der Kontexte in einem Kontext-Repository können die Teammitglieder ihre Arbeitsumgebung um die Kontextinformationen anderer Mitglieder erweitern (ACTIVE Consortium, 2011b).

3.4.2. Mylyn

Bei Mylyn handelt es sich um ein Eclipse Plug-In, das in der Entwicklungsumgebung eine Aufgaben-fokussierte Benutzeroberfläche bietet. Die erste Version dieser Anwendung entstand im Rahmen einer Doktorarbeit von Kersten (2007). Der Autor behandelte in diesem Zusammenhang hauptsächlich zwei Problematiken bei der Softwareentwicklung:

- **Information overload problem**
  Bei der Arbeit mit Systemen, die den Entwickler mit Informationen versorgen, werden diese immer auf Basis des zugrunde liegenden Systems dargestellt. Beispielsweise werden die Ergebnisse einer Websuche anhand eines Suchalgorithmus ermittelt, bewertet und dargestellt. In diesem Fall werden auch Informationen angezeigt, die für den Benutzer irrelevant sind. Die für ihn interessanten Informationen muss er sich folglich aus der Vielzahl der angebotenen herausfiltern.
- **Context loss problem**
  Während der Bearbeitung einer Aufgabe generiert ein Entwickler Wissen und

> sammelt sich Informationen zur Bewältigung der Aufgabe an. Diese werden in der Regel, wenn überhaupt, erst nach Beendigung der Aufgabe dokumentiert. Da es häufig vorkommt, dass der Entwickler unterbrochen wird und eine andere Aufgabe bearbeitet werden muss, besteht die Gefahr, dass er diese Informationen, den *Kontext*, zur eigentlichen Aufgabe verliert. Als Folge muss er sich diese nach der Unterbrechung wieder neu aufbauen.

(Kersten, 2007, S. 1)

In seiner Doktorarbeit setzt sich Kersten mit der Reduzierung der beiden genannten Probleme in der Entwicklungsumgebung (IDE) eines Softwareentwicklers auseinander. In diesem Rahmen wurde das PlugIn Mylyn für die Entwicklungsumgebung Eclipse (The Eclipse Foundation, 2011a) entwickelt. Es reduziert die Informationsflut in der IDE und zeigt nur jene Ressourcen an, die für die aktuelle Arbeitsaufgabe relevant sind. Auf Basis der Tätigkeiten eines Entwicklers in der Entwicklungsumgebung wird ein Kontext erzeugt, in dem die Ressourcen nach ihrer Verwendung und Relevanz erfasst werden.

Um dies zu realisieren, stellt Mylyn einige Connectoren (Mylyn Team, 2011b) zu Aufgabenverwaltungssystemen (Ticketsystemen) wie beispielsweise JIRA (Atlassian Pty Ltd., 2011b) zur Verfügung. In der Entwicklungsumgebung wählte der Entwickler ein Ticket aus und markiert die Bearbeitung als aktiv. Mylyn registriert nun während der Tätigkeit des Entwicklers alle verwendeten Ressourcen und erstellt so einen Kontext zur aktuellen Arbeitsaufgabe. Dieser kann bei der nächsten Bearbeitung der Arbeitsaufgabe verwendet werden, so dass nur die relevanten Resourcen angezeigt werden (Mylyn Team, 2011a). Zusätzlich kann der Kontext mit anderen Mitarbeitern geteilt werden, beispielsweise wenn die Arbeitsaufgabe übertragen wird.

3.4.3. Mining Software Repositories

Das Forschungsfeld *Mining Software Repositories* (MSR) beschäftigt sich mit Systemen, die die Daten in Software Repositories analysieren, um zusätzliche interessante Informationen über Software-Projekte und -Systeme aufzudecken. In diesem Zusammenhang wird seit 2004 eine jährliche Konferenz abgehalten, die das Ziel hat, den

Titel	Zusammenfassung
Using Software Repositories to Investigate Socio-technical Congruence in Development Projects	Untersuchung der Daten in einem Repository einer Versionsverwaltung um Erkenntnisse zu gewinnen über die Verbindungen zwischen der sozialen Struktur im Entwicklunsteam und dem entwickelten Software-System (Valetto u. a., 2007).
Mining search topics from a code search engine usage log	Analyse der Log-Dateien einer der populärsten Code Suchmaschinen Koder, um häufige Suchthemen zu ermitteln und zu kategorisieren. Durch die Kategorisierung können Erkenntnisse über das Suchverhalten der Anwender und die Effizienz solcher Code Suchmaschinen gewonnen werden (Bajracharya und Lopes, 2009).

Tabelle 3.2.: Forschungsarbeiten im Bereich von Mining Software Repositories (MSR)

Austausch über die Forschung und Anwendung von MSR zu fördern (9th Working Conference on Mining Software Repositories, 2011). Sie findet im Rahmen der *International Conference on Software Engineering* (ICSE) statt.

Unter Software Repositories werden Systeme verstanden, die während des Software-Entwicklungsprozesses verwendet werden und diesen in irgendeiner Form dokumentieren. Dazu gehören beispielsweise:

- Versionsverwaltungssysteme, wie CVS oder Subversion
- Aufgabenverwaltungssysteme, wie Jira oder Trac
- Kommunikationsarchive, wie E-Mail-Listen

(Kagdi u. a., 2007)

Diese Systeme beinhalten viele implizite Informationen über den Entstehungsprozess eines Software-Systems und der damit einher gehenden Projektarbeit. MSR versucht diese Informationen in Verbindung zu bringen und im Hinblick auf eine bestimmte Fragestellung zu analysieren.

In diesem Rahmen wurden bereits eine Vielzahl verschiedener Ansätze mit unterschiedlichen Fragestellungen realisiert. In der Tabelle 3.2 werden beispielhaft zwei Arbeiten genannt, die sich mit dieser Thematik beschäftigen.

3.5. Fazit Analyse

Die zunehmende Globalisierung der Softwareindustrie, die sich stetig verändernden technologischen Möglichkeiten und der Wandel von einer Industrie- zu einer Wissensgesellschaft stellen wachsende Anforderungen an die Kollaboration und Effizienz von Entwickler (siehe Kapitel 3.1). Um diesen gerecht werden zu können, ist eine Umgestaltung der vorherrschenden Kollaborationssyteme angebracht und hilfreich. Dies spiegelt sich auch in der wachsenden Relevanz des Forschungsgebiets der *Enterprise 2.0*-Systeme wieder (siehe Kapitel 3.2).

Als grundlegende Erfolgsfaktoren für den Einsatz solcher Systeme in Unternehmen hebt Koch (2008) die einfache Handhabung und den Fokus auf dem Nutzen für den Einzelnen hervor: also ein positives Nutzen-Aufwand-Verhältnis für den einzelnen Anwender. Als Hürde zeigen sich dabei die unterschiedlichen Dokumentations- und Kommunikationsarten der verschiedenen Mitarbeiter. In einem Unternehmen sollte das dokumentierte Wissen in einer einheitlichen Struktur vorliegen, um allgemeingültige Zugriffe darauf zu gewährleisten. Aus diesem Grund müssen entweder die Dokumentationen der einzelnen Mitarbeiter im Nachhinein angepasst oder den Mitarbeitern klare Dokumentationsregeln vorgeben werden. Das wiederum führt zu einem zusätzlichen Aufwand für den Einzelnen und einer verminderten Motivation zur Nutzung (Koch, 2008, S. 3). Um dies zu verhindern, verfolgt das *Home Office 2.0*-Projekt den erst genannten Ansatz und setzt auf eine nachträgliche Anpassung der Dokumentationen der Mitarbeiter (dazu mehr in Kapitel 4.2).

Im Zusammenhang mit den beschriebenen Herausforderungen der heutigen Softwareentwicklung wurden in 3.3 zwei Szenarien vorgestellt. Diese sollen von dem zu entwickelnden Discourse-Tool behandelt werden. Sie beschäftigen sich mit

- der Unterstützung des Entwicklers beim Aufbau des Kontextes der aktuellen Arbeitsaufgabe und
- dem Schaffen eines Bewusstseins über die Tätigkeiten anderer Mitarbeiter an einer Datei.

Die Betrachtung vorhandener relevanter Forschungen und Projekte zeigt, dass sich die meisten Arbeiten aus dem Gebiet der Kontext-Unterstützung für Entwickler um den Aufbau eines Diskurses im Bezug auf eine konkrete Arbeitsaufgabe (z.B. Jira-Ticket) drehen. Dies ist auch bei Mylyn der Fall (siehe Kapitel 3.4.2). In der vorliegenden Arbeit

wird im Gegensatz dazu versucht, den Entwickler beim Aufbau des Kontexts durch einen Dateidiskurs zu unterstützen.

Der Fokus dieser Arbeit liegt demnach auf der Erstellung eines Diskurses für eine bestimmte Ressource. Da die direkte Entwicklungshistorie dieser Ressource in dem Repository einer Versionsverwaltung gespeichert ist, wird als Basis für den Discourse-Prototypen angenommen, dass die entsprechende Ressource mit einer Versionsverwaltung in Verbindung steht. Dadurch können ausgehend von der Versionsverwaltung Verbindungen zu Daten und Informationen anderer Repositories hergestellt werden (siehe Kapitel 3.4.3).

Die Idee des ACTIVE Projektes (siehe Kapitel 3.4.1), ein Kontext-Repository zu verwenden, ist bereits in die Planung des *Home Office 2.0*-Projekts (siehe Kapitel 4.2) mit eingeflossen. Es ermöglicht, dass gezielt Informationen und Prozesse im Team geteilt werden und sich die Mitarbeiter, trotz unterschiedlicher Standorte, den aktuellen Tätigkeiten der Kollegen bewusst sind.

In der Analyse wurden einige Anforderungen, sowohl funktionale als auch nicht-funktionale, deutlich, die im folgenden Abschnitt zusammengefasst werden.

Das zu entwickelnde Discourse-Tool soll dem Entwickler das Arbeiten an einer Ressource vereinfachen, indem deren Entwicklungsprozess strukturiert dargestellt wird. Da in Softwareunternehmen die Bearbeitung und Betrachtung einer Ressource in der Regel in einer Entwicklungsumgebung stattfindet, wird die Darstellung der Diskursinformationen einer Ressource in die Entwicklungsumgebungen der Mitarbeiter eingebunden. Die Informationen werden aus Daten aus verschiedenen Werkzeugen generiert, die bei der Softwareentwicklung Anwendung finden. Dazu gehören beispielsweise Versionsverwaltungssysteme, Aufgabenverwaltungssysteme oder Wikis. Der Diskurs soll den Entwickler dabei unterstützen, den Kontext der aktuellen Arbeit bei einem Projektwechsel oder Neueinstieg schneller aufzubauen und so die Produktivität zu steigern. Dazu sollen mögliche für die Ressource relevante Wiki-Seiten, Arbeitspakete und vorherige Bearbeiter aufgelistet werden, um sich selbstständig oder in Zusammenarbeit schneller in die aktuelle Arbeitsaufgabe an der Ressource einzufinden. Die wohl größte Herausforderung ist das Ermitteln der wirklich relevanten Informationen. Das heißt, dass möglichst alle interessanten Informationen angezeigt werden und möglichst wenige irrelevante Angebote aufgelistet werden.

Allerdings ist in einem agilen Softwareentwicklungsteam mit weak oder collectiv Code

ownership (Fowler, 2006) nicht nur der vergangene Entwicklungsprozess einer Datei von Interesse, sondern auch deren aktuelle Entwicklung. Daher soll das Discourse-Tool in der Entwicklungsumgebung als „Peripheral Awareness“-System (siehe Kapitel 2.1.3) realisiert werden, welches den Bearbeiter auf andere Entwicklungen an der Ressource aufmerksam macht und so vor möglichen Konflikten warnt.

Bei der Darstellung der Diskursinformationen für den Entwickler ist darauf zu achten, dass dies bedarfsorientiert geschieht, das heißt, dass sie sich nicht aufdrängen. Zudem sollten die Informationen übersichtlich und strukturiert angezeigt werden, damit der Entwickler diese schnell und einfach aufnehmen kann. Für die Aufnahme der Awareness-Informationen ist es wichtig, dass der Aufwand so gering wie möglich gehalten wird. So wird sichergestellt, dass der Entwickler nicht in seinem Wissensgenerierungsprozess unterbrochen wird (Gross und Koch, 2007, S. 62).

4. Design & Realisierung

Dieses Kapitel beschreibt das Design und die Realisierung des Discourse-Tools, das zur Umsetzung der Analyseergebnisse entworfen wurde. Dazu wird zunächst die Vision skizziert, die sich aus den entwickelten Szenarien (siehe Kapitel 3.3) ergibt. Anschließend wird das *Home Office 2.0*-Projekt vorgestellt, welches das Framework für das Discourse-Projekt liefert. In den darauf folgenden Kapiteln wird zunächst die erarbeitete Architektur für das Tool im Ganzen dargestellt und schließlich die einzelnen identifizierten Komponenten derselben im Detail erläutert. Zum Abschluss des Kapitels wird das Design zusammengefasst und aus verschiedenen Blickwinkeln heraus evaluiert.

4.1. Vision

Die Vision des Discourse-Teilprojekts ist die Entwicklung eines Tools zur Unterstützung eines Entwicklers bei der Teamarbeit, das sich ohne zusätzlichen Aufwand in die vorhandenen Arbeitsstrukturen eingliedern lässt und bei den beschriebenen Szenarien behilflich ist.

Dazu soll das Hauptwerkzeug des Entwicklers, die Entwicklungsumgebung, um zusätzliche Komponenten erweitert werden. Diese unterstützen ihn zum einen dabei, sich den Kontext der aktuellen Arbeitsaufgabe rasch aufzubauen und so schneller die effektive Wissensgenerierung beginnen zu können. Zum anderen stärken sie das Bewusstsein über die Kollegen und deren Tätigkeiten, also Teil eines Teams zu sein.

Dabei wird die Entwicklungsumgebung des Mitarbeiters mit einem Server verbunden, der über ein Kontext-Repository verfügt. Dieses beinhaltet, in einer universellen, allgemeingültigen Struktur, Daten von den Werkzeugen, die das Team zur Kollaboration und Wissensgenerierung verwendet. Anhand einer Analyse dieser Daten kann so zum einen zusätzliches Wissen erzeugt und zum anderen die Kollaboration durch einen

gemeinschaftlichen Kontext gestärkt werden. Bei den Werkzeugen handelt es um Versionsverwaltungen, Wikis, Blogs, Ticketsysteme und andere Software die von den Mitarbeitern zur Kollaboration verwendet wird.

Für die beschriebenen Szenarien (siehe Kapitel 3.3) erzeugt die Analyse, anhand der Daten im Kontext-Repository, auf dem Server einen Diskurs über die bisherige Entwicklung an einer Ressource. Dieser Diskurs wird anschließend in der Entwicklungsumgebung des Mitarbeiters übersichtlich dargestellt, der sich so den Kontext seiner aktuellen Arbeitsaufgabe schneller aufbauen kann.

Um zusätzlich eine Awareness über aktuelle Vorgänge an der Ressource zu schaffen, wird in der Entwicklungsumgebung mittels Signalfarben auf den derzeitigen Entwicklungsstatus hingewiesen. Es wird also angezeigt, ob und wie akut andere Mitarbeiter im Netzwerk die Ressource bearbeiten. Dafür informiert der Server alle Mitarbeiter in regelmäßigen Abständen über die aktuellen Tätigkeiten der Kollegen.

4.2. Projekt *Home Office 2.0*

In diesem Kapitel wird das Projekt *Home Office 2.0* vorgestellt. Dabei liegt der Fokus allerdings ausschließlich auf den Informationen, die für die vorliegende Arbeit besonders relevant sind. Für einen tiefergehenden Einblick werden Panier u. a. (2010) und Panier u. a. (2011) empfohlen.

Das Projekt mit dem Codenamen `LuPanKu` wurde im Rahmen des Masterstudiengangs an der Hochschule für Angewandte Wissenschaften (HAW) Hamburg im Wintersemester 2009/10 entwickelt. Dessen primäres Ziel ist die Unterstützung von Softwareentwicklern bei ihrer Arbeit in verschiedenen Teams und Projekten. Es widmet sich dabei der Thematik des *Home Office 2.0*-Szenarios mit dem Fokus auf der Softwareentwicklung. In diesem sind die Mitglieder eines Entwicklerteams zeitweise an verschiedenen Arbeitsplätzen (verschiedene Standorte, Heimarbeitsplatz) tätig, wodurch die direkte Kommunikation zwischen den Mitglieder erschwert wird (Panier u. a., 2010, S. 5).

Dieser Problematik wirkt das *Home Office 2.0*-Szenario entgegen, indem eine virtuelle Nähe zwischen den Teammitgliedern erzeugt wird, die die Kollaboration unterstützt (Panier u. a., 2010, S. 8). Dazu werden Web 2.0-Technologien verwendet, die bereits

länger sowohl im privaten als auch im Arbeitsleben (siehe Kapitel 3.2) Einzug gehalten haben. Sie stärken das Bewusstsein über die Teammitglieder und deren Arbeit und erzeugen so eine virtuelle Nähe. Allerdings werden dabei meist Insellösungen genutzt, die jeweils nur einen Teil der Kollaboration abdecken.

An diesem Punkt setzt das `LuPanKu`-System an, welches die Daten aus diesen einzelnen Teillösungen verknüpft und so die Auswertung von Meta-Informationen ermöglicht. Es werden Verbindungen zwischen den Informationen verschiedener Entwicklungswerkzeuge erfasst und erstellt, was neue Möglichkeiten der Kollaboration bietet, wie z.B. die Suche nach einem geeigneten Experten für ein spezielles Themengebiet oder das Aufzeigen eines sozialen Netzwerks (Panier u. a., 2010, S. 5).

An die Unterstützung des Entwicklers bei der verteilten Teamarbeit knüpft eine weitere Vision des Projekts an: die Schaffung eines mobilen Arbeitsplatzes. Dieser erlaubt es dem Entwickler, von verschiedenen Standorten (Büro, zu Hause, unterwegs)aus aktiv an der Projektarbeit teilzunehmen und in den Projektkontext eingebunden zu sein.

Ein ebenfalls wichtiger Aspekt des *Home Office 2.0*-Projekts ist die Unterstützung des Wissensmanagements in Unternehmen. Hierbei gilt es das Know-how der einzelnen Wissensarbeiter strukturiert zu verwalten. Dies ermöglicht einen besseren Zugang zu dem gesammelten Unternehmenswissen und macht die vorhandenen Kompetenzen sichtbar.

4.2.1. Architektur

Als grundlegendes Architektur-Muster für das `LuPanKu`-System wurde Online Analytical Processing (OLAP) gewählt (Codd u. a., 1993, S. 5). Dabei werden die Rohdaten aus den angebunden Fremdsystemen (z.B. Wikis, Versionsverwaltungen, Ticketsystemen) geladen und die Analysen dieser Daten auf den `LuPanKu`-Server ausgelagert. Dadurch bleiben die Fremdsysteme unbelastet von aufwändigen Analysen und ihr alltäglicher Gebrauch wird nicht beeinflusst.

Die schematische Darstellung der Systemarchitektur des `LuPanKu`-Systems (siehe Abb. 4.1) gliedert sich in drei Funktionsbereiche:

- Die **Extraction** ist verantwortlich für die Aquirierung der Rohdaten aus den Fremdsystemen. Diese sammelt der Collector-Service, der manuell oder automatisch

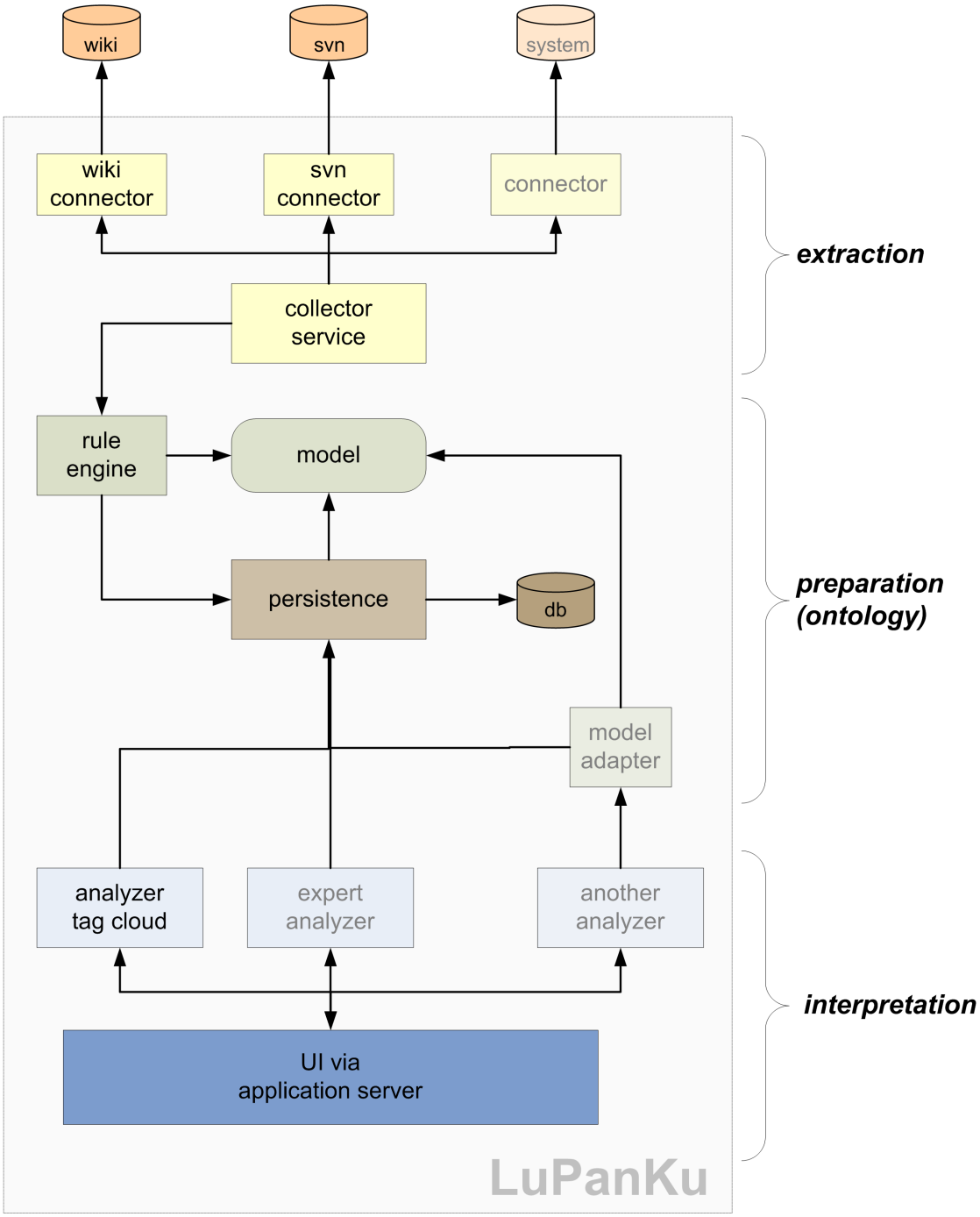


Abbildung 4.1.: Schematische Darstellung der Systemarchitektur des `LuPanKu`-Systems (Panier u. a., 2010, S. 20)

angestoßen werden kann, anhand der verschiedenen Implementationen der Connector-Schnittstelle (Panier u. a., 2010, S. 21).

- Der Bereich der **Preparation** ist für die Formatierung und Persistierung der Rohdaten zuständig. Die Rule-Engine formatiert diese zunächst anhand des `LuPanKu`-Datenmodells (siehe Kapitel 4.2.2). Dabei können für die verschiedenen Connectoren jeweils unterschiedliche Regeln definiert werden, die festlegen, wie die von den Fremdsystemen erhaltenen Rohdaten auf das `LuPanKu`-Datenmodell gemappt werden. Anschließend werden die formatierten Daten durch das Persistence-Modul dauerhaft in der Datenbank gespeichert (Panier u. a., 2010, S. 21).
- Bei der **Interpretation** handelt es sich um Module, die aus den persistierten Daten Informationen generieren. Die so genannten „Analyzer“ untersuchen die Daten im Hinblick auf eine bestimmte Fragestellungen und geben die ermittelten Informationen an eine Benutzeroberfläche weiter (z.B. einen Webbrowser oder eine Entwicklungsumgebung) (Panier u. a., 2010, S. 22).

Die Architektur des `LuPanKu`-Systems wurde möglichst modular gestaltet, um eine individuelle Anpassung des Systems an die Bedürfnisse verschiedener Unternehmen zu ermöglichen. So können beispielsweise problemlos Connectoren oder Analyzer jeweils unternehmensspezifisch in das System eingebunden werden, und zwar je nachdem welche Werkzeuge bei der Softwareentwicklung verwendet werden. Aus diesem Grund wurde eine Implementation der „Open Services Gateway initiative (OSGi)“-Spezifikation als Framework (siehe Kapitel 4.4.1) für das System gewählt, da es nicht nur einen modularen Aufbau aus einzelnen Komponenten mit klar definierten Schnittstellen bietet, sondern auch eine Lebenszyklus-Verwaltung, die es ermöglicht, Module zur Laufzeit des Systems hinzuzufügen, zu starten oder zu entfernen. Dieser Aspekt ist hinsichtlich eines dauerhaften Betriebs in einem Unternehmen von besonderem Interesse.

4.2.2. Datenmodell

Das Datenmodell ist Teil der `LuPanKu`-Kernanwendung (siehe Abbildung 4.1) und stellt eine Struktur für die Abbildung der gesammelten Daten zur Verfügung.

Die Aufgabe dieser Komponente ist die Bereitstellung der gesammelten Daten in einer allgemein gültigen, systemunabhängigen und verabredeten Form. Sie soll die Daten

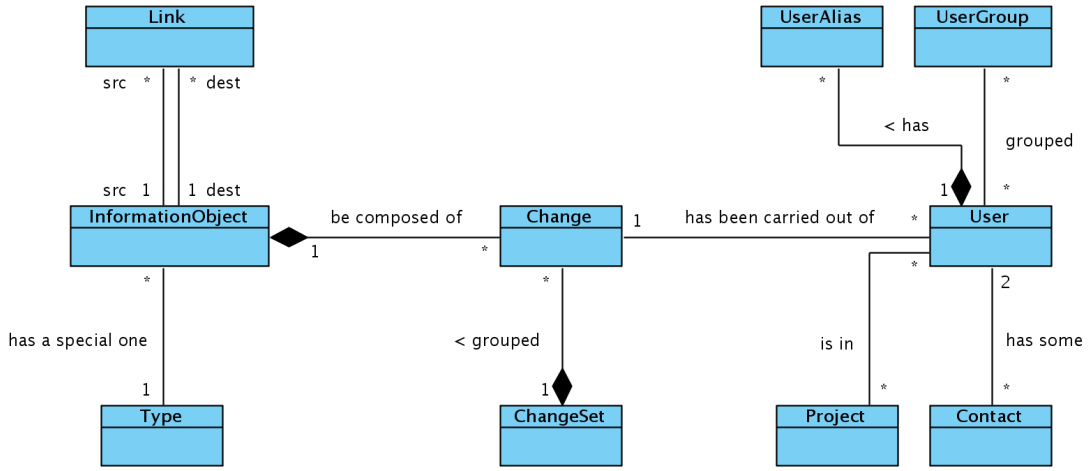


Abbildung 4.2.: UML-Klassendiagramm des verwendeten Datenmodells (Panier u. a., 2010, S. 22)

von ihren Ursprungssystemen entkoppeln und für zukünftige Analysen und Betrachtungen homogen aus der `LuPanKu`-Datenbank offerieren (ein UML-Klassendiagramm des *Models* ist in Abbildung 4.2 zu finden). Das heißt, dass alle Rohdaten, die durch den *Collector Service* gesammelt werden, mit Hilfe der *Rule Engine* auf dieses Modell abgebildet werden können. Dabei ist es unabhängig, aus welchem System diese stammen.

Eine zentrale Entität des Datenmodells ist das *Information-Object*. Dieses repräsentiert eine universelle Information, beispielsweise eine Wiki-Seite, ein Ticket einer Aufgabenverwaltung oder eine Resource, die von einer Versionsverwaltung administriert wird. In diesem *Information-Object* sind allerdings nur die Metadaten zur eindeutigen Identifizierung eines Objekts enthalten, wie die URL zu einer Wiki-Seite oder zu einem Ticket. Die Inhalte des *Information-Object* und der damit verbundene Entwicklungsverlauf sind in den so genannten *Change*s enthalten, die mit dem Objekt verbundenen sind (siehe Abbildung 4.2).

4.2.3. Einordnung Discourse in LuPanKu-Projekt

Die Bereiche Extraction und Preparation (siehe Abb. 4.1) des LuPanKu-Systems wurden bereits während des Masterstudiums in gemeinschaftlicher Arbeit mit dem *Home Office 2.0*-Projektteam realisiert. Dabei wurden Connectoren für die meistgenutzten Entwicklerwerkzeuge implementiert (The Eclipse Foundation, 2010, S. 15-17). Dazu gehören Subversion (SVN) (The Apache Software Foundation, 2011) als Versionsverwaltung, Confluence (Atlassian Pty Ltd., 2011a) als Wiki-Software und JIRA (Atlassian Pty Ltd., 2011b) sowie Trac (Edgewall Software, 2011) als Projekt- und Aufgabenmanagement-Systeme.

Die Zielsetzung der vorliegenden Arbeit fällt nun in den Bereich der Interpretation. Dabei handelt es sich um die Implementierung

- eines Analyzers, der aus den Daten der LuPanKu-Datenbank die Discourse-Informationen generiert,
- eines PlugIns, welches dem Entwickler die Informationen in der Entwicklungsumgebung präsentiert,
- und der Kommunikation der beiden Module miteinander.

Die Einordnung des Teilprojekts Discourse in das LuPanKu-Projekt ist in Abbildung 4.3, einer vereinfachten Darstellung der LuPanKu-Architektur, anhand der roten Markierung ersichtlich.

4.3. Architektur Discourse

Wie bereits beschrieben sind die Aufgaben des Discourse-Tools die Analyse der Daten aus den Entwicklerwerkzeugen und die Darstellung der generierten Informationen in der IDE. Anhand dieser Anforderungen können die zwei Komponenten Analyzer und PlugIn in der Entwicklungsumgebung identifiziert werden (siehe Abbildung 4.4).

Beide werden auf jeweils unterschiedlichen Systemen ausgeführt und die Kommunikation über das Internet realisiert. Daher wird eine lose Kopplung dieser Komponenten benötigt, damit einerseits eine voneinander unabhängige Ausführung möglich ist, die volle Funktionalität des Tools jedoch erst bei einer bestehenden Verbindung genutzt

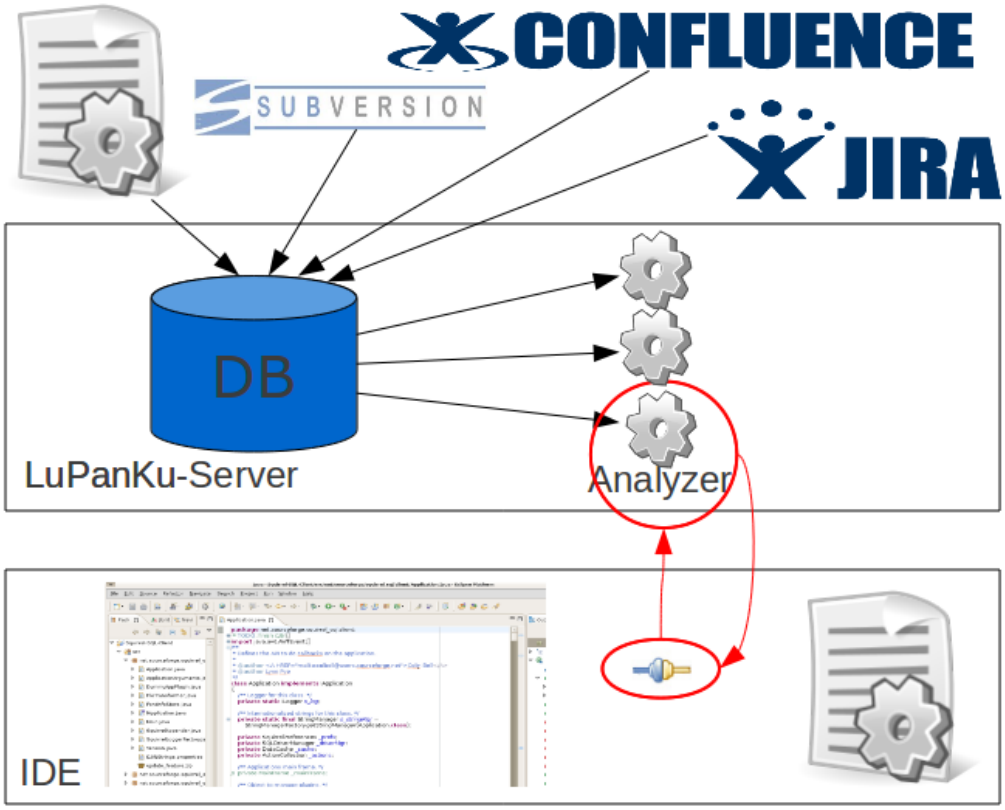


Abbildung 4.3.: Einordnung des Teilprojekts Discourse in das `LuPanKu`-Projekt

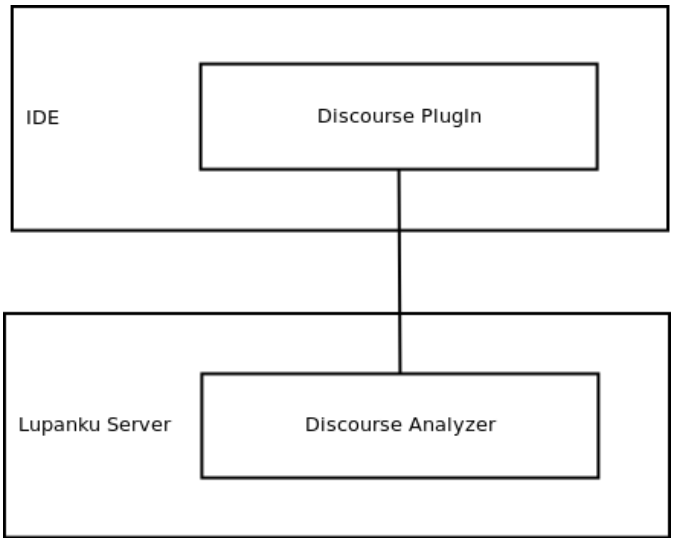


Abbildung 4.4.: Komponenten Discourse

werden kann. Entsprechend werden sowohl der Analyzer als auch das PlugIn in der Entwicklungsumgebung als voneinander unabhängige Module realisiert, um unabhängige Lebenszyklen für beide zu ermöglichen und n-m-Beziehungen der Komponenten zueinander zu gewährleisten. Das heißt, dass sich ein Analyzer mit mehreren Entwicklungsumgebungen wie auch eine Entwicklungsumgebung sich mit mehreren Analyzern verbinden kann (siehe Abbildung 4.3). Dieser Aspekt ist vor allem in Hinblick auf die verschiedenen Analyzer, die im Verlauf des Projekts *Home Office 2.0* noch entworfen werden, entscheidend. Für das Discourse-Tool bietet sich daher eine Service-orientierte Architektur (SOA) an.

Aufgrund ihrer zentralen Bedeutung wurde auch die Kommunikation als Komponente identifiziert. Da sowohl der Analyzer als auch das IDE-PlugIn nicht eigenständig laufende Anwendungen sind, bedarf es in dieser Arbeit auch einer gesonderten Betrachtung des Frameworks, in dem die Komponenten ausgeführt werden.

Aus der vorgestellten Architektur ergeben sich demnach die folgenden zu betrachtenden Komponenten:

Discourse-Analyzer

– Der Analyzer greift über die `LuPanKu`-Datenbank auf die Rohdaten der Entwicklertools zu

und generiert daraus eine Liste der Discourse-Informationen, die für die jeweilige Ressource relevant sind.

PlugIn für Entwicklungsumgebung – Die Ergebnisse der Discourse-Analyse werden dem Entwickler in der Entwicklungsumgebung zur Verfügung gestellt. Dabei sind die besonderen Anforderungen an Darstellungen in Entwicklungsumgebungen zu beachten (siehe Kap. 3.5).

Framework – Der Analyzer wird als Teil des `LuPanKu`-Systems und die Darstellung als Teil der Entwicklungsumgebung realisiert. Daher werden die Frameworks der beiden Komponenten betrachtet.

Kommunikation – Eine sichere und stabile Kommunikation der beiden Komponenten in verschiedenen Frameworks über das Internet ist Voraussetzung für die Funktionalität des Discourse-Tools.

Die einzelnen Komponenten werden in den folgenden Kapiteln detailliert erläutert.

4.3.1. Service-orientierte Architekturen

Als Service-orientierte Architektur (SOA) wird eine konzeptuelle, abstrakte Software-Architektur bezeichnet, die eine Infrastruktur vorsieht, in der Module ihre Schnittstellen anhand von Diensten (auch Services genannt) in einer Service-Registrierung über ein Netzwerk bekannt geben. Andere angeschlossene Module verwenden diese Registrierung, um die Dienste aufzufinden und anzufordern (Schmatz, 2007, S. 2).

SOA kann aus zwei verschiedenen Blickwinkeln betrachtet werden:

- Aus Sicht des **Business** stellt SOA eine Struktur dar, in der die Aktivitäten eines Geschäftsprozesses auf einzelne Dienste abgebildet werden und so eine hohe Wiederverwendbarkeit der einzelnen Aktivitäten (z.B. Kreditwürdigkeit prüfen) gewährleistet ist (Weber u. a., 2010, S. 23).

- Aus der Perspektive der **Softwareentwicklung** erweitert die Serviceorientierung das Komponentenmodell. Die Betrachtung der Komponenten als Services ermöglicht eine weitere Abstraktion von der zugrundeliegenden Implementierung der Komponenten und definiert mit dem Dienst einen klaren Interaktionspunkt mit der Komponente (Weber u. a., 2010, S. 24).

Für die vorliegende Arbeit ist die Sicht der Softwareentwicklung relevant.

Die einzelnen Module können sich auch auf verschiedenen Systemen befinden, die über das Internet miteinander verbunden sind. Durch eine „Remote Services“-Schnittstelle nach außen, die von der Service-Registrierung bereitgestellt wird, können auch Dienste auf entfernten Systemen aufgefunden und angefordert werden.

Diese Trennung von Diensten und einer verwaltenden Instanz, der Service-Registrierung, hilft dabei Programmier-Paradigmen, wie die Modularität, Redundanzfreiheit und Wiederverwendbarkeit, (Mall, 2004, S. 136) zu fördern und einzuhalten.

Für einen tiefer gehenden Einblick in SOA verweist der Autor etwa auf Melzer (2010), Dunkel u. a. (2008) oder Erl (2005).

4.4. Framework

Die Komponenten Analyzer und IDE-PlugIn werden, wie beschrieben, beide nicht als eigenständige Anwendungen, sondern als PlugIns für verschiedene Frameworks realisiert. Ersterer wird dabei in das Framework des `LuPanKu`-Servers und letztgenannter in das einer Entwicklungsumgebung eingebunden.

Beide Frameworks müssen auf einer Service-orientierten Architektur (SOA, siehe Kapitel 4.3.1) basieren, um anhand von „Remote Services“ eine Kommunikation der Module über das Internet gewährleisten zu können.

Bei der Entwicklung des *Home Office 2.0*-Projekts wurden diese sowie die folgenden Anforderungen an das Framework bereits beachtet (siehe dazu Panier u. a. (2010, S. 29)):

Modularität
– Die einzelnen Module des Frameworks müssen gekapselt und voneinander isoliert sein.

Dabei ist zu beachten, dass die Sichtbarkeit und der Zugriff individuell modellierbar sind.

Services mit klaren Schnittstellen – Jedes Modul kann eigene Dienste zur Verfügung stellen und anderen Modulen durch eine Service-Registrierung zugänglich machen. Dadurch werden ihre Verbindungen zueinander klar definiert. Diese Services müssen auch über die Grenzen des lokalen Systems hinaus auffindbar sein („Remote Services“).

Dynamische Komponenten – Die einzelnen Module müssen unabhängige Lebenszyklen besitzen, damit nicht jedes Mal, wenn ein Modul gestartet, gestoppt oder upgedatet wird, das gesamte System neu gestartet werden muss.

Im Rahmen des *Home Office 2.0*-Projekts wurden verschiedene Frameworks analysiert und anschließend die Spezifikation der Open Services Gateway initiative (OSGi) gewählt, da sie alle oben beschriebenen Anforderungen erfüllt.

Für diese Spezifikation existieren verschiedene Open-Source-Implementationen, wie zum Beispiel Equinox vom Eclipse Projekt (The Eclipse Foundation, 2011c), Felix von der Apache Foundation (Apache Software Foundation, 2011) oder Knopperfish (OSGi Alliance, 2011b). Die Wahl im Rahmen des *Home Office 2.0*-Projekts fiel letztendlich auf die Implementation Equinox, da schon bei der Planung mögliche Verbindungen zu der Entwicklungsumgebung Eclipse vorhersehbar waren. Zudem war darüber bereits Wissen im Team vorhanden und durch die große Eclipse-Community ein guter Support gewährleistet (Panier u. a., 2010, S. 30).

4.4.1. OSGi - Open Services Gateway initiative

Bei der OSGi-Alliance handelt es sich um ein weltweites Konsortium von Entwicklern, die an einer offenen Spezifikation zur Erstellung eines modularen, vernetzten Softwaresystems arbeiten (OSGi Alliance, 2011a). Das Ziel ist eine weite Verbreitung des OSGi-Frameworks bei Middleware-Systemen und so die Sicherung der Interoperabiltät verschiedener Applikationen und Services durch die Spezifikation.

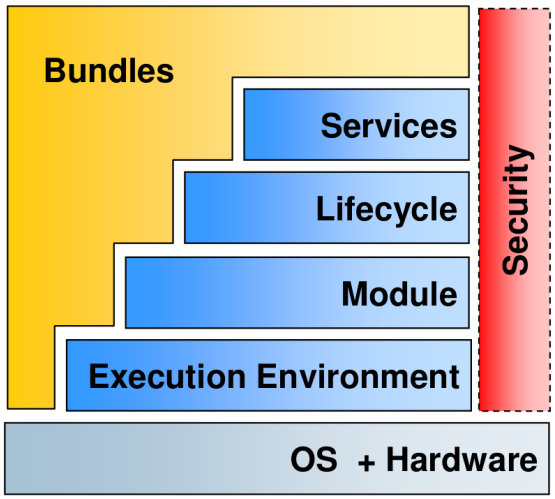


Abbildung 4.5.: OSGi Framework (Watson und Kaegi, 2007)

Ein OSGi-Framework stellt auf einer Java Virtual Machine (JVM) eine standardisierte Umgebung zur Verfügung auf der einzelne, voneinander isolierte Module dynamisch verwaltet werden können. Diese so genannten *Bundles* können zur Laufzeit einer solchen OSGi-Implementation (oder auch *Container*) installiert, aktualisiert, gestartet und gestoppt werden. Dadurch bedarf das System bei einem Update eines Moduls keines Neustarts (Ebert, 2011, Kap. 8).

Das Framework ist in mehrere Schichten aufgeteilt (siehe Abbildung 4.5):

Execution Environment – Bei dem **Execution Environment** handelt es sich um die Java Virtual Machine, auf der das OSGi-Framework ausgeführt wird.

Module Schicht – Die **Module Schicht** verwaltet die Abhängigkeiten und Sichtbarkeit der *Bundles*. Dies wird durch je einen eigenen Classloader pro Modul erreicht (Weber u. a., 2010, S. 6). Damit ein Modul Klassen von anderen verwenden kann, muss es diese explizit anfordern. Das OSGi-Framework stellt dann sicher, dass ein Modul erst gestartet werden kann, wenn alle Abhängigkeiten erfüllt sind. So wird

das Risiko einer ClassNotFound-Exception zur Laufzeit verringert.

Dabei unterstützt das Framework die Versionierung von *Bundles*, so dass auch mehrere Versionen eines *Bundles* gleichzeitig in einem Framework aktiv sein können.

Lifecycle Schicht – Die **Lifecycle Schicht** ist für die oben beschriebene dynamische Verwaltung der *Bundles* verantwortlich.

Service Schicht – Die **Service Schicht** verwaltet die Schnittstellen der Module. Dazu wird eine Service-Registrierung verwendet, in der die einzelnen Module Services anmelden, aufspüren und sich ihnen verschreiben können (OSGi Alliance, 2011d). Auch diese Verwaltung ist dynamisch und ermöglicht das Hinzufügen oder Entfernen von Services zur Laufzeit. Die Service-Registrierung verwaltet auch die „Remote Services“, über die auch Module in entfernten Systemen erreicht werden können (dazu mehr in Kapitel 4.5).

Security – Das OSGi-Framework erweitert die in Java bereits vorhandenen Sicherheitsmechanismen so, dass die Berechtigungen von Bundles zur Laufzeit beschränkt werden können (OSGi Alliance, 2011d).

(Watson und Kaegi, 2007)

Die *Bundles* stellen die zentralen Komponenten eines OSGi-Frameworks dar. Sie können etwa mit JAR-Dateien verglichen werden, in der die Java Klassen und eingebundenen Ressourcen gekapselt sind. Gemäß der Spezifikation von OSGi muss jedes *Bundle* eine *Manifest*-Datei enthalten, in der die Abhängigkeiten zu anderen *Bundles* festgehalten sind. Dazu gehören zum einen solche, die für die Ausführung der Komponente benötigt werden und zum anderen Java-Pakete, die diese Komponente exportiert, also anderen *Bundles* zur Verfügung stellt (Ebert, 2011, Kap. 8). Anhand der *Manifest*-Datei werden die Dienste der *Bundles* bei der Service-Registrierung vermerkt (OSGi Alliance, 2011d).

Die dynamische Verwaltung der Services der Module durch die Service-Registrierung in der JVM ermöglicht eine lose Kopplung der Module mit klaren Schnittstellen. Zusätz-

lich bietet das OSGi-Framework selbst einige Standard-Dienste, z.B. für Logging, Http, Konfiguration etc (Ebert, 2011, Kap. 8).

Die für die vorliegende Arbeit relevante Implementierung der OSGi-Core-Spezifikation ist Equinox vom Eclipse-Projekt (The Eclipse Foundation, 2011c), da sie sowohl beim `LuPanKu`-Server Anwendung findet als auch das Grundgerüst für die Entwicklungsumgebung Eclipse bildet. Sie basiert auf dem vierten Release der Spezifikation.

4.4.2. Equinox

Das Equinox Projekt wurde 2003 ins Leben gerufen, als die Eclipse-Entwicklungsumgebung auf ein PlugIn-System umgerüstet werden sollte (The Eclipse Foundation, 2011c). Damals wurde entschieden, die OSGi-Spezifikation zu verwenden, wobei die Equinox Version des aktuellen Indigo Release der IDE (Juni 2011) bereits auf der OSGi-Spezifikation 4.3 (OSGi Alliance, 2011c) basiert.

Equinox ist zudem derzeit die populärste OSGi-Implementierung, was sie vor allem der weiten Verbreitung der Eclipse-Entwicklungsumgebung zu verdanken hat. Sie basiert nicht nur selbst auf dieser OSGi-Implementierung, sondern stellt auch einige Werkzeuge zur einfachen Entwicklung zusätzlicher Bundles für Equinox zur Verfügung (Weber u. a., 2010, S. 33).

4.5. Kommunikation

Für die Übertragung der Informationen des Discourse-Analyzer zum Discourse-PlugIn wird eine Kommunikation zwischen der Eclipse-IDE und dem `LuPanKu`-Server benötigt.

Wie in 4.3 beschrieben basiert das Discourse-Tool auf einer Service-orientierten Architektur. Daher wird die Kommunikation über eine zentrale Instanz (Service Registry) realisiert, mittels derer die zwei Komponenten ihre Schnittstellen veröffentlichen und sich so gegenseitig aufrufen können. Da beide Module in ein Framework der OSGi-Spezifikation eingebunden werden, ist es sinnvoll zu prüfen, welche Möglichkeiten diese hinsichtlich entfernter Kommunikation bietet. In diesem Zusammenhang hat sich das *Distributed OSGi* positiv hervorgehoben (Weber u. a., 2010, S. 20).

4.5.1. Distributed OSGi

Das OSGi-Framework bietet, wie in Kapitel 4.4.1 dargelegt wurde, eine Serviceschicht, um verschiedene Module in einem Framework mit geringer Kopplung zu verbinden. Dabei können die einzelnen Bundles über Services miteinander kommunizieren. Das Framework stellt dazu eine Service-Registrierung zur Verfügung, um eigene Dienste anzumelden, die anderer aufzufinden und diese anzufordern. Bei diesen Services handelt es sich um Implementationen eines Java Interfaces, die lokal in der JVM ausgeführt und über Referenzen angesprochen werden (OSGi Alliance, 2011d).

Der OSGi-Spezifikation 4.2 (OSGi Alliance, 2011c) wurde im Kapitel 13 das Konzept der *Remote Services* hinzugefügt, welches die Kommunikation mit Modulen außerhalb der lokalen JVM ermöglicht. Dabei wird die bereits bekannte OSGi-Services Technologie verwendet, um entfernte Kommunikation zwischen Frameworks bereitzustellen. So werden lokale und entfernte Dienste in der Serviceschicht des OSGi-Frameworks analog behandelt und in der Service-Registrierung angemeldet. Das Framework ist dann verantwortlich für die Herstellung der Kommunikation und die Übertragung. Da somit die Transportschicht für die Bundles durchsichtig wird, garantieren die Remote-Services vollständige Transparenz bezüglich der Lokation eines Services (Weber u. a., 2010, S. 20). Um einen Dienst außerhalb der JVM bekannt zu machen und als Remote Service zu verwenden, sind entsprechende Export-Eigenschaften in der Manifest-Datei zu setzen (sie Kapitel 4.4.1), die in der Spezifikation definiert sind. Dabei können verschiedene Transport-Technologien (wie RMI oder SOAP) festgelegt werden, was eine Anbindung auch anderer als OSGI-Systeme erlaubt (OSGi Alliance, 2011c).

Die OSGi-Remote-Services ermöglichen demnach die Entwicklung eines modularen Komponentenmodells, dessen Kommunikation auf lokalen und verteilten Services basiert. Da die Kommunikation durch das OSGi-Framework geregelt wird und die konkrete Übertragungstechnologie weggekapselt ist, kann sie problemlos ausgetauscht werden (Weber u. a., 2010, S. 21).

4.5.2. Eclipse Communication Framework

Die Funktionalität der Remote Services wird durch zusätzliche Komponenten im OSGi-Framework zur Verfügung gestellt. Auf Basis der gewählten OSGi-Implementation Equinox existieren verschiedene Projekte, um diese Komponenten zu realisieren. Die po-

pulärsten sind Rienna (The Eclipse Foundation, 2011d) und Eclipse Communication Framework (ECF) (The Eclipse Foundation, 2011b). Im Rahmen des *Home Office 2.0*-Projekts wurde das ECF-Projekt gewählt, da es eine höhere Flexibiltät für die Remote-Services bietet und zusätzlich die für andere Teilprojekte des *Home Office 2.0*-Projekts relevante Chat-Funktionalität unterstützt (Panier u. a., 2010, S. 15).

Das ECF-Projekt erlaubt die Entwicklung von verteilten Anwendungen, wobei sowohl Client-Server- als auch Peer-to-Peer-Architekturen unterstützt werden (ECF Team, 2010). Der Fokus des ECF-Projekts liegt dabei auf Werkzeugen zur Kommunikation und Kollaboration, wie Chat oder geteilte Editoren (ECF Team, 2011a).

Das ECF verwendet das Prinzip der Kommunikations-Container. Dabei handelt es sich um Bundles, die die Kommunikation nach außen anhand von Proxies gewährleisten und von den lokalen Bundles anhand der Service-Schnittstellen in der Service-Registrierung aufgefunden werden können. Für jede Verbindung zu einem anderen OSGi-Framework wird ein Proxy-Bundle durch das ECF erstellt. Über dieses werden die Service-Schnittstellen in die lokale Service-Registrierung eingetragen und können so von den anderen Bundles aufgefunden werden.

Das ECF stellt zur Entwicklung von Remote Services zwei zentrale Schnittstellen zur Verfügung:

Discovery-API – stellt eine protokoll-unabhängige Schnittstelle bereit, um den eigenen Remote-Service zu veröffentlichen, die Services anderer zu finden und sie anzufordern. (ECF Team, 2011b)

Remote-Service-API – bietet eine Schnittstelle, um Remote Services in OSGi-Frameworks unabhängig von der zugrunde liegenden Transporttechnologie zugänglich zu machen. (ECF Team, 2011b)

4.5.3. `LuPanKu` Remote Services

Im *Home Office 2.0*-Projekt wurde in gemeinschaftlicher Arbeit bereits die Grundlage zur Realisierung einer Kommunikation entworfen. Da in den bisher entwickelten Szenarien des *Home Office 2.0*-Projekts `LuPanKu` als zentraler Server verwendet wird (Panier u. a., 2011, S. 6-7), wurde auf einen Discovery-Dienst verzichtet. Somit müssen sich die Klienten mit einem feststehenden Server verbinden und übernehmen die Konfiguration selbst.

Dazu wurde das Bundle *Services API* implementiert (siehe QuellCode 4.1 und 4.2), in dem sich die als Java-Interfaces beschrieben Service Definitionen befinden. Diese Schnittstellenkomponente wird sowohl auf dem Server als auch dem Client installiert.

```
package org.aysada.lupanku.services.api;

public interface IClientListener
{
  public void registerClient(String clientId, String clientServiceURL);
  public void submitContext(Set<String> contextInfos);
  public void unregisterClient(String clientServiceURL);
  public void registerDiscourseClient(String clientServiceURL);
  public void unregisterDiscourseClient(String clientServiceURL);
  public void callDiscourse(HashMap<String, Object> clientMap);
}
```

Quellcode 4.1: IClientListener Schnittstelle

```
package org.aysada.lupanku.services.api;

public interface ICallBackService
{
  void handleContextOfColleague(Set<String> contextInfos);
  void handleDiscourse(HashMap<String, Set<String>> discourseInformation);
}
```

Quellcode 4.2: ICallBackService Schnittstelle

IClientListener – Die Schnittstelle definiert einen Listener, der auf die Aufrufe der Klienten wartet und sich auf dem Server befindet.

ICallBackService – Die ICallBackService-API wird von Klienten implementiert, um sich mit einem Listener verbinden und von diesem aufgerufen werden zu können.

Ferner wurde in gemeinschaftlicher Arbeit das PlugIn `ClientListener` für den `LuPanKu`-Server realisiert, welches das `IClientListener`-Interface implementiert. Der `ClientListener` verwendet die Komponenten des ECF und setzt die Funktionalität der Remote Services um. Beim Starten des Bundles wird ein Remote Service Listener gestartet, der an einem festgelegten Port auf Aufrufe der Klients wartet. Diese können sich über die `IClientListener`-Schnittstelle mit dem Server verbinden. Für jeden beim Server angemeldeten Klienten wird eine Instanz der `ICallBackService`-Schnittstelle auf dem Server geführt, über die die Methoden der Klienten aufgerufen werden können.

Ein entwickelter Klient muss zum einen das `ICallBackService`-Interface implementieren, um vom Server aufgerufen werden zu können, und zum anderen eine Instanz des `IClientListener`-Interfaces führen. Diese verweist auf den verbundenen Server und ermöglicht die Aufrufe der auf ihm befindlichen Methoden.

4.6. Integration in Entwicklungsumgebung

Die Discourse-Informationen sollen für den Entwickler bequem aus der Entwicklungsumgebung zugreifbar sein, um einen Programm- oder Fensterwechsel während der Arbeitsaufgabe zu vermeiden. Daher wird ein PlugIn erarbeitet, das in die Entwicklungsumgebung eingebunden werden kann und auf die GUI der IDE zugreift, um die Informationen anzuzeigen. In diesem Zusammenhang bedarf es daher einer Entwicklungsumgebung, die die Erweiterung der eigenen Funktionalität und der GUI ermöglicht.

Bei der Darstellung der Informationen in der Entwicklungsumgebung ist zum einen zu beachten, dass diese bedarfsorientiert erfolgt und zum anderen, dass der Aufwand für die Aufnahme der *Peripheral Awareness*-Informationen so gering wie möglich gehalten wird (siehe Kapitel 3.5).

Als Entwicklungsumgebung, in die das Tool integriert werden soll, wurde Eclipse gewählt (The Eclipse Foundation, 2011a), da es sich um die wohl meistgenutzte IDE handelt und entsprechend auch von allen Teammitgliedern des `LuPanKu`-Projekts verwendet wird. Zudem stellt sie ein offenes und plattform-unabhängiges System dar, das einfach erweiterbar ist und auf der OSGi-Implementation Equinox basiert.

4.6.1. Eclipse-Entwicklungsumgebung

Bei Eclipse handelt es sich um ein Open-Source-Projekt der unabhängigen, gemeinnützigen Organisation „The Eclipse Foundation“, die als Schirmherr der Eclipse-Community agiert, welche sich wiederum aus Programmierern unterschiedlicher Softwareindustrien zusammensetzt. Der Fokus der Projekte im Eclipse-Kontext liegt auf der Entwicklung einer kostenfreien, quelloffenen Plattform zur Unterstützung der Softwareentwicklung (The Eclipse Foundation, 2011a). Die wachsende Eclipse-Gemeinschaft

gewährt zum einen einen guten Support und zum anderen die Verwendung der vielzähligen von Mitgliedern erstellten Produkte.

Seit dem dritten Release basiert die Eclipse-IDE auf dem Framework Equinox (siehe Kapitel 4.4.2), das die OSGi-Spezifikation implementiert. Die Eclipse-Anwendungen bestehen also aus OSGi-Bundles und werden auf dem Equinox-Framework ausgeführt. Allerdings werden Bundles im Zusammenhang mit Eclipse häufig als PlugIns bezeichnet (Ebert, 2011, Kap. 8). Damit verfügt die IDE über ein dynamisches PlugIn-System, das es ermöglicht, einzelne PlugIns als eigenständige Einheit zu entwickeln, zu testen und weiterzugeben. Darüber hinaus erlaubt dieses modulare Komponentenmodell das Hinzufügen und Entfernen von PlugIns zur Laufzeit, ohne dass die Anwendung neu gestartet werden muss (Bolour, 2003). Das PlugIn-System gestattet es Entwicklern, die IDE-Funktionalität unkompliziert zu erweitern, beispielsweise um neue Menüs oder Editoren.

Die GUI der Entwicklungsumgebung ist hierarchisch strukturiert, wobei der Workbench die Wurzel bildet. Er bildet den Rahmen für alle Fenster und Dialoge und ist mit einem Workspace verbunden. Dieser wiederum beinhaltet in Projekten strukturiert die Ressourcen (Dateien und Ordner), die mit der Entwicklungsumgebung bearbeitet werden können. Die zentralen Komponenten des Workbench sind die Workbench-Windows (siehe Abbildung 4.6), welche je eine Menüleiste (menu bar), Werkzeugleiste (toolbar), Abkürzungsleiste (shortcut bar), Statusleiste (status line) und eine oder mehrere Perspectives beinhaltet. Bei letztgenannter handelt es sich um eine Zusammenstellung von verschiedenen Views und Editoren. Sie ist dabei abhängig von der Aufgabe, die in der Perspective erledigt werden soll, beispielsweise eine Java-Debug-Perspective. Editoren ermöglichen das Bearbeiten von Ressourcen und können von mehreren Perspectives geteilt werden. Die grafischen Komponenten Views unterstützen die Bearbeitung von Ressourcen durch die Darstellung zusätzlicher Informationen. Views können beispielsweise die Hierarchie von Ressourcen abbilden oder die Eigenschaften des aktiven Editors anzeigen (Thrower u. a., 2011).

Da es sich bei Eclipse um ein Open-Source-Projekt handelt, wurden bereits einige Entwicklungsumgebungen erstellt, die, wenngleich sie nicht den Namen Eclipse tragen, auf dessen Technologie basieren. Dazu gehört neben weiteren *Aptana* von Appcelerator (2011) eine IDE zur Web-Entwicklung.

Bei der Erstellung von Eclipse-PlugIns ist zu beachten, dass die Eclipse-Entwicklungsumgebung zur Darstellung aller grafischer Elemente das Java Standart Widget Toolkit

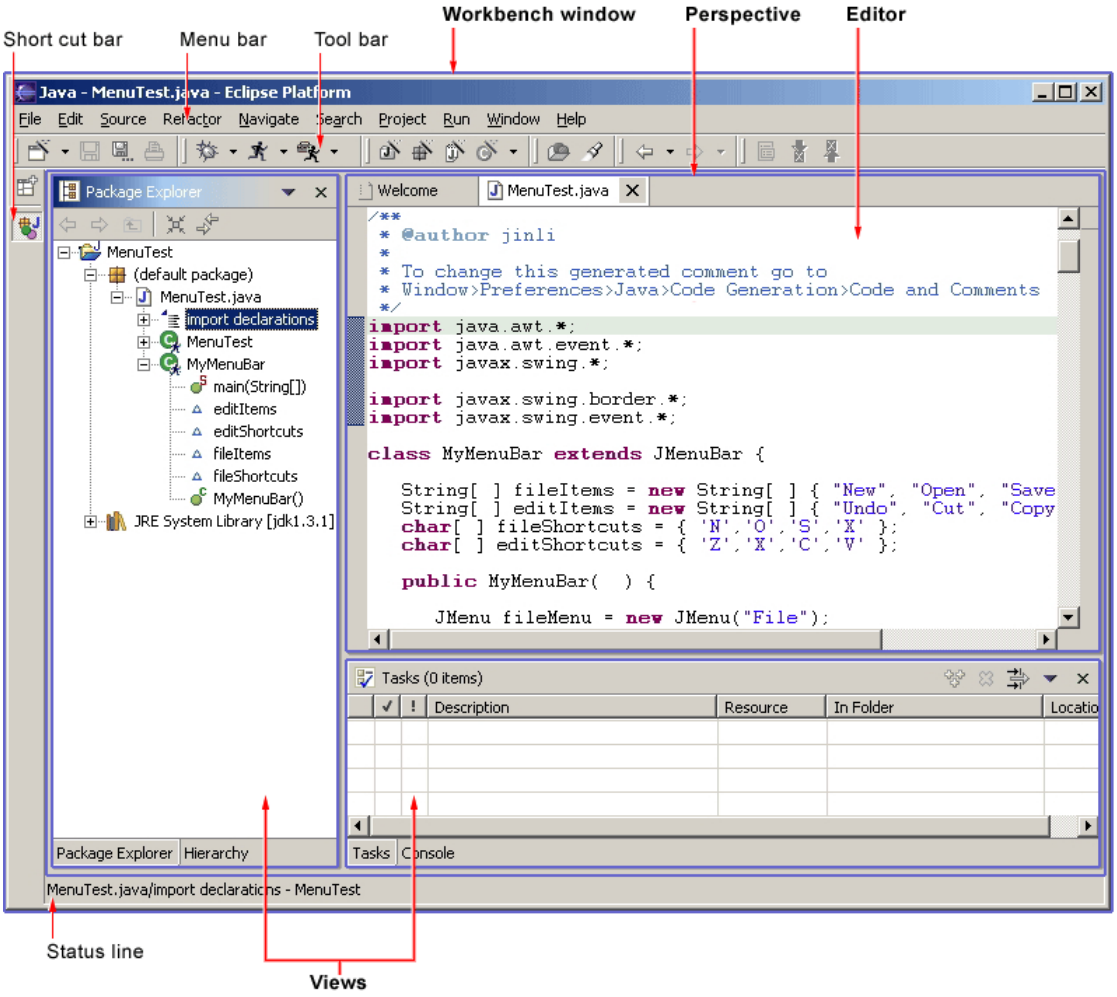


Abbildung 4.6.: Eclipse Workbench Window Struktur (Thrower u. a., 2011)

(SWT)verwendet. Dieses müssen daher auch alle Plug-Ins einsetzen, wenn sie in die GUI der Entwicklungsumgebung integriert werden sollen.

4.6.2. Eclipse-PlugIn

Bei PlugIns handelt es sich um die kleinste funktionale Einheit innerhalb von Eclipse (Bolour, 2003). Sie können die Funktionalität der Entwicklungsumgebung um einzelne Aspekte erweitern. Dabei kann die IDE durch unterschiedliche Komponenten ausgebaut werden. So kann ein PlugIn beispielsweise ein View oder einen Editor in das Workbench Fenster einbinden oder der Entwicklungsumgebung eine weitere Aktion hinzufügen. Da die Eclipse-IDE auf Equinox basiert, können die einzelnen PlugIns über Services verknüpft werden.

Ein PlugIn der Eclipse-IDE wird in einer XML Manifest Datei mit dem Namen „plugin.xml“ beschrieben. Diese muss sich in dem PlugIn-Ordner befinden und gibt der eclipse-IDE Aufschluss darüber, was diese beachten muss, um das PlugIn zu aktivieren und zu starten. Dazu verwendet Eclipse einen PlugIn Management Kernel (Eclipse-Plattform bzw. Eclipse-Runtime) und mehrere Kern-PlugIns, die mit jeder Ausführung von Eclipse gestartet werden. PlugIns, die nicht zu letzteren gehören, werden durch die Eclipse-Runtime erst aktiviert, sobald sie von anderen benötigt werden (Bolour, 2003).

Um trotz der großen Anzahl verschiedener PlugIn-Entwicklern eine intuitive Entwicklungsumgebung zu behalten, wurden einige UI-Guidelines zur Erstellung von PlugIns formuliert (siehe dazu Thrower u. a. (2011)).

Das Discourse Eclipse-PlugIn dient der Darstellung der durch den Discourse Analyzer generierten Informationen und der *Awareness*-Signalfarben in der Eclipse-basierten Entwicklungsumgebung. Nach einer Analyse der möglichen Präsentationen von Informationen in der Eclipse-IDE hat sich die Komponente *View* als passendes Element zur Darstellung der Informationen hervorgehoben, da bei der Discourse-Darstellung keine Bearbeitung von Ressourcen erfolgt, sondern nur zusätzliche Informationen darüber angezeigt werden sollen.

Bei der Entwicklung eines Views müssen besonders die UI-Guidelines ab Punkt 7.5 berücksichtigt werden. Um die sie betreffenden Aspekte zu verdeutlichen, sind die unterschiedlichen Elemente eines View in Abbildung 4.7 dargestellt:

Erscheinung – Ein View besteht aus einer Titelleiste, einer Werkzeugleiste, einem Pulldown-Menü und einer eingebetteten Kontrolle.

Initialisierung – Wenn ein View geöffnet wird - unabhängig ob beim Start der IDE oder erst im Nachhinein - sollte dessen Inhalt vom Status der Perspektive abhängig sein. So erhält beispielsweise der Outline-View Informationen über die aktuell offene und ausgewählte Ressource.

Menü – Das Pulldown-Menü soll keine Aktionen enthalten, die sich auf die Auswahl im View oder einzelne Objekte desselben beziehen, sondern nur solche, die für das View allgemein gelten, wie etwa dessen Darstellung.

Kontextmenü – Das Kontextmenü ist das Gegenstück zum Pulldown-Menü. Die darin befindlichen Aktionen sollen sich auf die Selektion im View, also auf dessen einzelne Objekte, beziehen und nicht für das View allgemein gelten.

Werkzeugleiste – Die Werkzeugleiste wird für die Darstellung der meist genutzten Aktionen des View verwendet. Jede Aktion, die sich auf der Toolbar befindet muss sich auch im Menü, entweder im Pulldown- oder im Kontextmenü, wiederfinden.

Icon – Der View Icon befindet sich in der Titelleiste eines jeden Views und sollte dessen Funktionalität andeuten.

Persistierung – Im Unterschied zu Editoren sollten Veränderungen, die in einem View durchgeführt werden, sofort gespeichert werden. Wenn z.B. eine Datei in dem Navigation-View umbenannt wird, sollte dies auch direkt im Workspace, also im Datei-Verzeichnis, übernommen werden. So kann der Zustand des Views auch über die aktuelle Sitzung hinaus persistiert und beim Start der nächsten Sitzung die Informationen wieder angezeigt werden.

(Thrower u. a., 2011)

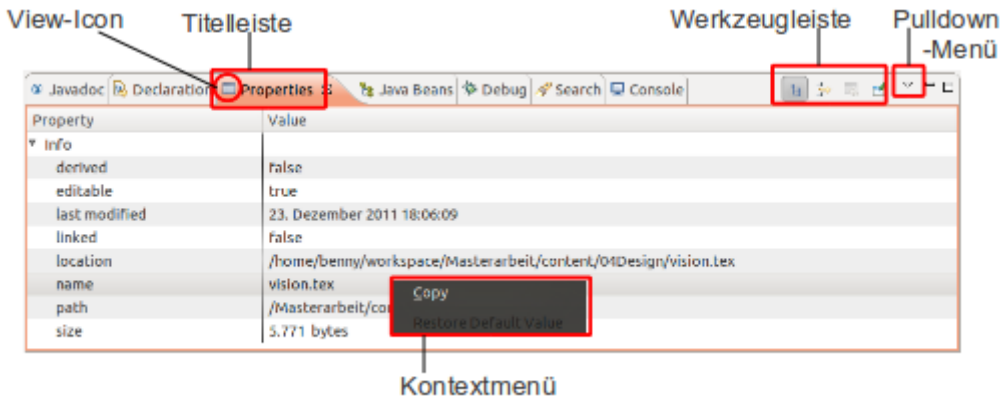


Abbildung 4.7.: Beispiel einer Eclipse-View

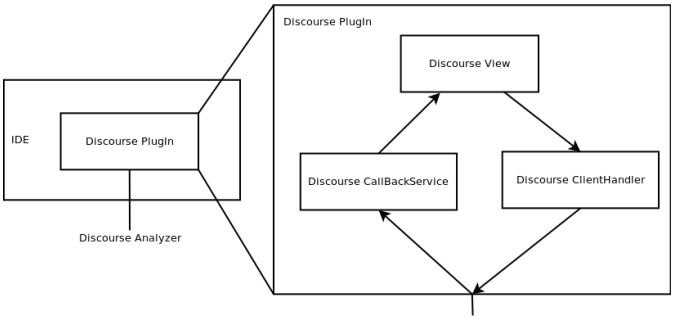


Abbildung 4.8.: Komponenten Discourse-PlugIn

4.6.3. Discourse-PlugIn

Das Discourse-PlugIn, welches im Rahmen dieser Arbeit entwickelt wurde, dient der Darstellung der Discourse-Informationen und der Schaffung einer Awareness über die Entwicklung anderer Mitarbeiter. Die benötigten Informationen dazu erhält es von dem Discourse-Analyzer.

Für das PlugIn können drei Hauptkomponenten identifiziert werden (siehe Abbildung 4.8):

Discourse-View – Dabei handelt es sich um die grafische Komponen-

te, das View, in der Entwicklungsumgebung. Er stellt die vom Analyzer erhaltenen Informationen und den Awareness-Zustand einer Ressource dar.

Discourse-ClientHandler – Der *DiscourseClientHandler* ist für die Anmeldung des Klienten beim `LuPanKu`-Server und die Aufrufe der entfernten Methoden verantwortlich.

Discourse-CallBackService – Der *CallBackService* empfängt die Daten vom Analyzer und gibt sie an das View weiter.

Das PlugIn muss sich, um Informationen zu erhalten, mit dem Discourse-Analyzer verbinden, der auf dem `LuPanKu`-Server aktiv ist. Allerdings soll der Mitarbeiter wählen können, ob er sich in der aktuellen Situation mit dem System verbinden möchte oder nicht. Daher wird die Anmeldung beim Server nicht automatisch vorgenommen, sondern erst durch einen Aufruf der „Connect“-Aktion in der Werkzeugleiste des Views (siehe Abbildung 4.9).

Da die Discourse-Informationen datei-spezifisch sind, hängt deren Darstellung im View von der im Editor aktuell fokussierten Ressource ab. So muss bei jedem Selektionswechsel im Editor auch die Präsentation im Discourse-View angepasst werden. Um nicht bei jedem Wechsel die Daten vom Discourse-Analyzer erneut anfordern zu müssen, führt das View eine Liste mit allen gegenwärtig in der Entwicklungsumgebung geöffneten Ressourcen (*ResourceDiscourse*, siehe Anhang A.1). In dieser ist nicht nur jede Ressource selbst anhand eines *IResource*-Objekts gespeichert, sondern auch die möglicherweise bereits angefragten Discourse-Informationen sowie der aktuelle Awareness-Status der Datei (*AwarenessState*, siehe Anhang A.1).

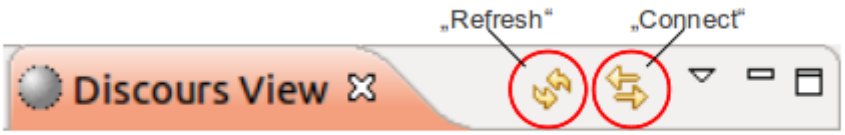


Abbildung 4.9.: Titelleiste & Werkzeugleiste des DiscourseView

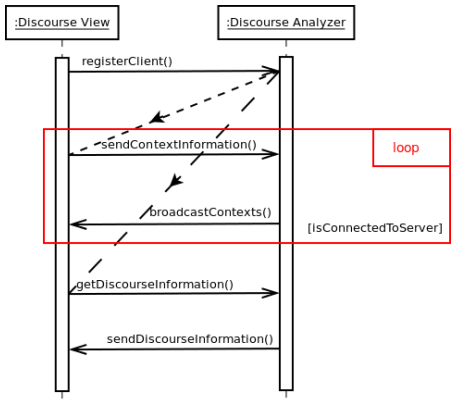


Abbildung 4.10.: Sequenz Diagramm Kommunikation

Awareness

Wenn sich die View erfolgreich mit dem `LuPanKu`-System verbunden hat, wird ein Thread gestartet, der alle fünf Minuten die aktuell in der Entwicklungsumgebung offenen Ressourcen an den Server sendet (siehe Abbildung 4.10). Dabei werden allerdings nur jene Dateien berücksichtigt, die mit einer Versionsverwaltung in Verbindung stehen (siehe Kapitel 3.5).

Durch die bestehende Verbindung empfängt der Entwickler zudem automatisch in regelmäßigen Abständen Informationen über die derzeit geöffneten Dateien der anderen Mitarbeiter. Dazu sendet der `LuPanKu`-Analyzer eine Liste aller aktuell in den angeschlossenen IDEs geöffneten Ressourcen (dazu später mehr in 4.7.1). Wenn sich darin eine in der eigenen Entwicklungsumgebung geöffnete Datei befindet, wird der Entwickler durch das Discourse-View darauf aufmerksam gemacht. Dies geschieht anhand einer Veränderung des Icons des Views (siehe Abbildung 4.9), das durch die Signalfarbe Rot auf eine aktuelle Bearbeitung an der Ressource hinweist. Zudem wird der Awareness-State in der Liste der Ressourcen gespeichert.

Für die Darstellung der Awareness in der Entwicklungsumgebung wurden die Zustände `UNKNOWN`, `BUSY`, `LATELY` und `FREE` festgelegt (siehe Tabelle 4.1).

Die Bedeutung, die dem Icon dadurch zugesprochen wird, widerspricht unglücklicherweise der UI-Guideline bezüglich Icons (siehe Kapitel 4.6.2), welche besagt, dass das Icon eine Auskunft über die Funktionalität des Views geben soll. Es wurde sich dennoch

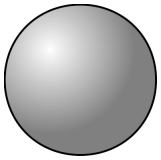
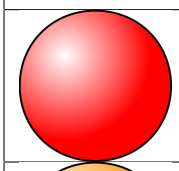
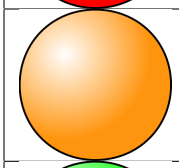
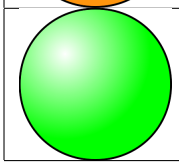
Icon	Konstante	Beschreibung
	UNKNOWN	Es sind keine Informationen über die Entwicklung an der Datei vorhanden. Das kann daran liegen, dass das View noch nicht mit dem LuPanKu-Server verbunden ist oder dass die Datei in keine Versionsverwaltung eingebunden ist.
	BUSY	Die Datei ist aktuell in der Entwicklungsumgebung eines anderen Mitarbeiters geöffnet. Bei der Bearbeitung ist mit Konflikten bei Commits zu rechnen.
	LATELY	Die Datei ist zwar nicht aktuell in einer anderen Entwicklungsumgebung geöffnet, wurde aber in den letzten zwei Tagen bearbeitet.
	FREE	Die Datei wird gerade nicht von einem anderen Entwickler bearbeitet. Es ist also nicht mit Konflikten zu rechnen.

Tabelle 4.1.: Awareness States

für das Icon als Awareness-Darstellung entschieden, da es auch sichtbar ist, wenn das View nicht im Vordergrund oder nur minimiert geöffnet ist (siehe Tabelle 4.2). Erst dieser Umstand ermöglicht das Schaffen einer ununterbrochenen *Peripheral Awareness*.

Discourse-Informationen

Auch die Abfrage von Discourse-Informationen zu einer Datei vom LuPanKu-Server ist möglich, sobald eine Verbindung zum System aufgebaut wurde (siehe Abbildung 4.10). Da dies allerdings nicht unerhebliche Rechen- und Netzwerkleistung erfordert und nicht unbedingt zu jeder Ressource Informationen angefordert werden sollen, wurde beschlossen, die Abfrage der Informationen nicht automatisch sondern manuell durch den Aufruf der „Refresh“-Aktion auszuführen (siehe Abbildung 4.9). Um allerdings durch den Aufruf nicht die ganze Entwicklungsumgebung zu blockieren, wird auch hierfür ein ei-

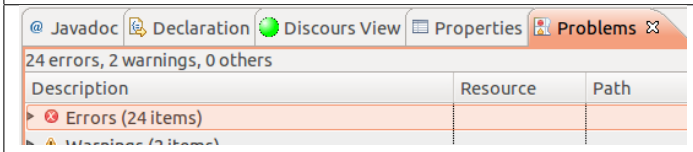
Hintergrund	Minimiert
	

Tabelle 4.2.: Discourse View im Hintergrund und minimiert

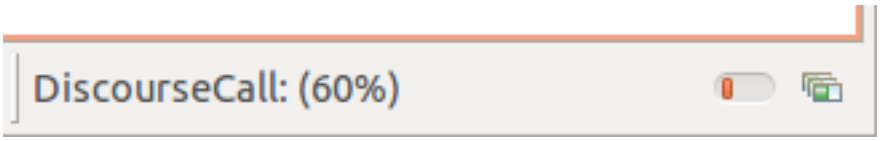


Abbildung 4.11.: Anzeige Fortschritt Discourse-Call in der Statusleiste

gener Thread gestartet und in der Statusleiste der Fortschritt der Abfrage in Prozent angezeigt (siehe Abbildung 4.11).

Bei der Anfrage an den Analyzer wird die Versionsverwaltungs-ID der Ressource als eindeutige Identifizierung verwendet. Der Analyzer sendet nach der Anfrage eine Hash-Map mit den Discourse-Informationen an das View zurück, das diese in der Liste der Ressourcen zu der passenden Ressource abspeichert.

Für die Anzeige der Informationen im Discourse-View wurde die Baumdarstellung gewählt, die diese in Kategorien gliedert und einen übersichtlichen Zugriff gewährt (siehe Abbildung 4.12). Bei der Entwicklung des Discourse-Prototypen wurden folgende Kategorien festgelegt:

Previous Owner – Eine Liste der Mitarbeiter, die die Datei zuvor bearbeitet haben. Anhand dieser können Ansprechpartner zu Entwicklungsprozessen identifiziert werden.

Commit-Messages – Meldungen, die bei den Commits der Ressource formuliert wurden. Sie können eine Auskunft über die Arbeitsaufgaben geben, in deren Zusammenhang die Ressource bearbeitet wurde.

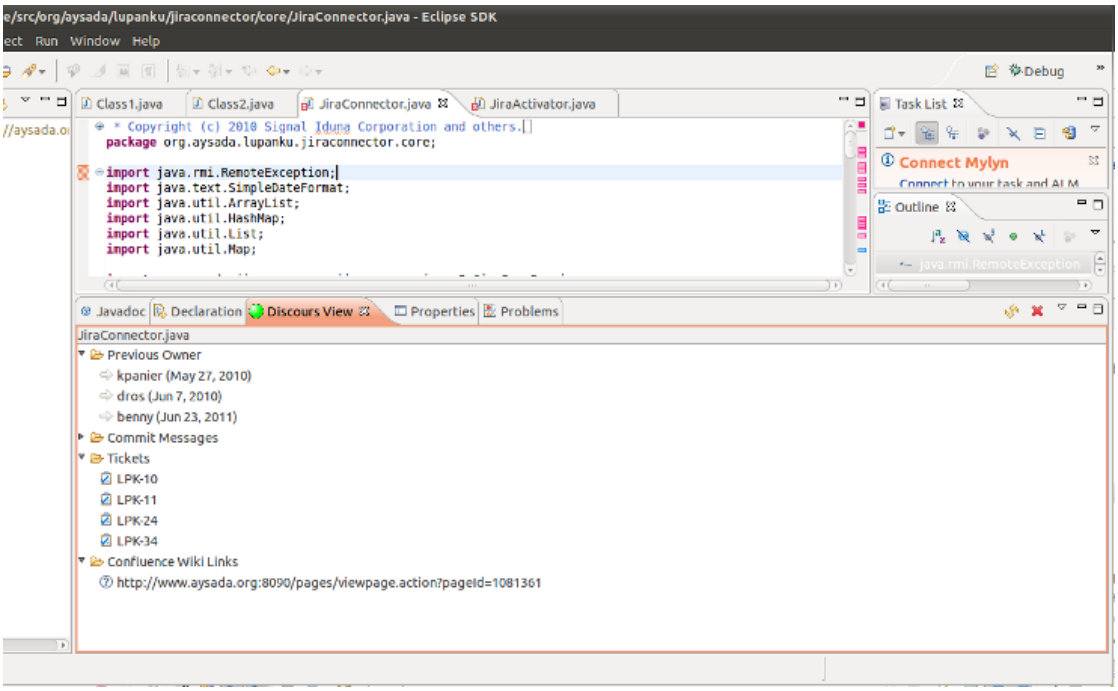


Abbildung 4.12.: Discourse View in Eclipse-IDE mit Discourse-Informationen

Tickets – Eine Liste der Jira-Tickets im Zusammenhang mit der Ressource, die ebenfalls den Arbeitskontext der vorherigen Bearbeitung darlegen können.

Confluence Wiki Links – Eine Übersicht über Wiki-Seiten, die für die bisherige Entwicklung an der Ressource relevant sein könnten. So können mögliche Fragen vielleicht mit bereits dokumentiertem Wissen beantwortet werden, ohne dass dafür ein anderer Mitarbeiter bei seiner aktuellen Aufgabe unterbrochen werden muss.

Klassendiagramm

In der Abbildung 4.13 ist ein vereinfachtes Modell der Klassen des Discourse Eclipse-PlugIns dargestellt. Eine detailliertes Klassendiagramm, das auch Methoden und Attri-

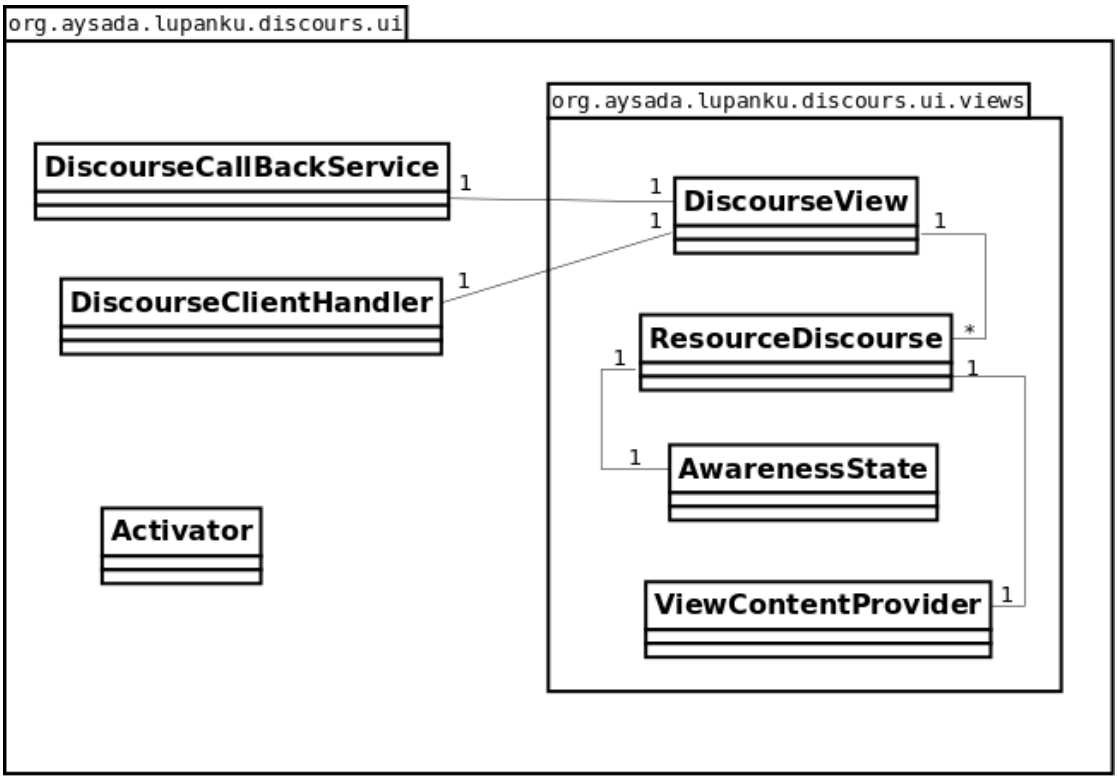


Abbildung 4.13.: vereinfachtes Modell der Klassen des Discourse-PlugIns

bute berücksichtigt, ist im Anhang A.1 zu finden. Da es sich bei dem Paket *org. aysada. lupanku.discourse.ui* um ein PlugIn für das Equinox Framework von Eclipse handelt, beinhaltet es neben den Java-Klassen und Sourcen auch eine *plugin.xml*-Datei zur Verwaltung der Abhängigkeiten des PlugIns (siehe Kapitel 4.6.2) sowie eine *Activator*-Klasse. Diese stellt die Schnittstelle zum Framework dar und wird von diesem zum Starten und Stoppen des PlugIns verwendet.

Im Folgenden wird ein Überblick der erzeugten Klassen gegeben:

DiscourseView
– Diese Klasse erbt von *ViewPart* und ist damit die zentrale Klasse, die die grafische Darstellung des Views realisiert. Sie ist zusätzlich verantwortlich für das Führen einer Liste aller in der IDE geöffneter Ressourcen und sie verarbeitet jegliche Benutzeraktionen.

ResourceDiscourse – Diese Klasse repräsentiert eine offene Ressource in der IDE. Neben der Datei selbst als `IResource`-Instanz verwaltet sie zusätzliche Informationen, wie den aktuellen Awareness Status oder die vom Analyzer gesendeten Discourse-Informationen.

ViewContentProvider – Der *ViewContentProvider* ist verantwotlich für die hierarchische Strukturierung der Discourse-Informationen. Er wird der *DiscourseView* von jedem *ResourceDiscourse*-Objekt individuell bereitgestellt, so dass diese die strukturierten Informationen abbilden kann. Dazu implementiert der *ViewContentProvider* die Interfaces `ITreeContentProvider` und `IStructuredContentProvider` vom `org.eclipse.jface`-Paket.

AwarenessState – Dabei handelt es sich um eine *Enumeration* von Java, die die vier Zustände `UNKNOWN`, `BUSY`, `LATELY` und `FREE` beinhaltet.

DiscourseCallBackService – Der *DiscourseCallBackService* ist eine Implementation der `ICallBackService`-Schnittstelle (siehe Quellcode 4.2) und damit die Verbindung zum `LuPanKu`-Server. Bei der Anmeldung auf dem Server wird eine Referenz auf dieses Objekt auf dem Server hinterlegt, über die er dem Discourse Eclipse-PlugIn Daten senden kann.

DiscourseClientHandler – Der *DiscourseClientHandler* ist verantwortlich für den Verbindungsaufbau zum `LuPanKu`-Server und das Senden der Anfragen zu diesem. Er meldet sich beim Server mit einem *DiscourseCallBackService*-Objekt an und ruft die entfernten Methoden des Servers auf.

4.7. LuPanKu-Analyzer

Bei dem Discourse-Analyzer handelt es sich um einen Analyzer im LuPanKu-System (siehe Abbildung 4.1). Diese fallen in den Funktionsbereich der Interpretation und haben die Aufgabe, die in der LuPanKu-Datenbank gespeicherten Rohdaten anhand einer Fragestellung zu analysieren und daraus Informationen zu generieren. Sie stehen demnach mit dem *Persistence*-Modul des LuPanKu-Servers in Verbindung und rufen über dieses die Rohdaten aus der Datenbank ab.

Der Analyzer läuft als PlugIn des LuPanKu-Systems. Daher wird auch er als OSGi-*Bundle* realisiert (siehe Kapitel 4.4.1). Zudem muss ein Analyzer im LuPanKu-System die IAnalyzer-Schnittstelle des *Home Office 2.0*-Projekts (siehe Quellcode 4.3) implementieren, um als Analyzer vom LuPanKu-System erkannt und dem entsprechend angesprochen werden zu können.

```
public interface Analyzer
{
  public final static String ID = "org.aysada.lupanku.core.Analyzer";

  public void process(HashMap<String, Object> information);
}
```

Quellcode 4.3: IAnalyzer Schnittstelle

Für den LuPanKu-Server wurde bereits in gemeinschaftlicher Arbeit eine Implementation des Interfaces IClientListener realisiert, der ClientListener (siehe Abbildung 4.14). Über diese Remote Service Schnittstelle verbinden sich Klienten mit dem LuPanKu-System(siehe Kapitel 4.5.3). Der ClientListener gibt die Informationen an das Analyzer Bundle anhand der IAnalyzer-Schnittstelle (process(HashMap<String, Object> information)) weiter. Das IAnalyzer-Interface wurde im Hinblick auf verschiedene Typen von Analyzern im Rahmen des *Home Office 2.0*-Projekts, die jeweils verschiedene Anforderungen haben, möglichst allgemein gehalten.

4.7.1. Discourse-Analyzer

Die Fragestellung beim Discourse-Analyzer bezieht sich auf die Entwicklung an einer Ressource:

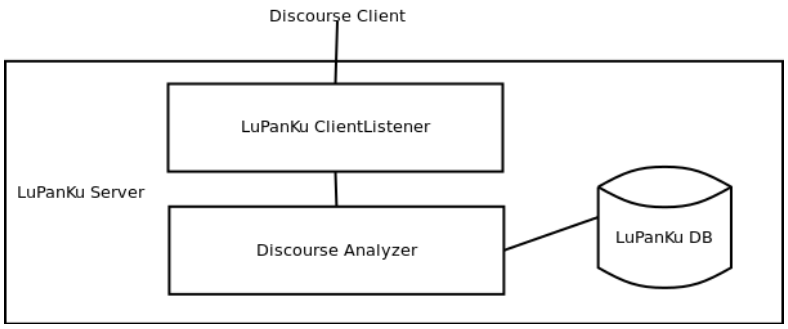


Abbildung 4.14.: Discourse-Analyzer Komponente auf dem `LuPanKu`-Server

Durch welche Werkzeuge wurden, während der bisherigen Entwicklung an einer Ressource, welche Informationen (bzw. welches Wissen) produziert?

Als Vorraussetzung für das Discourse-Tool wurde in Kapitel 3.5 die Verwendung einer Versionsverwaltung im Unternehmen benannt. Da die Discourse-Funktionalität nur für Ressourcen realisiert wird, die mit einer Versionsverwaltung verbunden sind, ist der erste Schritt der Analyse nahe liegend. Anhand der SVN-Sourcen können Informationen über den direkten Entwicklungsverlauf der Datei aus der Datenbank ausgelesen werden. Die größere Herausforderung ist die Erstellung von Verbindungen zu Daten anderer Entwicklungswerkzeuge, die bisher nicht direkt mit der Ressource verbunden sind, wie etwa Aufgaben-Tickets oder Wiki-Seiten. An dieser Stelle gibt es verschiedene Herangehensweisen. Für den Prototypen des Discourse-Analyzers wurde zum einen die einfache Suche nach dem Dateinamen in der `LuPanKu`-Datenbank und zum anderen die Analyse der Commit-Nachrichten anhand von fest definierten Regeln bei Commits gewählt.

Darüber hinaus stellt der Discourse-Analyzer eine Plattform zur Verfügung, die alle angemeldeten Klienten vermerkt und so die erwünschte Awareness über die Aktivitäten anderer Mitarbeiter erzeugen kann.

Beim entwickelten Prototypen des Discourse-Analyzers werden zunächst die Daten eines Subversion Systems, eines JIRA Ticket Systems sowie eines Confluence Wikis analysiert (für Referenzen siehe Kapitel 4.2.3).

Da die `IAnalyzer`-Schnittstelle so allgemein wie möglich gehalten wurde, empfängt der Discourse-Analyzer die Informationen zum Methoden-Aufruf in der *HashMap* der

`process`-Methode (siehe Quellcode 4.3). In dieser sind sowohl die Information darüber enthalten, welche Methode der Discourse-Analyzer ausführen soll, als auch die für ihren Aufruf benötigten Parameter.

Discourse-Informationen

Wie in Kapitel 3.5 beschrieben, wird die URL einer Ressource in einer Versionsverwaltung zur eindeutigen Identifizierung der Datei verwendet. Anhand dieser wird zunächst das `InformationObject` (siehe Kapitel 4.2.2) der SVN-Ressource in der `LuPanKu`-Datenbank ermittelt. Aus dieser können nun, über das `InformationObject`, die zugehörigen `Change`-Objekte, also der Entwicklungsverlauf, abgefragt werden. Dabei handelt es sich um die einzelnen Commits, in denen die Datei enthalten war. Durch diese Daten können die vorherigen Bearbeiter (**Previous Owner**), die Commit-Nachrichten (**Commit Messages**) und die Bearbeitungszeiträume ermittelt werden.

Da für andere `InformationObject`s, wie etwa Wiki-Seiten oder Tickets von Aufgabenverwaltungssystemen, keine eindeutigen Identifizierer aus der Ressource direkt abgeleitet werden können, ist die größere Herausforderung die Erstellung von Verbindungen zu diesen.

Eine Herangehensweise ist den Namen der Ressource als Schlagwort für eine Suche in der *Change*s-Tabelle zu verwenden. Dabei werden alle *Change*s und damit auch alle zugehörigen *InformationObject*s aufgezeigt, in denen der Name der Ressource vorkommt. Dabei ist zu beachten, dass nur jene *InformationObject*s ermittelt werden, bei denen es sich nicht um SVN-Sourcen handelt. So können beispielsweise **Wiki-Seiten**, in denen die Datei erwähnt wird, angezeigt werden. Problematisch ist dabei, dass die Dateinamen nicht zur eindeutigen Identifizierung genutzt werden können. So ist etwa in jedem Java-Projekt, das als Equinox-Bundle implementiert wird, eine Datei mit dem Namen „plugin.xml“enthalten, wodurch in den Suchergebnissen häufig auch unrelevante Informationen enthalten sind. Bei eindeutigen Bezeichnern, wie beispielsweise „JiraConnector.java“, sind die Ergebnisse der Suche relativ genau. Diese Problematik kann umgangen werden, indem statt dem Namen der Datei der Paket- oder Projekttitel als Schlagwort für die Suche verwendet wird.

Eine andere Herangehensweise, die sich als exakt erwiesen hat, ist die Analyse der Commit-Nachrichten, die im Zusammenhang mit der Ressource verfasst wurden. Eine

große Unterstützung dabei ist, wenn die Mitarbeiter zur Verwaltung der zugeteilten Arbeitsaufgaben ein PlugIn in der Entwicklungsumgebung verwenden (z.B. JIRA-PlugIn, (Atlassian Pty Ltd., 2009)). Da diese bei Commits ausgehend von der Entwicklungsumgebung einen fest definierten Präfix an die Commit-Nachricht anhängen, können die passenden Tickets anhand des Strings ermittelt werden. So fügt beispielsweise das JIRA-PlugIn für Eclipse den Namen und den Status des Tickets in einem String an die Commit-Nachricht an.

Der Prototyp des Discourse-Analyzers untersucht daher alle im ersten Schritt ermittelten Commit-Nachrichten anhand des fest definierten Präfix, um den Namen der im Rahmen der Ressource erstellten **Tickets** herauszufinden.

Wenn die Liste mit Discourse-Informationen erstellt ist, überträgt der Analyzer die Daten durch den Aufruf der `handleDiscourse`-Methode des `ICallBackService`- Interfaces an den Klienten (siehe Quellcode 4.2).

Awareness

Der Discourse-Analyzer führt ein Verzeichnis aller angemeldeten Klienten und deren aktuell geöffneten Ressourcen[1] (`DiscourseClient`). Dieses wird durch die Aufrufe der Benutzer regelmäßig aktualisiert (siehe Kapitel 4.6.3). Bei jedem Aufruf stellt der Analyzer für jeden Klienten eine Liste aller offenen Ressourcen der anderen Kollegen zusammen und überträgt sie an diesen mittels der `handleContextOfColleague`-Methode des `ICallBackService`-Interfaces (siehe Quellcode 4.2).

Wenn ein Klient sich abmeldet oder seinen Kontext einen gewissen Zeitraum nicht mehr sendet, wird er aus dem Verzeichnis des Analyzers entfernt.

Klassendiagramm

Auch für den Discourse-Analyzer ist in Abbildung 4.15 eine vereinfachte Darstellung der Klassen gezeigt. Das detaillierte Klassendiagramm, welches auch Methoden und Attribute enthält, ist im Anhang A.2 zu finden. Da es sich bei dem Paket *org.aysada.lupanku.*

[1] wird in diesem Kapitel auch als Kontext des Klienten bezeichnet

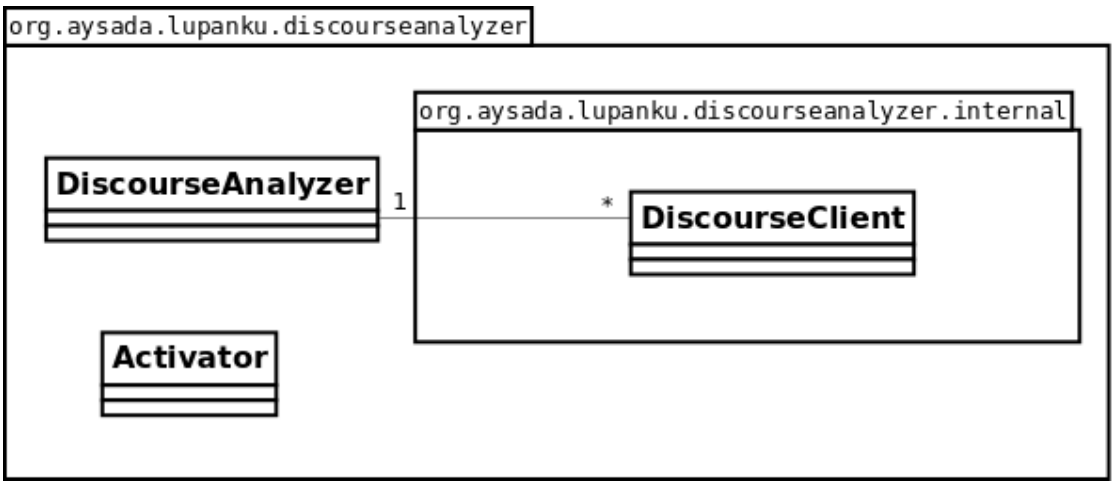


Abbildung 4.15.: vereinfachtes Modell der Klassen des Discourse-Analyzers

discourseanalyzer ebenfalls um ein PlugIn für das Equinox Framework von Eclipse handelt, beinhaltet es auch eine *plugin.xml*-Datei zur Verwaltung der Abhängigkeiten des PlugIns sowie eine *Activator*-Klasse als Schnittstelle zum OSGi-Framework.

Im Folgenden wird ein Überblick der erzeugten Klassen gegeben:

DiscourseAnalyzer – Der *DiscourseAnalyzer* ist die zentrale Klasse des *org.aysada.lupanku.discourseanalyzer*-Pakets. Er ist sowohl für das Erstellen der Discourse-Informationen, und somit für die Analyse der Daten aus der `LuPanKu`-Datenbank zuständig, als auch für das Führen eines Verzeichnisses aller angemeldeter Discourse-Klienten (`DiscourseClient`) und deren Kontexte. Darüber hinaus sendet er über die `ICallBackService`-Instanz, die im `DiscourseClient` gespeichert ist, regelmäßig die Kontexte der Kollegen an die einzelnen Klienten.

DiscourseClient – Diese Klasse repräsentiert einen Discourse-Klienten, der sich beim Discourse-Analyzer angemeldet hat. Sie beinhaltet sowohl die Referenz auf den Klienten anhand einer `ICallBackService`-Instanz als auch deren aktuellen Kontext. Zusätzlich wird mit jedem Aufruf durch den Klienten, bei dem er seinen Kontext sendet, der Zeitpunkt festgehalten. Dadurch können

Klienten aus der Liste des *DiscourseAnalyzer* entfernt und damit abgemeldet werden, wenn sie sich über einen längeren Zeitraum nicht mehr melden.

4.8. Fazit & Evaluation

Ausgehend von den Anforderungen, die in der Analyse ermittelt wurden, wurde in diesem Kapitel ein mögliches Design für das Discourse-Tool und die Realisierung eines Prototypen entwickelt. In Anlehnung an die Vision für das Discourse-Projekt wurde eine Service-orientierte Architektur (SOA) als grundlegendes Konzept gewählt. Als zentrale Komponenten der Architektur wurden dabei das Discourse-PlugIn für eine Entwicklungsumgebung und der Discourse-Analyzer im `LuPanKu`-System identifiziert. Bei diesen handelt es sich um getrennte Komponenten, die über eine lose Kopplung verbunden sind.

Technische Auswertung

Diese beiden Komponenten werden als Teil eines Equinox-Frameworks, einer Implementierung der OSGi-Spezifikation, realisiert. Dieses bietet eine modulare, dynamische Verwaltung der Elemente mit einer service-orientierten Verbindung. Das Discourse-PlugIn wird in eine Entwicklungsumgebung und der Discourse-Analyzer in den `LuPanKu`-Server integriert. Zur Verbindung und Kommunikation der Komponenten wurde das ECF gewählt, welches problemlos in das Equinox-Framework eingebunden werden kann und dieses um die Funktionalität der Remote-Services erweitert.

Als Entwicklungsumgebung, in die der Discourse-Prototyp integriert werden soll, wurde in dieser Arbeit Eclipse von der Eclipse Foundation gewählt, da es sich um die wohl meistgenutzte IDE handelt und von allen Teammitgliedern des *Home Office 2.0*-Projekts verwendet wird. Zudem stellt sie ein offenes und plattform-unabhängiges System dar, das einfach erweiterbar ist und auf der OSGi-Implementation Equinox basiert. Dem Entwickler sollen zusätzliche hilfreiche Informationen zur Bearbeitung einer Ressource in der IDE angeboten werden. Daher wurde nach einer Analyse der grafischen Oberfläche von Eclipse die visuelle Komponente *View* in der IDE als passendes Mittel zur Darstellung der Informationen gewählt. Zusdem wird das Icon des Views zur

Schaffung einer Awareness über die Tätigkeiten der Mitarbeiter verwendet, indem es in verschiedenen Signalfarben dargestellt wird.

Der Discourse-Analyzer wird in das `LuPanKu`-System integriert. Dazu implementiert er die `IAnalyzer`-Schnittstelle des Systems, um von diesem als solcher aufgefunden und aufgerufen werden zu können. Er analysiert die Daten in der `LuPanKu`-Datenbank anhand einer gegebenen Ressource in einer Versionsverwaltung, um relevante Daten über deren Entwicklungsprozess zu erhalten. Der Prototyp versucht Informationen zu ermitteln über:

- vorherige & aktuelle Bearbeiter der Ressource,
- Jira-Tickets, die im Zusammenhang mit der Bearbeitung der Ressource erstellt wurden,
- sowie Wikiseiten, die relevante Informationen zum Entwicklungsprozess beinhalten.

Dazu verwendet er zum einen eine einfache Stichwort-Suche in der `LuPanKu`-Datenbank mit dem Namen der Datei und zum anderen eine String-Analyse der Commit-Nachrichten, in denen die Datei enthalten war.

Bei der weiteren Entwicklung des Discourse-Projekts kann die Awareness optimiert werden, indem eine Unterscheidung zwischen bearbeiteten und nur geöffneten Ressourcen in der IDE getroffen wird. Um dies zu realisieren, könnte eine Verbindung zum Eclipse-PlugIn Mylyn (siehe Kapitel 3.4.2) erstellt und dessen Kontext verwendet werden.

Design Auswertung

Die grundlegende Service-orientierte Architektur unterstützt Programmier-Paradigmen, wie Modularität, Redundanzfreiheit und Wiederverwendbarkeit. Die beiden Komponenten, das IDE-PlugIn und Analyzer, sind nur lose miteinander gekoppelt und laufen auch ohne Verbindung fehlerfrei.

Auch das gewählte OSGi-Framework bietet Möglichkeiten zur modularen Entwicklung mit klar definierten Schnittstellen. Ferner gewährleistet die Lebenszyklus-Verwaltung des Frameworks eine dynamische Erweiterung bzw. Anpassung einzelner Module, ohne dass das gesamte System neu gestartet werden muss.

Durch diese Nutzung von Standards (OSGi) in Kombination mit Design-Patterns (SOA) ist die Anforderung an die Modularität sehr gut erfüllt.

Diese Modularität ermöglicht zudem eine problemlose Erweiterung bzw. Wiederverwendung des Tools. So kann der Discourse-Analyzer etwa auch von anderen Analyzern des `LuPanKu`-Systems aufgerufen werden, die anhand seiner Ergebnisse weitere Untersuchungen durchführen können. Außerdem erlaubt die Modularität eine Verknüpfung mit weiteren Teilprojekten des *Home Office 2.0*-Projekts, wie dem *Virtual Project Office* (VPO) oder der *Expertensuche* (siehe (Panier u. a., 2011, S. 5-9)). So könnten beispielsweise Informationen des VPO über aktuelle Tätigkeiten der Kollegen in den Discourse-Informationen (vorherige Bearbeiter) mit angezeigt werden.

Um die Wiederverwendbarkeit des IDE-PlugIns im Hinblick auf weitere Teilprojekte des *Home Office 2.0*-Projekts zu optimieren, kann die Kommunikation der Entwicklungsumgebung mit dem `LuPanKu`-Server in einem eigenen Modul verpackt werden (siehe Abbildung 4.8, `ClientHandler` und `CallBackService`). Dadurch könnten dann mehrere IDE-PlugIns (z.B. VPO) über eine gemeinsame Schnittstelle mit dem Server kommunizieren.

Benutzer Auswertung

Da es sich bei *Home Office 2.0* um ein noch junges Projekt handelt, traten bei den Tests zur Funktionalität und UserExperience verschiedene Schwierigkeiten auf, die diese erschwerten. Zum einen ist das zugrunde liegende `LuPanKu`-System selbst noch nicht ausführlich getestet worden, so dass bei der Entwicklung des Discourse-Tools immer wieder kleine Fehler im System aufgetreten sind, die nicht auf den entwickelten Discourse-Prototypen zurückzuführen waren und dennoch behoben werden mussten. Zum anderen sind noch nicht ausreichend viele Daten in der Datenbank vorhanden, um umfangreichere Tests durchzuführen. Daher sind diese nur im Rahmen des *Home Office 2.0*-Projekts durchgeführt, nicht aber in eine konkrete Teamarbeitssituation eines Unternehmens eingebunden worden. Dies ist dann der nächste Schritt zum Testen des Prototypen.

Aus den genannten Gründen können hier nur subjektive Wahrnehmungen zur Behilflichkeit des Tools genannt werden und eine Aussage über dessen „fehlerfreie“ Funktionalität getroffen werden.

Das Discourse-PlugIn fügt sich reibungslos in die Entwicklungsumgebung des Mitarbeiters ein. Es erfordert keinen zusätzlichen Aufwand beim Ausführen der IDE, weil es als PlugIn mit dieser gestartet wird. Da der Entwickler die Verbindung mit dem `LuPanKu`-Server und das Abrufen von Discourse-Informationen zu einer Datei selbst initieren muss, wird die Performance der Entwicklungsumgebung durch das PlugIn nicht merklich beeinträchtigt.

Die zur Darstellung der Discourse-Informationen in der IDE gewählte Baumstruktur ermöglicht einen schnellen und strukturierten Überblick über den Diskurs der Datei. Allerdings sind Qualität und Behilflichkeit der Informationen stark von der Implementierung des `LuPanKu`-Systems abhängig, da der Umfang der vorhandenen Daten an die Verbindungen zu Softwarentwicklungswerkzeugen gekoppelt ist. Außerdem korreliert die Exaktheit der Ergebnisse des Analyzers mit den Regeln des Softwareentwicklungsprozesses in einem Unternehmen. Wie in Kapitel 4.7.1 beschrieben, können die Ergebnisse der Analyse deutlich verbessert werden, wenn z.B. feste Regeln für die Formatierung von Commit-Nachrichten im Betrieb bestehen.

Wie für andere *Enterprise 2.0*-Systeme ist auch der Erfolg des Discourse-PlugIn stark von der Akzeptanz der Mitarbeiter abhängig. Wenn wirklich alle Teammitglieder das Discourse-Tool verwenden und sich beim Arbeiten mit dem `LuPanKu`-Server verbinden, ist die Aussage, die das Tool über die Awareness trifft, durchaus hilfreich, um sich über andere Entwicklungen an einer Ressource bewusst zu sein. Dabei ist die Aufnahme der Awareness-Informationen anhand der Signalfarben einfach und ohne zusätzlichen Aufwand möglich.

Durch die nahtlose Integration des PlugIns in die Arbeitsumgebung eines Entwicklers und den Mehrwert, den das Tool den Daten der Entwicklerwerkzeuge abgewinnt, ist für die Akzeptanz durch die Mitarbeiter und für ein positives Kosten-Nutzen-Verhältnis eine gute Grundlage gelegt.

5. Fazit und Ausblick

5.1. Zusammenfassung & Fazit

Die vorliegende Untersuchung hat sich mit Möglichkeiten der Unterstützung kooperativer Projektarbeit eines Softwareentwicklers befasst. Der Fokus lag dabei auf dem effizienten schnellen Aufbau des Kontexts der aktuellen Arbeitsaufgabe sowie dem Schaffen eines Bewusstseins (Awareness) über die gegenwärtigen Tätigkeiten der Kollegen. In diesem Rahmen wurde vom Autor der Prototyp eines Werkzeugs entwickelt, das diese Thematik behandelt und die Teamarbeit durch Informationen zu einer bestimmten Ressource unterstützt. Das **Discourse-Tool** ist Teil des Projekts *Home Office 2.0* an der Hochschule für Angewandte Wissenschaften (HAW) Hamburg.

Zur Einführung in die Thematik wurden in Kapitel 2 die Bereiche und Begriffe der CSCW, des Wissensmanagements, des Kontexts und der agilen Softwareentwicklung vorgestellt und erläutert. Diese schaffen ein grundlegendes Verständnis für die weitere Arbeit und deuten bereits einige allgemeine Anforderungen an die Entwicklung von Systemen an, die sich mit der Unterstützung von Kooperationsprozessen beschäftigen. So kristallisierte sich beispielsweise die besondere Rolle der Unternehmen bei der Unterstützung des Wissensmanagements heraus.

Kapitel 3 widmete sich anschließend der Analyse einiger Herausforderungen an einen Softwareentwickler in der heutigen Arbeitswelt. Dies zeigte zum einen die Relevanz der Behandlung dieser Herausforderungen selbst, zum anderen macht es einige Anforderungen deutlich, die sich an Systeme ergeben, die in diesem Zusammenhang entwickelt werden. Das Arbeiten in verteilten Entwicklerteams stellt eine Herausforderung dar, da die für die Teamarbeit so wichtige direkte Kommunikation erschwert wird und zudem meist kein gemeinschaftlicher Projektkontext besteht. Außerdem ist das Vertrauen die Basis einer effizienten Teamarbeit, was bei Mitgliedern, die sich nicht persönlich kennen, nur schwer aufgebaut werden kann. Letztlich liegt es in der Verantwortung der

Unternehmen, diesen, ebenso wie anderen Problematiken, entgegenzuwirken und den Mitarbeitern entsprechende Möglichkeiten zu bieten, effizient zu kommunizieren und eine Vertrauensbasis zu schaffen. Als weiteres Beispiel hierfür wurde in der Analyse deutlich, dass ein strukturiertes Wissensmanagement der Problematik der kognitiven Überlastung von Angestellten begegnen kann, indem es dem Entwickler beim Aufbau des Kontexts einer aktuelle Arbeitsaufgabe hilft. Das Vorhandensein von Schatten-IT ist häufig ein Zeichen dafür, dass es einer Anpassung der im Unternehmen bereitgestellten Softwarewerkzeuge bedarf. Vor allem im Bereich der Kommunikation und Kollaboration erfüllen IT-Abteilungen häufig nicht die Anforderungen und Vorstellungen der Mitarbeiter, die daher die Verwendung selbstgewählter Systeme initieren.

Mit den genannten Herausforderungen beschäftigt sich das Forschungsgebiet *Enterprise 2.0*. Es untersucht Möglichkeiten von *Social Software*, um den Wissensaustausch, die Bildung eines sozialen Netzwerks auch über Standortgrenzen hinaus und somit die Kollaboration in Unternehmen zu unterstützen. Da der Erfolg des Einsatzes solcher Systeme stark von der Akzeptanz der Mitarbeiter abhängt, sollte bei der Einführung der Nutzen für den Einzelnen im Vordergrund stehen, so dass ein gutes Aufwand-Nutzen-Verhältnis für ihn entsteht. In dieses Forschungsgebiet fällt auch die vorliegende Arbeit.

In Kapitel 3.3 wurden Szenarien entworfen, die den Rahmen dieser Arbeit eingrenzen und die Funktionalität des Discourse-Prototypen verdeutlichen. Dabei wurde festgelegt, dass dieses dem Entwickler den Aufbau des Kontexts einer aktuelle Arbeitsaufgabe anhand eines Diskurses zur zu bearbeitenden Ressource erleichtern soll. Außerdem soll eine Awareness über aktuelle Entwicklungen an der Ressource durch andere Mitarbeiter geschaffen werden. In diesem Zusammenhang wurde deutlich, dass die Funktionalität und der Erfolg des Tools stark davon abhängen, dass es von allen Teammitgliedern verwendet wird. Wenn etwa nur die Hälfte der Mitarbeiter das Tool nutzt, ist die Aussage, die das Tool über die Awareness trifft, unvollständig.

Bei der Betrachtung relevanter wissenschaftlicher Arbeiten und Projekte wurde deutlich, dass ein großes Interesse an Forschungsergebnissen des *Enterprise 2.0* besteht. Dies zeigt sich mitunter an den Konsortien und Gremien, die sich, um diese Fragestellung zu behandeln, zusammengefunden haben. Die hier herangezogenen Projekte legten ihren Fokus ebenfalls auf die Unterstützung der Entwickler beim Aufbau der Kontexte aktueller Arbeitsaufgaben. Jedoch wird dabei meist kein Datei-, sondern eher ein Projekt- oder Aufgabendiskurs verwendet. Zudem zeigte das Forschungsgebiet MSR,

dass auch hier bereits Ansätze entwickelt wurden, anhand von Analysen der Daten aus Softwareentwicklungswerkzeugen zusätzliches Wissen zu generieren.

Im Fazit der Analyse wurden die einzelnen Kapitel zusammengefasst, deren Ergebnisse formuliert und die Anforderungen an die Entwicklung des Discourse-Tools identifiziert.

Dessen Design und die Realisierung erläuterte Kapitel 4, wozu zunächst ein visionäres Design für das Projekt entwickelt wurde. Dieses sieht eine Erweiterung der Entwicklungsumgebungen der Mitarbeiter vor, welche dem Entwickler wie beschrieben beim Kontextaufbau hilft und dessen Bewusstsein über die Aktivitäten der Kollegen fördert. Im Anschluss daran stellte Kapitel 4.2 das *Home Office 2.0*-Projekt vor, das den Rahmen des Discourse-Tools und der vorliegenden Arbeit bildet.

Im Kapitel 4.3 wurde die Architektur für das Discourse-Tool erarbeitet sowie dessen einzelnen Komponenten identifiziert. Diese wurden in den folgenden Kapiteln im Einzelnen detailliert betrachtet.

Als Framework für die beiden Hauptkomponenten (Discourse-PlugIn für IDE, Discourse-Analyzer) wurde Equinox, eine Implementation der OSGi-Spezifikation, festgelegt. Es bietet als Mittel der Kommunikation das ECF, welches die Remote Services bereitstellt.

Die für die Integration des Discourse-PlugIn gewählte Entwicklungsumgebung ist Eclipse. Es offeriert dem Anwender die Möglichkeit, sich mit einem `LuPanKu`-Server zu verbinden und anschließend Discourse-Informationen zu Ressourcen in der IDE abzufragen. Bei diesen handelt es sich um Diskurse der Ressourcen, wobei sowohl die direkten Entwicklungsprozesse von Dateien enthalten sind als auch ergänzende Auskünfte, wie relevante Arbeitspakete oder Wiki-Seiten, aufgeführt werden. Überdies erzeugt das PlugIn, sobald es mit dem Server verbunden, ist eine Awareness über die Tätigkeiten der Kollegen, in dem es dem Benutzer anzeigt, ob andere Mitarbeiter aktuell an der geöffneten Ressource arbeiten.

Der Discourse-Analyzer, der auf dem `LuPanKu`-Server läuft, generiert, anhand des Kontext Repository des `LuPanKu`-Systems, die Discourse-Informationen. Dazu ermittelt er zunächst den Entwicklungsverlauf der jeweiligen Ressource mit Hilfe der Versionsverwaltung und erstellt anschließend Verbindungen zu anderen Werkzeugen der Softwareentwicklung, wie Aufgabenverwaltungs- oder Wiki-Systemen. Zudem führt der

Analyzer eine Liste aller angemeldeten Klienten und deren Kontexte (geöffnete Ressourcen). Diese werden in regelmäßigen Abständen an die Klienten gesandt, wodurch die Awareness über die Aktivitäten geschaffen werden kann.

Der entwickelte Prototyp wurde im Kontext des *Home Office 2.0*-Projekts getestet und erfüllte die festgelegten Anforderungen. Er gliedert sich ohne Mehraufwand in die Arbeitsumgebung eines Entwicklers ein und stellt ihm bedarfsorientiert zusätzliche Informationen zu der zu bearbeitenden Datei zur Verfügung. Diese nahtlose Integration ermöglicht ein positives Kosten-Nutzen-Verhältnis für den Mitarbeiter, was wiederum dessen Akzeptanz stärkt. Der Analyzer generiert aus den Daten der Softwareentwicklungswerkzeuge einen Diskurs über die Datei und gewinnt so den Rohdaten einen Mehrwert ab. Dadurch werden neben den vorherigen Bearbeitern auch relevante Arbeitsaufgaben (Tickets) und Wiki-Seiten aufgezeigt.

5.2. Ausblick

Zwischen den individuellen Arbeitsweisen der Angestellten und den vom Unternehmen zur Verfügung gestellten Werkzeugen sowie dessen Philosophie bestehen häufig Inkompatibilitäten. Daher setzt die *Enterprise 2.0*-Forschung auf Kollaborationssysteme, die das Bedürfnis eines Unternehmens nach Skalier- und Kontrollierbarkeit erfüllen und sich gleichzeitig reibungslos in den Arbeitsrhythmus des Mitarbeiters integrieren. Letzteres ist essentiell, um deren Akzeptanz gegenüber den Tools, die der Betrieb bereitstellt, zu steigern und diese somit effizienter zu machen.

Die Ergebnisse des Projekts *Home Office 2.0*, welches ebenfalls diesen Ansatz verfolgt, werden aus der Sicht des Autors auch in den nächsten Jahren aktuell und von Interesse bleiben. Die Relevanz spiegelt sich unter anderem in der Zunahme der entwickelten *Enterprise 2.0*-Systeme. So setzen mittlerweile auch Unternehmen, die eigentlich aus dem Bereich der Netzwerktechnik stammen, auf die Entwicklung solcher Systeme. Ein Beispiel dafür ist die Plattform *Cisco Quad*[1] von *Cisco Systems*, die vor allem die soziale Interaktion und die Mobilität der Mitarbeiter stärken will.

Bei der Umsetzung entsprechender Systeme gilt es, sowohl neue Ideen als auch bereits entwickelte Lösungen regelmäßig zu evaluieren und jeweils an gegenwärtige Ge-

[1] http://www.cisco.com/web/products/quad/index.html

gebenheiten und aktuelle Erkenntnisse der Forschung anzupassen. So sollte auch das Discourse-Projekt als Teil von *Home Office 2.0* bei der weiteren Entwicklung kontinuierlich neu betrachtet und an die gegebene Situation angeglichen werden.

Allerdings zeigten die in Kapitel 3.1 vorgestellten Herausforderungen der Softwareentwicklung, dass der Unterstützung durch Kollaborationssysteme Grenzen gesetzt sind. Sie können beispielsweise den direkten Kontakt (*face-to-face*) in verteilten Teams bisher nicht adäquat ersetzen. Dieser hält allerdings, vor allem in einigen neuen Entwicklungsmethoden wie Scrum, einen besonderen Stellenwert inne und ist aus der Sicht des Autors auch mit den aktuellen Technologien noch nicht modellierbar. An diesem Punkt wird sich zeigen, ob sich zukünftig die Entwicklung der Methoden an den vorhandenen technischen und sozialen Strukturen (z.B. verteilte Teamarbeit) orientieren wird oder ob vielmehr diese den neuen Entwicklungsmethoden angepasst werden.

Literaturverzeichnis

[ACTIVE Consortium 2011a] ACTIVE CONSORTIUM: *Active - Enabling the Knowledge Powered Enterprise*. 2011. – URL http://www.active-project.eu/about-active.html. – Zugriffsdatum: 12. October 2011

[ACTIVE Consortium 2011b] ACTIVE CONSORTIUM: *Overview Of Akws And What It Offers To Your Organization*. 2011. – URL http://www.active-project.eu/publications/software-download/knowledge-workspace-software/overview.html. – Zugriffsdatum: 10. October 2011

[it agile 2011] AGILE it: *Was ist agile Softwareentwicklung?* 2011. – URL http://www.it-agile.de/wasistagilesoftwareentwicklung.html. – Zugriffsdatum: 20. November 2011

[Ambler 2002] AMBLER, Scott: *Bridging the Distance*. September 2002. – URL http://drdobbs.com/184414899. – Zugriffsdatum: 12. August 2011

[Andresen 2011] ANDRESEN, Judith: *Auf grosse Reise gehen - Über die Methodenwahl in Projekten*. Slideshare. Dezember 2011. – URL http://www.slideshare.net/janosch007/auf-groe-reise-gehen-ber-die-methodenwahl-in-projekten. – Zugriffsdatum: 05. December 2011

[Apache Software Foundation 2011] APACHE SOFTWARE FOUNDATION: *Apache Felix*. March 2011. – URL http://felix.apache.org/site/index.html. – Zugriffsdatum: 15. July 2011

[Appcelerator 2011] APPCELERATOR: *Aptana Studio 3*. 2011. – URL http://www.aptana.com/. – Zugriffsdatum: 07. November 2011

[Atlassian Pty Ltd. 2009] ATLASSIAN PTY LTD.: *Using JIRA in the Eclipse Connector*. Juli 2009. – URL http://confluence.atlassian.com/display/

IDEPLUGIN/Using+JIRA+in+the+Eclipse+Connector. – Zugriffsdatum: 23. June 2011

[Atlassian Pty Ltd. 2011a] ATLASSIAN PTY LTD.: *Confluence - Everyone on the same page*. 2011. – URL http://www.atlassian.com/software/confluence. – Zugriffsdatum: 21. August 2011

[Atlassian Pty Ltd. 2011b] ATLASSIAN PTY LTD.: *Jira - Issue and project tracker*. 2011. – URL http://www.atlassian.com/software/jira. – Zugriffsdatum: 21. August 2011

[Back und Seufert 2000] BACK, Andrea ; SEUFERT, Andreas: Computer Supported Cooperative Work (CSCW) - State of the Art und zukünftige Herausforderungen. In: *HMD - Praxis Wirtschaftsinformatik* 213 (2000), S. 5–22

[Bajracharya und Lopes 2009] BAJRACHARYA, Sushil ; LOPES, Cristina: Mining search topics from a code search engine usage log. In: *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*. Washington, DC, USA : IEEE Computer Society, 2009 (MSR '09), S. 111–120. – ISBN 978-1-4244-3493-0

[Bayan 2004] BAYAN, Ruby: *Shed light on shadow IT groups*. July 2004. – URL http://articles.techrepublic.com.com/5100-22_11-5247674.html. – Zugriffsdatum: 09. May 2011

[Beck u. a. 2001] BECK, Kent ; BEEDLE, Mike ; AL. et: Manifesto for Agile Software Development. (2001), Februar. – URL http://www.agilemanifesto.org/. – Zugriffsdatum: 06. December 2011

[Bolour 2003] BOLOUR, Azad: *Notes on the Eclipse Plug-in Architecture*. 2003. – URL http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html. – Zugriffsdatum: 23. November 2011

[Cadiz u. a. 2001] CADIZ, Jj ; CADIZ, Jj ; VENOLIA, Gina D. ; VENOLIA, Gina D. ; JANCKE, Gavin ; JANCKE, Gavin ; GUPTA, Anoop ; GUPTA, Anoop: Sideshow: Providing Peripheral Awareness of Important Information / Microsoft Research, Collaboration, and Multimedia Group. Redmond, WA 98052, September 2001. – Forschungsbericht. – URL http://research.microsoft.com/en-us/groups/coet/01-83.pdf

[Codd u. a. 1993] CODD, Edgar F. ; CODD, S B. ; SALLEY, C T.: Providing OLAP to User-Analysts: An IT Mandate. In: *Ann ArborMichigan* (1993), S. 24. – URL http://www.minet.uni-jena.de/dbis/lehre/ss2005/sem_dwh/lit/Cod93.pdf

[Dey 2001] DEY, Anind K.: Understanding and Using Context. In: *Personal Ubiquitous Comput.* 5 (2001), January, S. 4–7. – ISSN 1617-4909

[Drucker 1972] DRUCKER, Peter F.: *Concept of the corporation [by] Peter F. Drucker. 1972 ed. with a new pref. and new epilogue by the author*. [Rev. ed.]. John Day Co. New York„ 1972. – xxv, 319 p. S

[Drucker 1999] DRUCKER, Peter F.: Knowledge-worker productivity: the biggest challenge. In: *California Management Review* 41 (1999), Nr. 2, S. 79–94. – URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1679053

[Dunkel u. a. 2008] DUNKEL, Jürgen ; EBERHART, Andreas ; FISCHER, Stefan ; KLEINER, Carsten ; KOSCHEL, Arne: *Systemarchitekturen für Verteilte Anwendungen*. München : Hanser, 2008. – ISBN 978-3-446-41321-4

[Ebert 2011] EBERT, Ralf: *Eclipse RCP*. Bd. 2. Ralf Ebert, August 2011. – 230 S. – URL http://www.ralfebert.de/eclipse_rcp/osgi/

[ECF Team 2010] ECF TEAM: *OSGi 4.2 Remote Services and ECF*. The Eclipse Foundation. 2010. – URL http://wiki.eclipse.org/OSGi_4.2_Remote_Services_and_ECF. – Zugriffsdatum: 12. September 2011

[ECF Team 2011a] ECF TEAM: *ECF User Guide*. The Eclipse Foundation. July 2011. – URL http://wiki.eclipse.org/EUG:Users_Guide. – Zugriffsdatum: 27. August 2011

[ECF Team 2011b] ECF TEAM: *ECF/API Docs*. The Eclipse Foundation. 2011. – URL http://wiki.eclipse.org/ECF_API_Docs. – Zugriffsdatum: 04. September 2010

[Eckstein 2009] ECKSTEIN, Jutta: *Agile Softwareentwicklung mit verteilten Teams*. Heidelberg : dpunkt.verlag, 2009

[Edgewall Software 2011] EDGEWALL SOFTWARE: *The Trac Project*. 2011. – URL http://trac.edgewall.org. – Zugriffsdatum: 20. April 2011

[Ellis u. a. 1991] ELLIS, C. A. ; GIBBS, S. J. ; REIN, G.L.: Groupware: Some issues and experiences. In: *COMMUNICATIONS OF THE ACM*, 1991, S. 38–58

[Erl 2005] ERL, Thomas: *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2005. – ISBN 0131858580

[Fachgruppe CSCW] FACHGRUPPE CSCW: *Computer-Supported Cooperative Work*. – URL http://www.fgcscw.de/. – Zugriffsdatum: 17. April 2011

[Fowler 2006] FOWLER, Martin: *Code Ownership*. 2006. – URL http://martinfowler.com/bliki/CodeOwnership.html. – Zugriffsdatum: 17. March 2011

[Gasser 2009] GASSER, Urs: *DNAdigital - Wenn Anzugträger auf Kapuzenpullis treffen: Die Kunst, aufeinander zuzugehen*. Neckarhausen : Willms Buhse and Ulrike Reinhard, 2009. – ISBN 9783934013988

[Gennaro 2008] GENNARO, Rocco J.: Representationalism, Peripheral Awareness, and the Transparency of Experience. In: *Philosophical Studies* 139 (2008), S. 39–56

[Gross und Koch 2007] GROSS, Tom ; KOCH, Michael: *Computer-Supported Cooperative Work*. München : Oldenbourg, 2007 (Interaktive Medien). – URL http://www.gbv.de/dms/ilmenau/toc/516033204.PDF. – ISBN 9783486580006

[Hall 2007] HALL, Anja: Tätigkeiten und berufliche Anforderungen in wissensintensiven Berufen, Empirische Befunde. In: *BIBB/BAuA-Erwerbstätigenbefragung 2006, Bundesinstitut für Berufsbildung* (2007), Februar, Nr. Nr. 3-2007

[Handy 1995] HANDY, Charles: *Trust and the Virtual Organization*. Visionarymarketing.com. May 1995. – URL http://hbr.org/1995/05/trust-and-the-virtual-organization/ar/1. – Zugriffsdatum: 07. November 2011

[Helms 1991] HELMS, Rich: Distributed knowledge worker (DKW): a personal conferencing system. In: *Proceedings of the 1991 conference of the Centre for Advanced Studies on Collaborative research*, IBM Press, 1991 (CASCON '91), S. 115–125. – URL http://dl.acm.org/citation.cfm?id=962111.962122

[Herbsleb 2007] HERBSLEB, James D.: Global Software Engineering: The Future of Socio-technical Coordination. In: *Future of Software Engineering* 0 (2007), S. 188–198. ISBN 0-7695-2829-5

[Herbsleb und Mockus 2003] HERBSLEB, James D. ; MOCKUS, Audris: An Empirical Study of Speed and Communication in Globally Distributed Software Development. In: *IEEE Trans. Softw. Eng.* 29 (2003), June, S. 481–494. – URL http://dl.acm.org/citation.cfm?id=1435631.859041. – ISSN 0098-5589

[Hunt 2008] HUNT, Andy: *Pragmatic Thinking and Learning: Refactor Your Wetware (Pragmatic Programmers)*. Pragmatic Bookshelf, 2008. – ISBN 1934356050, 9781934356050

[Kagdi u. a. 2007] KAGDI, Huzefa ; COLLARD, Michael L. ; MALETIC, Jonathan I.: *A survey and taxonomy of approaches for mining software repositories in the context of software evolution*. 2007

[Kersten 2007] KERSTEN, Mik: *Focusing knowledge work with task context*. Vancouver, BC, Canada, Canada, Dissertation, 2007. – AAINR26735

[Kirsh 2000] KIRSH, David: A Few Thoughts on Cognitive Overload. In: *Intellectica* 30 (2000), S. 19–51

[Koch 2009] KOCH, Michael: *Computer-Supported Cooperative Work (CSCW)*. Januar 2009. – URL http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/daten-wissen/Informationsmanagement/IT-Infrastruktur/Informations--und-Kommunikationstechnologien/computer-supported-cooperative-work-cscw. – Zugriffsdatum: 14. April 2011

[Koch 2008] KOCH, Prof. Dr. M.: Enterprise 2.0 ... Social Software in Unternehmen / Universität der Bundeswehr München Forschungsgruppe Kooperationssysteme. September 2008. – Forschungsbericht

[Koeckritz 2006] KOECKRITZ, Oliver: Geschichte und Konzepte von Collaborative Workspaces. In: *Seminararbeit im Fach AW1* (2006). – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2006/koeckritz/abstract.pdf

[Lynch u. a. 1990] LYNCH, Kevin J. ; SNYDER, Joel M. ; VOGEL, Douglas R. ; MCHENRY, William K.: The Arizona Analyst Information System: supporting collaborative research on international technological trends. In: *Proceedings of the IFIP WG 8.4 confernece on Multi-user interfaces and applications*. Amsterdam, The

Netherlands, The Netherlands : Elsevier North-Holland, Inc., 1990, S. 159–174. – URL http://portal.acm.org/citation.cfm?id=133482.133506. – ISBN 0-444-88750-1

[Mall 2004] MALL, R.: *Fundamentals of Software Engineering*. Prentice-Hall of India, 2004. – ISBN 9788120324459

[Marquis 2006] MARQUIS, Hank: *The Rise of Shadow IT*. September 2006. – URL http://www.cioupdate.com/reports/article.php/11050_3633056_1/The-Rise-of-Shadow-IT.htm. – Zugriffsdatum: 07. May 2011

[McAfee 2006] MCAFEE, Andrew P.: Enterprise 2.0: The Dawn of Emergent Collaboration. In: *MITSloan Management Review* 47 (2006), Nr. 3, S. 21–28. – URL http://sloanreview.mit.edu/the-magazine/articles/2006/spring/47306/enterprise-the-dawn-of-emergent-collaboration/

[McKinney und Whiteside 2006] MCKINNEY, Vicki R. ; WHITESIDE, Mary M.: Maintaining distributed relationships. In: *Commun. ACM* 49 (2006), March, S. 82–86. – URL http://doi.acm.org/10.1145/1118178.1118180. – ISSN 0001-0782

[Melzer 2010] MELZER, Ingo (Hrsg.): *Service-orientierte Architekturen mit Web Services: Konzepte – Standards – Praxis*. 4. Heidelberg : Spektrum, 2010

[9th Working Conference on Mining Software Repositories 2011] MINING SOFTWARE REPOSITORIES, The 9th Working Conference on: *MSR 2012*. 2011. – URL http://2012.msrconf.org/. – Zugriffsdatum: 07. December 2011

[Mylyn Team 2011a] MYLYN TEAM: *Mylyn / User Guide*. Eclipse Foundation. 2011. – URL http://wiki.eclipse.org/index.php/Mylyn/User_Guide. – Zugriffsdatum: 10. October 2011

[Mylyn Team 2011b] MYLYN TEAM: *Mylyn Extensions*. Eclipse Fondation. 2011. – URL http://wiki.eclipse.org/index.php/Mylyn_Extensions. – Zugriffsdatum: 15. October 2011

[North und Güldenberg 2008] NORTH, Klaus ; GÜLDENBERG, Stefan: *Produktive Wissensarbeit(er): Antworten auf die Management-Herausforderung des 21. Jahrhunderts ; mit vielen Fallbeispielen ; Performance messen, Produktivität steigern, Wissensarbeiter entwickeln*. 1. Aufl. Wiesbaden : Gabler, 2008. – ISBN 9783834907387

[OSGi Alliance 2011a] OSGi Alliance: *About the OSGi Alliance*. 2011. – URL http://www.osgi.org/About/HomePage. – Zugriffsdatum: 23. September 2011

[OSGi Alliance 2011b] OSGi Alliance: *Knopperfish*. 2011. – URL http://www.knopflerfish.org/. – Zugriffsdatum: 14. April 2011

[OSGi Alliance 2011c] OSGi Alliance: *OSGi Alliance Specifications*. 2011. – URL http://www.osgi.org/Specifications/HomePage?section=2#Release4. – Zugriffsdatum: 19. November 2011

[OSGi Alliance 2011d] OSGi Alliance: *OSGi Technology*. 2011. – URL http://www.osgi.org/About/Technology. – Zugriffsdatum: 23. September 2011

[Panier 2011] Panier, Karsten: Home Office 2.0 - Virtual Project Office. (2011). – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master10-11-seminar/panier/folien.pdf

[Panier u. a. 2010] Panier, Karsten ; Holsten, Matthias ; Kirstgen, Benjamin: *Projektbericht Home-Office 2.0*. 2010. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2010-proj1/panier.pdf

[Panier u. a. 2011] Panier, Karsten ; Holsten, Matthias ; Kirstgen, Benjamin: *Projektbericht Home-Office 2.0 - Second Milestone*. Februar 2011. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master10-11-proj2/holsten-kirstgen-panier.pdf

[Perez u. a. 2009] Perez, Jose Manuel G. ; Grobelnik, Marko ; Ruiz, Carlos ; Tilly, Marcel ; Warren, Paul: Using task context to achieve effective information delivery. In: *CIAO '09: Proceedings of the 1st Workshop on Context, Information and Ontologies*. New York, NY, USA : ACM, 2009, S. 1–6. – URL http://portal.acm.org/citation.cfm?id=1552262.1552265. – ISBN 978-1-60558-528-4

[Raden 2005] Raden, Neil: Shedding Light on Shadow IT. In: *DSSResources.COM* (2005), Februar. – URL http://dssresources.com/papers/features/raden/raden02262005.html. – Zugriffsdatum: 24. April 2011

[Schilit u. a. 1994] Schilit, B. ; Adams, N. ; Want, R.: Context-Aware Computing Applications. In: *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*. Washington, DC, USA : IEEE Computer Society, 1994, S. 85–90.

– URL http://dl.acm.org/citation.cfm?id=1439278.1440041. – ISBN 978-0-7695-3451-0

[Schmatz 2007] SCHMATZ, Mark: *Generische kontext-sensitive Aspekte für Service-Orientierte Architekturen*. Bonn, Rheinische Friedrich-Wilhelms Universität, Diplomarbeit, Februar 2007

[Sherman 2004] SHERMAN, Rick: Shedding Light on Data Shadow Systems. In: *DM Direct* (2004). – URL http://www.athena-solutions.com/bi-brief/may04-issue12.html. – Zugriffsdatum: 24. April 2011

[Shumarova und Swatman 2008] SHUMAROVA, Elitsa ; SWATMAN, Paul A.: Informal eCollaboration Channels: Shedding Light on Shadow CIT. In: *Proc. Bled eConference 2008*, 2008

[Spafford 2004] SPAFFORD, George: *The Dangers that Lurk behind Shadow-IT*. February 2004. – URL http://itmanagement.earthweb.com/career/article.php/3308481/The-Dangers-that-Lurk-Behind-Shadow-IT.htm. – Zugriffsdatum: 24. April 2011

[The Apache Software Foundation 2011] THE APACHE SOFTWARE FOUNDATION: *Apache Subversion - Enterprise-class centralized version control for the masses*. 2011. – URL http://subversion.apache.org/. – Zugriffsdatum: 25. November 2011

[The Eclipse Foundation 2010] THE ECLIPSE FOUNDATION: *The Open Source Developer Report*. 2010. – URL http://www.eclipse.org/org/community_survey/Eclipse_Survey_2010_Report.pdf

[The Eclipse Foundation 2011a] THE ECLIPSE FOUNDATION: *eclipse*. 2011. – URL http://www.eclipse.org/org/. – Zugriffsdatum: 12. October 2011

[The Eclipse Foundation 2011b] THE ECLIPSE FOUNDATION: *Eclipse Communication Framework*. 2011. – URL http://www.eclipse.org/ecf/. – Zugriffsdatum: 20. October 2011

[The Eclipse Foundation 2011c] THE ECLIPSE FOUNDATION: *Equinox*. 2011. – URL http://www.eclipse.org/equinox/. – Zugriffsdatum: 20. October 2011

[The Eclipse Foundation 2011d] THE ECLIPSE FOUNDATION: *Riena Platform Project*. 2011. – URL http://www.eclipse.org/riena/. – Zugriffsdatum: 20. October 2011

[Thrower u. a. 2011] THROWER, Woody ; KOREN, Eitan ; SUEN, Remy Chi J. u. a.: *User Interface Guidelines*. Juni 2011. – URL http://wiki.eclipse.org/User_Interface_Guidelines. – Zugriffsdatum: 29. November 2011

[Valetto u. a. 2007] VALETTO, Giuseppe ; HELANDER, Mary ; EHRLICH, Kate ; CHULANI, Sunita ; WEGMAN, Mark ; WILLIAMS, Clay: Using Software Repositories to Investigate Socio-technical Congruence in Development Projects. In: *Proceedings of the Fourth International Workshop on Mining Software Repositories*. Washington, DC, USA : IEEE Computer Society, 2007 (MSR '07), S. 25–. – URL http://dl.acm.org/citation.cfm?id=1269039. – ISBN 0-7695-2950-X

[von Thun 1981] VON THUN, Friedemann S.: *Miteinander reden 1 - Störungen und Klärungen. Allgemeine Psychologie der Kommunikation.* Hamburg : Rowohlt, 1981

[Watson und Kaegi 2007] WATSON, Thomas ; KAEGI, Simon: *The Equinox Project*. Webinar. 2007. – URL http://www.eclipse.org/community/training/webinars/071127_Equinox_Webinar.pdf

[Weber u. a. 2010] WEBER, B. ; BAUMGARTNER, P. ; BRAUN, O.: *OSGi für Praktiker: Prinzipien, Werkzeuge und praktische Anleitungen auf dem Weg zur "kleinen SOA"*. München, Wien : Hanser Fachbuchverlag, 2010. – ISBN 9783446420946

[Willke 1998] WILLKE, Helmut: Organisierte Wissensarbeit. In: *Zeitschrift für Soziologie* 27 (1998), Nr. 3, S. 161–177. – URL http://www.uni-bielefeld.de/soz/globalgov/Lit/Willke_ZfS.pdf

[Winograd 2001] WINOGRAD, Terry: Architectures for context. In: *Hum.-Comput. Interact.* 16 (2001), Dezember, S. 401–419. – ISSN 0737-0024

[Worthen 2007] WORTHEN, Ben: User Management - Users Who Know Too Much and the CIOs Who Fear Them. In: *CIO Magazine* (2007), February. – URL http://www.cio.com/article/28821/User_Management_Users_Who_Know_Too_Much_and_the_CIOs_Who_Fear_Them_?page=1&taxonomyId=3119. – Zugriffsdatum: 05. May 2011

A. Klassendiagramme

A.1. Klassendiagramm Discourse Eclipse-PlugIn

Aus Platzgründen wurden einige private Methoden und Variablen, die für das Verständnis nicht relevant sind aus der Darstellung rausgelassen.

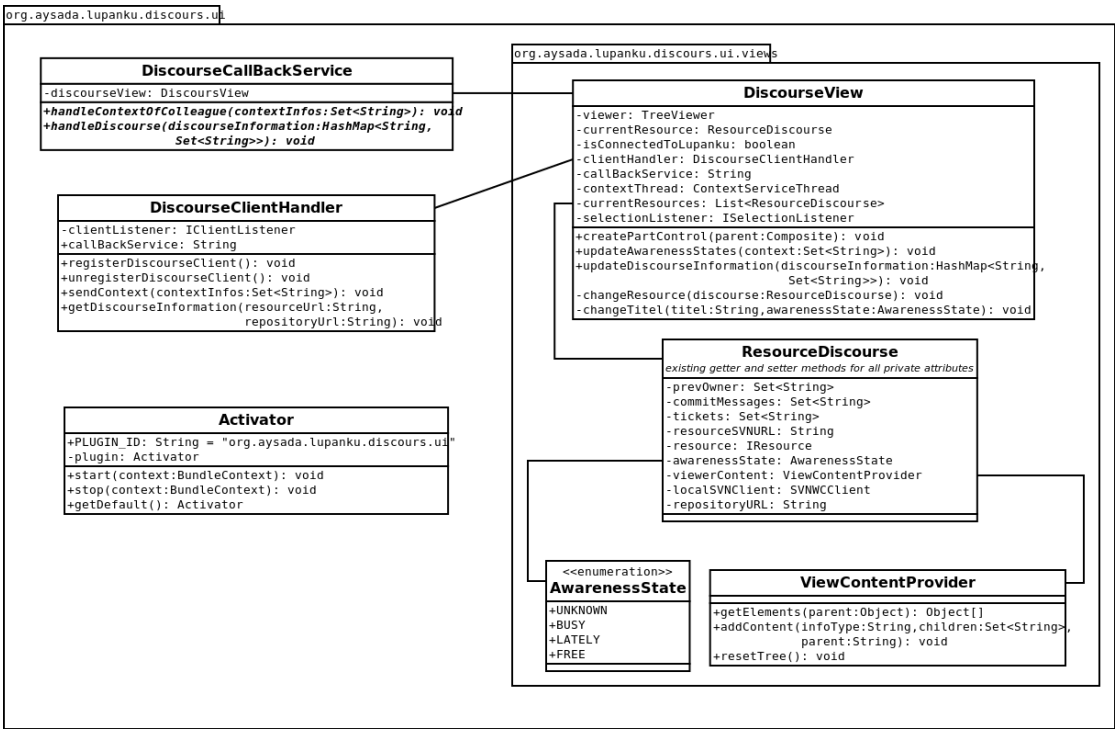


Abbildung A.1.: Klassendiagramm Paket org.aysada.lupanku.discourse.ui

A.2. Klassendiagramm Discourse-Analyzer

Aus Platzgründen wurden einige private Methoden und Variablen, die für das Verständnis nicht relevant sind aus der Darstellung rausgelassen.

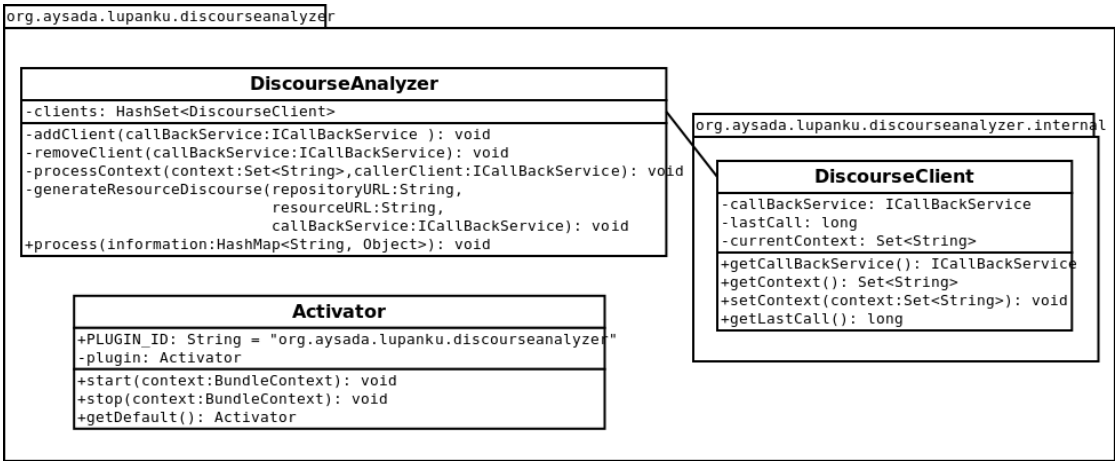


Abbildung A.2.: Klassendiagramm Paket org.aysada.lupanku.discourseanalyzer

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Ort, Datum Unterschrift