Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektbericht Home-Office2.0

Matthias Holsten, Benjamin Kirstgen,
Karsten Panier, Daniel Wojtucki

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Matthias Holsten, Benjamin Kirstgen,
Karsten Panier, Daniel Wojtucki

Projektbericht Home-Office2.0

LuPanKu Projekt - First Milestone

Projektbericht eingereicht im Rahmen von PR1
im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Professor: Prof. Dr. Kai von Luck

Abgegeben am 31. August 2010

Inhaltsverzeichnis

1 Einleitung

In der heutigen globalisierten Wissensgesellschaft gibt es eine stetige Änderung der Arbeitsformen und die daraus resultierenden Anforderungen an einen Mitarbeiter. Dies erfordert, neben der Anpassung der Arbeitsmethoden, auch eine bessere Arbeitsunterstützung für den Mitarbeiter bzw. ein Unternehmen. Aus diesem und anderen Gründen (bspw. der Vorbeugung von *Schatten-IT*, vgl. Abschnitt 2.1) werden vermehrt Neue Medien eingesetzt, welche sich bereits in der Internet-Community bewährt haben. Die als *Web2.0* bezeichneten Anwendungen, wie bspw. *Facebook* [12], *GoogleMail* [21], *Twitter* [45] oder *Wikipedia* [33], erhalten zurzeit Einzug in moderne wie auch traditionelle Unternehmen und werden dort als *Enterprise2.0*-Technologien bezeichnet.

Das hier beschriebene Projekt mit dem Codenamen `LuPanKu` betrachtet dabei das Szenario *Home-Office2.0*. Dabei geht es um die Unterstützung von Entwicklerteams, wobei Teile des Teams zeitweise über mehrere Standorte verteilt arbeiten. Für eine generelle Arbeitsunterstützung der Entwickler, speziell aber für das verteilte Arbeiten, werden Enterprise2.0-Technologien eingesetzt. Vorallem in der Softwareentwicklung werden schon länger einzelnen Anwendungen aus dem Enterprise2.0-Kontext genutzt. In der Regel werden diese Anwendungen allerdings als einzelnen eigenständige Insellösungen verwendet, die für die Unterstützung einer bestimmten Domäne bereitgestellt werden. Dies geschieht ohne ein integratives Konzept, womit eine Vielzahl der Vorteile, die sich aus der Nutzung von integrierten Enterprise2.0-Anwendungen ergeben, ungenutzt bleiben.

`LuPanKu` erarbeitet in dem beschriebenen Szenario zwei wesentliche Bereiche. In dem konzeptionellen Bereich werden verschiedene Strategien zum Einsatz von Enterprise2.0-Technologien in dem Szenariokontext entwickelt. Die ersten Ansätze dieser Konzepte und Lösungen werden u.a. im Kapitel 3 vorgestellt, fließen aber auch immer wieder an anderer Stelle mit ein. Der zweite, anwendungsbezogene Bereich betrifft die Entwicklung eines Softwaresystems, welches die genannten Insellösungen inhaltlich verknüpft und somit eine große Bandbreite an relevanten Fragen aus dem täglichen Arbeitsleben beantworten kann. Dazu gehört bspw. die Suche nach einem geeigneten Experten für ein spezielles Themengebiet oder das Aufzeigen eines sozialen Netzwerks anhand von Entwickleraktivitäten im Source-Code (vgl. Abschnitt 5.6).

Die vorliegende Arbeit stellt die Ergebnisse des ersten Projektreleases vor. Dabei werden in Kapitel 2 die bereits teilweise genannten Begrifflichkeiten, die in dem derzeitigen Projektrahmen relevant sind, erläutert bzw. für dieses Projekt definiert. In Kapitel 3 werden die Gründe und Problematiken in der Softwareentwicklung aufgezeigt, aus denen das Projekt `LuPanKu` entstanden ist. Desweiteren werden verschiedene Ideen zur Lösung der Probleme diskutiert. Aus einer Teilmenge dieser Ideen und Strategien ist der aktuelle

Projektrahmen, welcher in Kapitel 4 beschrieben ist, entstanden. Er bildet die aktuellen Anforderungen an das erste Release des `LuPanKu`-Systems. Das daraus entstandene System wird detailliert in den Kapiteln 5 und 6 beschrieben. Die Erfahrungen und Probleme die bei der Projektarbeit entstanden, sind in Kapitel 7 dokumentiert und sollen bei der zukünftigen Weiterführung des Projekts unterstützen. Abschließend stellt das Kapitel 8 die naheliegenden nächsten Projektschritte für das kommende Release vor.

2 Begriffsklärung

Für den Entwurf einer Vision für das LuPanKu-Projekt werden zunächst die notwendigen Begriffe und Themenbereiche diskutiert. Dazu zählt der Ansatz von *Enterprise2.0* sowie das Szenario *Home-Office2.0*. Abschließend wird die Arbeitssituation von Wissensarbeitern und speziell Softwareentwicklern betrachtet, die den Anwenderkreis darstellen.

2.1 Enterprise2.0

Einer der wichtigsten Aspekte bei der Arbeit eines Entwicklers ist das erstellte und verfügbare Wissen (engl. „Knowledge Worker"). Das vorhandene und neu gesammelte Wissen effektiv zu verwalten und einen schnellen Zugriff darauf zu ermöglichen, ist für ein Unternehmen daher von großem Interesse. In vielen Unternehmen lief der Informationsfluss zwischen Entwicklern verschiedener Abteilungen häufig erst über Projekt- und Abteilungsleiter. Ursächlich hierfür sind bestehende Unternehmensstrukturen. Eine derartige Kommunikation kostet jedoch unnötigen Aufwand, der durch eine direkte Verbindung der Arbeiter verschiedener Abteilungen eines Unternehmens verhindert werden kann.

Zum anderen kann der Informationsaustausch durch das Etablieren eines sozialen Netzwerks im Unternehmen mit Hilfe von *Social Software* gefördert werden. Unter *Social Software* werden Anwendungen verstanden, die die menschliche Interaktion unterstützen und dazu Technologien wie Weblogs, Wikis, Gruppeneditoren, Social Tagging, Instant Messaging (IM) und Präsenz-Awareness-Systeme verwenden [24]. McAffee prägte dazu 2006 den Begriff *Enterprise2.0* , mit dem er sich auf Systeme bezog, die aus dem Bereich Web2.0 stammen und die Praktiken und Ergebnisse der *Wissensarbeiter* eines Unternehmens sichtbar machen. Dazu zählt er auch implizites Wissen (*tacit knowledge*) und *Best Practices* [28]. Der Begriff umfasst aber nicht nur die Werkzeuge selbst, sondern auch eine Veränderung in der Unternehmensstruktur, die durch die soziale Vernetzung der Mitarbeiter entsteht [28].

Die kommende Generation der Digital Natives ist ein weiterer Indikator für die Relevanz von *Enterprise2.0*. Dabei handelt es sich um junge Menschen, die mit den digitalen Technologien aufgewachsen sind. Sie formen ihre Identitäten online, besonders durch soziale, digitale Netzwerke. Sie sind auf die neuen Möglichkeiten der Interaktion, Kommunikation und Informationsbeschaffung geprägt, die die vernetzte Welt bietet. Daher ist es zum Vorteil für das Unternehmen, wenn diese Möglichkeiten auch auf Unternehmensebene zur Verfügung gestellt werden [20].

Ein weiterer, im Kontext von *Enterprise2.0* relevanter Begriff ist die Schatten-IT. Dabei handelt es sich um IT im Unternehmen, die außerhalb der unternehmensweiten IT-Struktur steht (z.B. eigene USB-Sticks, heruntergeladene Softwaretools) [34]. Sie entsteht, wenn Individuen oder Teams in einem Unternehmen denken, dass sie ihre Arbeit schneller und effizienter ausführen können, wenn sie nicht an die IT im Unternehmen gebunden wären. Daher ist diese oft ein Indikator für eine den Service-Ansprüchen der Mitarbeiter nicht gerecht werdende IT-Abteilung [38]. *Enterprise2.0*-Systeme sollten dem Auftreten von Schatten-IT entgegenwirken, bei der wertvolles Wissen möglicherweise nur den Verfassern zugänglich ist. Um dieses zu umgehen, bietet bspw. die Verwendung eines firmeninternen, in ein *Enterprise 2.0*-System eingebundenes IM die Möglichkeit, Erkenntnisse der Kommunikation allen Mitarbeitern schnell zugänglich zu machen und durch deren Dokumentation den Einstieg in die Thematik zu einem späteren Zeitpunkt zu erleichtern.

2.2 Home-Office2.0

Home-Office2.0 soll die Möglichkeit der Steigerung der Work Life Balance bieten, indem Arbeit und Privates besser in Einklang gebracht werden. Manchmal ist eine räumliche Entfernung aufgrund der Unternehmensstruktur zwingend notwendig. Diese Zerstreuung der Mitarbeiter über verschiedene Standorte nennt man verteilte Teams. In solch einer Konstellation werden die Aufgaben meistens an einen der Standorte gebunden. Ist dies nicht der Fall, spricht man von einem verstreuten Team. Gerade für Unternehmen bietet es einen Mehrwert kostbare Experten standortunabhängig einsetzen zu können [10]. Soll dieser Mehrwert genutzt werden, so ergeben sich verschiedene Herausforderungen an das Projektteam [32]:

- keine direkte Kommunikation
- kein spontaner Informationsaustausch unter den Kollegen
- Besprechungen nur per Telefon oder Videokonferenz

Gerade in agilen Projekten wird die direkte Kommunikation als ein zentraler Erfolgsfaktor des Projektes angesehen [1].

Für eine effektive Kommunikation im Team sind gefühlte Nähe und Vertrauen notwendig [10]. Vertrauen lässt sich am besten im direkten Gespräch von Angesicht zu Angesicht aufbauen. Studien zeigen, dass elektronische Kommunikation gut funktioniert, wenn vorher eine persönliche Beziehung aufgebaut wird [29].

Daher wird im *Home-Office2.0*-Szenario davon ausgegangen, dass sich das Team regelmäßig vor Ort trifft. Für die Zeit, in der das Team verteilt arbeitet, soll eine virtuelle Nähe die Arbeit und Kommunikation unterstützen. Dafür muss sich jedes Teammitglied der Arbeit der Kollegen bewusst sein. Bewährte Kommunikations- und Kollaborationsplattformen sollen hierbei genutzt und integriert werden. Die Diskussionen stehen dabei im Kontext einer Aufgabe. Dadurch dass die Informationen miteinander verknüpft werden, soll es, an der Diskussion unbeteiligten oder auch später ins Projekt kommenden

Mitarbeitern, erleichtert werden dem Diskurs zu folgen. Werden die für eine Aufgabe relevanten Informationen gesammelt und bereitgestellt, kann ein Mitarbeiter zudem leichter mit Unterbrechungen umgehen [32].

2.3 Wissensarbeiter

Der Begriff *Wissensarbeiter* (engl. „Knowledge Worker“) wurde 1959 von P. Drucker geprägt. Er bezeichnet Arbeiter, deren Tätigkeiten die Erstellung und Verarbeitung von Wissen sind [9].

Die Definition des Begriffs wurde bis heute von verschiedenen Forschern weiterentwickelt. Eine treibende Kraft ist Thomas Davenport. Er definierte 1996 vier Kerntätigkeiten von *Wissensarbeitern*:

> *„First, employees may be primarily engaged in finding existing knowledge. Secondly, employees may be involved in creating new knowledge. Existing knowledge be packaged by employees for other consumers or may be applied to a production process or problem.“* [41]

Außerdem identifizierte er folgende Arbeitsfelder:

- Recherche
- Produktentwicklung
- Marketing
- Ausbildung
- Beratung

Wissensarbeiter erzeugen aus ihrem Wissen wieder neues Wissen. Sie entwickeln es stetig weiter und passen es kontinuierlich an. Um *Wissensarbeiter* in einem Unternehmen produktiv einsetzen zu können, existieren verschiedene Anforderungen an das Unternehmen. Ebenso werden auch Anforderungen an *Wissensarbeiter* gestellt, um sich in das Unternehmen und dessen Arbeitsabläufe zu integrieren. Zwei wesentliche Anforderungen an *Wissensarbeiter* sind Teamwork und Wissensmanagement. Das Unternehmen kann zur Erfüllung dieser Anforderungen einen geeigneten Rahmen bieten. Die Anforderung Teamwork beschrieb Willke 1998:

> *„Die Erarbeitung und Weiterentwicklung von Wissen wird von mehreren Menschen gemeinsam durchgeführt. Dies stellt hohe Anforderungen an die Kommunikationsfähigkeit und Sozialkompetenz eines Wissensarbeiter. Es ist nicht nur wichtig Wissen zu erlangen, er muss auch in der Lage sein, dieses anderen zu vermitteln. Damit wird Wissensarbeit zu einem soziologischen Thema, in dem untersucht wird wie Menschen sich für erfolgreiche Wissensarbeit organisieren können.“* [50]

Für das Unternehmen bedeutet dies, den Wissensaustausch der Kollegen untereinander aktiv zu fördern. Darüber hinaus muss es eine Infrastruktur für das Wissensmanagement bereitstellen.

Letzteres ist besonders wichtig, da der Mitarbeiter für die Erfüllung seiner Aufgaben nicht das gesamte Wissen vorrätig haben kann. Er benötigt Zugriff auf ausgelagertes Wissen. Diese Informationen beschreibt Andy Hunt als den Exocortex. In ihm befinden sich die Informationen, die wir aus verschiedenen Medien für die Erweiterung des Gedächtnisses heranziehen bspw. aus Büchern, Notizen oder dem Internet [22].

2.3.1 Experte

Jeder Mitarbeiter eines Unternehmens arbeitet an bestimmten Problemen und Aufgaben, somit erarbeitet er sich ein Wissen in speziellen Themenbereichen. Das Wissen ist auf die Mitarbeiter im Unternehmen unterschiedlich verteilt. Einige von ihnen entwickeln sich zu Experten in ihrem jeweiligen Themengebiet. In Wikipedia wird Experte als ein Schlagwort definiert, das eine Person bezeichnet, welche *„[...] über überduchschnittlich umfangreiches Wissen auf einem oder mehreren bestimmten Sachgebieten oder über spezielle Fähigkeiten verfügt“* [47].

Durch einen Lernprozess kann sich der Mitarbeiter vom Anfänger zum Experten entwickeln. In diesem Entwicklungsprozess wird nicht nur neues Wissen angesammelt. Durch den Lernprozess ändert sich auch das Verständnis für die Thematik. Es werden neue Strategien zur Lösung von Problemen oder für das Lernen entwickelt. Das mentale Modell über die Materie ändert sich dabei. Dies hat deutliche Konsequenzen auf die Art wie ein Mitarbeiter mit Problemen umgeht und seine Lösungsstrategien vermitteln kann. Durch Erfahrungen bildet sich eine Intuition, die für Lösungsansätzegenutzt werden kann. Diese kann der Mensch nicht einfach erklären, da er auf *sein Gefühl* hört [22].

Aus den Änderungen des mentalen Modells bei der Entwicklung vom Anfänger zum Experten, lassen sich Stufen der Expertise ableiten. Ein Modell zur Beschreibung dieser Stufen wurde von den Brüdern Hubert und Stuart Dreyfus entwickelt. Das Dreyfus-Modell differenziert dabei fünf Stufen.

Anfänger – Anfänger haben wenig oder keine Erfahrung in dem relevanten Thema. Dadurch sind sie in Fehlersituationen leicht zu verwirren und haben nicht die Fähigkeit das Problem zu lösen. Sie benötigen für ein erfolgreiches Lernen auch bei kleinen Schritten Erfolgserlebnisse. Für eine effektive Arbeit in dem Themenbereich benötigen sie klare kontextfreie Regeln.

Fortgeschrittene Anfänger – Diese Gruppe kann außerhalb der Regeln in dem Themenbereich agieren. Fortgeschrittene Anfänger haben erste Erfahrungen in dem relevanten Themenbereich oder in verwandten Themen. Dies ermöglicht es ihnen sich zur

Lösung ihres aktuellen Problems, gezielt Punkte aus dem Bereich herauszusuchen. Sie kennen jedoch die zugrunde liegende Theorie und das Gesamtbild nicht. Auch diese Gruppe kann nicht eigenständig mit Fehlersituationen umgehen.

Kompetent Handelnder – Mitarbeiter der dritten Stufe können konzeptionelle Modelle für den Domainbereich entwickeln und mit diesen weiterarbeiten. Diese Gruppe ist auch in der Lage selbstständig mit Fehlersituationen umzugehen und dafür Lösungen zu entwickeln. Ihre Arbeitsweise ist mehr durch Erfahrung und eigenständige Planung geprägt. Dennoch fehlt ihnen die Erfahrung, um erkennen zu können welche Details sie bei der Problemlösung fokussieren müssen. Mitarbeiter dieser Stufe sind gut zur Einweisung von Anfängern in den Themenbereich geeignet.

Gewandt Handelnder – Mitarbeiter der Stufe gewandt Handelnder kennen das Gesamtbild des Themenbereiches. Sie sind in der Lage zu kritischer Selbstreflexion und können ihre Lösungsstrategien anpassen. Neben dem Lernen aus eigener Erfahrung, können sie auch aus Erfahrungsberichten von Kollegen lernen. Diese Gruppe kann besser mit kontextsensitiven Regeln arbeiten, da sie über genügend Erfahrung verfügen, um situationsabhängige Entscheidungen zu treffen.

Experte – Experten verfügen über einen reichen Erfahrungsschatz, der sie bei ihrer Arbeit unterstützt und lenkt. Ihre Entscheidungen und Lösungsideen werden verstärkt durch ihre Intuition beeinflusst. Sie erkennen sehr schnell welche Details wichtig sind und welche nicht. Dadurch arbeiten sie besonders effektiv bei der Lösung von Problemen.

Das Modell zeigt, wie sich der Lernprozess verändert und welche Hilfsmittel gebraucht werden. Die Regeln eines Anfängers müssen kontextfrei sein, während ein Mitarbeiter der vierten Stufe besser mit kontextsensitiven Regeln arbeiten kann. Berücksichtigt man auf welcher Stufe des Dreyfus-Modell sich ein Mitarbeiter befindet, kann man ihm geeignete Ansprechpartner und Hilfsmittel zur Verfügung stellen. So ist ein Experte mit seinen intuitiven Entscheidungen nicht unbedingt geeignet, um einem Anfänger kontextfreie Regeln zu geben [8].

2.3.2 Alte Projekte oder „Old Software never dies“

Das Dreyfus-Modell zeigt die Bedeutung von Erfahrungen auf. Diese Erfahrungen werden bei der Arbeit erworben und führen zu Ergebnissen, die einen Mehrwert für das Unternehmen bedeuten. Gerade erfahrene Mitarbeiter werden benötigt, um unerfahrene Kollegen

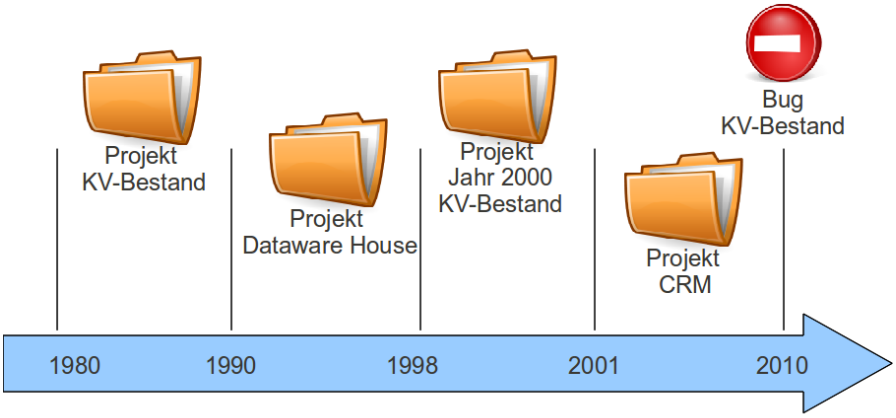

Abbildung 2.1: Szenario für die Arbeit in alten Projekten

zu unterstützen oder neue Projekte voranzutreiben. Sie tragen zum Wissenstransfer und der Etablierung einer gemeinsamen Vision bei [10].

Dies führt dazu, dass *Wissensarbeiter* mit der Zeit in mehrere Projekte involviert werden.

Bei der Softwareentwicklung bedeutet dies, dass Softwareentwickler bei der Arbeit an neuen Projekten durch Wartungsaufgaben an produktiven Systemen unterbrochen werden. Software wird länger als geplant produktiv betrieben, da sie im Gegensatz zu Hardware keine Materialermüdungen aufzeigt, die einen Austausch notwendig machen. Software unterliegt dennoch einem Alterungsprozess, da sich die Umgebung, in der die Anwendung läuft, ändert oder neue Anforderungen gestellt werden. Ein bekanntes Beispiel hierfür ist das Jahr 2000 Problem.

In Abb. 2.1 ist ein entsprechendes Szenario dargestellt. Ein Entwickler hat zunächst in einem Projekt zur Realisierung eines Krankenversicherungs-Bestandssystems gearbeitet. Danach wechselt er zu einem anderen Projekt. Für die Jahr 2000 Umstellung muss er wieder an dem alten System arbeiten. Zu einem späteren Zeitpunkt kommt ein erst kürzlich entdeckter Fehler in dem Bestandssystem hinzu.

3 Visionen

In diesem Kapitel werden die Visionen und Ideen, die zu dem Projekt `LuPanKu` geführt haben dargelegt. Durch intensive Zusammenarbeit der Projektmitglieder wurden die Möglichkeiten, Herausforderungen und Grenzen des Projekts abgesteckt. Dadurch entstand ein Katalog von Fakten der aktuellen Situation in Softwareprojekten, als auch in der Anwendung von Enterprise2.0-Technologien. Aus diesem Katalog wurden neben den konzeptionellen Ansätze für den Umgang mit Enterprise2.0-Technologien im Home-Office-Bereich auch Anforderungen und erste Schritte für die Entwicklung eines Softwaresystems erarbeitet. Die Umsetzung dieser Ideen ist im Projektrahmen (siehe Kapitel 4) konkretisiert wurden.

3.1 Das „Social Network“ automatisiert aufzeigen

Im Szenario *Home-Office2.0* geht es unter anderem darum, das Bewusstsein für die Mitglieder und deren Wissensaustausch in verstreuten Teams (vgl. Abschnitt 2.2) zu verbessern. Dafür wird das soziale Netzwerk des Teams betrachtet, welches direkten Einfluss auf die Struktur des Systems hat. Dieser Zusammenhang wird Conway's Law genannt.

Melvin Conway formulierte es 1968 so:

> *„Organizations which design software systems are constrained to produce design which are copies of the communication structures of the organizations.“* [35]

Dieser Zusammenhang zwischen der sozialen Struktur eines Teams und dem Design von Software wird *Socio-technical Congurence* genannt und ist seit 2008 ein aktuelles Forschungsthema des Software Engineering [4]. Dabei wird der Deckungsgrad und die Wechselwirkung zwischen der Struktur eines Softwaressystems und dem sozialen Netzwerk des Teams betrachtet.

In einem sozialen Netzwerk wird die Nähe und Art der sozialen Bindungen von Menschen untereinander betrachtet. Die Qualität der Bindungen beeinflusst die Wahl des Kollegen, mit dem ein Mitarbeiter die Lösung eines Problems erarbeitet [7].

Die Informationen über das soziale Netzwerk in einem Projekt geben der Projektleitung Hinweise darauf, wie sie den Entwicklungsprozess oder die Projektstruktur optimieren können [46]. In agilen Projekten kann in der Retrospektive [23] ein menschliches Soziogramm erstellt werden. Dabei verteilen sich die Mitarbeiter des Teams im Raum. Jeder einzelne soll sich dabei relativ zu seinen Kollegen positionieren. Die Regel dabei lautet,

sich umso näher neben jene Kollegen zu platzieren, je enger sie mit diesen zusammenarbeiten und entsprechend umgekehrt. In verteilten Teams ist dies aufgrund der räumlichen Entfernung nicht möglich und muss durch eine Umfrage ersetzt werden [10].

Eine Alternative zu der Umfrage ist die Betrachtung der vom Team erzeugten Artefakte. Typische Artefakte in einem Software Projekt sind dabei:

- Dokumentation
- Anforderungen
- Fehler
- Quellcode

Für `LuPanKu` müssen diese Artefakte revisionssicher durch Werkzeuge verwaltet werden. Diese Anforderung ist in den meisten Projekten erfüllt. Als typische Werkzeuge hierfür werden von `LuPanKu` folgende betrachtet:

- Wikis
- Anforderungs- und Fehlermanagement
- Versionsverwaltungssysteme

Diese Werkzeuge speichern die Aktivitäten der Nutzer mit einem Zeitstempel ab. Aus diesen Informationen und den Abhängigkeiten der Artefakte untereinander kann das soziale Netzwerk abgeleitet werden [46].

Die Vision von `LuPanKu` ist es, das soziale Netzwerk aus den Ergebnissen des Entwicklungsprozesses heraus zu ermitteln und aufzuzeigen. Dadurch soll dem Team die Chance gegeben werden, seine Struktur und den Prozess besser anzupassen.

3.2 Expertensuche

Wie in Abschnitt 2.3.1 erläutert existieren viele verschiedene Expertisen in einem Unternehmen. Jedoch ist die Existenz und die Ansiedlung des Wissens häufig nicht bekannt. Diese Problematik wächst mit der Größe des Unternehmens. In kleinen Unternehmen ergeben sich mehr Möglichkeiten für den Erfahrungsaustausch der Mitarbeiter untereinander. In großen Unternehmen ist dies in der Qualität nicht mehr möglich, da sich ein Großteil der Mitarbeiter nicht persönlich kennt. Diesem Umstand begegnet man mit Organisationsstrukturen, in denen Kompetenzzentren für identifizierte Problembereiche gegründet werden. Trotz dieser Strukturen werden Lösungen redundant entwickelt, da die Problemstellungen nicht in jedem Fall einem Kompetenzzentrum zugeordnet werden können.

Jede erarbeitete Problemlösung sollte gut dokumentiert werden und zentral abrufbar sein. Jedoch sind Dokumentationen häufig zu oberflächlich oder nicht vollständig. Ein Grund hierfür liegt darin, dass ein Experte über tiefes Wissen zu dem Thema verfügt und

die Informationen, die ein Anfänger benötigt, nicht mehr als relevant erachtet. Außerdem werden die Dokumentationen nach der initialen Erstellung nicht immer weitergepflegt.

Bei einem direkten Gespräch können Fragen gestellt werden. Dieses führt zu einer für den Empfänger besser verständlichen Erklärung. Zudem wird nicht dokumentiertes Wissen transferiert. Lars Mählmann fasst diesen Umstand in seiner Arbeit „*Deliver who I mean*“ [30], wie folgt zusammen: *„Viele der Informationen sind schneller verfügbar, wenn man die richtigen Personen kennt, bzw. fragt.“*

Wie soll die *richtige* Person angesprochen werden, wenn sie nicht bekannt ist. Es ist möglich, bei einer bestehenden Dokumentation den Autor anzusprechen, jedoch muss dieser nicht die Expertise haben. Er kann sich schon lange aus dem Thema entfernt oder es abgegeben haben oder es existieren mehrere Experten im Unternehmen für diese Domäne. Diese Umstände zeigen auf, dass der einfache Ansatz den Autor eines entsprechenden Dokumentes zu nennen, nicht ausreicht. Auch Lars Mählmann erkennt: *„Die Herausforderung besteht darin, die richtige Person zu finden“*. Diese Aufgabe soll durch `LuPanKu` vereinfacht werden.

Die Abbildung 3.1 symbolisiert auf welche Weise die Vision der Expertensuche funktionieren kann. Ein Fragesteller (asker) stellt eine Suchanfrage an das System (abgebildet durch ein Zahnrad). `LuPanKu` analysiert diese auf Grundlage der extrahierten Daten aus den Quellsystemen, dem *knowledge space*. Der *knowledge space* ergibt sich aus allen Exocortex Informationen eines jeden Teammitglieds. Er enthält das gesamte Teamwissen, das in elektronischer Form verfügbar ist. Die Analyse beantwortet die Suchanfrage sowohl, wie eine herkömmliche Suchmaschine, mit inhaltlichen Antworten, z.b. Artikeln, Dokumenten, Wikieinträgen, als auch mit dem besten Ansprechpartner für diese Fragestellung. Die Ansprechpartner können aus Einzelpersonen oder Gruppen bestehen.

Für Profis und Experten ist das Beschäftigen mit einfachen Fragestellungen weniger attraktiv. Daher reagieren sie bei Anfragen von Ihnen unbekannten Kollegen eher abweisend [22]. Allerdings ist die Hilfsbereitschaft bei Personen zu der sie einen persönlichen Kontakt pflegen (reziproke Beziehung) höher [39].

Aus diesem Grund berücksichtigt `LuPanKu` bei der Expertensuche das soziale Netzwerk des Benutzers. Sind Experte und Fragesteller nicht direkt miteinander bekannt, so ist eine Anzeige mehrerer Kollegen geplant, die sowohl mit dem Fragesteller als auch mit dem Experten bekannt sind.

Der *Analyzer* für die Expertensuche agiert sowohl proaktiv als auch reaktiv. Er analysiert proaktiv das Wissen der Mitarbeiter und bewertet ihren Wissensstand in einer Domäne. Bei Bearbeitung der Suchanfrage muss der *Analyzer* reaktiv agieren und die Antworten dynamisch ermitteln.

3.3 Verbesserung Home-Office2.0

Wie im Kapitel 2.2 beschrieben, soll beim Szenario *Home-Office2.0* durch Kommunikations- und Kollaborationswerkzeuge eine virtuelle Nähe zwischen Teammitgliedern erzeugt

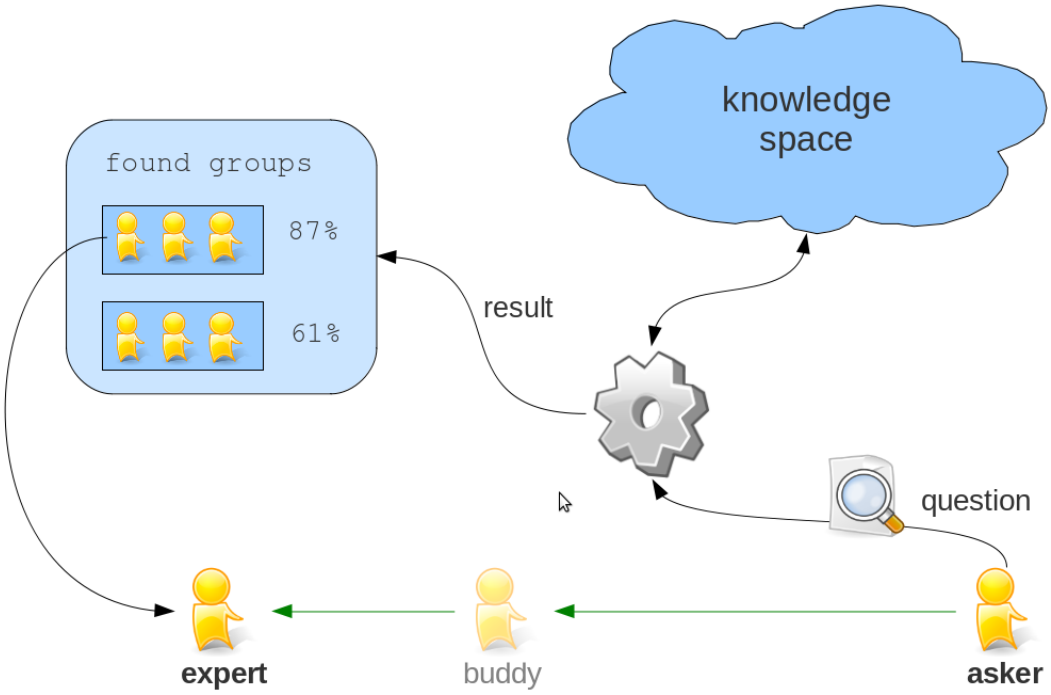

Abbildung 3.1: Schematische Darstellung der Expertensuche

werden. Ein Ansatz, um dies zu unterstützen, ist die Kontexterfassung. Durch diese kann der Diskurs verschiedener Arbeitsplätze (zu Hause, Mobil) sowohl zur selben, als auch zu verschiedenen Zeiten besser verfolgt werden. So kann ein Mitarbeiter beispielsweise das Design einer Komponente eines Kollegen besser nachvollziehen, wenn der Entscheidungsprozess (Diskussionen, Recherchen, verworfene Entwürfe) an das Design gebunden ist.

Eine effiziente Kontexterfassung ist nicht nur für das Szenario *Home-Office2.0* von Interesse, sondern auch für andere Bereiche des Arbeitslebens. So arbeiten viele Mitarbeiter in mehreren Projekten zur selben Zeit bzw. über die Zeit. Der Kontext vereinfacht den Einstieg in ein zurückliegendes Projekt (vgl. Kapitel 2.3.2) und den Wechsel zwischen verschiedenen aktuellen Projekten. Dies gilt auch für die Einarbeitung neuer Mitarbeiter.

Ein entscheidender Vorteil der Kontexterfassung ist, dass an verschiedenen Standorten arbeitende Mitarbeiter, besser in den sozialen Bezug des Untenehmens bzw. des Projektes eingebunden werden. Diese Bindung wird durch das Wissen der einzelnen Mitarbeiter über die aktuellen Tätigkeiten und Probleme der Teammitglieder hergestellt. Die Stati und Tätigkeiten der Teammitglieder könnten hierfür in einem virtuellen Raum dargestellt werden.

4 Projektrahmen

Damit aus den vielfältigen Ideen, die in Kapitel 3 vorgestellt wurden, ein konkretes Projekt entstehen kann, ist es nötig einen klaren Projektrahmen zu erstellen. Im Gegensatz zu den Projektvisionen muss der Projektrahmen verständliche und detaillierte Anforderungen definieren. Die Anforderungen sind dabei die Lösung eines Teilproblems der Visionen und sind in einer festgelegten Zeit zu erarbeiten.

Ein Teil des Rahmens besteht in der Festlegung des Anwenderkreises. Da Begriffe wie *Enterprise2.0* oder *Home-Office2.0* in der Regel für ein breites Spektrum von Anwendern (bzw. einer Community) definiert sind, wird für das Projekt `LuPanKu` aussschließlich das Anwenderprofil des Softwareentwicklers berücksichtigt. Somit soll `LuPanKu` also der Unterstützung in Software-Projekten dienen und gezielt von Softwareentwicklern genutzt werden.

In dem ersten Release des `LuPanKu`-Projekts stand vor allem die Planung und Implementierung eines soliden und leicht erweiterbaren Grundsystems im Vordergrund. Das Grundsystem soll dabei folgende Eigenschaften aufweisen:

- Extraktion von heterogenen Daten aus Fremdsystemen
- Aufbereitung der Daten auf Basis einer erweiterbaren Ontologie
- Archivierung der aufbereiteten Daten
- Analyse, Auswertung und Präsentation der Daten

Das damit geforderte Grundsystem bildet eine große Schnittmenge mit den sog. *Information-Retrieval-Systemen* [48]. Somit wurden einige Architektur- bzw. Systementscheidungen auf Grundlage von bewährten Konzepten aus dem Bereich *Information-Retrieval* getroffen.

Ein weiteres Ziel für das erste Release ist eine einfache Umsetzung einer Expertensuche, welche auf den gesammelten und aufbereiteten Daten basiert. Damit soll vor allem eine erste Sicht auf den Datenbestand implementiert werden, welche exemplarisch für weiterführende Projekte genutzt werden kann.

Desweiteren wurde festgelegt, welche fremden *Enterprise2.0*-Systeme innerhalb des ersten Release angebunden werden. Dabei entschieden sich die Projektmitglieder aufgrund eigener Erfahrungen für Systeme, die häufig als Werkzeuge in Softwareprojekten eingesetzt werden:

- Wiki-Systeme (vertreten durch *Confluence* [25])

- Ticket- bzw. Bugtracking-Systeme (vertreten durch *Trac* [37] und *Jira* [26])
- Versionsverwaltungssysteme (vertreten durch *Subversion* [6])

Für das `LuPanKu`-Projekt wurde als Implementierungssprache Oracle Java festgelegt. Für die Entwicklung wurde als Datenbankmanagementsystem HSQLDB eingesetzt, welche ggf. für den Produktivbetrieb ohne Einschränkungen ausgetauscht werden kann.

5 Architektur

Das System `LuPanKu` soll verschiedene Systeme auswerten und anpassbar an unterschiedliche Projektbedürfnisse sein. Als grundlegendes Architektur-Muster wurde *Online Analytical Processing* (kurz *OLAP*) gewählt. Dadurch werden die zu untersuchenden Systeme nicht durch aufwändige Analysen belastet [5].

Die Komponenten der Architektur werden als austauschbare Plug-Ins entwickelt. Dadurch wird das System an konkrete Projektbedürfnisse anpassbar und kann individuell erweitert werden.

Ein Plug-In implementiert eine oder mehrere Schnittstellen und registriert dann seine Implementation als Service in einer Service Registry. Es wird dabei eine 1..n Beziehung zwischen Schnittstelle und Implementation angeboten. Dadurch können Komponenten beim Lookup eines Services anhand von Eigenschaften entscheiden welche Implementation im aktuellen Systemkontext besser geeignet ist. Dies ermöglicht spezialisierte Lösungen für ein Werkzeug auf mehreren Schichten der Architektur.

5.1 Systemüberblick

Der in Abbildung 5.1 dargestellte Systemüberblick des `LuPanKu`-Systems, lässt sich in drei Funktionsabschnitte unterteilen (wie in der Grafik dargestellt). Unter dem Begriff *Extraction* können alle Systembestandteile zusammengefasst werden, welche Rohdaten aus Fremdsystemen sammeln, diese in sogenannte *Changes* gruppieren und zur weiteren Verarbeitung bereit stellen. Die Kategorie *Preparation* stellt Module für die Aufbereitung und Formatierung der Rohdaten in ein eigenes Datenmodell bereit. Darunter fällt auch das Modul *Persistence*, welches das Datenmodell persistieren und somit einen dauerhaften Datenbestand aufbauen kann. Der dritte Abschnitt, *Interpretation*, beschreibt Module, welche auf den bestehenden Daten Analysen durchführen, Daten interpretieren und diese über eine Benutzeroberfläche (UI) präsentieren. Die Analyse und Interpretation wird von sogenannten *Analyzern* durchgeführt. Ein *Analyzer* ist dabei für eine bestimmte Fragestellung implementiert, welche an den Datenbestand gestellt werden soll. Die errechneten „Antworten“ werden per Application-Server als Webseite dargestellt. Somit kann ein Nutzer von `LuPanKu` seine Anfragen an das System über einen herkömmlichen Webbrowser stellen.

Für eine bessere Sicht auf das *Big Picture* werden die einzelnen Komponenten der verschiedenen Funktionsbereiche nachfolgend aufgelistet und skizziert. Eine ausführliche Beschreibung ist in den jeweils angegebenen Abschnitten zu finden.

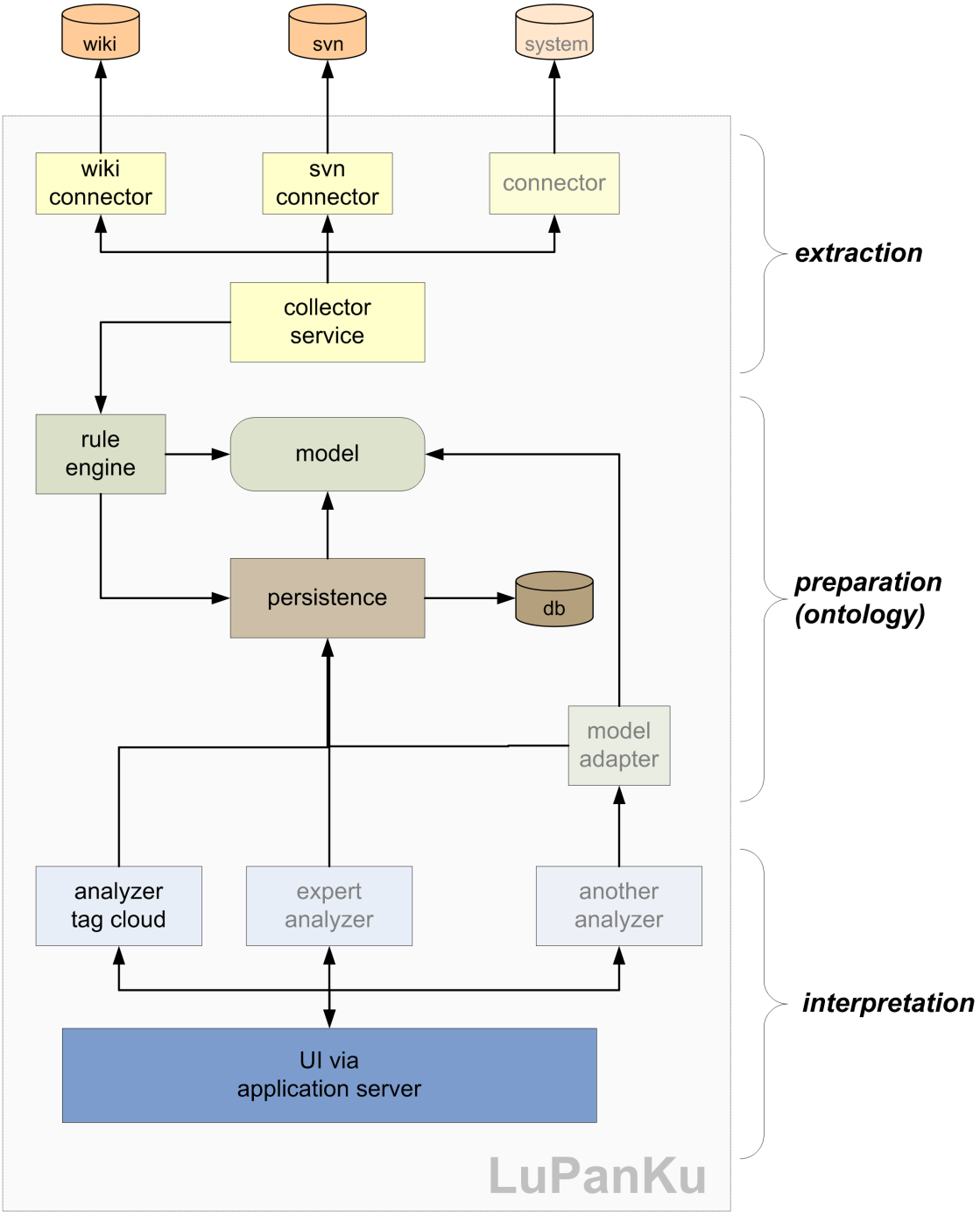

Abbildung 5.1: Schematische Darstellung der Systemarchitektur des `LuPanKu`-Systems

Extraction

Connector – Ein *Connector* ist eine Schnittstelle zwischen dem *Collector Service* und einem bestimmten Fremdsystem. Wird der *Connector* aufgerufen, sucht der *Connector* im Fremdsystem nach geänderten oder neuen Daten, die es seit seiner letzten Bearbeitung gibt. Diese Datensätze werden als *Changes* bezeichnet, also dem Datendelta zwischen zwei Zeitpunkten. Der *Connector* wird in Abschnitt 5.4 und 6.3 beschrieben.

Collector Service – Diese Komponente kennt alle verfügbaren *Connectoren* und Konfigurationen der abzufragenden Fremdsysteme. Je nach Einstellung, wird der Collector über einen automatischen Job oder manuell gestartet. Der Collector startet wiederum alle *Connectoren* und sammelt ihre *Changes* ein. Die *Changes* können dann von der *Rule Engine* weiterverarbeitet werden.

Preparation

Model – Das *Model* ist das eigene Datenmodell von `LuPanKu`. Dieses generische Datenmodell dient zur Abbildung der Daten und deren Beziehungen aus den Fremdsystemen. Somit werden verschiedene heterogene Datensätze in dieses eine homogene Datenmodell transferiert und mit zusätzlicher Semantik angereichert. Auf dem Datenmodell können so Analysen durchgeführt werden, die auf den einzelnen Fremdsystemen bzw. deren Daten nicht möglich sind. Das sogenannte *Model* wird in Abschnitt 5.2 beschrieben.

Rule Engine – Dieses Modul stellt Regeln bereit, welche die Rohdaten in das eigene Datenmodell übersetzten können. Über einen Mechanismus ist es möglich, für einen bestimmten *Connector* neue Regeln zu implementieren, welche ausschließlich mit *Changes* aus dem eigenen Fremdsystem arbeiten. Die *Rule Engine* wird im Abschnitt 5.5 und 6.4 beschrieben.

Persistence – Das *Persistence*-Modul sorgt für die Persistierung des Datenmodells. Dabei ist das Datenmodell selbst mit Informationen für diese Schicht angereichert. Somit können zur Laufzeit normale Java-Objekte aus den Rohdaten aufgebaut werden und mit Hilfe dieses Moduls einfach persistiert werden. Analog können aus den Daten in der Datenbank wieder Java-Objekte erzeugt werden. Das Datenbankmanagementsystem selbst kann dabei i.d.R. beliebig gewählt werden (bspw. *MySQL*, *Postgres*, *HSQLDB* oder *DB2*). In Abschnitt 6.2 wird dieses Modul ausführlicher erläutert.

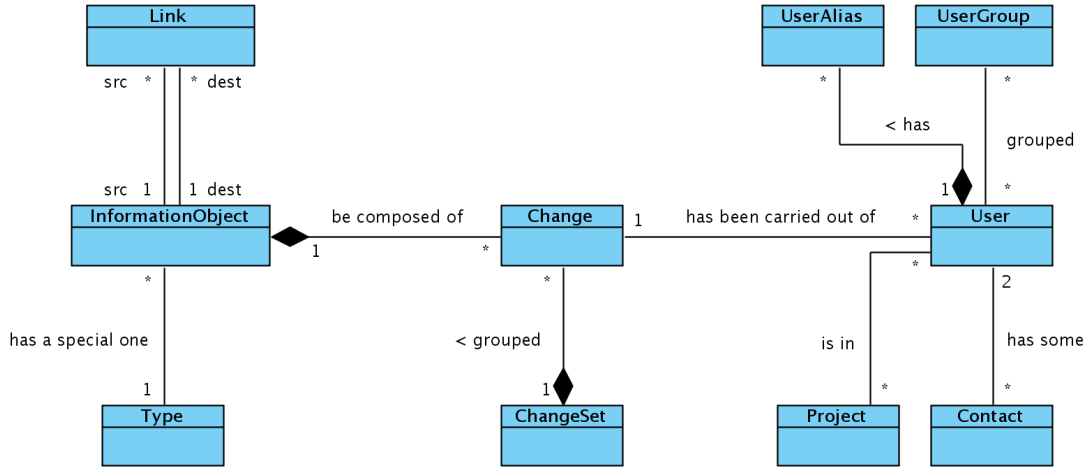

Abbildung 5.2: UML-Klassendiagramm des verwendeten Datenmodells

Interpretation

Analyzer – Ein *Analyzer* ist eine spezielle Implementierung, um eine bestimmte Fragestellung auf Grundlage der verfügbaren Daten zu beantworten. Somit führt der *Analyzer* Berechnungen auf den vorhandenen Daten aus und erstellt ein Ergebnis, welches über ein UI dem Nutzer präsentiert werden kann. Der *Analyzer* wird in Abschnitt 5.6 beschrieben.

Model Adapter – Für die stark spezialisierten *Analyzer* kann es nötig sein, das vorhandene Datenmodell in eine geeignetere Form zu transformieren. Diese Aufgabe übernimmt der sogenannte *Model Adapter* und bildet damit eine Schnittstelle zwischen einem bestimmten *Analyzer* und dem eigentlichen Datenmodell. Diese Komponente wurde in dem aktuellen `LuPanKu`-Release noch nicht entwickelt, ist allerdings als zukünftiges Feature für das nächste Release geplant, wie in Kapitel 8 beschrieben ist.

5.2 Datenmodell

Das Datenmodell oder kurz *Model* ist Teil der Kernanwendung (vgl. Abschnitt 5.1) und stellt eine Struktur für die Darstellung der gesammelten Daten bereit.

Das Ziel dieser Systemkomponente ist die Bereitstellung der gesammelten Daten in einer allgemeingültigen, systemunabhängigen und verabredeten Form. Es soll die Daten von ihren Ursprungssystemen entkoppeln und für zukünftige Analysen und Betrachtungen

homogen zur Verfügung stellen (ein UML-Klassendiagramm des *Models* ist in Abb. 5.2 zu finden).

Das heißt, dass alle Rohdaten, die durch den *Collector Service* gesammelt werden, mit Hilfe der *Rule Engine* auf dieses *Model* abgebildet werden können. Dabei ist es unabhängig, aus welchem System die Rohdaten stammen. Das Design des Datenmodells versucht, mit einer geringen Anzahl von Entitäten und Assoziationen, ein möglichst generisches Modell darzustellen. Unstimmigkeiten beim Mapping der Rohdaten auf dieses *Model* werden nicht durch eine Anpassung des Datenmodells erreicht, sondern i.d.R. durch die *Rule Engine* ausgeglichen.

Desweiteren wurde die Entität `ModelObject` eingeführt, um bei Bedarf eine einfache und flexible Möglichkeit zu schaffen, jede beliebige Entität des *Models* mit neuen Attributen erweitern zu können. Jede Entität des *Models* ist somit eine Spezialisierung des `ModelObject`s. Über eine Schlüssel-Werte-Map ist es somit möglich, neue Attribute anzulegen und so eine gewünschte Model-Entität zu erweitern. Das Mapping von abweichenden Datentypen auf ein neues Attribut (bspw. Zahlen, Daten oder Listen), wird wiederum in der *Rule Engine* gelöst. Die Id des `ModelObject`s wird für eine Persistierung der Daten benötigt (siehe Abschnitt 6.2).

Nachfolgend werden alle weiteren Entitäten des *Models* einzeln aufgeführt und dessen Funktion erläutert:

InformationObject – Diese Entität stellt neben dem `User` eine der Hauptentitäten dar. Es handelt sich dabei um ein generisches `InformationObject`, also ein Objekt für die Präsentation einer universellen Information. Dabei kann es sich bei der Information um ein konventionelles Dokument (Wiki-Seite, Code-Datei), aber auch um ein Task-Ticket (z.B. aus *Trac* oder *Jira*) oder einen Termin aus einem elektronischen Kalender handeln. Dabei darf das `InformationObject` **nicht** als Kopie der eigentlichen Information verstanden werden. Es repräsentiert die Information aussschließlich durch Metadaten und Assoziationen zu anderen Entitäten. Sofern der Inhalt eines `InformationObject`s in das *Model* aufgenommen werden soll, muss dies über die Entität `Change` geschehen.

Link – Um einen Zusammenhang zwischen zwei `InformationObject`s darstellen zu können, kann diese Entität genutzt werden. Die Verbindung ist dabei unidirektional modelliert, somit können die Quelle und das Ziel explizit dargestellt werden. Dabei ist die Art des Zusammenhangs nicht näher definiert und sollte nicht mit einem tatsächlichen, technischen (Web)Link in Verbindung gebracht werden.

Type – Der Typ eines `InformationObject`s wurde bewusst als eigene Entität und nicht als Attribut des `InformationObject`s modelliert. `Type` soll funktional wie ein *Enum*, also eine Reihe von Konstan-

ten, eingesetzt werden. Damit soll die Doppeldefinition von InformationObject-Typen verhindert werden (bspw. „txt“ und „text“).

Change – Der `Change` ist immer ein fester Bestandteil eines `InformationObject`s. Der `Change` beschreibt den Zustand des `InformationObject`s zu einem bestimmten Zeitpunkt. So kann im `Change` bspw. der gesamte Inhalt oder ein Teil aus dem eigentlichen Dokument abgebildet werden. Durch zwei oder mehr `Change`s ergibt sich damit eine nachvollziehbare History von Änderungen an einem `InformationObject`. Über das Änderungsdatum und die Assoziationen zu `User` oder `ChangeSet`, lässt sich u.a. ein Nutzerkontext konstruieren und auswerten.

ChangeSet – Um eine logische Gruppierung von `Change`s zu erreichen, wurde die Entität `ChangeSet` eingeführt. Führt ein Nutzer bspw. eine Reihe von Änderungen an verschiedenen Dokumenten durch, die in einem gemeinsamen Kontext stattfinden, können diese mit dem `ChangeSet` kenntlich gemacht werden. Bspw. können diese Dokumentenzusammenhänge in Versionierungssystemen gefunden werden. So kann der Nutzer eine Reihe von Änderungen in verschiedenen Dokumenten einchecken, welche mit einem einheitlichen Kommentar und einer gemeinsamen Versionsnummer versehen werden.

User – Bei dieser Entität handelt es sich um die Präsentation eines menschlichen Anwenders oder ein anwendendes Fremdsystem aus Sicht des `LuPanKu`-Systems. Dabei ist es zweckmäßig, dass jeder Anwender nur durch eine Instanz von `User` abgebildet wird. Für verschiedene Fremdsystemnutzer eines Anwenders können Instanzen der Entität `UserAlias` eingesetzt werden.

UserAlias – Um ein Mapping zwischen einem einzigen `LuPanKu`-Nutzer und dessen Aliase in verschiedenen Fremdsystemen zu erreichen, wurde diese Entität realisiert. Ausführlich wird dieser Mechanismus in Abschnitt 5.3 erläutert.

UserGroup – Die `UserGroup` realisiert, ähnlich wie das `ChangeSet`, die Möglichkeit zusammenhängende `User` zu gruppieren. Dies erweitert die Möglichkeiten detaillierte User-bezogene Kontexte abzubilden.

Project – Dient ebenfalls zur Gruppierung von `Usern`, allerdings mit der semantischen Bedeutung der Projektzugehörigkeit, was explizit bei der Softwareenticklung der Fall ist.

Contact – Ein `Contact` ist, analog zum `Link`, eine Verknüpfung zwischen zwei `Usern`. Damit können bidirektionale Beziehungen zwischen Anwendern modelliert werden.

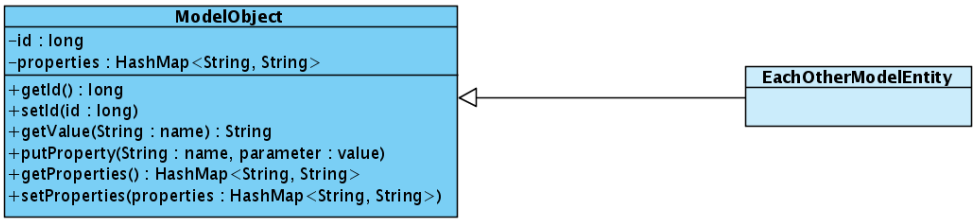

Abbildung 5.3: UML-Klassendiagramm der Klasse `ModelObject`

5.3 Benutzermanager

Der Benutzermanager ist ein schlichter Mechanismus, der die logische Verbindung zwischen einem im `LuPanKu`-System definierten Nutzer (Entität `User`) und dessen Nutzer-Alias (Entität `UserAlias`) in einem Fremdsystem herstellt. Auch wenn dies kein technisch komplexer Mechanismus ist, soll doch die Funktionsweise an dieser Stelle erläutert werden. Schließlich ist die korrekte semantische Bindung zwischen `InformationObject` und `User` sehr wichtig und würde im fehlerhaften Fall drastische Auswirkungen auf die Datenanalyse und -ansicht haben.

So hat jede `User`-Entität mindesten eine bis viele `UserAlias`-Entitäten. Bei der Verarbeitung der Rohdaten muss jeder ausgelesene Datensatz neben einem Informations-Objekt auch den dazugehörigen Besitzer beinhalten. Wird der User-Alias-Name im System gefunden, kann das betreffende `InformationObject` dem systemeigenen `User` zugeordnet werden. Ist der Nutzer noch nicht bekannt, so wird für den neuen `UserAlias` gleichzeitig ein neuer systemeigener `User` angelegt. Dieser trägt dann denselben Namen wie sein Alias. Somit ist der Einsatz von `LuPanKu` auch ohne eine vorherige Konfiguration möglich. Es empfiehlt sich allerdings besonders bei der Verwendung von mehreren `Connectoren` für jeden realen Nutzer auch einen einzigen System-Nutzer anzulegen und die zugehörigen `UserAlias`-Entitäten für die verschiedenen Systeme anzugleichen.

5.4 Connector API

Während der Arbeit an einem Projekt werden verschiedene Werkzeuge zur Kollaboration und Kommunikation verwendet (z.B. Wikis, Versionsverwaltungssystem, Bugtrackingsystem), die unterschiedliche Daten erzeugen. Um einen Kontext der aktuellen Arbeit erstellen zu können, müssen diese Daten aus den verschiedenen Systemen entnommen und auf eine einheitliche Struktur gebracht werden.

Die Grundidee hierfür ist eine einfache, allgemeingehaltene Schnittstelle, über die eine Verbindung zu verschiedenen Fremdsystemen erstellt wird. Durch eine Abfrage der

Änderungen der erstellten Daten in regelmäßigen Abständen kann ein Kontext gebildet werden.

Die Vorgehensweise, wie die Daten konkret ausgelesen werden, wird erst in den Implementierungen der *Connectoren* (z.B. Confluence, SVN, JIRA) individuell festgelegt. Dadurch ist das nachträgliche Anfügen weiterer Fremdsysteme ohne erheblichen Aufwand möglich.

5.5 Rule Engine

Für das tatsächliche Mapping von Rohdaten aus dem *Collector Service* auf das eigene *Model* (vgl. Abschnitt 5.2), ist das Modul *Rule Engine* zuständig. Zusammen mit dem *Model* bildet die *Rule Engine* eine Ontologie für das `LuPanKu`-System, mit der die Rohdaten aus den Fremdsystemen interpretiert werden können.

Dieses Modul hält Regeln bereit, um die eingehenden Rohdaten aus den Fremdsystemen zu interpretieren und zu formatieren. Unter Verwendung des eigenen Datenmodells werden diese optimierten Daten in die bereits bestehenden Datenstrukturen integriert und persitiert.

Die *Rule Engine* stellt dabei eine Schnittstelle bereit, die es den Entwicklern von neuen *Connectors* ermöglicht, eigene Regeln zu implementieren. Somit ist ein höchstes Maß an Flexibilität für die Aufbereitung der Daten vorhanden. Zusammen mit der *Property*-Erweiterung des *Models* (vgl. Abschnitt 5.2) können beliebige Datenstrukturen und Datenformen im `LuPanKu`-System umgesetzt werden.

Neben der erwähnten Schnittstelle wird allerdings eine Standardimplementierung der Regeln angeboten. Mit dieser können die meisten Rohdaten der Fremdysteme auf das *Model* gemappt werden. Für spezielle Datenausprägungen genügt dies in der Regel aber nicht.

5.6 Analyzer

`LuPanKu` soll Informationen aus den, mittels der *Connectors*, gesammelten Daten generieren. Diese Aufgabe übernehmen die Plug-Ins mit *Analyzer*-Funktionalitäten. Die Leitlinie für den Entwurf eines *Analyzers* ist die Beantwortung einer bestimmten Frage. Diese Leitlinie soll zu einem modularen und unabhängigen Entwurf des *Analyzers* führen. Die Aufgaben eines *Anaylzers* können mehrere Aspekte des `LuPanKu`-Systems betreffen. Aus diesem Grund reicht es nicht aus, eine Schnittstelle zu bedienen. Ein Plug-In, das *Analyzer*-Funktionen anbietet, hat die Möglichkeit, wie ein Eclipse Plug-In [15], mehrere Erweiterungspunkte des Systems zu nutzen. Für den *Analyzer* sind folgende Erweiterungspunkte vorgesehen:

Analyzer – Ein *Anaylzer*-Service wertet den bestehenden Datenbestand aus und generiert dadurch Informationen zur Beantwortung einer Benutzerfrage. Dafür werden zwei Arten der Implementation angeboten:

- *BatchAnalyzer*
  Dieser Typ wird als Hintergrundprozess gestartet und ist für langwierige Analysen gedacht.
- *OnlineAnalyzer*
  Dieser Typ führt seine Analyse auf Grund einer direkten Benutzerinteraktion aus und muss daher sein Ergebnis in einer kurzen Antwortzeit bereitstellen.

Model Adapter – *Anaylzer* erzeugen auf Basis der vorhanden Daten neue Informationen, die eventuell ebenfalls gespeichert werden sollen. Das *Model* kann mit den gewonnenen Daten vom *Anaylzer* ergänzt werden. Damit diese Erweiterungen keine Nebeneffekte auf andere Komponenten haben, wird das *Model* nach dem Adapter Muster vervollständigt [19].

UI Darstellung – Für die Darstellung der gewonnenen Informationen bringt der *Anaylzer* eine Benutzeroberfläche in Form eines Portlets mit.

Komplexere Fragestellungen kann ein *Anaylzer* durch die Verwendung von anderen *Anaylzer* beantworten. Für das `LuPanKu`-Projekt befinden sich folgende *Anaylzer* in der Planung oder Entwicklung:

TagCloudAnalyzer – Fragestellung: *Was sind aktuell bearbeitete Themen einer bestimmten Person?*; Modulkomplexität: *Gering*
Der *Anaylzer* erweitert die Benutzeroberfläche des *Benutzermanagers*, vgl. 5.3 und stellt eine Tag Cloud mit aktuellen Themen des Benutzers dar. Hierfür werden die aus dem Wiki System extrahierten Stichworte verwendet.

TaskArtefaktAnalyzer – Fragestellung: *Welche Artefakte wurden für eine Aufgabe erzeugt oder modifiziert?*; Modulkomplexität: *Gering*
In Kapitel 3.1 wurden mehrere Werkzeuge für die Softwareentwicklung vorgestellt. Für eine fachlichen Aufgabe können in mehren dieser Werkzeuge Artefakte erzeugt oder modifiziert werden. Der *TaskArtefaktAnalyzer* rekonstruiert den fachlichen Zusammenhang über die Werkzeuggrenzen hinweg. Dafür werden Aufgabenschlüssel in Kommentaren und Stichwörtern zu den Modifikationen gesucht.

DependencyChangesetAnalyzer – Fragestellung: *Welche Abhängigkeiten hat ein Entwickler in einer Software geändert?*; Modulkomplexität: *Gering*
Für die Erstellung eines *Socio-technical Networks* [46] werden die Softwareabhängigkeiten eines Check-In Vorgangs benötigt. Der *Anaylzer* betrachtet dafür die statischen Quellcode Abhängigkeiten der Änderungen an der Software.

ArtefaktBasedSocialNetwork – Fragestellung: *Wie ist das Soziale Netzwerk im Team aufgebaut?*; Modulkomplexität: *Hoch*
Dieser *Anaylzer* erstellt ein Soziales Netzwerk auf Basis der Aktivitäten der Benutzer in den Kollobarationswerkzeugen, vgl. 3.1.

ExpertAnalyzer – Fragestellung: *Wer ist ein Experte für ein bestimmtes Thema?*; Modulkomplexität: *Hoch*
Dieser *Anaylzer* soll zu einem bestimmten Thema passende Experten suchen, vgl. 3.2.

PathToExpert – Fragestellung: *Welche Verbindungen gibt es zu einem Experten?*; Modulkomplexität: *Hoch*
Die in Kapitel 3.2 dargestellte Suche nach Verbindungen zu einem Experten, löst dieser *Anaylzer* auf.

6 Implementierung

6.1 OSGi

Eine Anforderung an das System `LuPanKu` ist der modulare Aufbau durch Plug-Ins. Die Funktionalitäten der Plug-Ins sollen mittels Services angeboten werden. Im Java-Umfeld erfüllt die OSGi Technologie diese Anforderungen [3]. Bei OSGi handelt es sich um eine Spezifikation für einen dynamischen Components Container. Diese Spezifikation wurde 1999 für Embedded Systems entwickelt und wird mittlerweile in Enterprise Anwendungen wie der SOA Platform Swordfish [11] verwendet. Aktuell liegt das vierte Release der Spezifikation vor. Für das `LuPanKu`-Projekt sind dabei folgende Eigenschaften interessant:

OSGi Services
- OSGi stellt eine Infrastruktur zur Verwaltung von dynamischen Services bereit. Ein Service ist dabei ein POJO (Plain Old Java Object). Dieses Objekt wird unter einem Namen und mit Properties in der Service Registry angemeldet. Für einen Plug-In Mechanismus ist es eine bewährte Praxis, den Service unter dem Namen seine fachlichen Schnittstellen zu registrieren. Dabei können mehrere Services an einen Namen gebunden werden. Der Konsument kann dann anhand der Properties entscheiden, welche Service Implementation für ihn geeignet ist.

Komponenten Trennung
- In einem OSGi Container werden Komponenten voneinander isoliert. Dies wird durch einen eigenen Classloader pro Komponente erreicht. Damit eine Komponente Klassen von anderen Komponenten verwenden kann, muss sie diese explizit anfordern. Dabei werden nicht nur unterschiedliche Klassennamen berücksichtigt, sondern auch deren Versionen. Dies erlaubt es in einer Instanz des Containers mehrere Versionen einer Komponente zu betreiben. Dadurch bietet OSGi ein mit dem vom Buildsystem Maven [14] vergleichbares Dependency Management. Der Container stellt zur Laufzeit sicher, dass eine Komponente nur dann gestartet werden kann, wenn alle ihre Abhängigkeiten erfüllt sind. So wird verhindert, dass es zur Laufzeit zu ClassNotFound-Exceptions kommt.

Schnittstellendefinition – Die herkömmliche Steuerung für die Sichtbarkeit von Klassen in vielen Sprachen ist für den Aufbau von komponentenbasierten Systemen unzureichend. Martin Fowler schlägt daher die Einführung von *Published Interfaces* vor [17]. OSGi bietet eine solche Unterstützung für Komponenten. Der Entwickler entscheidet welche Packages andere Komponenten sehen können. Dadurch lässt sich eine Trennung zwischen Schnittstelle und Implementation erreichen, die zur Übersetzungs- und Laufzeit geprüft wird.

Dynamisches Komponenten – OSGi unterstützt das Installieren und Aktualisieren von Komponenten zur Laufzeit. Dadurch ist es nicht notwendig das System bei der Installation oder Aktualisierung eines Plug-Ins neu zu starten.

Es gibt verschiedene quelloffene Implementierungen dieser Spezifikation:

- Equinox vom Eclipse Projekt [16]
- Felix von der Apache Foundation [13]
- Knopperfish [2]

Für die Implementation von `LuPanKu` fiel die Wahl auf Equinox, da weitere Komponenten wie die Benutzeroberfläche mit Eclipse-Technologien [15] entworfen wurden. Zudem war im Projektteam Wissen über diese OSGi Implementation vorhanden und eine gute Unterstützung durch die Eclipse IDE gegeben.

6.2 Persistierung mit Hibernate

Wie in Kapitel 5 erwähnt, ist `LuPanKu` an die Architektur eines *OLAP*-Systems angelehnt. Für die durchzuführenden Analysen wird ein dementsprechender Datenbestand benötigt, welcher von den produktiv genutzten Daten entkoppelt ist. Dazu werden die durch den *Collector Service* gesammelten Rohdaten mit Hilfe der *Rule Engine* auf das *Model* abgebildet. Die Rohdaten werden in zeitlichen Intervallen (bspw. täglich) gesammelt und Änderungen am eigenen Datenbestand durchgeführt (wodurch u.a. die im Abschnitt 5.2 erwähnten `Change`-Entitäten gebildet werden). Damit dieser eigene Datenbestand jederzeit für eine Auswertung bereit steht, muss das gesamte *Model* inklusive Daten persistiert werden.

Zu diesem Zweck nutzt `LuPanKu` das Framework *JBoss Hibernate* [36] und die *JPA Annotations* [31]. Dabei wird das implementierte *Model*, welches einzig aus POJOs (*Plain Old Java Objects*) besteht, mit den JPA Annotations erweitert und so mit den benötigten Datenbankinformationen angereichert. Über die Hibernate-Konfiguration `hibernate.cfg.xml` werden die Klassen für die Persistierung bekannt gegeben. Beim Start des Frameworks wird somit aus den Informationen der Annotations und der eigenen Informationen

der Java-Klassen eine Datenschnittstelle zwischen dem Datenbankmanagementsystem und der Java-Applikation aufgebaut.

Mit Hilfe von Hibernate ist ein konstantes Arbeiten mit Java-Objekten auch bei Transaktionen mit der Datenbank möglich. Somit muss der Entwickler sich nicht um das Befüllen, Erstellen oder Sichern seiner Datenobjekte kümmern und keine eigenen SQL-Abfragen erstellen. Er kann seine Daten wie gewohnt auf objektorientierte Weise verwalten und nutzen. Allerdings stellt Hibernate trotzdem Funktionen für das Arbeiten mit nativen SQL-Statements zur Verfügung.

Für die Steuerung der Persistenzschicht und die Arbeit mit den Daten, stehen bei `LuPanKu` unter dem Package `org.aysada.lupanku.core.database` dafür folgenden Basisklassen bereit:

HibernateUtil – In der `HibernateUtil`-Klasse werden Methoden zur Steuerung von Hibernate bereitgestellt. Dazu gehören bspw. die grundlegenden Methoden `initialize()` und `shutdown()` zum Starten und Stoppen des Frameworks. Diese können an geeigneter Stelle in der eigentlichen Anwendung aufgerufen werden, um anschließend auf die persistenten Daten zugreifen zu können.

QueryManager – Diese Klasse implementiert alle Methoden die nötig sind, um auf gewünschte Datenobjekte zuzugreifen oder diese persitent zu machen. Oftmals können bei den `Get`-Methoden Beispielobjekte übergeben werden, zu denen versucht wird ein passendes „reales“ Objekt zu finden. So kann bspw. ein `UserAlias`-Objekt mit einem gesuchten Aliasnamen erzeugt und an die Methode übergeben werden. Diese sucht das dazu passende `User`-Objekt und gibt es, wenn eines gefunden wird, zurück.

6.3 Connector-Implementierung

Wie im Kapitel 5.4 beschrieben, soll es sich hierbei um eine einfache, allgemein gehaltene Schnittstelle zu den verschiedenen Kollaborations-Werkzeugen handeln.

Die Connector API deklariert zwei grundlegende Methoden für die Verbindung zu und die Datenerhebung von anderen Software-Systemen.

```
public void connect(ConnectorConfig config) throws Exception;

public List<Map<String, String>> getLastChanges() throws Exception;
```

Listing 6.1: Methoden Connector-API

Zum einen die `Connect`-Methode, in der eine Verbindung zu dem entsprechenden Fremdsystem aufgebaut wird. Als Parameter erwartet diese Methode ein *ConnectorConfig-*

Objekt. Die *ConnectorConfig*-Klasse muss vor dem Verbindungsaufbau für jedes Fremdsystem individuell konfiguriert werden und beinhaltet alle nötigen Verbindungsinformationen:

- den Benutzernamen zum Login in das Fremdsystem
- das Passwort passend zum Benutzernamen
- die URL zum Fremdystem (z.B. Confluence: http://www.webadr.com/rpc/xmlrpc)
- den Zeitpunkt der letzten Datenerhebung (LastPollIdentifier)
- den Typ des Fremdsystems
- weitere Eigenschaften, die individuell von den unterschiedlichen Systemen angefordert werden können und hierfür in eine Liste mit Schlüssel-Werte-Paaren eingetragen werden

Zum anderen eine `getLastChanges`-Methode mit der die aktuellen Änderungen in den Daten des Fremdsystems abgerufen werden. Als Rückgabewert stellt diese Methode eine Liste von Maps mit den Änderungen seit dem letzten Abruf zur Verfügung. Der letzte Abruf ist in der *ConnectorConfig* festgehalten, die bei der `connect`-Methode übergeben wurde.

6.3.1 Trac

Der *Connector* zum Ticket- und Bugtrackingsystem *Trac* basiert auf einer von *Trac* mitgelieferten XML-RPC Schnittstelle. Sie liefert Methoden, um die benötigten Informationen zu extrahieren. *Trac* wurde als zu untersuchendes System gewählt, obwohl es nicht vom Projekt selbst genutzt wird, da es ein häufig verwendetes quelloffenes System in der Softwareentwicklung ist und sowohl ein Ticketsystem als auch ein Wiki enthält. In diesem Stadium der Entwicklung wurde sich auf das Ticket- und Bugtrackingsystem beschränkt, eine Erweiterung des *Connector*s zur Extraktion der Wikidaten ist noch vorgesehen. Wie in Abschnitt 6.3 beschrieben, implementiert dieser *Connector* das geforderte Interface.

Somit wird eine Methode `connect` bereitgestellt, an welche die *ConnectorConfig* übergeben wird. Die *ConnectorConfig* des *TracConnector*s muss pflichtmäßig die Felder:

- URL
- User
- Password
- Type

enthalten. Weiterhin sind optional die Felder:

- LastPollIdentifier
- Property(encoding)

STING-RÜCKGABEWERT	TICKETINFORMATION
SYSTEM_URI	track_(Trac System URL)
USER_NAME	Ticket-Besitzer
CHANGE_DATE	Erstellungsdatum bzw. Änderungsdatum
INFORMATION_OBJECT_URI	Ticketnummer
CHANGE_TYPE	Änderungstyp
INFORMATION_OBJECT_TYPE	trac_ticket
CHANGE_MESSAGE	Welche Änderung ist passiert

Tabelle 6.1: Mapping der Rückgabewerte der Methode `getLastChanges()` auf die Trac Inhalte

zu befüllen.

Die Parameter URL, User und Password identifizieren das zu analysierende System und den Benutzer. Dem Benutzer müssen im *Trac*-System besondere Rechte zugesprochen werden, damit er über *XML-RPC* Daten extrahieren darf. Er muss die Genehmigung haben, um `TICKET_VIEW` und `XML_RPC` ausführen zu dürfen. Der Parameter Type dient zur Selbstidentifikation. Um mitzuteilen, zu welchem Zeitpunkt die letzte Abfrage stattgefunden hat, wird im Parameter LastPollIdentifier der Timestamp [49] als `Long` angegeben. Wenn dieser Parameter nicht gesetzt wird, ist die Annahme, dass noch nie Daten aus dem System extrahiert wurden.

Für die Rückgabe der Inhalte wurde eine sehr flexible Form gewählt. Die Rückgabe wird in einer `Map` aus `String`s durchgeführt und später durch eine entspreche Rule umgewandelt (vgl. Abschnitt 5.5). Der *TracConnector* verwendet bisher die *DefaultConnectorRule*, dazu füllt er die in der Tabelle 6.1 abgebildeten Schlüssel-Werte-Paare. Die `SYSTEM_URI` identifiziert das System aus dem die Daten stammen. Sie setzt sich aus dem einleitenden *„trac_“* und der URL des System zusammen. In einem Ticketsystem werden alle Tickets mit einer eindeutigen Nummer versehen, diese wird auch in `LuPanKu` als Identifikationsnummer des `InformationObject`s benutzt. Ebenso wird zu jeder Änderung, die an einem Ticket vorgenommen wurde, die Art der Änderung gespeichert, was ebenfalls als `CHANGE_TYPE` zurückgeliefert wird. Der Schlüssel-Rückgabewert `CHANGE_MESSAGE` gibt eine Zusammenfassung der Änderung wieder.

6.3.2 Confluence

Der *Confluence-Connector* implementiert das *Connector*-Interface und wird verwendet, um eine Verbindung zu einem *Confluence*-Wiki aufzubauen und aktuelle Eintragungen auszulesen.

Da *Confluence*, vom australischen Unternehmen Atlassian Software Systems, in vielen Unternehmen und auch im aktuellen Projekt als Wiki verwendet wird, wurde festgelegt, dass einer der ersten *Connectoren* des Projekts der zu *Confluence* ist.

Zu Beginn der Umsetzung des *Connectors* wurde überlegt die Abfrage der Daten nur

über XMLRPCs zu implementieren. Allerdings bietet die XMLRPC-API von *Confluence* nicht die Möglichkeit, die geforderten Daten auf einen gewissen Zeitraum zu begrenzen. Daher müssten zuerst alle Datensätze der *Confluence*-Datenbank ausgelesen und kontrolliert werden, ob sie in dem gewünschten Zeitraum liegen. Da dies zu einem erheblichen Performance-Problem führen kann, wurde eine alternative Umsetzung gewählt. Der *Connector* liest die aktualisierten Elemente des Wikis mit Hilfe einer Feed-URL des *Confluence* aus und verwendet XMLRPCs, um genauere Informationen zu den einzelnen Neuerungen zu erhalten.

Dazu wird in der `connect`-Methode ein XMLRPC-Client instanziiert und mit den Werten aus der *ConnectorConfig* (vg. Kapitel 6.3) eine Verbindung zum *Confluence*-Wiki aufgebaut. Dabei wird ein Token-String erzeugt, der in den späteren Methodenaufrufen zur Identifizierung der Verbindung benötigt wird.

In der `getLastChanges`-Methode wird zunächst die Feed-URL verfasst. Der URL-String wird dynamisch erstellt, so dass festgelegt werden kann welche Daten aus welchem Zeitraum abgefragt werden. Der Zeitraum wird anhand des *LastPollIdentifiers* der *ConnectorConfig* festgelegt.

Mit einem DocumentBuilder werden anschließend die Elemente des Feeds ausgelesen und in einer Liste gespeichert. Bei diesen Elementen kann es sich um Wikiseiten, Blogs, Anhänge, Kommentare oder Nachrichten handeln. Die einzelnen Elemente in der Liste sind als zusammenhängende Strings mit zusätzlichen Feed-Informationen gespeichert und müssen daher vor der weiteren Bearbeitung in einzelne Teilstrings zerlegt werden. Für jedes Element wird anschließend ein XMLRPC-Methodenaufruf ausgeführt, mit dem alle relevanten Informationen des Elementes vom *Confluence*-Wiki abgefragt werden. Dabei wird der oben bereits erwähnte Token-String benötigt.

Die Informationen über die Änderungen werden anschließend in einer `HashMap` mit dem jeweils passenden Key gespeichert. Die verschiedenen `HashMaps` der einzelnen Elemente werden in einer Liste zusammengefügt. Diese Liste wird als Rückgabewert definiert.

Danach loggt sich das System beim *Confluence*-Wiki wieder aus und die Session wird beendet.

6.3.3 JIRA

Der JIRA-Connector wird verwendet, um mit der Connector-Schnittstelle eine Verbindung zu JIRA-Systemen aufzubauen und Informationen über Aufgaben und Fehlermeldungen auszulesen.

Alle JIRA-Systeme stellen RPC-Plugins zur Verfügung, welche einen Zugriff auf die JIRA-Datenbank mit SOAP oder XMLRPCs ermöglichen [44]. Die benötigten Klassen um die SOAP RPCs zu realisieren, können durch den WSDL-[1]Deskriptor der JIRA

[1] Bei WSDL handelt es sich um eine auf XML-basierende Metasprache mit der Klassen, Methoden und Datentypen plattform- und betriebssystemunabhängig beschrieben werden können.

Installation ermittelt werden. Mit Hilfe des Web-Service-Client-Assistenten von eclipse kann ein komplettes Abbild der Klassen erstellt werden. Dieser Assistent arbeitet mit der SOAP-Engine von Apache Axis und liest die WSDL-Deskription aus. Daraus wiederum kann eine Java-Bibliothek mit allen gültigen entfernten Methodenaufrufen kreiert werden.

Der JIRA-*Connector* importiert diese Bibliothek und baut in der `connect`-Methode über die Login-Methode der `JiraSoapService`-Schnittstelle eine Verbindung zum JIRA-System auf. Wie in Kapitel 6.3 beschrieben werden die Zugangsdaten dem *Connector-Config*-Objekt entnommen.

In der `getLastChanges`-Methode werden die Änderungen der Daten aus dem aktuellen Zeitraum ermittelt, der durch den *LastPollIdentifier* festgelegt wird. Dazu wird dynamisch eine Abfrage erstellt, die auf die Daten im gewünschten Zeitraum beschränkt ist. Bei den resultierenden Elementen handelt es sich um JIRA-Issues, deren Informationen auf eine Liste von `Hashmaps` abgebildet werden. Die Liste wird anschließend als Rückgabewert der `getLastChanges`-Methode definiert und die Verbindung zum JIRA-System wird abgebaut.

6.3.4 Subversion

`LuPanKu` soll verschiedene Versionsverwaltungssysteme unterstützen. Als erstes System wurde Subversion [6] gewählt. Die Wahl fiel auf dieses Werkzeug, da es zum einen für das `LuPanKu`-Projekt selbst verwendet wird und außerdem das Werkzeug eines interessierten Kunden ist.

Subversion ist ein quelloffener Vertreter der zentralen Konfigurationsmanagement-Werkzeuge.

Im Gegensatz zu verteilten Repository Servern wie Git [42], arbeitet Subversion mit lediglich einem Server. Das bietet den Vorteil, dass für die Extraktion der Metadaten eines Teamrepositories nur ein Server abgefragt werden muss. Zudem ist es von Vorteil, dass bestehende Konfigurationen von Quellcode nicht auf Veränderungen geprüft werden müssen. Subversion versioniert bei jedem Checkin die gesamte Struktur des Repositories und erzeugt so implizit eine Baseline.

Für den Zugriff auf Subversion wurde die Java Bibliothek SVNKit [27] gewählt, welche den Zugriff auf den Server nur mit Java Technologie realisiert. Die von `LuPanKu` benötigten Metadaten werden über das Subversion Kommando *Log* ermittelt. Durch dieses erhält man die Revisionen von Subversion mit dem Kommentar und der Information, welche Dateien modifiziert, geändert oder gelöscht wurden.

6.3.5 Code Dependencies

In Kapitel 3.1 wurde der Zusammenhang zwischen dem sozialen Netzwerk und dem zu entwickelnden Softwareprodukt aufgezeigt. Dieser Zusammenhang lässt sich durch ein

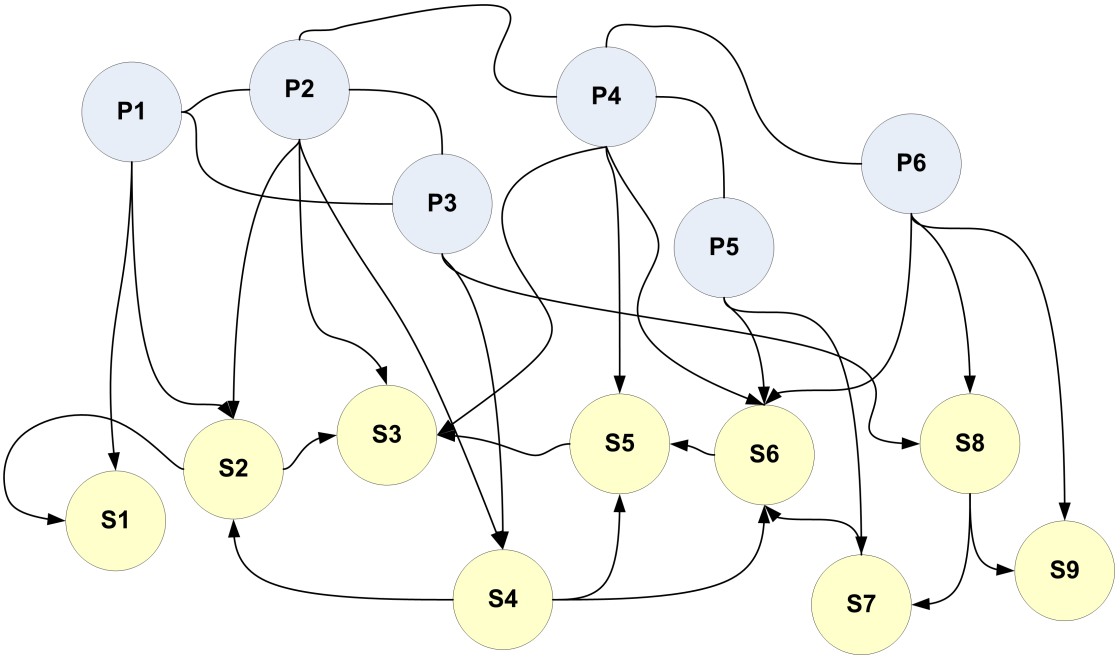

Abbildung 6.1: Schematische Darstellung eines *Socio-technical Congurence* Network

Socio-technical Congurence Network [46] darstellen. Zur Erstellung eines solchen Netzwerkes benötigt man zum einen das soziale Netzwerk der Entwickler und dann den Abhängigkeitsgraphen der Software. Durch die Informationen aus dem Versionsverwaltungssystem lassen sich diese beiden Netzwerke miteinander verbinden.

Die Analyse eines *Socio-technical Congurence* Network gibt Aufschluss über ein mögliches Potential zur Verbesserung der Kommunikation im Team. Wird beispielsweise eine hohe Abhängigkeit von Klassen untereinander ohne eine korrespondierende Beziehung im sozialen Netzwerk der Entwickler festgestellt, kann dies ein Indiz für eine notwendige Verbesserung der Kommunikation oder des Entwicklungsprozesses sein.

In Abb. 6.1 wird ein mögliches Netzwerk dargestellt. Die Knoten in der oberen Ebene stellen das soziale Netzwerk der Personen da. Auf der unteren Ebene werden die Abhängigkeiten der Softwareartefakte untereinander aufgezeigt. Das Netzwerk zeigt dabei, das es keine Verbindung zwischen den Personen P3 und P6 gibt, obwohl sie das gleiche Softwareartefakt bearbeiten.

Eine Möglichkeit zum Aufbau eines solchen Abhängigkeitsgraphen der Software ist die statische Analyse der Typenbeziehungen im Quellcode. Zwar berücksichtigt diese Analyse nicht die dynamischen Zusammenhänge der Software, liefert jedoch wertvolle Informationen. Diese lassen sich durch vorhandene Werkzeuge mit geringem Aufwand ermitteln.

Zudem kann die statische Codeanalyse Hinweise auf die Expertise des Entwicklers geben. Zum Beispiel kann sich das Expertenwissen über eine Bibliothek dadurch äußern, dass

ein Entwickler regelmäßig Abhängigkeiten zu den Klassen einer bestimmten Bibliothek erzeugt oder bearbeitet.

Für die statische Analyse wird die Java-Bibliothek Dependency Finder [40] genutzt. Diese Bibliothek bietet eine API zur Analyse von Java Classfiles. Das Konzept von Dependency Finder ist nicht die Ausgabe eines Ergebnisses der vom Client geforderten Classfile Analyse, welches dieser wiederum auswerten muss. Stattdessen registriert der Client Listener in der Analyse Klasse `ClassfileLoader` zu bestimmten Ereignissen und kann direkt an dieser Stelle die relevanten Informationen extrahieren.

Somit lassen sich komplette Abhängigkeitsgraphen der Software auf Klassenebene aufbauen und speichern. Diese Daten können dann von dem in Abschnitt 5.6 beschriebenen `DependencyChangesetAnalyzer` verwendet werden.

6.4 Rule Engine

Die derzeitigen Implementierungen der *Rule Engine*, welche in Abschnitt 5.5 erläutert wurde, befinden sich im Package `org.aysada.lupanku.core.rules`. Dabei wird die Funktionaltität durch drei Klasse bereitgestellt.

Das Interface `Rule` stellt eine einzig Methode (`process(Changes)`) bereit, an welche die Rohdaten in Form einer Liste mit `Changes` übergeben wird. Diese Methode sorgt dann für die Aufbereitung der Rohdaten, baut das eigene Datenmodell auf und sorgt dafür, dass dieses Modell persitiert wird.

Die passende Regel für einen bestimmten *Connector* kann mit dem `RuleManager` gefunden werden, welcher durch die verwendeten *OSGi Services* (vgl. Abschnitt 6.1) unterstützt wird. Wurde für einen bestimmten *Connector* keine eigene Regel implementiert (oder zumindest über den `RuleManager` nicht gefunden), wird die Standardregel in Form einer Instanz der Klasse `DefaultConnectorRule` verwendet.

Die `DefaultConnectorRule` erzeugt im aktuellen Release zwar ein einfaches Modell und persistiert dieses, befindet sich aber hauptsächlich in einem experimentellem Stadium der Entwicklung. Eine verbesserte Standardregel ist für das kommende Release geplant.

6.5 Benutzeroberfläche

Die Anwendung soll zentral und ohne weiteren Aufwand erreichbar sein. Dafür bietet sich eine Web-Anwendung an, da diese von verschiedenen Computern im Netzwerk ohne zusätzliche Softwareinstallation erreichbar ist. Durch den Plug-In zentrierten Ansatz von `LuPanKu` muss sich die Oberfläche dynamisch an diese anpassen können.

Ein großer Anwenderkreis werden Entwickler sein. Die Web-Anwendung eignet sich bei Abstimmungen und Besprechungen. Bei der Entwicklung am Quellcode ist eine bessere Integration innerhalb der Entwicklungsumgebung wünschenswert. In diesem Fall sollte

ein Medienbruch durch die direkte Integration von `LuPanKu` in die IDE vermieden werden. Die Daten der Entwicklungsumgebung können zur Bereitstellung von kontextbezogenen Informationen genutzt werden.

Aus oben genannten Gründen wurde entschieden die Eclipse Rich Ajax Platform (RAP) für die Entwicklung zu nutzen. Diese bietet folgende Vorteile für die Softwareentwicklung:

Komfortable Oberflächen – Durch die Verwendung von Ajax kann die Usability der Web-Anwendung erhöht werden.

Plug-In-basierte Oberflächen – Die im Eclipse Umfeld etablierte View und Editor Technik ist in der RAP Umgebung verfügbar. Dadurch lässt sich die in Eclipse bewährte Technik zur Integration von Plug-Ins in einer Oberfläche nutzen.

Single Sourcing – Das Programmiermodell von RAP ist auf Ebene des Binärcodes vom Client identisch mit dem von SWT. Dies bedeutet, dass eine Oberfläche, die auf Basis von RAP für das Web entwickelt wurde, auch als Rich Client funktioniert. Dafür wird die RAP Bibliothek durch die von SWT ersetzt. Zwischen einer Rich Client Anwendung und einer Web-Anwendung gibt es Unterschiede, die berücksichtigt werden müssen. Beispielsweise steht einer Web-Anwendung aus Sicherheitsgründen kein Dateizugriff auf den Client zu. Daher fehlen die Öffnen- und Speichern-Dialoge und müssen durch Download- und Upload-Mechanismen nachgebildet werden. Dennoch lässt sich ein großer Teil des Oberflächencodes wiederverwenden, da die Besonderheiten der Zielplattform in eigene Plug-Ins ausgelagert werden können.

Bei einer RAP-Anwendung wird die Oberflächenlogik auf dem Server verwaltet. Dies sorgt für eine hohe Netzwerklast. Andere Werkzeuge wie das Google Widget Toolkit halten die Logik auf dem Client im Webbrowser und reduzieren dadurch den Netzwerkverkehr. Bei der Verwendung von RAP sollte man daher die Anzahl der Nutzer berücksichtigen.

7 Erfahrungen

Das Projekt wurde im Sommersemester 2010 neu erstellt, daher stand anfangs der Teambildungsprozess im Vordergrund. Nach Tuckmans Teamentwicklungsmodell [43] muss jedes Team vier Stadien durchlaufen, um sich tatsächlich zu einer kollaborierenden Gemeinschaft zu entwickeln:

- Formierung (*forming*) - Zusammenkommen und Ziele definieren
- Konflikte austragen (*storming*) - Konflikte, die auch erste Konsequenz bzgl. der Rollen haben
- Regeln und Leitlinien definieren (*norming*)
- Zusammenarbeiten (*performing*) - um gemeinsame Ziele zu erreichen

Zu Beginn des Semesters wurde rege über die Möglichkeiten und Grenzen des Projektes diskutiert. Aus den Zielen und Vorstellungen der Teammitglieder wurde in dieser Formierungsphase eine gemeinsames Leitbild für das Projekt entwickelt.

Dies führte zum einen zu den Visionen für das Projekt (siehe Kapitel 3), aber gleichzeitig förderte dieser Prozess auch das Vertrauen im Team. Dabei wurde deutlich, dass eine große Herausforderung darin besteht, die unterschiedlichen Vorstellungen der Teammitglieder zu vereinbaren, da ihr Fokus auf unterschiedlichen Aspekten des Projektes liegt.

In der zweiten Phase wurde ein gemeinsames technisches Verständnis des Systems entwickelt. Beispielsweise war das Erstellen eines möglichst generischen Datenmodells, von intensiven Diskussionen begleitet. Diese Diskussionen zeichneten ein schärferes Gesamtbild über das zu entwickelnde System und zeigten bisher unbeachtete Problemstellungen und Komplexitäten der zu analysierenden Werkzeuge auf. Dabei bildeten sich Verantwortungsschwerpunkte für die Mitglieder des Teams heraus.

Das Team entwickelte Regeln und Leitlinien für die weitere Zusammenarbeit. Dazu zählt zum einen ein einheitlicher Quellcode Standard und zum anderen die konsequente Verwendung der beschriebenen Kollaborationswerkzeuge. Das Projekt folgt damit der Regel *„Eat your own dog food“* [18]. Als Leitlinie für das System wurde eine modulare Architektur gewählt. Deren Komponenten und über welche Schnittstellen diese miteinander interagieren, wurde detailliert erörtert.

Nach dem Entwurf der Architektur wurde an den technischen Komponenten des Systems gearbeitet. Dabei wurden Erfahrungen im Umgang mit den ausgewählten Werkzeugen gemacht. Hierbei zeigte sich deren Potential die Tätigkeiten von Teams zu unterstützen,

die zeitweise verteilt arbeiten. Es bedarf allerdings einer hohen Selbstdisziplin, die Informationen in den verschiedenen Werkzeugen mit der notwendigen Qualität zu pflegen. Gerade diese Qualität ist eine wichtige Vorraussetzung für die Analyse durch `LuPanKu`.

Dadurch dassw sich die Analyzer noch in der Entwicklung befinden, kann nicht abschließend bewertet werden in wie weit der Ansatz geeignet ist um die Vision von *HomeOffice2.0* zu erreichen. Dies ist für Projekt 2 geplant.

8 Ausblick

Im Rahmen der Lehrveranstaltung „Projekt 1“ wurde die Architektur für `LuPanKu` entwickelt und mit der Implementation des Systems begonnen. Dabei wurden zuerst die Infrastrukturkomponenten (wie *Persistence* und *Service Maintenance*) sowie die in Abschnitt 6.3 beschriebenen *Connectors* realisiert.

Für das „Projekt 2“ (Wintersemester 2010/2011) liegt der Schwerpunkt auf der Implementation und Verwendung der geplanten *Analyzer*, wie sie in Abschnitt 5.6 beschrieben werden. Dieses Vorgehen verifiziert das entworfene Datenmodell und den Ansatz der Modelladapter. Zu diesem Zeitpunkt wird auch ersichtlich, ob die Qualität der abgefragten Meta-Informationen für die Analysen ausreichend sind. Wie in Kapitel 3.3 beschrieben, bilden die Kontexte in denen sich die Entwickler befinden, eine Grundlage für die Verfolgung des Projektdiskurses. Daraus ergibt sich die Fragestellung ob die vorhandenen Daten ausreichend Informationen bieten.

Nach einer Verifikation der Daten aus dem Entwicklungsprozess des `LuPanKu`-Projekts, soll eine Auswertung von produktiven Unternehmensdaten erfolgen, um die Tauglichkeit des Systems im *Enterprise2.0*-Umfeld zu prüfen.

Für weitere Analysen muss evaluiert werden, welche zusätzlichen Daten neben den Meta-Informationen notwendig sind. Hierbei sind *Textmining*-Algorithmen oder die Verwendung der Fremdsystem-eigenen Suchmechanismen denkbar. Ebenfalls ist eine Überprüfung der aktuellen Fremdsystem-Auswahl sinnvoll.

Das Ziel von „Projekt 2“ besteht in dem Aufbau einer passenden Infrastruktur, die für die nachfolgenden Masterarbeiten ausreichend ist. Diese werden sich intensiv mit den Themen der Expertensuche und der Unterstützung von verteilten Teams beschäftigen.

Literaturverzeichnis

[1] Agile: Manifesto for Agile Software Development. (2010), 02. – URL http://agilemanifesto.org/

[2] Alliance, OSGi: *Knopperfish*. – URL http://www.knopflerfish.org/. – Zugriffsdatum: 04. July 2010

[3] Alliance, OSGi: *OSGi*. 2010. – URL http://www.osgi.org/Main/HomePage. – Zugriffsdatum: 04. July 2010

[4] Cataldo, Marcelo ; Easterbrook, Steve ; Damian, Daniela ; Herbsleb, James ; Devanbu, Premkumar ; Mockus, Audris: 2nd international workshop on socio-technical congruence (STC 2009). (2009), S. 476–477. ISBN 978-1-4244-3495-4

[5] Codd, E.F.: Providing OLAP (on-line analytical processing) to user-analysts: an IT mandate. (1993)

[6] CollabNet: *Subversion*. – URL http://subversion.tigris.org/. – Zugriffsdatum: 04. July 2010

[7] Degenne, Alain ; Forse, Michel: *Introducing Social Networks (Introducing Statistical Methods)*. Sage Pubn Inc, 1999. – URL http://www.amazon.de/Introducing-Social-Networks-Statistical-Methods/dp/0761956042%3FSubscriptionId%3D192BW6DQ43CK9FN0ZGG2%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0761956042. – ISBN 0761956042

[8] Dreyfus, Hubert L. ; Dreyfus, Stuart E. ; Athanasiou, Tom: *Mind over machine: the power of human intuition and expertise in the era of the computer*. New York, NY, USA : The Free Press, 1986. – ISBN 0-02-908060-6

[9] Drucker, Peter: *Concept of the Corporation*. Transaction Publishers, 1993

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

[11] Eclipse: *Eclipse Swordfish*. – URL http://www.eclipse.org/swordfish/. – Zugriffsdatum: 14. Juli 2010

[12] Facebook: *Facebook*. – URL http://www.facebook.com. – Zugriffsdatum: 20. August 2010

[13] Foundation, Apache S.: *Apache Felix.* – URL `http://felix.apache.org/site/index.html`. – Zugriffsdatum: 04. July 2010

[14] Foundation, Apache S.: *Apache Maven.* – URL `http://maven.apache.org/`. – Zugriffsdatum: 04. July 2010

[15] Foundation, Eclipse: *Eclipse.* – URL `http://www.eclipse.org/`

[16] Foundation, Eclipse: *Equinox.* – URL `http://www.eclipse.org/equinox/`. – Zugriffsdatum: 04. July 2010

[17] Fowler, Martin: *Public versus Published Interfaces.* 2002

[18] Gamma, Erich ; Beck, Kent: *Contributing to Eclipse: Principles, Patterns, and Plugins.* Redwood City, CA, USA : Addison Wesley Longman Publishing Co., Inc., 2003. – ISBN 0321205758

[19] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns.* Reading, MA : Addison Wesley, 1995

[20] Gasser, Urs: *Die Digital Natives.* DNADIGITAL - Wenn Anzugträger auf Kapuzenpullis treffen. Februar 2009. – URL `http://www.scribd.com/doc/12544534/DNAdigital-Wenn-Kapuzenpullis-auf-Anzugtraeger-treffen`

[21] Google: *Google Mail.* – URL `http://mail.google.com`. – Zugriffsdatum: 20. August 2010

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

[23] Kerth, Norm: *Project Retrospectives. A Handbook for Team Reviews.* Dorset House, 2001

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

[25] Ltd., Atlassian P.: *Confluence.* – URL `http://www.atlassian.com/software/confluence`. – Zugriffsdatum: 20. August 2010

[26] Ltd., Atlassian P.: *Jira.* – URL `http://www.atlassian.com/software/jira`. – Zugriffsdatum: 20. August 2010

[27] Ltd., TMate S.: *SVNKit.* – URL `http://svnkit.com/`. – Zugriffsdatum: 04. July 2010

[28] McAfee, Andrew P.: Enterprise 2.0: The Dawn of Emergent Collaboration. In: *MITSloan* 47 (2006), Spring, Nr. 3, S. 21–28

[29] McKinney, Vicki R. ; Whiteside, Mary M.: Maintaining distributed relationships. In: *Commun. ACM* 49 (2006), Nr. 3, S. 82–86. – ISSN 0001-0782

[30] Mählmann, Lars: Deliver who I mean, automatische Erstellung von Personenprofilen in großen Unternehmen. (2009). – URL `http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/maehlmann.pdf`. – Zugriffsdatum: 15. August 2010

[31] ORACLE: *Oracle TopLink JPA*. – URL `http://www.oracle.com/technetwork/middleware/ias/index-085257.html`. – Zugriffsdatum: 20. August 2010

[32] Panier, Karsten: Home Office 2.0 - Collaborate Working. (2010). – URL `http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw1/Panier/bericht.pdf`. – Zugriffsdatum: 14. Juli 2010

[33] Project, A W.: *Wikipedia*. – URL `http://www.wikipedia.org`. – Zugriffsdatum: 20. August 2010

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

[35] Randell, B.: Software engineering in 1968. (1979), S. 1–10. ISBN none

[36] RedHead Inc., JBoss by: *Hibernate - Relational Persistence for Java & .NET*. – URL `http://www.hibernate.org`. – Zugriffsdatum: 20. August 2010

[37] Software, Edgewall: *The Trac Project*. – URL `http://trac.edgewall.org`. – Zugriffsdatum: 20. August 2010

[38] Spafford, George: *The Dangers that Lurk Behind Shadow IT*. Februar 2004. – URL `http://itmanagement.earthweb.com/career/article.php/3308481/The-Dangers-that-Lurk-Behind-Shadow-IT.htm`

[39] Stroebe, Wolfgang ; Jonas Klaus ; R.C. Hewstone Miles: *Sozialpsychologie. Eine Einführung*. Berlin, Heidelberg : Springer-Verlag, 2002 (Springer-Lehrbuch)

[40] Tessier, Jean: *Dependency Finder*. – URL `http://depfind.sourceforge.net/`. – Zugriffsdatum: 04. July 2010

[41] Tom Davenport, Julian B.: *Knowledge work as organizational behavior*. BlackweIiPublishersLtd, 2000. – 287–304 S

[42] Torvalds, Linus: *GIT*. – URL `http://git-scm.com/`. – Zugriffsdatum: 04. July 2010

[43] Tuckman, B.: Developmental sequence in small groups. In: *Psychological Bulletin* 63 (1965), S. S. 384–389

[44] Turner, Jeff: *Creating a SOAP Client*. Oktober 2005. – URL `http://confluence.atlassian.com/display/JIRA/Creating+a+SOAP+Client`

[45] Twitter: *Twitter*. – URL `http://www.twitter.com`. – Zugriffsdatum: 20. August 2010

[46] Valetto, Giuseppe ; Helander, Mary ; Ehrlich, Kate ; Chulani, Sunita ; Wegman, Mark ; Williams, Clay: Using Software Repositories to Investigate Socio-technical Congruence in Development Projects. (2007), S. 25. ISBN 0-7695-2950-X

[47] Wikipedia: *Experte*. – URL `http://de.wikipedia.org/wiki/Experte`. – Zugriffsdatum: 24. August 2010

[48] Wikipedia: *Information Retrieval*. – URL `http://de.wikipedia.org/wiki/Information_Retrieval`. – Zugriffsdatum: 18. August 2010

[49] Wikipedia: *Timestamp*. – URL `http://en.wikipedia.org/wiki/Timestamp`. – Zugriffsdatum: 29. June 2010

[50] Willke, Helmut: Organisierte Wissensarbeit. In: *Zeitschrift für Soziologie 3* (1998), S. 161–177