Projektergebnis

Stand der Technik der zu integrierender Kerntechnologien

Projektidentifikation:	WAVES – Wissensaustausch bei der verteilten Entwicklung von Software
Ergebnis ID:	E9
Arbeitspaket(e):	AP2, AP4-AP8
Autor(en):	Hans-Jörg-Happel, Volker Kuttruff, Tim Romberg, Peter Szulman, Max Völkel
Koordinator:	FZI Forschungszentrum Informatik

1 Überblick

Das vorliegende Dokument fasst den im Rahmen von WAVES ermittelten Stand der Technik zusammen. Das Dokument ist hierbei lediglich ein „Snapshot“ des im WAVES-Wiki http://team.waves.fzi.de zu findenden Inhalts. Da im Laufe des Projekts der Stand der Technik permanent aktualisiert wird, stellt das Wiki die zu bevorzugende Informationsquelle dar.

Der Aufbau des Dokuments orientiert sich (im Gegensatz zu den querverlinkten Wiki-Seiten) an den einzelnen Arbeitspaketen, zu denen jeweils der Stand der Technik zusammengetragen wurde. Dies sind im Einzelnen:

- Wissensprozesse im Software Engineering (AP2)
- Metadaten-Speicher (AP4)
- Knowledge Desktops und Semantische Wikis (AP5)
- Sharing Engine (AP 7)
- Werkzeugintegration (AP 8)
- Ontologien zum Software-Lifecycle-Management (AP2, übergreifend)

2 Wissensprozesse im Software Engineering

Bei agilen Ansätzen nach Cockburn sind Kommunikationsbarrieren sowie Zeit- und Energiekosten für Ideentransfer ein großes Manko der verteilten Arbeit. Mit dem Aufbau von Vertrauen durch überzeugende Kompetenz (Paul & McDaniel, 2004) könne diese Hürde überwunden werden. Software-Communites, wie in der Open-Source-Gemeinde ansatzweise etabliert (z.B: GNU enterprise (Elliot & Scacchi, 2003)), bieten Entwicklern die Möglichkeit, ihr Wissen zielorientiert aufzustocken und ihr Können unter Beweis zu stellen. Bereits (Gibbs et al., 1990) stellen fest, dass sog. "Software Information Systems" Vorreiter der Software Communities - für "large scale class reuse" erst nützlich werden, wenn man sie öffentlich, also unternehmensübergreifend, nutzt.

Insgesamt können also Communities of Practice (CoP, Wenger et al., 2002) und Communities of Interest (CoI) eine enorme Rolle in der Erzeugung, Verbreitung und Aktualisierung von Spitzenqualifikationen im Software Engineering spielen. Dieses Potenzial wird montan aber nur in ersten Experimenten und Überlegungen untersucht (vgl. (Fægri et al., 2005; Dingsøyr et al., 2004)).

Tatsächlich tragen agile Methoden der Softwareentwicklung dieser Tatsache insofern Rechnung, als sie in ihren Vorgehensmodellen intensiv die Kommunikation und das systematische Kombinieren der Erfahrungen verschiedener Know-How-Träger umsetzen, so dass obige Lernmethoden indirekt zum Tragen kommen. Sie bewirken also lokal, projektbezogen, ähnliche Effekte, wie wir sie hier insgesamt als Ziel verfolgen. Allerdings sind solche Ansätze nicht auf räumlich, zeitlich oder organisatorisch entkoppelte Kooperationen abgestellt, sie nutzen Kooperation nur punktuell, statt die commu-

nity-weite Wissensbasis weiterzuentwickeln, und sie binden nicht systematisch externe Wissensquellen ein (vgl. auch (Kähkönen, 2005)).

Web-basierte Community Portale. Auf der anderen Seite sind CoPs / CoIs im Wissensmanagement (Davenport, 1998; Probst et al., 1999; Mentzas et al., 2002) etablierte Ansätze, die aber wenig kontext- und projektspezifisch orientiert sind.

Semantische Community Web Portale (Hartmann & Sure, 2004; Davies et al., 2004) realisieren ähnliche Funktionalitäten, jedoch mit verfeinerten Informationsintegrationsmechanismen aufgrund ontologiebasierter Metadaten, und mit fortgeschrittenen, ontologiebasierten Informationssuch-, -auswertungs- und -visualisierungsmöglichkeiten.

Aufbauend auf dem Experience Factory Konzept von Basili und Rombach (Basili et al., 1994; Althoff et al, 2001) sind die wichtigsten Arbeiten in diesem Kontext sicher im Umfeld des Fraunhofer IESE und der dort initiierten Workshop-Reihe Learning Software Organizations entstanden. Dort werden Lessons Learned und Prozessverbesserungen entlang formaler Prozessmodelle gesammelt. Verfeinerte ontologiebasierte Beschreibungen von Artefakten und Kontexten sind die Basis fallbasierter Retrieval-Verfahren (vgl. z.B. (Decker et al., 2002)). Der Ansatz ist primär unternehmensintern gedacht, während einer unserer Forschungsschwerpunkte gerade das Zusammenspiel von internen und externen Wissensquellen und Austauschmechanismen sein wird. Im indiGo Projekt (Decker et al., 2004) werden moderne Groupware-Instrumente zur kollaborativen Wissenserzeugung eingesetzt. Auch hier geht es allerdings nur um Lernen über Software-Entwicklungsprozesse, nicht um allgemeine Themen und nicht um externe Quellen, so dass auch Kontextbegriffe hier a priori schon enger eingegrenzt sind. Im Gegensatz zu indiGo sind wir auch mit Wikis und Blogs, aktiven Wissensinhalten, Werkzeuganbindung, Code-Analyse zur Kontextbestimmung und Social Networking Software zur Kooperationsunterstützung, vielseitiger und aktueller hinsichtlich der Lern-Instrumentierung.

Weiterführende Informationen hierzu sind zum Beispiel auf Wikipedia zu finden:

- Personal Knowledge Management (PKM)
- Organizational Learning
- Community of Practice

2.1 Software-Engineering in verteilten Umgebungen

Autoren zur diesem Thema sind (lt. Jan Romberg):

- Stacy Hibino
- Becky Grinter
- James Herbsleb
- Audris Mockus
- Dave Weiss

Stichworte: "Geography of coordination", "Dealing with Distance"

3 Metadaten-Speicher

Metadaten-Speicher sind spezialisierte Datenbanken. Klassische Datenbanken weisen ein fixes Schema (Tabellen, Datentypen von Spalten) auf, nach dem die Daten organisiert sind. Metadaten sind oft durch eine unregelmässigere Struktur gekennzeichnit, die sich vor allem zur Laufzeit ändern kann. Daher müssen Metadaten-Speicher mit sich verändernden Datenschemas zurechtkommen. Das Schema der Daten wird dabei gemeinsam mit den Daten gespeichert.

Ein weiterer unterschied von Metadaten-Speichern zu klassichen, relationalen Datenbanken sind die Abfragemöglichkeiten. Metadaten-Speicher erlauben oft mit Hilfe eines eingebauten logischen Schliessers (aka Reasoner, Inferencing) Abfragen zu beantworten, die nicht explizit in den Daten stehen.

Das typische Beispiel dazu:

- Daten: A ist vom Typ X, X ist eine Spezialisierung von Y.
- Abfrage: Ist A vom Typ Y?
- Anwort: Ja.

Ein vom W3C verabschiedeter Standard für Metadaten ist RDF, das "Resource Descrption Framework". Es stellt alle Daten als Aussagen (Statements) der Form "subject, property, object" da. Diese Grundaussagen werden auch "triple" genannt. Ein "Triplestore" ist also eine auf die Speicherung und Abfrage von RDF spezialisierte Datenbank.

Manche Triplestores beschränken sich auf eine reine Datenhaltungsfunktion. Andere unterstützen Reasoning. Typischerweise wird RDFS-Reasoning unterstützt. RDFS (RDF Schema) ist eine einfache Ontologiebeschreibungssprache mit den wesentlichen Konstrukten:

- Klasse, Unterklasse
- Relation, Unterrelation
- Label und Kommentar
- Definitions- und Wertebereich von Relationen

Die Semantik von RDFS ist teilweise kontra-intuitiv, erzeugt dafür aber keine logischen Widersprüche.

Da sich für manche Anwendungen das Konzept der Triples als zu beschränkt oder umständlich erwies, um beispielsweise die Herkunft der Daten (provenance) mitzuspeichern, wurde das Modell auf "Quads" erweitert. Dieses Konzept ist auch unter den Namen "Named Graphs" und "Contexts" bekannt.

Manche Triplestores ermöglichen eine Abfrage über die vom W3C normierte Sprache "SPARQL". SPARQL selbst macht bist jetzt keine Aussage, über die vom Triplestore verwendete Semantik.

Die meisten Metadaten-Speicher sind in Java geschrieben, es gibt aber auch für fast jede andere Sprache Produkte. Obwohl der Markt noch überweigend von open-source

Produkten geprägt ist, gib es mittlerweile auch kommerzielle Anbieter wie Oracle, Aduna (support für Sesame), Nokia. Unterscheiden lassen sich Triplestores weiterhin nach Datenhaltungskomponente einteilen: In-memory, relational data base oder native triple store Implementierung. Die handhabbare Datenmenge variiert stark zwischen den Produkten, die meisten Produkte haben ein Skalirungsproblem, vor allem wenn Reasoning genutzt wird.

Im Java-Bereich besonders bekannt sind die beiden open-source Projekte Jena (http://jena.sf.net) und Sesame (http://openrdf.org). In C ist Redland die führende open-source Implementierung.

Bei Triplestores hat man drei Ebenen von APIs, von denen eine jeweils auf der anderen aufsetzt. Auf der untersten Ebene ist die triple-orientierte Schicht mit Befehlen wir "füge triple hinzu", "lösche triple" und "suche triple". Darauf aufbauend sind Resource-orientierte APIs mit Befehlen wie: "füge Klasse hinzu", "lies Kommentar der Resource", "setze Wertebereich der Relaton x auf y". Auf der obersten Ebene werden manchmal objektorientierte APIs verwendet mit Befehlen wie "erzeuge Person", "setze Vorname" und weiteren Bereichs-spezifischen Befehlen.

Im Kontext des WAVES-Projekts werden sollten bei der Auswahl eines geeigneten Metadatenspeichers folgende Anforderungen berücksichtigt werden:

- Wichtig ist vor allem die Art der benötigten Inferenzmöglichkeiten, das zu erwartende Datenvolumen und benötigte Schnittstellen. Sollen Daten z. B. über HTTP abgefragt werden, so ist SPARQL-Unterstüzung wichtig. Triplestores verhalten sich zudem unterschiedlich, wenn Daten häufig oder selten (nie?) gelöscht werden. Wir sind die Einsatzzwecke zu klären.
- Im Java-Bereich kann zur Entkopplung von einer spezifischen API das open-source Projekt RDF2Go (http://rdf2go.ontoware.org) verwendet werden. Es unterstützt Triple- und Quadmodelle und hat Adapter für Jena, Sesame und YARS. Darauf aufbauend kann RDFReactor (http://rdfreactor.ontoware.org) genutzt werden, um Java-Entwicklern einen möglichst einfachen Zugang zur RDF-Welt zu geben, ohne das Einzelheiten des Triple-Modells bekannt sein müssen.

3.1 Versionierte Metadaten-Speicher

Aus Sicht von Polarion sollte die gesamte Persistenz in Subversion erfolgen.

Die bisher am FZI (und auch bei Empolis) realisierten Technologien zur Erfassung von semantischen Relationen, zur Speicherung und Suche nach Metadaten (und Volltexten?) basieren dagegen auf Datenbanktechnologien.

Auf beiden Seiten sind spiegelbildliche Lücken zu erkennen:

- Polarion muss im Subversion Daten redundant halten, z.B. von Hand eigene Indizes auf die Metadaten verwalten
- Die metadatenbasierten Anwendungen des FZI (und Empolis) würden eigentlich eine robuste Versionierung benötigen, wie Subversion sie liefert. (In Mediawiki wird Versionierung von Hand auf der Datenbank realisiert).

Im Hinblick auf Offline-Zugriff und zwischen mehreren Organisationen gesharete Wissens-Speicher werden zumindest mächtige Transaktions-Funktionen verlangt, Subver-

sion würde schon ein gutes Modell für die nötige Interaktion liefern (Einchecken von Änderungen, Konfliktlösung).

4 Knowledge Desktops und Semantische Wikis

Im Moment werden abwechselnd Wikis und der Desktop als "führendes" UI-Paradigma von Waves angeführt. Dies erscheint zunächst als Widerspruch. Allerdings besteht hier gerade eines der Ziele des Projekts - nämlich einen Plattform zu schaffen, die die individuelle Produktivität des Desktops mit der flexiblen Kollaboration von Wikis und anderer browserbasierter Social Software vereinigt. Heutige Wikis sind nur ein Notnagel, um mit die Beschränkungen von HTTP und HTML zu umgehen. Ziel der zukünftigen Plattform ist es, dass keine Medienbrüche oder sonstigen Barrieren mehr bestehen zwischen der persönlichen Wissenssphäre (heutiger Desktop) und denen des Teams und der Öffentlichkeit.

4.1.1 Knowledge Desktop

"Knowledge Desktop" bezeichnet weniger ein konkretes Produkt, als vielmehr die Absicht, Wissensarbeit am Computer zu verbessern. Dabei sind vor allem zwei Schnittstellen im Blickfeld:

- Die Schnittstelle vom Mensch zu Wissenseingabe- und Strukturierung am Computer. Dabei stehen Bemühungen, Inhalte auch zu formalisieren im Mittelpunkt.
- Die Schnittstellen zwischen verschiedenen Applikationen auf dem Desktop. Beim Konvertieren gehen oft unnötigerweise Strukturinformationen verloren.

Im Bereich Knowledge Desktops (auch als "semantische Desktops" bekannt, http://www.semanticdesktop.org), lassen sich in zwei Grundansätze einteilen:

- Die einen wollen den Desktop mit all seinen Anwendungen ersetzen, und eine integrierte Anwendung für Emails, Termine, Notizen, Adressen und Bookmarks erstellen (Haystack, DeepaMehta[?], OpenIRIS[?], Chandler, ...)
- Ein anderer Ansatz besteht in der Anreicherung und Vernetzung von Anwendungen (z.B. über Plugins). Dieser Ansatz wird beispielsweise von Gnowsis und NEPOMUK verfolgt.

Keines der semantischen Desktop-Projekte hat eine Stabilität erreicht, so dass man neue Produkte darauf aufbauen könnte.

4.2 Semantische Wikis

Wikis sind entstanden als kollaborative, webbbasierte Informationssysteme mit folgenden Characteristika:

- Es ist einfach, neue Inhalte hinzuzufügen
- Es ist einfach, Inhalte zu strukturieren (mit Hilfe der Wiki-Syntax)
- Es ist einfach, Inhalte zu vernetzen (mit Hilfe der Seitentitel). Diese Eigenschaft ist vermutlich das Merkmal mit der grössten Unterscheidungskraft im Vergleich mit anderen Editoren. Semantische Wikis sind dann Wikis, bei denen man Inhalte nicht nur strukturieren, sondern auch einfach formalisieren kann. In der

Praxis bedeutet dies Momentan, das Seiten getypt werden können, und das die Wiki-Links (Verweise auf andere Wiki-Seiten) ebenfalls einen Typ bekommen können. Es existieren eine Reihe von semantischen Wikis (siehe http://wiki.ontoworld.org/index.php/SemWiki2006). Es gibt auch Wikis ohne kollaborative Aspekte (Persönliche Wikis) und Wikis ohne Wiki-Syntax (WYSIWYG-Wikis). Damit verwischt die Grenze zwischen Wikis und z.B. Word immer stärker. Es bleibt das Kriterium der einfachen Vernetzung übrig. Ein Wiki ist im Herzen also ein Werkzeug, um vernetztes Wissen zu erstellen, zu pflegen und zu nutzen.

4.3 Technische Randbedingungen

Die meisten Implementierungen bieten Java- oder HTTP-basierte APIs über die RDF-Daten ausgetauscht werden. Auch eine zentrale Busarchitektur zur Kommunikation ist meist vorhanden. Die zentrale Datenhalten erfolgt meist in RDF, hier sollte vorhande Inrastruktur wiederverwendet werden. Dabei ist zu beachten, das RDF allein wenig aussagt. Wichtig ist auch die Struktur der RDF-Daten, welche mit Ontologien beschrieben werden kann.

4.4 Browser als Plattform

Der Browser bietet uns auf zwei Arten eine Plattform zur Auslieferung von Technologie:

- Über AJAX, d.h. entweder
  - Dynamisches HTML mit Java Script oder
  - Flash mit Java Script
- Über Plugins, im Falle von Firefox:
  - XUL + Java Script (schlecht dokumentiert, schlecht zu debuggen, teilweise instabil und schwer vorhersagbares Verhalten)
  - XPCOM-Erweiterungen, üblicherweise mit C++ (recht komplex, schlecht dokumentiert, eventuell schwer mit Firefox-Weiterentwicklung zu synchronisieren, dafür sehr performant)

In vielen anderen Bereichen des Projekts wird Java verwendet, mit Java kennen sich alle Partner auch offenbar am Besten aus. Java-Applikationen und Plugins lassen sich relativ gut über Eclipse an die Entwickler-Zielgruppe ausliefern (siehe Abschnitt hierzu), andere Zielgruppen kann man über Java-GUIs weniger gut erreichen als über den Browser.

4.4.1 Browser-Plugins im Social Software / Semantic Web Bereich

- **Outfoxed** - Resultat einer Diplomarbeit an der Uni Osnabrück. Reichert Webseiten mit sozialer Information an, beispielweise durch die Benutzer hoch bewertete Google-Resultate.
- **Piggy-Bank** - Folgeprojekt von Haystack (MIT) als Firefox-Plugin. Auf den ersten Blick eine interessante Lösung des oben angesprochenen Java/JavaScript-Problems, da große Teile des Codes in Java geschrieben sind. Kann man per XPCOM also auf Java zugreifen? Der zweite Blick zeigt: Kann man nicht. Es handelt sich um eine relativ typische Client-Server-Lösung mit Server (RDF-

Backend) in Java und Client in Javascript, nur dass der Server auf dem gleichen Rechner läuft.

5 Sharing Engine

Die Aufgabe der Sharing-Engine kann allgemein so formuliert werden:

Gegeben ist ein räumlich verteiltes Informationssystem mit vielen, grundsätzlich gleichberechtigten Teilnehmern (man könnte auch "Datenbank" sagen). Ein Teil der darin enthaltenen Informationen gehört räumlich/netzmäßig eng konzentrierten Teams, manche nur einzelnen Teilnehmern, andere wiederum größeren Gruppen von Teilnehmern, bis hin zu öffentlichen Informationen.

Öffentliche bzw. weit verbreitete Informationen können von einzelnen Teilnehmern oder Kleingruppen annotiert werden. Anfangs private Information kann in größere Gruppen hinein publiziert werden.

Gesucht sind möglichst einfache Mechanismen, Architekturprinzipien, Protokolle, Rechtekonzepte, um gute Antwortzeiten, hohe Datensicherheit (gegen Ausfall und unerlaubten Zugriff), gerechte Lastverteilung auf Netzknoten, geringe Netzlast usw. zu gewährleisten. Ein Abfallprodukt soll letztlich die Möglichkeit zum Offline-Arbeiten auf Laptops sein.

Wir vermuten, dass es im Bereich verteilte Datenbanken, Peer2Peer-Systeme schon Antworten darauf gibt. Auch bei Produkten wie Lotus Notes oder Groove lässt sich vielleicht etwas finden.

Informationsquellen: Workshop-Serie Databases, Information Systems and Peer-to-Peer Computing (DBISP2P): Homepage und Programm 2005, 2006, DBLP-Proceedings 2004, 2003

Eventuell könnte Subversion für unsere Implementierung eine zentrale Rolle spielen.

6 Werkzeugintegration

Ziel dieses Abschnitt ist es, einen Überblick über bestehende Entwicklungsdatenbanken wie Bugtracking-Systeme, Anforderungsverwaltungssysteme, Versionsverwaltungssysteme, Buildsystemen, Ressourcenplanungssysteme, Teststeuerungssysteme zu geben. Manche von diesen Werkzeugen kommen als potentielle Bestandteile für die in Waves entstehende Werkzeuginfrastruktur in Frage. Einen separaten Abschnitt widmen wir solchen Werkzeugen, die es ermöglich en, Metadaten aus Quelltext zu extrahieren. Zum Schluss wird Eclipse als ein mögliches Integrationsplattform für die Waves-Tools vorgestellt.

6.1 Untersuchung bestehender Entwicklungsdatenbanken

6.1.1 Bugtracking-Systeme

Bugtracking-Systeme dienen zur Erfassung und Dokumentation von Programmfehlern. Mit ihnen werden, oft interaktiv und im Internet, auch Status- oder Feature-Berichte geschrieben. Daneben nehmen sie auch Verbesserungsvorschläge und Wünsche der Nutzer oder allgemeine Vorgänge auf. Bei manchen Projekten spricht man dann zum Beispiel von Metabugs, wo ein Bug ein Element einer Aufgabenliste darstellt. Bei ande-

ren Projekten spricht man stattdessen von „Issues“ (Angelegenheiten), da sich dieser Ausdruck nicht auf Programmfehler beschränkt.

6.1.1.1 Werkzeuge

- **Bugzilla**: Bugzilla ist ein in Perl geschriebenes Open-Source-Programm, das unter der Mozilla Public License steht und damit kostenlos erhältlich ist. Es setzt auf die CGI-Schnittstelle eines Webservers auf; die Benutzung erfolgt über eine HTML-Oberfläche. Als Datenbank dient standardmäßig MySQL[?]. Es ist unter Linux, Mac OS X und Windows lauffähig.

- **Mantis**: Mantis ist eine in PHP geschriebene Open-Source-Software, die unter der GNU General Public License steht und damit kostenlos erhältlich ist. Sie basiert auf PHP und benötigt daher einen Webserver. Der Benutzer bedient Mantis über eine HTML-Oberfläche. Als unterlagerte Datenbank dient MySQL[?]. Es ist unter Linux, Mac OS X, Windows, OS/2 und Unix lauffähig. (http://www.mantisbt.org/)

- **Trac**: Trac ist ein freies, webbasiertes Projektmanagement-Werkzeug. Es enthält eine webbasierte Oberfläche zum Betrachten von Subversion-Repositories, ein Wiki zum kollaborativen Erstellen und Pflegen von (z.B.) Dokumentation, und einen Bug-Tracker zum Erfassen und Verwalten von Programmfehlern und Erweiterungswünschen. Trac ist in Python implementiert, und kann via CGI, FastCGI[?] oder mod_python betrieben werden. Trac ist modular geschrieben und kann damit durch Plugins erweitert werden. (http://trac.edgewall.org/)

- **Track+**: Track+ ist ein Werkzeug für das Projektmanagement. Da Track+ webbasiert und unter einer Open Source-Lizenz verfügbar ist, wird es in vielen Projekten für Aufgabenmanagement und als Bugtracker eingesetzt. Track+ nutzt einen Webserver oder Applikationsserver, wie z.b. Tomcat oder JBoss, und einen Datenbankserver, wie z.B. MySQL[?] oder PostgreSQL[?]. Vorgänge (Issues) können von jedem mit entsprechenden Rechten ausgestatteten Benutzer eingegeben werden, und können dann einem Bearbeiter zugeordnet werden. Jeder Vorgang durchläuft einen Lebenszyklus, der gekennzeichnet ist durch einen Status und einen Verantwortlichen. Jeder Vorgang kann mit Beschreibungen, Dokumenten jeder Art und Verweisen auf andere Vorgänge verknüpft werden. Vorgänge können hierarchisch organisiert werden. Track+'s Interpretation eines Vorgangs ist recht allgemein gehalten; z.B. kann ein Vorgang eine Anforderung, ein Fehler (Bug), eine Problemmeldung, ein Meilenstein, ein Risiko usw. sein. Vorgangstypen können vom Benutzer definiert werden und lassen sich jeweils mit eigenen Arbeitsabläufen (Workflows) verknüpfen. Übersichten lassen sich in textueller Form und grafisch, z.B. als Gantt-Diagramme darstellen. (http://www.trackplus.com/)

- **Jira**: Jira ist ein Bug-Tracking, Issue-Tracking und Projekt-Management Werkzeug. Über ein Dashboard bietet es eine einheitliche Sicht für alle Benutzer an. Zur Verwaltung der Bugs/Issues gehört auch ein mächtiges Filter– bzw. Sortierungsmechanismus und Email-Benachrichtigung dazu. (http://www.atlassian.com/software/jira/tour/)

- **Bugzero**: Bugzero ist ein Web-basiertes Bug/Defekt/Issue/Änderungsverfolgungssystem, das in einer verteilten Team-basierten Software-Entwicklung eingesetzt werden kann. Es kann auch Helpdesks unterstützen, indem es erleichtert Kunden-Feedbacks, Änderungs/Erweiterungsvorschläge zu verwalten. Das Werkzeug ist pur in Java imp-

lementiert, benutzt eine SQL-Datenbank und ist kompatibel mit gängigen Browsern. (http://www.websina.com/bugzero/)

6.1.2 Versionsverwaltungssysteme

Unter einer Versionsverwaltung versteht man ein System, welches typischerweise in der Softwareentwicklung zur Versionierung und um den gemeinsamen Zugriff auf Quelltexte zu kontrollieren, eingesetzt wird. Hierzu werden alle laufenden Änderungen erfasst und alle Versionsstände der Dateien in einem Archiv mit Zeitstempel und Benutzerkennung gesichert. Es wird sichergestellt, dass jeder Benutzer mit dem aktuellen Stand arbeitet oder auf Wunsch auf die archivierten Stände zugreifen kann. Dadurch ist eine Versionsverwaltung nicht nur für professionelle Entwickler in großen Teams, sondern auch für einzelne Entwickler interessant. Es kann jederzeit eine ältere Version aufgerufen werden, falls eine Änderung nicht funktioniert und man sich nicht mehr sicher ist, was nun alles geändert wurde.

6.1.2.1 Werkzeuge

- **CVS**: Concurrent Versions System (CVS, nicht zu verwechseln mit CSV) ist ein Software-System zur Versionsverwaltung von Dateien, hauptsächlich Software-Quelltext. Das ursprüngliche CVS ist ein reines Kommandozeilen-Programm, aber es wurde für alle gängigen Betriebssysteme mindestens eine grafische Oberfläche entwickelt, zum Beispiel TortoiseCVS und WinCVS für Windows, MacCVS für den Apple Macintosh und Cervisia für KDE unter Linux. LinCVS und SmartCVS funktionieren auf Windows, Linux und Mac OS X. CVS erfreute sich besonders in der Open-Source-Gemeinde großer Beliebtheit. So wurde es bei den meisten großen Open-Source-Projekten verwendet. Allmählich wird CVS durch andere Entwicklungen wie Subversion ersetzt.

- **PVCS**: PVCS ist ein Software Configuration Management System der Firma Merant. PVCS eignet sich für Projekte jeder Größenordnung: PVCS deckt vom intelligenten Versionieren von Source-Dateien über umfassende Kommunikation im Team bis hin zum prozess-gesteuerten Software Configuration Management alle Anforderungen, insbesondere hochkomplexer Projekte ab.

- **VSS** (Visual Source Safe): Visual SourceSafe (VSS) ist ein Programm von Microsoft zum Verwalten von Software- Entwicklungen (Quellcode), wie z. B. VisualBasic-, C++ oder C-Programme, in einer Datenbank.

Microsoft Visual SourceSafe besitzt außerdem eine Versionsverwaltung, welche beim "Auschecken" (Laden des Quellcodes aus der Datenbank) automatisch die Versionsnummer im Quellcode hochzählt, wenn die entsprechenden Felder vorhanden sind.

- **SubVersion**: Subversion (SVN) ist eine Open-Source-Software zur Versionsverwaltung. Da es viele Schwächen des in Entwicklerkreisen sehr beliebten Programms CVS behebt, wird Subversion oft als dessen Nachfolger bezeichnet, obwohl es sich um ein eigenständiges Projekt handelt. Es ist jedoch absichtlich von der Bedienung sehr ähnlich gehalten. Zusätzlich zu vielen neuen Features werden fast alle Funktionen von CVS unterstützt. Verschiedene kostenfreie Import-Werkzeuge (CVS, PVCS, VSS, ClearCase, MKS) sind ebenfalls erhältlich. Subversion wurde unter einer Lizenz im Stil der Apache-Lizenz veröffentlicht.

- **Rational** ClearCase: ClearCase ist ein System zur Versionsverwaltung von Quellcode und anderen Softwareentwicklungsdaten. Der ClearCase Client in-

tegriert sich transparent in die entsprechenden Dateimanager (Explorer, Total-Commander[2] etc.), so dass der Zugriff auf die entsprechenden Dateien ganz normal über gewohnte Tools möglich ist.

6.1.3 Build-Systeme

- **make**: ist ein Computerprogramm, das Shellskript-ähnlich Kommandos in Abhängigkeit von Bedingungen ausführt. Es wird hauptsächlich im Unixbereich bei der Softwareentwicklung eingesetzt. Genutzt wird es beispielsweise, um in einem Projekt, das aus vielen verschiedenen Dateien mit Quellcode besteht, automatisiert alle Arbeitsschritte (Übersetzung, Linken, Dateien kopieren etc.) zu steuern, bis hin zum fertigen ausführbaren Programm.

- **ANT**: Im Gegensatz zu make ist Ant plattformunabhängig - zumindest solange auf der Zielplattform eine Java-Laufzeitumgebung (JRE) verfügbar ist. Gesteuert wird Ant durch eine XML-Datei, die so genannte Build-Datei. In dieser wird zunächst ein Projekt definiert, welches Targets (deutsch „Ziele“) enthält. Diese sind vergleichbar mit Funktionen in Programmiersprachen und enthalten u. a. Aufrufe von Tasks (deutsch „Aufgaben“). Ein Task ist ein untrennbarer Arbeitsschritt. Zwischen den Targets sollten Abhängigkeiten definiert werden. Beim Aufrufen eines Targets löst Ant diese Abhängigkeiten auf und arbeitet die Targets entsprechend ab. Wenn man ein einzelnes Target definiert hat, welches direkt oder indirekt Abhängigkeiten zu allen anderen Targets hat, so genügt es, dieses aufzurufen und Ant führt dann alle notwendigen Arbeitsschritte in der richtigen Reihenfolge aus. Beim Projekt kann dieses Target als default angegeben werden. Ant ist ein offenes System mit exakt definierten Schnittstellen, was bedeutet, dass es z. B. durch selbst erstellte Tasks beliebig erweitert werden kann. Viele Java-Werkzeuge bieten Unterstützung für Ant, wodurch die Einbindung oftmals sehr einfach ist. Außerdem lässt es sich auch in eigene Anwendungen – z. B. Installationsprogramme – einbinden, um die verschiedensten, meist Batch-artigen, Aufgaben zu übernehmen.

- **Maven**: Maven ist eine Weiterentwicklung des Apache-Ant-Projekts, welches auch weiterhin den Kern von Maven ausmacht. Maven basiert auf Java und ist ein Build-Management-Tool, welches die Erzeugung und Verteilung von Java-Programmen managen soll. Maven basiert auf einer Plugin-Architektur, die es ermöglicht, Plugins für verschiedenste Anwendungen (compile, test, build, deploy, checkstyle, pmd, scp-transfer) auf das Projekt anzuwenden, ohne diese explizit installieren zu müssen.

6.1.4 Anforderungsverwaltungssysteme

Die Anforderungsanalyse ist ein Teil des Software- und Systementwicklungsprozesses. Sie dient dazu, die Anforderungen des Auftraggebers an das zu entwickelnde System (vielfach ein Anwendungsprogramm) zu ermitteln. Diese Anforderungen werden dabei in einem Dokument, dem sogenannten Anforderungskatalog, schriftlich fixiert. Hierbei kann die Software Requirements Specification zum Einsatz kommen.

6.1.4.1 Werkzeuge

Auf dem Markt sind eine Vielzahl von Anforderungsverwaltungssystemen vorzufinden. Die bekanntesten sind:

- Acept 360

- Active Focus von Xapware
- Agility von Agile Edge
- AnalystPro von Goda Software
- Caliber-RM von Borland
- C.A.R.E. von Sophist Group
- Cradle von 3SL
- Clear Requirements Workbench (CRW) von LiveSpecs Software
- DocuBurst von Teledyne Brown Engineering
- Doors von Telelogic
- Focal Point
- Gatherspace
- GMARC von Computer System Architects
- iRise von iRise.com
- IRqA von TCP Sistemas e Ingeniería
- Jalsoft von Jalsoft
- Leap SE von Leap Systems
- MKS Requirements 2005 von MKS
- MockupScreens von Igor Jese
- Objectiver von Cediti
- OPEN Process Framework (OPF) von Firesmith Consulting
- Rally von Rally Software Development
- RDT von IgaTech Systems
- Reconcile von Compuware
- Reqtify von TNI-Valiosys
- Requisite Pro von IBM Rational
- Rhapsody von Telelogic (formerly I-Logix)
- ScenarioPlus
- Serena RTM von Serena
- SpeeDEV von SpeeDEV
- Statestep von Statestep
- Steeltrace (formerly Catalyze) von SteelTrace

- Teamcenter von UGS
- Team-Trace von WA Systems
- Truereq von Truereq Inc.
- Vital-Link von Compliance Automation
- The Volere Template von The Atlantic Systems Guild
- WIBNI von Project Toolbox
- XTie-RT von Teledyne Brown Engineering (TBE)

Es würde die Rahmen dieses Dokumentes sprengen jedes Werkzeug einzeln vorzustellen. Deshalb verweisen wir hier auf eine sehr gute Zusammenfassung von den wichtigsten Merkmalen dieser Werkzeuge. Hier ist außerdem ein sehr guter Vergleich von existierenden Anforderungs-Management-Werkzeugen zu lesen. Zum Vergleich werden die Werkzeuge anhand einer Vielzahl von Kriterien untersucht.

6.1.5 Ressourcenplanungssysteme/Enterprise-Resource-Planning-Systeme

Der Begriff Enterprise Resource Planning bezeichnet die unternehmerische Aufgabe, die in einem Unternehmen vorhandenen Ressourcen (wie zum Beispiel Kapital, Betriebsmittel oder Personal) möglichst effizient für den betrieblichen Ablauf einzuplanen. Der ERP-Prozess wird in Unternehmen heute häufig durch Software-ERP-Systeme unterstützt.

6.1.5.1 Werkzeuge

Ein großer Teil des weltweiten ERP Marktes ist zwischen den Anbietern SAP (R/3, mySAP), Oracle (E-Business Suite), PeopleSoft (von Oracle gekauft), Sage (in Deutschland Office Line und Classic Line), und Microsoft (Axapta und Navision) aufgeteilt. Alle großen Anbieter bieten zu den oben genannten Funktionsbereichen (z. B. Finanzwesen und Materialwirtschaft) Module an, die von den Kunden auf ihre individuellen Bedürfnisse angepasst werden können (durch sogenanntes Customizing). Seit einiger Zeit gibt es auch freie Software für ERP, zum Teil auch unter lizenzgebührenfreien Open Source Lizenzen. Freie ERP-Software wird von ERP-Anbietern angeboten, die auf Basis dieser Software kostenpflichtige Dienstleistungen erbringen. Einige verfügbare ERP-Software Systeme mit offengelegtem Quellcode sind zurzeit: AvERP, Compiere, IntarS, Lx-Office, SQL-Ledger, und webERP.

6.1.6 Teststeuerung

Als Software-Test bezeichnet man in der Informatik ein mögliches Verfahren zur teilweisen Verifikation und Validierung eines Programms. Ein Software-Test dient der Qualitätssicherung einer neu erstellten oder geänderten Software. Dabei geht es prinzipiell darum, das tatsächliche Verhalten mittels Testfällen und gemäß eines Testplans zu untersuchen und die Ergebnisse mit den erwarteten Ergebnissen (Anforderungskatalog, Normen usw.) zu vergleichen und zu dokumentieren.

6.1.6.1 Werkzeuge

Die einzelnen Test-Verfahren vorzustellen würde die Rahmen dieses Dokumentes sicherlich sprengen. Da der Fokus dieses Arbeitspaketes stark auf Werkzeugen liegt, fokussieren wir uns jetzt auf die Automatisierung von Tests, bzw. welche Tools entwicklungsbegleitend, im Rahmen einer kollaborativen Softwareentwicklung eingesetzt werden können. Die Liste ist sicherlich nicht vollständig.

- **JUnit**: JUnit ist ein Framework zum Testen von Java-Programmen, das besonders für automatisierte Unit-Tests einzelner Units (meist Klassen oder Methoden) geeignet ist. Es basiert auf Konzepten, die ursprünglich unter dem Namen SUnit für Smalltalk entwickelt wurden.
- **NUnit**: NUnit ist ein Unit-Test-Framework für die .NET-Sprachen. Die Konzepte sind die gleichen, die bei JUnit benutzt werden.
- **CppUnit**: CppUnit ist ein C++-Unit-Test-Framework. Es ist dem Java-Tool JUnit nachempfunden.
- **Hyades**: Hyades ist ein freies (Open Source) Werkzeug, welches die Möglichkeit bietet Software automatisiert auf ihre Qualität zu überprüfen. Dafür dient es mit einer integrierten Test-, Ablaufverfolgungs- und Überwachungsumgebung um Interoperabilität zwischen den einzelnen Testprozessen zu gewährleisten. Dabei basiert Hyades auf dem Eclipse-Framework.

6.1.7 Polarion [pol]

Die Hauptfunktionalitäten des Polarion Application Lifecycle Management Systems sind:

- Anforderungsmanagement
- Projektplanung, Projekttracking
- Change-Management, inklusive Bug-Tracking
- Konfigurationsmanagement (entweder auf SubVersion oder auf SAP NetWaver basierend)
- Task-Management
- Dokumentation und Reporting
- Metriken und Audit
- Zentralisiertes Build-Management
- Zugang auf sämtliche Polarion-Funktionalitäten aus aller Welt über einen Browser

Polarion enthält ein Tutorial, das die Einarbeitung in die Software enorm erleichtert. Nach dem Einloggen in das Polarion-System geht man zu den eigenen Tasks. Die Schritte vom Tutorial sind in der Form von Tasks definiert.

Die Architektur von Polarion kann wie folgt charakterisiert werden:

- Benutzte Technologien

  - Subversion (siehe oben)
  - Apache Web Server
  - Apache Maven (siehe oben)
  - OpenSymphony[?] [opensym]: OpenSymphony[?] bietet eine Vielzahl von qualitativ hochwertigen J2EE-Komponenten. Es beinhaltet eine Vielzahl von Unterprojekten, wie zum Beispiel Quartz.
  - Quartz [quartz] : Quartz ist ein Job Scheduling Framework, das im Grunde genommen in beliebigen Java-Anwendung benutzt werden kann. Das Framework skaliert sehr gut, kann nicht nur bei den kleinsten Anwendungen sondern auch bei komplexen E-Commerce-Systemen eingesetzt werden. Der Scheduling Algorithmus von Quartz kann sogar mit mehreren zehntausend Jobs umgehen.

- Repository
  - Das Repository ist Subversion
  - Das Repository enthält den Software-Quelltext (z.B. Java-Dateien), aber auch projektbezogene Dokumente, die in Polarion gespeichert werden (z.B. Produkt-Anforderungen usw.)
  - Die projektbezogenen Polarion-Metadaten sind in der Form von XML-Dateien auch in der Repository hinterlegt.
  - In SVN können auch Daten von anderen Repositories importiert werden. Es sind Importers für CVS, PVCS, VSS, ClearCase und MKS vorhanden.
  - Vermutlich gibt es entsprechende Java-Klassen, die den Umgang mit den Polarion-Metadaten und der Repository erleichtern. (Zugänglichkeit der API?)

6.2 Extraktion von Metadaten aus Quellcode

6.2.1 SiSSy (Structural Investigation of Software Systems) [sissy]

SISSy ermittelt automatisch den aktuellen Zustand der inneren Qualität eines Softwaresystems. Damit lässt sich eine degenerative Entwicklung des Systems frühzeitig erkennen und verhindern. So kann der laufende Wartungsaufwand minimiert und gleichzeitig die innere Qualität verbessert werden. Dies führt zu einer signifikanten Reduktion der Softwarekosten.

SISSy untersucht einen Quelltext auf über 50 verschiedene Problemmuster, die einen negativen Einfluss auf Eigenschaften wie Analysier- und Modifizierbarkeit, Prüfbarkeit und Stabilität sowie Zeit- und Verbrauchsverhalten haben.

Die Problemmuster sind als Verletzungen anerkannter Entwurfsprinzipien definiert und damit durch konkrete Verbesserungsvorschläge gezielt behebbar.

Während der Analyse erstellt SISSy ein abstraktes Modell des Quellcodes und legt es in einer relationalen Datenbank ab. Dieses Modell ist dabei eine Instanz des SISSy-Systemmodells, welches als eine Ontologie für Softwarestrukturen aufgefasst werden

kann. Die Problemmuster sind als SQL-Anfragen implementiert, welche auf die Datenbank zugreifen. Diese Vorgehensweise ermöglicht eine einfache Erweiterung des Problemmuster-Katalogs.

SISSy ist während des kompletten Lebenszyklus eines Softwaresystems einsetzbar. Mit Hilfe von SISSy kann schon während des Programmierens des Quellcodes überwacht werden, ob die vorgegebene Architektur auch durchgehend eingehalten wird.

Nach der Implementierung des Systems überwacht SISSy kontinuierlich sämtliche Änderungen und Erweiterungen. Auf Basis eines Referenz-Checks lassen sich während der gesamten Software-Evolution Qualitätsänderungen im Quelltext aufzeigen:

Bei folgenden Qualitätsanalysen werden die neu gewonnenen Systemdaten mit den Soll-Werten verglichen. Durch ein Visualisierungs-Tool wird ersichtlich, welche Problemmuster im Vergleichszeitraum verstärkt auftraten bzw. welche Probleme behoben wurden.

Weiterhin können die aus der Analyse gewonnenen Daten mit bereits gesammelten Werten aus früheren Analysen verglichen werden. Eine anonymisierte Datenbank erlaubt es dem Kunden, sein Softwaresystem mit anderen auf dem deutschen Markt eingesetzten Systemen zu vergleichen. Dies ermöglicht eine umfassende Einschätzung des Sachverhalts durch den Kunden und zeigt ihm die Konkurrenzfähigkeit des eigenen Systems auf.

SISSy ist branchenübergreifend einsetzbar und kann Quellcodes unterschiedlicher Sprachen (Java, C/C++, Delphi) und beliebiger Länge analysieren.

6.3 Eclipse-basierte Werkzeugintegration

Eclipse ist ein Open-Source-Framework, das meist als Entwicklungsumgebung (IDE) genutzt wird. Bis einschließlich zur Version 2.1 war Eclipse als erweiterbare IDE konzipiert. Seit Version 3.0 ist Eclipse selbst nur der Kern, der die einzelnen Plugins lädt, die dann die eigentliche Funktionalität zur Verfügung stellen. Diese Funktionalität nennt sich Rich Client Platform (kurz RCP) und basiert auf dem OSGi-Standard. Sowohl Eclipse als auch die Plugins sind vollständig in Java implementiert. Als GUI-Framework zur Erstellung der grafischen Oberfläche wurde SWT verwendet. Zur Darstellung der GUI-Komponenten basiert SWT ähnlich wie AWT auf den nativen GUI-Komponenten des jeweiligen Betriebssystems. Eclipse ist daher nicht plattformunabhängig, wird aber für 11 verschiedene Systeme und Architekturen bereitgestellt. Die Plugins lassen sich durch den Download direkt in Eclipse von einem Update-Server oder durch einfaches Entpacken installieren. Falls in Waves Eclipse als Integrationsplattform gewählt wird, bietet sich der Plugin-Mechanismus als ein mögliches Kandidat an.

7 Ontologien zum Software-Lifecycle-Management

7.1 Einführung

7.1.1 Ontologien

Ontologien sind formale Konzeptualisierungen, die in einer bestimmten Domäne gemeinsam genutzt werden. Ontologien beschreiben die Domäne üblicherweise anhand von Konzepten und ihren Beziehungen untereinander. Typische Anwendungsfelder

sind Informationsintegration, Medizininformatik und in zunehmenden Maße auch Software Engineering (vgl. z.B. [hoo1]).

Im Software Engineering bieten sich wiederum eine Vielzahl konkreter Anwendungszwecke für Ontologien. Beispielsweise wird unter dem Stichwort "Ontology-driven Architecture" der Einsatz von Ontologien als Artefakte im Software-Design diskutiert, aus dem entsprechend dem Paradigma der modellgetriebenen Entwicklung weitere Artefakte wie z.B. Quellcode abgeleitet werden (vgl. [w3c]).

7.1.2 Ontologien für den Software-Lebenszyklus

In WAVES sollen Ontologien vornehmlich dazu dienen, den Wissensaustausch zwischen Werkzeugen und Entwicklern zu verbessern. Somit ist das Ziel der in WAVES zu benutzenden Ontologien, die Beschreibung der Domäne Software Engineering selbst. Im Folgenden unterscheiden wir dabei Ontologien die den Aufbau von Artefakten - d.h. Software-Strukturen beschreiben, und solche, die den Entwicklungsprozess bzw. Lebenszyklus von Software erfassen. Der Zweck dieser Ontologien ist zunächst eine formale Beschreibung der Domäne, mit dem Ziel eine nachvollziehbare Beschreibung davon zu bekommen, was etwa eine "Komponente" ist. Dies hilft zum einen der Kommunikation von Entwicklern, aber auch für den Datenaustausch über Werkzeuggrenzen hinweg. Schliesslich ermöglicht es eine leichtere Wartung von Softwaresystemen, da ein einheitliches Modell zur Verfügung steht, um Informationen aus unterschiedlichen Quellen unabhängig vom ursprünglichen Format zu Verknüpfen.

Dieses Dokument soll einen Überblick über vorhandene Arbeiten über Ontologien zur Beschreibung von Softwarestrukturen und des Software-Lebenszyklus geben. Von besonderem Interesse ist dabei zum einen die Existenz von Standards zur Beschreibung, sowie die Arbeiten der Konsortialpartner in diesem Bereich. Dabei handelt es sich um das Modell zur Beschreibung des Softwarelebenszyklus von Polarion sowie diverse Vorarbeiten am FZI. Im Kontext dieser Einführung ist es zunächst nicht von Belang, ob diese Konzeptualisierungen technisch anhand einer Ontologiesprache, anhand von relationalen Schemata, UML oder einfach durch harte Codierungen repräsentiert werden.

7.1.3 Bewertungsrahmen

Im Folgenden wird ein grobes Raster zur Charakterisierung der Ontologien vorgestellt. Aufgrund der Heterogenität der untersuchten Modelle wird dabei eine qualitative Beschreibung dem Einsatz exakter Metriken vorgezogen. Teile des Bewertungsrahmens orientieren sich an der Arbeit von Oberle ([dob1], S. 43ff).

Folgende Kategorien werden verwendet:

- Anwendungsdomäne/Zweck der Ontologie (application/reference)
- Gegenstand/Hauptelemente/Konzepte: In dieser Dimension wird kurz skizziert, was die Ontologie beschreibt. Dazu werden jeweils einige charakteristische Beispiel-Konzepte aufgefährt.
- Spezifizität: Die Spezifität von Ontologien richtet sich nach der Abstraktionsebene der beschriebenen Konzepte. "Generic" Ontologies beinhalten eher allgemeingültige Konzepte wie "Komponente", "Ereignis" oder "Prozess", "core" Ontologies definieren Konzepte die in einer Menge von Domänen gültig sind

(z.B. "Software-Komponente", während "domain" Ontologies für eine Domäne spezifische Konzepte definieren (z.B. "EJB-Komponente"). Die Abgrenzung zwischen den drei Ebenen ist häufig nicht eindeutig möglich. In gut gestalteten Ontologien, sollte jedoch eine Zuordnung erkennbar sein, die sich auch darin äussert, das Konzepte der niedrigeren Schichten von generischen Konzepten abgeleitet sind.

- Struktur und Abhängigkeiten: Hier wird beschrieben, ob die Ontologie in mehrere Teilontologien modularisiert ist, oder andere Ontologien importiert.
- Standards/Interoperabilität/Erweiterbarkeit: Falls die Ontologie internationale Standards berücksichtigt oder spezifische Erweiterungsmechanismen vorsieht, wird dies hier erwähnt.
- Beschreibungsprache: Hier wird das Format beschrieben, in dem die Ontologie vorliegt. Beispiele sind z.B. Sprachen zur Wissensrepräsentation wie OWL oder RDF.
- Expressiveness: Mit der Ausdrucksmächtigkeit der Ontologie wird die Detailliertheit der Axiomatisierung der Konzepte bezeichnet. Während "schwergewichtige" (heavyweiht) Ontologien eine möglichst präzise Axiomatisierung anstreben, ähneln "leichtgewichtige" (lightweight) Ontologien häufiger Taxomomien, benötigen werniger Ausdrucksstarke Beschreibungssprachen und erlauben effizieteres Schlussfolgern (reasoning).

Dieser Bewertungsrahmen dient als Struktur für die folgenden Beschreibungen. Eine detaillierte Auflistung in Tabellenform findet sich in Abbildung 1.

7.2 Ontologien für Softwarestrukturen

In diesem Abschnitt werden verschiedene Metamodelle bzw. Ontologien für Softwarestrukturen vorgestellt. Diese lassen sich ebenfalls grob in zwei Gruppen bündeln:

- Ontologien zur Beschreibung von Quelltextstrukturen orientieren sich stark an programmierspezifischen Konzepten
- Ontologien zur Beschreibung von Softwarearchitekturen beschreiben Komponenten und Konnektoren eines Systems auf einer abstrakteren Ebene - unabhängig von konkreten Programmiersprachen

7.2.1 Welty

Chris Welty hat sich als einer der ersten Forscher mit der Anwendung von Beschreibungslogiken auf Probleme im Software Engineering befasst. Seine Ziel war dabei, die Wartbarkeit grosser Softwaresysteme mit Hilfe so genannter "Software Information Systems" (SIS; [dev1], [wel2]) zu steigern. Seine Hauptthese ist, dass Programmiersprachen als Standardrepräsentation von Wissen im Software Engineering für typische Wartenungsaufgaben schlecht geeignet sind. Sie beschreiben Wissen prozedural und richten sich eher auf die Ausführung von Code als auf die Abfrage von Wissen hin aus.

Daher hat er, aufbauend auf der LaSSIE Ontologie [dev1]), eine Ontologie zur programmiersprachen-unabhängigen Beschreibung von Softwarestrukturen [wel3] entworfen. Diese Vernetzt die Elemente des Quellcodes explizit und bietet somit eine Gesamtsicht auf das Wissen über die Zusammenänge von Klassen, Methoden und Variablen.

7.2.2 COHSE

Das COHSE Projekt (http://cohse.semanticweb.org/software.html) beschäftigt sich mit einer verbesserten Verlinkung von Internetseiten mit Hilfe von Semantic Web Technologien. Ein Anwendungsfall war dabei die Java API Dokumentation. Ziel war ein höherer Verlinkungsgrad des Wissens innerhalb der Webseiten, die die API beschreiben.

Aus diesem Grunde wurde eine COHSE Java Ontologie entwickelt, mit deren Hilfe Konzepte auf Webseiten entdecket, annotiert und verlinkt werden. Die Ontologie hat drei Grundkonzepte Programming concept (Generelles Konzept der Programmierung - unabhängig von einer konkreten Programmiersprache), Java Concept (Java-spezifische Konzepte) und Technical Concept (Technische Konzepte, die keine Programmierkonzepte sind). Allerdings erfasst sie nicht sämtliche Konzepte von Java, da sie auf den konkreten Anwendungsfall eines Teils der API Dokumentation zugeschnitten ist.

7.2.3 IEEE 1471

Im IEEE Standard 1471 [ieee1] wird ein konzeptueller Rahmen für die Beschreibung von Softwarearchitekturen definiert. Ziel ist eine Methoden-neutrale, Format-unabhängige Konzeptualisierung der Kernelemente von Softwarearchitekturen, um einen einheitlichen Rahmen für die Kommunikation über Softwarearchitekturen bereitszustellen.

Das Modell spezifiziert zwölf Kernelemente wie "System", "Stakeholder", "View" oder "Concern" und ihre Beziehungen untereinander. Da es als Referenzmodell gedacht ist, und nicht zur direkten Anwendung, liegt es nicht in einer formalen Sprache vor.

7.2.4 SiSSy-Metamodell

Im Rahmen eines Forschungsprojektes QBench (http://www.qbench.de) wurde maßgeblich vom FZI (Forschungszentrum Informatik an der Universität Karlsruhe) ein Metamodell zur Beschreibung von objekt-orientierter Software entwickelt. Mittels Faktenextraktoren werden aus dem Quelltext des Softwaresystems, das in einer der Sprachen Java, C++ oder Delphi implementiert ist, ein Systemmodell aufgebaut. Das Systemmodell wird persistent in einer SQL-Datenbank abgespeichert und dient als Grundlage für weitere Analysen wie Software-Qualitätsanalyse und -Verbesserungsmassnahmen, Visualisierungen der extrahierten Software-Strukturen, usw.

Das SiSSY-Metamodell (Structural Improvement of Software Systems) vereinigt eine Vielzahl von Eigenschaften von den Programmiersprachen C++, Java, Delphi und C#. Es werden zum Beispiel Pakete, Dateien, Klassen, Members (Methoden und Attribute), globale Funktionen, globale Variablen, Referenzen u.a. Aufrufe, Read-/Write-Zugriffe, Vererbungsbeziehungen, Aggregationen, Typzugriffe und Anweisungen (Statement) unterschieden. Das Metamodell definiert die Syntax und Semantik deren möglichen Instanzen, von den so genannten Systemmodellen. Ein Systemmodell enthält zum Beispiel alle Pakete, Klassen, Methoden, Attribute usw. die in dem jeweiligen Softwaresystem vorzufinden sind.

7.2.5 FAMOOS Information Exchange Model (FAMIX)

FAMIX [tic1] ist ähnlich zu SiSSy ein Metamodell zur Beschreibung der Konzepte objektorientierter Programmiersprachen. Es wurde entwickelt, um Informationen zwischen Werkzeugen Auszutauschen, die verschiedene konkrete objektorientierte Sprachen nutzen. Daher nutzt FAMIX das Format CDIF, das für den Datenaustausch zwischen Entwicklungstools entwickelt wurde. Über einen Plugin-Mechanismus können bei Bedarf weitere Programmiersprachen in das FAMIX-Modell integriert werden.

7.3 Ontologien für Software Lebenszyklen

7.3.1 Core Ontology of Software

Im Rahmen seiner Arbeiten zum semantischen Verwaltung von Middleware [dob1] hat Oberle eine Reihe von Ontologien zur Beschreibung von Konzepten der kompenentenbasierten und service-orientierten Entwicklung erstellt (http://cos.ontoware.org/). Ziel ist dabei die Unterstützung eines Systemadminstrators bei der Verwaltung serverbasierter Anwendungen, indem z.B. Abhängigkeiten zwischen Bibliotheken formal spezifiziert werden.

Bei der Konzeptualisierung der Ontologie wurden vor allem zwei Ziele verfolgt: eine präzise formale Definition von überladenen Begriffen aus dem Software Engineering ("Komponente", "Service") sowie in Beschreibungsschema zur Unterstützung von Middleware-Verwaltung (u.a. durch die Abbildung von Abhängigkeiten zwischen Bibliotheken, Lizenzen, etc.). Beachtenswert ist, dass die Ontologie an Top-Level Ontologie DOLCE [dol1] angepasst ist und in drei Teilontologien Core Software Ontology, Core Ontology of Software Components sowie Core Ontology of Services modularisiert ist.

7.3.2 Dhruv

Dhruv [aan1, aan2] ist ein System zur Unterstützung von Problemlösungsprozessen in Web-Communities, das auf Semantic-Web Technologien basiert. Das konkrete Anwendungsszenario des Prototypen sind Open Source Entwicklergemeinschaften und Defekte (Bugs) in der zu entwickelnden Software. Eine Ontologie soll dabei helfen, die elektronische Kommunikation der Entwickler mit den Bug-Reports sowie betroffenen Regionen im Quellcode zu verknüpfen, und somit den Problemlösungsprozess effizienter zu machen.

Zentrale Entitäten sind dabei die community (Entwickler), ihre Interaktionen sowie Inhalte (z.B. Mailverkehr auf der Entwickler-Mailungsliste. Wissen darüber wird in drei Arten von Ontologien erfasst. Zwei Content-Ontologien - eine bezüglich Software-Strukturen (basierend auf den Arbeiten von Welty) sowie eine Bug-Taxonomie - beschreiben die Struktur vorhandener Artefakte. Eine Interaktions-Ontologie beschreibt die Kommunikationsflüsse zwischen den Entwicklern und die Community-Ontologie definiert die verschiedenen Rollen, die in den Problemlösungsprozess involviert sind.

7.3.3 Falbo

Falbo et al. [fal1] arbeiten an der Vision einer ontologiebasierten Entwicklungsumgebung (ODE). Dazu wurden verschiedene Tool-Prototypen entwickelt, die im Hintergrund mit verschiedenen Ontologien arbeiten, die den gesamten Software-Lebenszyklus in Form von Prozessen, Werkzeugen und Qualität beschreiben.

Die Ontologie spezifiziert Konzepte wie etwa "Software Process", "Activity", "Artifact", "Input", "Output" und liefert damit eher ein Metamodell für Software-Lebenszyklen. Die konkrete Ausprägung einzelner Phasen bzw. Definition von Prozessen muss mit den entsprechenden Werkzeugen der ODE erfolgen.

7.3.4 Polarion

Der Polarion-Suite liegt eine teilweise konfigurierbare Objekt-Struktur zugrunde, mit Anforderungen, Bugs, Änderungen, Code-Modulen, usw. Darauf aufbauend werden Beziehungen (Änderung erfolgte-aufgrund-von Anforderung) und (Workflow-)Regeln (Bug-Ticket darf geschlossen werden, wenn alle untergeordneten Bug-Tickets geschlossen sind) definiert. Es handelt sich also im obigen Sinne um eine Ontologie. [pol]

7.4 Zusammenfassung

Die Ergebnisse der Betrachtung verschiedener Modelle sind in der folgenden Tabelle zusammengefasst.

Name	Anwendungsdomäne/Zweck der Ontologie (application/reference)	Gegenstand/Hauptelemente/ Konzepte	Spezifizität (generic/core/ domain)	Struktur und Abhängigkeiten	Standards/Interoperabilität /Erweiterbarkeit	Beschreibungsprache (OWL, RDF...)	Expressiveness (heavyweight/lightweight)
Welty (Code-Level)	Software Information System (application)	Software-object, software-value, program, method, data-type	core	Verknüpfung mit Domänenontologie	-	Classic DL	lightweight
COHSE	Verlinkung von Internetseiten (application)	Programming concept, Java concept, Class, Attribute, Method	domain	-	-	DAML+OIL	lightweight
SiSSy	Software-Qualitätsanalyse (application)	Class, Function, Variable	core	-	?	?	lightweight
FAMIX	Datenaustausch zwischen Werkzeugen (application)	Class, method, attribute	core/domain	-	Language plugins für verschiedene Programmiersprachen	CDIF	lightweight
IEEE 1471	Beschreibung von Softwarearchitekturen (reference)	Zwölf Basiskonzepte (u.a. System, Stakeholder, View, Concern)	core	-	IEEE	Keine	lightweight
Welty (Architectecture-Level)	Architekturdokumentation pflegen (application)	Component, Service, Interface, Connector	core	-	-	„Prolog-like language“	lightweight
COS	Management von Middleware/Application Servern (application)	Software, Liberaries, Services, Input, Output	core	DOLCE, Modularisiert in vier Teilontologien	-	KAON/OWL-DL	lightweight
Dhruv	Verkünpfung von Artefakten und Kommunikation einer Community (application)	Quellcode (angelehnt an Welty), Personen/Rollen, Bugs und Nachrichten/Diskussionen	domain	Modularisiert in vier Teilontologien (Community, Interactions, Code, Bugs)	-	OWL-DL	lightweight
Falbo	Integration von Software-Werkzeugen (application)	Software Process, Activity, Artifact, Input, Output	core	-	Werkzeuge zum Erstellen von Instanzen (z.B. Software-Prozesse)	Lingo	lightweight
Polarion	Software-Lifecycle Management (application)	?	?	?	?	?	?

Abbildung 1: Vergleich der betrachteten Ontologien

Während es im Bereich der Software-Strukturen einige Ontologien gibt, die von konkreten Programmiersprachen abstrahieren, und somit breit einsetzbar sind, gibt es kein allgemein verwendbares Modell für den Software-Lebenszyklus. Die vorhandenen Modelle sind entweder zu abstrakt (z.B. Falbo), decken nur ein spezifisches Anwendungsszenario ab (Dhruv) oder liegen nicht formal in einer Wissensrepräsentation vor.

8 Literatur

[aan1] Ankolekar, A., Sycara, K., Herbsleb, J., Kraut, R., and Welty, C. 2006. Supporting online problem-solving communities with the semantic web. In Proceedings of the 15th international Conference on World Wide Web (Edinburgh, Scotland, May 23 - 26, 2006). WWW '06. ACM Press, New York, NY, 575-584.

[aan2] Anupriya Ankolekar. Towards a Semantic Web of Community, Content and Interactions. Ph.D. Thesis September 2005 CMU-HCII-05-103

[cub1] Davor Cubranic, Gail C. Murphy, Janice Singer, Kellogg S. Booth: Hipikat: A Project Memory for Software Development. IEEE Trans. Software Eng. 31(6): 446-465

[dev1] Devanbu, R. J. Brachman, P. G. Selfridge and B. W. Ballard: LaSSIE? - A Knowledge-Based Software Information System, ACM Comm., 34(5), 1991, pp. 34-49.

[dob1] Daniel Oberle: Semantic Management of Middleware. Springer, New York. February 2006.

[dol1] WonderWeb? Deliverable D17. The WonderWeb? Library of Foundational Ontologies and the DOLCE ontology. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A., Schneider, L. Preliminary Report (ver. 2.0, 15-08-2002)

[fal1] Ricardo de Almeida Falbo, Daniel O. Arantes, Ana Candida Cruz Natali: Integrating Knowledge Management and Groupware in a Software Development Environment. PAKM 2004: 94-105

[hoo] Steffen Staab, Rudi Studer: Handbook on Ontologies Springer 2004

[ieee1] IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html

[maven] Maven-Webseite: http://maven.apache.org

[moc1] Audris Mockus, James D. Herbsleb: Expertise browser: a quantitative approach to identifying expertise. ICSE 2002: 503-512

[omg1] OMG: UML 2.0 Infrastructure Specification,. Version:M¨arz 2005. http://www.omg.org/cgi-bin/doc?formal/05-07-05.pdf. – Online–Ressource, Abruf: 2006-06-12

[opensym] OpenSymphony-Projekt: http://www.opensymphony.com/

[pol] Webseite von Polarion: http://www.polarion.org/

[qbench] Webseite des QBench-Projekts: http://www.qbench.de

[Quartz] Quartz-Projekt: http://www.opensymphony.com/quartz/

[sag1] Sager Tobias. Coogle - A Code Google Eclipse Plug-in for Detecting Similar Java Classes. Diploma Thesis. Department of Informatics, University of Zurich December 2005.

[sissy] SiSSy-Webseite: http://sissy.fzi.de

[tic1] Tichelaar, S., Steyaert, P., and Demeyer, S. (1999). FAMIX 2.0: The FAMOOS information exchange model.

[w3c] Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering. http://www.w3.org/2001/sw/BestPractices/SE/ODA/

[wel1] Welty, C.A., and Ferrucci D.A.: A Formal Ontology for Re-Use of Software Architecture Documents. ASE, 1999, pp. 259-262.

[wel2] Welty, C.A.: Software Engineering.In: Description Logic Handbook, 2003, pp. 373-387.

[wel3] Christopher A. Welty. An Integrated Representation for Software Development and Discovery. Ph.D. Thesis, Rensselaer Polytechnic Institute, 1995

[ye1] Yunwen Ye, Gerhard Fischer: Reuse-Conducive Development Environments. Autom. Softw. Eng. 12(2): 199-235 (2005)

