Stefan Leonhard Strell

Ontologie in Quality-Aware Software Engineering

MASTER THESIS

zur Erlangung des akademischen Grades

Diplom-Ingenieur

FAKULTÄT FÜR TECHNISCHE WISSENSCHAFTEN
Institut für Angewandte Informatik
Forschungsgruppe Application Engineering

Begutachter: *o.Univ.-Prof. Dr. Dr. h.c. Heinrich C. Mayr*

Vorbegutachter: *Dr. phil. Volodymyr Shekhovtsov*

April 2016

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich

- die eingereichte wissenschaftliche Arbeit selbständig verfasst und andere als die angegebenen Hilfsmittel nicht benutzt habe;
- die während des Arbeitsvorganges von dritter Seite erfahrene Unterstützung, einschließlich signifikanter Betreuungshinweise, vollständig offengelegt habe;
- die Inhalte, die ich aus Werken Dritter oder eigenen Werken wortwörtlich oder sinngemäß übernommen habe, in geeigneter Form gekennzeichnet und den Ursprung der Information durch möglichst exakte Quellenangaben (z.B. in Fußnoten) ersichtlich gemacht habe;
- die Arbeit bisher weder im Inland noch im Ausland einer Prüfungsbehörde vorgelegt habe und dass
- die zur Plagiatskontrolle eingereichte digitale Version der Arbeit mit der gedruckten Version übereinstimmt.

  Ich bin mir bewusst, dass eine tatsachenwidrige Erklärung rechtliche Folgen haben wird.

Klagenfurt, 19.04.2016

(Unterschrift) (Ort, Datum)

Anmerkungen

In dieser Arbeit werden Ergebnisse aus Quellen, wie z. B. Interviews verwendet, die einer Geheimhaltungsvereinbarung unterliegen. Diese wurden vor der Verwendung in dieser Masterarbeit anonymisiert. Aufgrund dieser Tatsache werden diese Ergebnisse ohne Angaben von Quellen oder in direkter oder indirekter Weise nachvollziehbaren Mitteln verwendet.

Vorwort

Immer wieder wird in den Medien von Semantic Web, Internet der Dinge (engl.: Internet of Things, IoT), Industrie 4.0 und ähnlichen technischen Schlagworten gesprochen. Was diese jedoch genau bedeuten, ist den durchschnittlichen LeserInnen meist unklar. In der heutigen Zeit, in der die Preise für Hardware stetig sinken und die Performanz dieser dadurch für die AnwenderInnen ins Unüberschaubare ansteigen, wird immer häufiger von Problemen mit Software berichtet. Zum einen sollen die Sicherheitslücken minimiert werden und die Qualität immer weiter ansteigen, zum anderen wird aber zumeist vergessen, welche Leistung teilweise hinter diesen Softwareprodukten steckt. Viele IT-ExpertInnen und WissenschaftlerInnen beschäftigen sich seit Jahren bzw. Jahrzehnten immer wieder mit der Frage, wie die Software Qualität gesteigert werden kann, ohne dass dabei die Preise ins Unermessliche steigen. Dazu wird nicht nur versucht, vorhandene Ansätze weiter zu verbessern, sondern es werden auch neue Möglichkeiten ausgelotet und durch wissenschaftliche Artikel, Bücher und andere Medien publiziert und in der Gesellschaft verankert.

Diese Masterarbeit wurde mit dem Ziel geschrieben, zum Einen Ontologien und Wissensverarbeitung, sowie die generellen Ziele in der qualitätsbewussten Softwareentwicklung und zum Anderen den Proof-of-Concept des Forschungsprojektes „QuASE“, bei dem ich Mitentwickler war, näher zu erläutern. Dabei handelt es sich nicht um ein Benutzerhandbuch für eine Softwaretechnik, sondern vielmehr um eine Erläuterung des wissenschaftlichen Hintergrunds. Als Zielgruppe sollten sich SoftwareentwicklerInnen und SystemadministratorInnen sehen, die die QuASE-Applikation weiterentwickeln oder zur Verwendung in ihrem Unternehmen oder ihrer Organisation vorbereiten.

Entscheidend für die Wahl dieses Themas war, dass die Mitarbeit am Forschungsprojekt an der Alpen-Adria-Universität Klagenfurt (AAU) mein Wahlfachprojekt darstellte. So konnte ich die Themen, denen ich mich dort und während meines Studiums zuwandte, bestmöglich in diese Arbeit einfließen lassen. Seit der Beendigung des Bachelorstudiums Informatik an der Hochschule für Angewandte

Wissenschaften Landshut (ehemals FH Landshut) konnte ich mich in den fast fünf Jahren des Masterstudiums auch beruflich weiterbilden und einiges an Erfahrungen hinsichtlich Software Qualität in der Praxis hier einfließen lassen.

Danksagung

Als Autor dieser Arbeit möchte ich mich bei allen, die mich in jeglicher Form unterstützt haben, bedanken. Ein besonderer Dank gilt hier vor allem den Angehörigen, die mich während meines gesamten Studiums vor allem mental unterstützten und auch einige Zeit auf meine Anwesenheit verzichten mussten. Hier seien vor allem mein Großvater Oswald Forman, meine Eltern Maria und Franz Strell und meine Geschwister Alexander, Martin und Wolfgang Strell (jeweils mit Familie) zu nennen.

Auch an meinen Freundeskreis sei ein herzlicher Dank für die Unterstützung und vor allem Kameradschaft gerichtet, die mir in so mancher Stunde mit Rat und Tat zur Seite standen. Ebenso den Studienkolleginnen und -kollegen gilt ein Dank für die gute Zusammenarbeit vor allem während der Vorbereitungen auf diverse Prüfungen und dem gemeinsamen Bestreiten des Studienalltags. Den Korrekturlesern, namentlich Walter Quendler, Anna Ackmann, Linda Stromberger und Matija Kucko, gilt auch ein Dank für die guten Hilfestellungen und Anregungen während dem Schreiben dieser Arbeit.

Ebenfalls bedanken möchte ich mich bei meinen ehemaligen Kolleginnen und Kollegen der Firma trinitec IT Solutions & Consulting GmbH, die mich während der ersten zwei Jahre des Masterstudiums begleitet, fachlich unterstützt und die mir einen Einblick in die Berufswelt schon während des Studiums gewährt haben. Ein weiteres Dankeschön möchte ich an die Kolleginnen und Kollegen der Forschungsgruppe Application Engineering richten, die mich während des Praxissemesters und des Schreibens dieser Arbeit vor allem fachlich beraten, aber auch mental den Rücken gestärkt haben und auch mit wertvollen Tipps und guten Gesprächen zur Seite standen. Insbesondere dem Projektteam, dessen Teil ich 10 Monate lang sein durfte, gilt hierfür ein besonderer Dank. Außerdem möchte ich mich auch bei den Kolleginnen und Kollegen an meiner aktuellen Wirkungsstätte, Jack Filter Produktions GmbH, für weitere Ideen und Anregungen, sowie der Möglichkeit zur Erweiterung der Fachkenntnisse bedanken.

Einen großen Dank möchte ich auch den Professorinnen und Professoren, sowie den Lehrbeauftragten der Informatik-Institute an der Alpen-Adria-Universität Klagenfurt für die exzellente Unterstützung und Ausbildung während der Studienzeit aussprechen. Insbesondere an die Betreuer der Arbeit, Prof. Dr. Dr. h.c Heinrich C.

Mayr und Dr. phil. Volodymyr Shekhovtsov, ohne die diese Arbeit wohl nicht in dieser Form zustande gekommen wäre, sei ein herzliches Dankeschön gerichtet.

Klagenfurt am 19. April 2016, Stefan Leonhard Strell, B. Sc.

Abstract

Note: This thesis is written in German. Only this part is translated to English.

This master thesis deals with the question, how to improve the quality awareness in software projects by exploiting appropriate ontologies and knowledge bases. To get a dedicated answer, you have to deal with the classical and primary solutions to improve quality and there must be a definition or working definition of the term software quality. Additionally, you should have a basic knowledge about the techniques of ontologies and its additional topics.

Furthermore this thesis should support a frame for the objectives of Quality-Aware Software Engineering in general and software toolsets and auxiliary functions in the special case. It is not only essential if these toolsets and functions gives competitive edges to individual development teams, but also as the previously existing knowledge and tools to support and facilitate the work of Software Engineers.

A “knowledge base”, “ontologies” and “the quality-aware Software Engineering” are perhaps to someone a so called “bohemian village”, so what do these keywords mean and where are they used? How can knowledge be stored in a sustainable way and how can it be expanded as needed without constantly developing new data structures? An answer to these and similar questions, as well as the quality of software and the research project QuASE related to this thesis will be addressed here.

However, the most important message of this thesis is not the great support and maximum revenue-improvement, but rather a qualitative assessment of what has so far not gone optimally, and where it can be attached to reduce or eliminate these weak spots. For this is also a stimulus for discussion and room for further improvement, which can and should go beyond the scientific work in this thesis.

Zusammenfassung

Diese Masterarbeit beschäftigt sich mit der Frage, wie die Qualitätsaspekte in Softwareentwicklungsprojekten mit Hilfe von Ontologien und Knowledge Bases besser berücksichtigt werden können. Um eine genaue Antwort auf diese Frage geben zu können, müssen einerseits die klassischen und ursprünglichen Möglichkeiten zur Qualitätsverbesserung und andererseits die Definition bzw. Arbeitsdefinition von Software Qualität gegeben sein. Zudem sollte ein Grundwissen über die Technologien der Ontologien und den damit verbundenen Themen vermittelt werden.

Weiters soll in dieser Arbeit die Zielsetzung der qualitätsbewussten Softwareentwicklung im Allgemeinen und der Software-Unterstützung durch eine Sammlung von Werkzeugen und Hilfsfunktionen im Speziellen ein Rahmen geboten werden. Hierbei ist nicht entscheidend, ob diese Vorteile für die einzelnen Entwicklerteams bringen, sondern vielmehr wie das bisher vorhandene Wissen und Werkzeuge die Arbeit unterstützen und erleichtern können.

Begriffe wie „Knowledge Base“, also eine Wissensdatenbank, „Ontologien“ oder „qualitätsbewusste Softwareentwicklung“ sind vielleicht für den einen oder anderen ein sog. „böhmisches Dorf“. Also was bedeuten diese Schlagworte und wie und wo werden sie eingesetzt? Wie kann man das Wissen nachhaltig speichern und bei Bedarf erweitern, ohne immer wieder neue Datenstrukturen zu entwickeln? Eine Antwort auf diese und ähnliche Fragen, sowie die Qualität von Software und das der Arbeit zugrunde liegende Forschungsprojekt QuASE sollen hier ebenfalls thematisiert werden.

Die wichtigste Botschaft dieser Masterarbeit ist jedoch nicht die Tool-Unterstützung und die größtmögliche Umsatz-Verbesserung, sondern vielmehr eine qualitative Einschätzung dessen, was bisher nicht optimal gelaufen ist und wo angesetzt werden kann, um diese Schwachstellen zu reduzieren oder auszumerzen. Hierzu soll in dieser Masterarbeit auch eine Anregung zur Diskussion und Raum für weitere Verbesserungen gegeben werden, die über diese wissenschaftliche Arbeit hinausgehen kann und soll.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Diese Masterarbeit stützt sich, wie der Titel erahnen lässt, im Wesentlichen auf Ontologien und der qualitätsbewussten Softwareentwicklung. Jeder der beiden Teile für sich behandelt ein breites Feld von bereits vorhandenen Wissen. Die qualitätsbewusste Softwareentwicklung korrespondiert dabei mit dem Themenbereich der Software Qualität und Ontologien werden in der Literatur zumeist in Verbindung mit Knowledge Bases und der Semantic Web Technologie genannt.

Qualität von Software und deren Verbesserung ist seit der Softwarekrise in den 1960er-Jahren ein Thema, das in Fachkreisen immer wieder neu aufgegriffen wird. Dabei ändern sich häufig aber nicht die Ziele, sondern die Wege zur Erreichung des Ziels eine hinsichtlich ihrer Qualität optimale Software zu entwickeln.

Seit den ersten eigenständigen Softwaresystemen und den ersten Programmiersprachen bis heute hat sich die IT-Welt zweifelsohne verändert. Neue, verbesserte Methoden, Entwicklungsumgebungen und neue Hardware-Standards erleichtern nicht nur die Arbeit in der Softwareentwicklung. Auch wenn sich die Ziele in ihrer Grundform nicht ändern, so werden sie dennoch komplexer und vor allem die Zusammenarbeit zwischen den verschiedensten Personengruppen sind die neuen Herausforderungen für die IT.

Grady Booch, einer der drei Erfinder der Unified Modeling Language (UML), lässt mit folgendem Zitat erahnen, wie schwer es sein kann, die Qualität von Software für alle Beteiligten zu definieren:

> *„Der Softwareentwicklungs-Amateur ist immer auf der Suche nach Wundern, sensationellen Methoden oder Werkzeugen, deren Anwendungen versprechen, Softwareentwicklung trivial zu machen. Der professionelle Softwareentwickler weiß, dass ein solches Patentrezept nicht existiert.*
>
> *– Grady Booch, in Object-Oriented Analysis and Design“* ([7], S. 25)

Auch andere ExpertInnen im Software Engineering sind dieser oder ähnlicher Meinung. So schreibt Heinrich C. Mayr, Informatik-Professor an der Universität Klagenfurt, in [8] von Problemen hinsichtlich der konkreten Definition von Software Qualität im Bereich des Software Engineerings. Auch wenn dieser Artikel bereits 1995 verfasst wurde, so sind Teile davon auch heute noch gültig, denn wie im späteren Verlauf der Arbeit gezeigt wird, ist eine eindeutige Formulierung von Software Qualität nicht möglich. Ziel der Forschungen im Bereich Software Engineering kann deshalb nur sein, sich einem optimalen Zustand anzunähern.

Ernest Wallmüller schreibt in [1], dass nach Boehm 1978 drei Fragen zur Problematik der Bestimmung von Qualität gestellt wurden:

1. *Problem der Definition von Software Qualität*
   Ist es überhaupt möglich, Definitionen der Eigenschaften und Merkmale eines Software-Produkts aufzustellen, die messbar sind und sich nicht überschneiden?
2. *Problem der Qualitätsprüfung*
   Wie gut kann man die Qualität eines Software-Produkts bzw. die Eigenschaften und Merkmale messen, welche die Qualität des Software-Produkts bestimmen?
3. *Problem der Qualitätslenkung*
   Wie kann man die Informationen über die Qualität des Produkts zur Verbesserung des Produkts im Lifecycle verwenden?
   ([1], S. 13)

Die erste Frage ließe sich mit einer einheitlichen Definition der Software Qualität und deren Qualitätsmerkmalen beantworten. Dazu sei allerdings gesagt, dass dies aktuell nicht bzw. noch nicht möglich ist. Man müsste hierzu die Definitionen für das konkrete Software-Produkt verwenden. Einen Überblick über dieses Thema bieten neben dem Abschnitt „Software Qualität“ in dieser Arbeit auch Werke von Wallmüller, Hoffmann und anderen namhaften AutorInnen.

Die Fragen zu Qualitätsprüfung und Qualitätslenkung werden im Kontext dieser Arbeit versucht zu beantworten. Mit dem Forschungsprojekt *Quality-Aware Software Engineering* (QuASE), also der qualitätsbewussten Softwareentwicklung, möchte das Institut für Angewandte Informatik an der Alpen-Adria-Universität Klagenfurt mit einem Proof-of-Concept zur Verbesserung der aktuellen Entwicklung beitragen. Dieses Projekt wurde bis Februar 2015 unter anderem von der Österreichischen Forschungsförderungsgesellschaft (FFG)[1] und einigen Kooperationspartnern in der Wirtschaft gefördert (vgl. [9, 10]). Ein nicht unerheblicher Teil dieser Arbeit befasst sich ebenfalls mit dem Thema Software Qualität und deren Auswirkungen auf Anforderungen an die resultierende Software und auf das Softwareprodukt selbst.

In der im Projekt QuASE entwickelten Software sind vor allem die Basiskonzepte der Knowledge Base und Ontologie ein Hauptbestandteil. Beide Begriffe wurden Ende des letzten Jahrhunderts vor allem durch die Erforschung der Technologie Semantic Web innerhalb der künstlichen Intelligenz geprägt. Hierzu gibt es einige, teilweise unterschiedliche Ansätze, mit denen Ontologien und Knowledge Bases auf-

[1] QuASE wurde durch das Bridge1-Programm der FFG gefördert. Nähere Infos hierzu siehe https://www.ffg.at/bridge1; Projekt-ID: 3215531

gebaut werden können. Eine zentrale Bedeutung haben hier standardisierte Beschreibungen und Beschreibungssprachen, die als Grundlagen dieser Ansätze dienen. Die Basiskonzepte von Knowledge Base und Ontologien, sowie deren Einsatz im Forschungsprojekt QuASE nehmen den größten Teil ein und sind dadurch zentraler Bestandteil der Arbeit.

Einen weiteren, für die aktuelle Entwicklung in der Qualitätsprüfung, nicht unerheblichen Anteil, stellen die Merkmale und die Messbarkeit von Qualität in Softwareprojekten dar. Diese müssen definiert sein und vorliegen, um eine Steuerung für zukünftige Projekte bereitstellen zu können. Die Qualitätslenkung basiert in der Regel auf sog. „best practices" aus der Vergangenheit und Evaluationen von abgeschlossenen und aktuellen Projekten, denn ohne Vorwissen gleicht dies einem Auto mit einer zeitverzögerten Lenkung, also erst Sekunden nachdem man gelenkt hat, wird sich das Auto in diese Richtung bewegen. Eine Automobilingenieurin würde dieses Auto als fehlerhaft beschreiben und es vom Markt nehmen. Bei einem Projekt würde man da wohl eher versuchen gegenzusteuern oder Kriterien, die als „weniger bedeutsam" gelten, zu revidieren, um verlorene Arbeitszeit wieder einzuholen. Doch was würde der Kunde[2] von zahlreichen Projekten halten, die entweder geringere Qualität als vorgegeben haben oder Mehrkosten verursachen? In den meisten Fällen wird der Kunde sich wohl um einen zuverlässigeren Partner umsehen. In der Softwareentwicklung ist dies aber aufgrund der zumeist kleineren Unternehmensstruktur kaum möglich. Deshalb sollte die Branche, auch im Interesse ihrer Kunden, verstärkt qualitätsbewusstes Entwickeln von Software leben.

Bevor jedoch auf die Klärung von Begriffen und in weiterer Folge auf das Thema der Arbeit eingegangen wird, werden in dieses Kapitel noch die grundlegende Literatur, die Motivation und Zielsetzung, sowie der Aufbau der Arbeit einfließen. Zur Zielsetzung gehört auch die Formulierung der Forschungsfrage, die einen zentralen Teil der Arbeit darstellen wird.

[2] Die Bedeutung von „der Kunde" ist in diesem Kontext nicht als Einzelperson gedacht, sondern als Unternehmen oder Organisation.

1.1 Basisliteratur und weiterführende Referenzen

Aufgrund der Vielzahl neuer Errungenschaften, vor allem im Bereich der Software Qualität, ist es kaum möglich den aktuellen Stand der Forschung konkret zu erfassen. Auch im Spektrum der Ontologien und Knowledge Bases gibt es einen stetigen Wandel durch Forschungsleistungen. Diese Arbeit stützt sich deshalb zum Teil auf, im Bezug auf das relativ junge Forschungsumfeld der Informatik, etwas ältere Literatur, also in einem Zeitfenster ab Mitte des 20. bis in die ersten Jahre des 21. Jahrhunderts, die allerdings noch immer ihre Gültigkeit besitzt. Auch da der Fokus in dieser Arbeit im Wesentlichen auf Basiskonzepten und Begriffsdefinitionen liegt, ist die Relevanz von neueren Konzepten z. B. der Ontologien, kaum gegeben. Ein weiterer Grund liegt darin, dass es vor allem in der Informatik und Technik einige Zeit dauert, bis neuere Artikel und Bücher von der Allgemeinheit akzeptiert werden. Dies folgt auch aus Sommervilles Werk *Software Engineering* [61]. Dort heißt es, die Informatik sei eine relativ junge Ingenieursdisziplin (vgl. [61], S. 14).

Einen großen literarischen Einfluss auf das Schreiben dieser Arbeit hatten die Werke *Handbook on Ontologies* von Staab und Studer [11], *Model Driven Engineering and Ontology Development* von Gašević, Djurić und Devedžić [12], sowie Giancarlo Guizzardis Dissertation *Ontological foundations for structural conceptual models* [6], die auf die Erklärung und Definition von Knowledge Base und Ontologie abzielen. Für die Grundlagen der Software Qualität und Meta-Modellierung wurden u. a. Wallmüllers Buch „*Software Qualitätsmanagement in der Praxis*“ [1] und die GI-Lecture Notes „*Meta-Modelling and Ontologies*“ von Brockmans, Jung und Sure [13] als Basis verwendet. Weiterführende Erläuterung zu *Quality-Aware Software Engineering* bieten die von Shekhovtsov und Mayr zu diesem Thema veröffentlichten Artikel (u. a. [10, 14]).

1.2 Motivation und Zielsetzung

Hintergrund zum Verfassen dieser Arbeit ist, wie eingangs erwähnt, das Forschungsprojekt *Quality-Aware Software Engineering*, das vom Institut für Angewandte Informatik der AAU durchgeführt wurde. Es bildet die Basis des zentralen Themas „Ontologien in der qualitätsbewussten Softwareentwicklung“. Ziel dieser Arbeit ist es, eine Antwort auf die folgende Forschungsfrage zu finden:

> *„Wie können Ontologien und Wissensdatenbanken die Arbeit der Akteure im Softwareentwicklungsprozess erleichtern und helfen Probleme und Missverständnisse zu reduzieren?“*

Bei der Beantwortung soll allerdings nicht nur eine Menge an Werkzeugen, ein sog. „Tool-Set“, vorgestellt werden, sondern vielmehr eine Auseinandersetzung mit den verschiedenen Bereichen und Möglichkeiten, eine Reduktion von Problemen und dadurch eine Verbesserung der Qualität zu erreichen. Eine weitere Zielsetzung ist darüber hinaus einen Überblick über die aus dem Forschungsprojekt resultierende Software und deren Einsatzmöglichkeiten zu bieten.

Zu einigen der Einsatzmöglichkeiten gibt es bereits fundiertes Wissen und Forschungsthemen, die in dieser Masterarbeit thematisiert werden. Aber auch im Hinblick auf neue, noch nicht gänzlich erforschte und erarbeitete Gebiete, wie dem Internet der Dinge und Industrie 4.0 werden Synergien und Möglichkeiten zur weiteren Verbesserung von Software- und IT-Systemen angeschnitten.

Ein Teil der Arbeit beschäftigt sich mit den Kernkonzepten der sogenannten Knowledge Base, ein auf Wissen, Wissenstransfer und Ontologien basierendes Datensystem. Hierzu gehören auch Grundlagen der Modellierung und Metamodellierung, welche die Strukturen dieses Wissens auf abstrakter Ebene darstellt. Auch sind die Grundlagen und Ausprägungen zu Software Qualität ein zentrales Thema um die Forschungsfrage beantworten zu können.

Der Hauptteil der Mitarbeit des Autors im Forschungsprojekt war zum einen die Erstellung und Weiterentwicklung der sog. Site-Modelle und zum anderen die Entwicklung und Anpassung der Schnittstellen dieser Modelle zu Knowledge Base und Ontologien. Zusätzlich wurden die Aufgaben Entwicklung und Test der Applikation unter allen Mitarbeitern aufgeteilt.

Da die Bereiche der Ontologien und Knowledge Bases ein fundiertes Fachwissen benötigen, wäre es in dieser Zeit kaum möglich gewesen so weit in die Tiefe der Materie einzusteigen, um ein fundiertes Fachwissen zu erzielen. Die Entwicklung der Applikation selbst wurde von insgesamt vier studentischen Entwicklern und einem hauptberuflichen Forscher entwickelt, wodurch ebenso nicht alle Bereiche von allen

gleich gut bekannt sein können. Durch die zum Teil automatisierten Tests und der Einbindung von verschiedenen Programmbibliotheken wird ein Gesamtüberblick eines einzelnen Entwicklers über das komplette System zusätzlich erschwert.

Die im vorherigen Abschnitt benannte Basisliteratur stützt sich, im Hinblick auf QuASE, jeweils nur auf ein Teilgebiet der Arbeit. Deshalb kann als weiteres Ziel dieser Arbeit auch die Verbindung dieser Gebiete genannt werden. Hierbei wird im Wesentlichen aber auf die Anwendung der derzeit bekannten Techniken und Technologien zur Verbesserung der Software Qualität eingegangen, die im weiteren Verlauf nochmals kurz reflektiert werden.

In der täglichen Arbeit eines Softwareentwicklers gibt es häufig Situationen, in denen die Qualität einer Anwendung kritisiert wird oder es Klärungsbedarf hinsichtlich Anforderungen und Zielsetzungen gibt. Vor allem dann, wenn die Entwicklerteams sehr klein sind, bzw. nur ein einzelner Entwickler sich um alle Bereiche des Projekts gleichermaßen kümmern muss ist der Fokus auf qualitativ hochwertige Software oft nicht in dem angedachten Maße gegeben. Natürlich dürfen die BenutzerInnen erwarten, dass die eingesetzte Software so funktioniert, wie sie spezifiziert wurde, allerdings sollten hierfür die Ziele und Anforderungen in besonderem Maße dem Machbaren angepasst werden. Auch diese teils persönliche Einschätzung als Experte motivierte den Autor zum Verfassen dieser Masterarbeit.

1.3 Aufbau der Arbeit

Die Arbeit ist unterteilt in Grundlagen und einen projektspezifischen Teil, der am Ende mit der Beantwortung der Forschungsfrage abschließt. Die Grundlagen beschäftigen sich mit der Definition und Erläuterung der Fachbegriffe und einigen technischen Details, die in weiterer Folge zum Verständnis dienen sollen. Im projektspezifischen Teil werden die thematische Einordnung der Forschungsfrage, das Forschungsprojekt und die daraus resultierende Software im Detail vorgestellt.

Zunächst wird im folgenden Kapitel eine Hinführung zum Thema mit einleitenden Definitionen der wichtigsten Begriffe und Schlagworte und den technischen Details gegeben. Anschließend wird auf die im Forschungsprojekt verwendeten Softwaresysteme und Frameworks kurz eingegangen und diese werden zum Verständnis kurz erklärt.

Die weiteren Kapitel beschäftigen sich mit der Forschungsfrage, dem Forschungsprojekt und seiner konkreten Umsetzung. Dabei wird zuerst über die qualitätsbewusste Softwareentwicklung im Allgemeinen, sowie über das Projekt selbst und anschließend über die Kern-Bestandteile Ontologie und Knowledge Base ein Überblick gegeben. In diesem Teil wird auch auf mögliche Erweiterungen und Synergien mit anderen, ähnlichen Konzepten eingegangen.

In weiterer Folge wird die Beantwortung der Forschungsfrage thematisiert. Hierzu werden Literatur und Forschungsbeiträge, sowie eine Reflexion des Autors über den aktuellen Stand der Software Qualität und deren Verbesserungsmöglichkeiten herangezogen.

Den Abschluss bilden die Zusammenfassung mit einer Reflexion über die Softwareentwicklung anhand des zuvor erarbeiteten Ergebnisses, sowie ein Ausblick auf mögliche anschließende Themengebiete. Im Fazit wird nochmals die persönliche Haltung des Autors zum Thema Software Qualität und Entscheidungshilfen durch Knowledge Bases wiedergegeben.

2 Begriffsdefinitionen & Grundlagen

In diesem Kapitel werden einige grundlegende Begriffe definiert und technische Grundlagen erklärt, welche einen im weiteren Verlauf der Arbeit benutzt werden und die für das Verständnis der weiteren Arbeit unerlässlich sind. Zunächst werden die beiden zentralen Begriffe Ontologie und Knowledge Base, sowie deren Zusammenhänge und die technische Ausführung erklärt. Anschließend folgen die Begriffe Software Qualität und Metamodell. Für den Begriff Software Qualität wurde in diesem Kapitel auch eine Arbeitsdefinition eingeführt, die in weiterer Folge als Grundlage für das Verständnis dienen soll. Weitere allgemeine Begriffe befinden sich im Glossar im Anhang.

2.1 Ontologie

Der Begriff *Ontologie* ist – im wissenschaftlichen Kontext gesehen – nicht nur im Bereich der Informationstechnologie zu finden, sondern spielt u.a. auch in der Philosophie eine zentrale Rolle. Denn lange bevor es die Wissenschaften der Informatik gab wurde der Begriff Ontologie und seine Bedeutung definiert.

2.1.1 Ontologie in der Philosophie

Der Ursprung des Begriffs Ontologie geht auf die Wissenschaft der Philosophie zurück. In diesem Kontext gibt es nach dem Internet-Lexikon von webster folgende, hier frei übersetzte, Definitionen:

- ein Zweig der Metaphysik, der sich mit der Natur und dem Zusammenhang des Seins beschäftigt
- eine spezielle Theorie über die Natur des Seins oder die Art der Dinge, die eine Existenz haben (vgl. [6, 15])

Giancarlo Guizzardi, ein Experte in Enterprise Modelling und Ontologien in der Informatik, schreibt in seiner Dissertation, der Begriff Ontologie habe sich im 17. Jahrhundert durch Rudolf Göckel und Jacob Lorhard in ihren Büchern, *Lexicon philosophicum* bzw. *Ogdoas Scholastica* (siehe auch Abbildung 2-1), parallel gebildet. Das lateinische Ursprungswort *Ontologia* bedeutet in wörtlicher Übersetzung die Studie der Existenz (vgl. [6], S. 52). In philosophischen Kreisen wurde der Begriff der Ontologie allerdings erst im 18. Jahrhundert durch Christian Wolffs Werk *Philosophia prima sive Ontologia* im Jahr 1730 populär (vgl. [6], S. 52). In

Guizzardis Werk ([6], S. 51 ff.) wird die Erläuterung dieses Begriffs zudem ausführlicher behandelt.

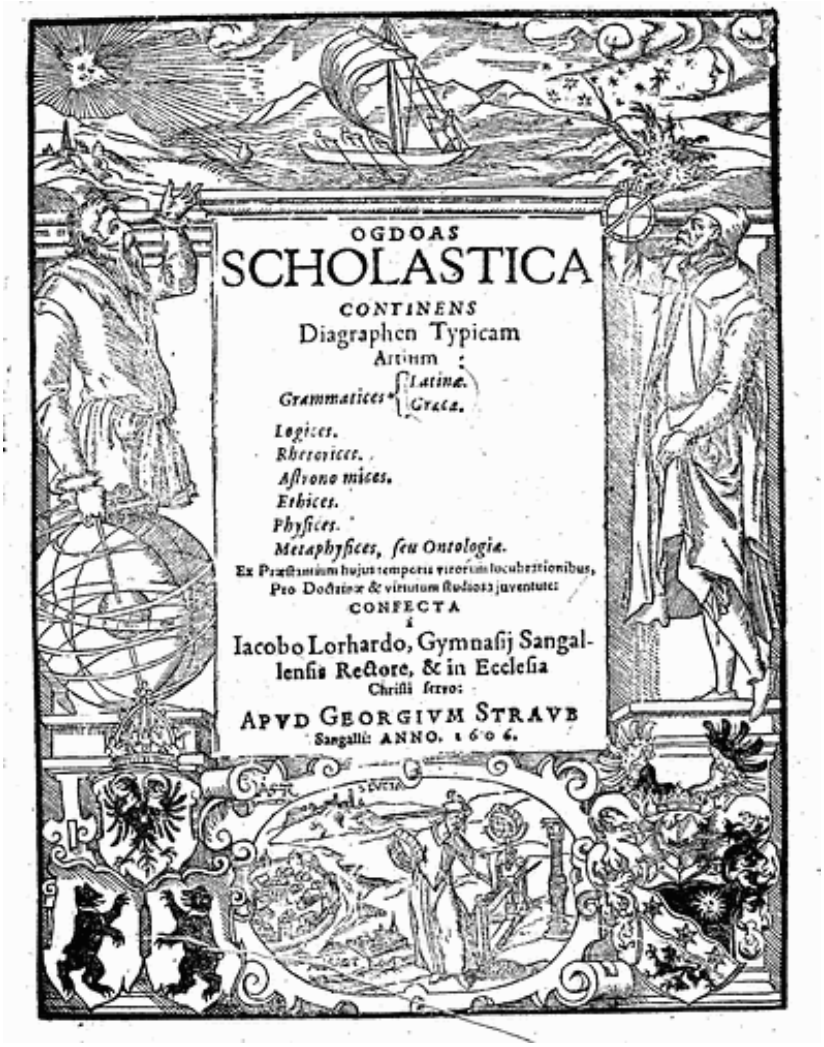

Abbildung 2-1: Cover von „Ogdoas Scholastica“ von 1606 (aus [6], Figure 3-1)

Kant, einer der führenden Philosophen, beschreibt in seinem Werk *Kritik der reinen Vernunft* die Metaphysik als *„eine bisher bloß versuchte, dennoch aber durch die Natur der menschlichen Vernunft unentbehrliche Wissenschaft“* ([16], S. 71). In einer weiteren Ausgabe dieses Werks heißt es, die Ontologie sei der Versuch *„Dingen überhaupt synthetische Erkenntnisse a priori in einer systematischen Doktrin zu geben“* ([17], S. 357). Diese Aussagen von Kant zeigen, dass die Begriffe der Metaphysik und Ontologie in der philosophischen Wissenschaft zwar verankert, aber nicht unumstritten sind.

2.1.2 Ontologie in der Technik

Seit einigen Jahrzehnten wird dieser Begriff auch in der Informationstechnologie und Informatik verwendet. Nach Guizzardi in [6] war Mealy einer der Ersten, der den Begriff Ontologie 1967 (vgl. [18]) in der Informatik benutzte. Allerdings dauerte es noch einige Jahre, bis Mitte der 1990er-Jahre das Interesse an diesem Thema wuchs. Guizzardi schreibt, dass in den ersten Jahren des 21. Jahrhunderts eine „Explosion" eintrat. In dieser Zeit, von dem Semantic Web Working Symposium (SWWS) 2001 bis zur dritten Auflage der International Semantic Web Conference (ISWC) 2003, stieg die Anzahl der eingereichten wissenschaftlichen Artikel um 400% (vgl. [6], S. 53).

Nach Staab und Studer war es am Anfang eine ziemlich kleine Forschungs-Community, die *Knowledge Acquisition Community*, die in diesem Bereich Forschungen anstrebte (vgl. [11]). Seit Beginn des 21. Jahrhunderts wurden in Verbindung mit der Technologie *Semantic Web* immer mehr Anstrengungen unternommen, um Ontologien in der Informatik zu verankern (vgl. [6], S. 53 ff.).

Die zum Großteil verschiedenen technischen Disziplinen des *Software Engineering* und *Artificial Intelligence* (AI) wurden laut Atkinson, Gutheil und Kiko durch die Errungenschaften um Semantic Web und Ontologien angenähert. Dies führte zu einem Austausch von „high level"-Ideen und -Methoden. Zwei der treibenden Kräfte waren der Trend zu eingebetteten intelligenten Systemen in nahezu allen Teilen unserer Umgebung und die steigenden Abhängigkeiten zwischen Internet und Gesellschaft (vgl. [19]).

Nach Gruber ist eine Ontologie eine explizite Spezifikation einer Konzeptualisierung (vgl. [20]). In diesem Kontext wird Konzeptualisierung als ein abstraktes Modell einiger Aspekte der realen Welt verstanden. (vgl. [21]).

Nach Gašević, Djurić und Devedžić in [12] definieren Chandrasekaran, Josephson und Benjamins „Ontologie" meistens im Kontext mit einer der beiden folgenden Beschreibungen:

- ein Repräsentationsvokabular, das oft zu Domänen oder einer Betrachtungseinheit spezialisiert wird
- ein Body of Knowledge, der spezielle Domänen mit Repräsentationssprachen beschreibt.

In beiden Fällen existiert immer eine assoziierte zugrundeliegende Datenstruktur, die die Ontologie repräsentiert (vgl. [12, 22]).

Nach Chandrasekaran et al. gibt es in der AI zwei breite Kategorien: *mechanism theories* (Theorien über Mechanismen) und *content theories* (Theorien über Inhalte), wobei die Ontologie der zweiten Kategorie zugeordnet wird (vgl. [22]).

Fuchs beschreibt in ihrer Dissertation [23] einen konkreten Anwendungsfall einer Ontologie. Dort kommt eine Ontologie für ein Fahrassistenzsystem zum Einsatz, das für die Repräsentation von Verkehrsobjekten (Schilder, Verkehrsteilnehmer, etc.) verwendet wird (vgl. [23], S. 68). Diese Ontologie wurde in weiterer Folge mit Hilfe von logischen Programm-Konstrukten zu einer Knowledge Base erweitert (vgl. [23]).

Ontologien werden mit verschiedenen Werkzeugen, Sprachen und Schemas modelliert. Einen Überblick darüber bietet der folgende Abschnitt. Die konkrete Umsetzung einer Ontologie im Forschungsprojekt QuASE wird in einem späteren Kapitel noch näher erläutert.

2.2 Beschreibungssprachen, Frameworks & Schemas für Ontologien

Zur Umsetzung von Knowledge Bases und Ontologien in einem Softwareprojekt werden zum einen die Ontology Web Language (OWL) und die Query Language SPARQL verwendet. Diese und weitere Sprachen, sowie Frameworks und Schemas werden in den weiteren Abschnitten beschrieben.

In diesem Abschnitt werden einige grundlegende Techniken zur Erstellung und Bearbeitung von Ontologien beschrieben. Zunächst wird auf zwei der Grundlagen aller Ontologien und Knowledge Bases, der Description Logic und dem Resource Description Framework und dessen Schema eingegangen. Abschließend wird in diesem Abschnitt auf die beiden für das QuASE-Projekt und dadurch auch für diese Masterarbeit wichtigen Grundlagen SPARQL und OWL eingegangen.

2.2.1 Description Logic

Die Description Logic (DL) ist eine Familie von Knowledge Representation Languages der auf Logik basierenden Repräsentationssprachen, die als Grundlage für Ontologien und elektronisch verarbeitetes Wissen dienen (vgl. [21]). Nach Gašević, et al. bauen Knowledge Bases, die mit einer solchen DL entwickelt wurden, auf einer *Terminologie*, der *TBox*, und einer *Assertion*, der *ABox*, auf. Die ABox nutzt hierbei das Vokabular der TBox. Weiters beinhaltet das Vokabular der DL Konzepte und Rollen, wobei die Konzepte eine Menge an *Individuals* bezeichnen (vgl. [12]). Nach Baader, Horrocks und Sattler ist der Name *Description Logics* durch das Konzept *description* zum einen und durch die *logic*-basierende Semantik entstanden. Eine dieser DLs ist die ausdrucksstarke DL SHIQ, auf deren Grundlage einige Web-Ontologie-Sprachen wie OIL, DAML+OIL und OWL entwickelt wurden. Diese DLs werden sowohl zum anfänglichen Designen, als auch zur Verbesserung einer Ontologie verwendet (vgl. [21]).

Anfänglich, 1980 bis 1990, wurden Systeme auf DLs wie KLONE und K-REP implementiert, später, 1990 bis 1995, wurden neue algorithmische Paradigmen in den DLs, die sog. *tableau-based algorithms*, ergänzt. Vertreter dieser Epoche sind KRIS und CRACK. Bis 2000 wurden darauf aufbauend hoch optimierte Systeme wie FaCT, RACE und DLP entwickelt. Diese wurden aus den ausdrucksstarken DLs wie SHIQ weiterentwickelt (vgl. [21]).

2.2.2 Resource Description Framework & -Schema

Das Resource Description Framework (RDF) ist ein Standard der W3C, der ein Framework zur Repräsentation von Informationen im Web definiert (vgl. [24, 25]). Im Wesentlichen wird das RDF für eine graphische Notation von Inhalten des Semantic Web verwendet und gilt auch als eine Beschreibung von Metamodellen und Ontologien (vgl. [24]). Nach Klyne und Carroll ist RDF ähnlich strukturiert wie der allgemein bekannte XML-Standard.

Hierzu gibt es nach McBride eine Beschreibungssprache, das Resource Description Framework Schema (RDFS). Beide wurden im Zuge des Semantic Web definiert und sind Web-Sprachen (vgl. [25]).

2.2.3 Ontology Web Language

Die Ontology Web Language (OWL) ist eine der Sprachen, die bei Ontologien und Wissensdatenbanken verwendet werden. Dies bezeichnet man in der Fachwelt auch als eine sog. Domain Specific Language (DSL), also eine Sprache, die sich auf eine bestimmte Kategorie bezieht. Nach Antoniou und van Harmelen ist die OWL ebenfalls vom W3C definiert worden, um das in der Ausdruckskraft limitierte RDF Schema zu erweitern (vgl. [26]).

Für eine Ontologie benötigt man folgende Hauptressourcen:

- eine wohldefinierte Syntax
- eine wohldefinierte Semantik
- effiziente Unterstützung in der Beweisführung
- ausreichende Ausdruckskraft
- angemessene Ausdrücke

  (vgl. [26])

Obwohl die XML-basierte RDF-Syntax nicht sehr benutzerfreundlich ist, wurden nach Antoniou et al. für DAML+OIL und OWL die Technologien aus RDF und RDFS als Grundlage verwendet. Durch die verschiedenen Ontologie-Entwicklungs-tools für die beiden Ontologie-Sprachen fällt dieser Nachteil nicht so sehr ins Gewicht. (vgl. [26]).

Grundsätzlich gibt es drei verschiedene Ausprägungen der OWL:

OWL Full: die komplette Sprache

OWL DL (Description Logic):
mit Beschränkungen der Nutzung von Konstruktoren in OWL und RDF

OWL Lite: limitierter Satz an Sprach-Konstruktoren

(vgl. [26])

Eine mögliche Syntax für Ontologien ist die sog. Manchester Syntax, die vom W3C zur Beschreibung definiert wurde. Sie verwendet für OWL 2 die Standard Notation der Backus-Naur-Form (BNF) (vgl. [27]). Die BNF ist eine formale Sprache zur Darstellung kontextfreier Grammatiken.

Für die Knowledge Base und Ontologie im Projekt QuASE wird aufgrund seiner Kompatibilität mit dem Werkzeug Protégé 4 diese Syntax mit der Ausprägung OWL DL verwendet.

2.2.4 SPARQL

Die Abfragesprache *SPARQL Protocol and RDF Query Language* (SPARQL) wird vom World Wide Web Consortium (W3C) folgendermaßen beschrieben: „SPARQL 1.1 ist eine Reihe von Spezifikationen, die Sprachen und Protokolle um RDF-Graphen im Web oder in einer RDF-Quelle abzufragen und zu manipulieren“ (vgl. [28]). Das W3C ist eine internationale Gemeinschaft, die offene Standards entwickelt, um das langfristige Wachstum des Web zu sichern (vgl. [29]).

Der Aufbau einer SPARQL-Query ähnelt dem der SQL-Query, wie im Beispiel der Abbildung 2-2 auf der nächsten Seite veranschaulicht wird. Zuerst sollten bei einer SPARQL-Query die verwendeten Prefixes deklariert werden, um diese dann in der eigentlichen Query verwenden zu können. Diese sind ähnlich zu den *include*- oder *import*-Statements der gängigen Programmiersprachen, binden diese allerdings zu einem Unified Ressource Identifier (URI), in der die zugehörigen Befehle definiert sind.

Im Unterschied zu einer SQL-Query gibt es bei SPARQL keine *FROM*-Bedingung. Die Ausgabe einer SPARQL-Query ist eine Tabelle, die alle in der *SELECT*-Zeile enthaltenen Variablen, die mit einem ”?“ am Beginn gekennzeichnet sind, wiedergibt.

Der grün umrandete Bereich in Abbildung 2-2 zeigt, wie ein *PREFIX* definiert wird. Im blau gekennzeichneten Bereich ist der SELECT-Befehl der SPARQL-Query, das die Ausgabe im orangenen Bereich ausführt. Hierbei kennzeichnet die erste Zeile die auszugebenden Variablen. In der Regel werden hierfür sog. Internationalized Resource Identifier (IRI) verwendet, die in der Manchester Syntax als Nachfolgemodell der URIs bezeichnet werden (vgl. [27]). Weitere Einzelheiten zum Aufbau der Query-Sprache SPARQL sind auch unter [27] nachzulesen.

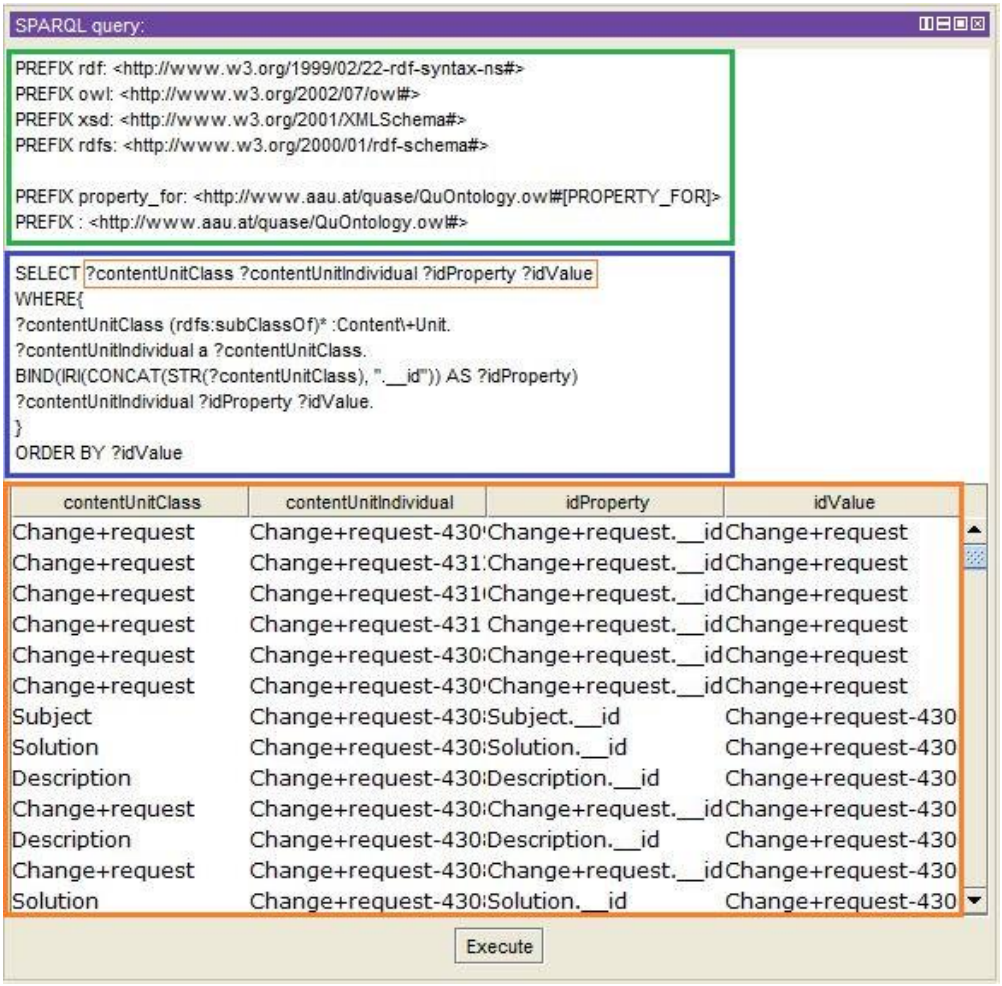

Abbildung 2-2: Beispiel einer SPARQL-Query in Protégé mit Ausgabe

Da die meisten SPARQL-Editoren jedoch nicht mit „Syntax-Highlighting“, also farblich markierte reservierte Wörter, ausgestattet sind, haben die Entwickler des Forschungsprojekts QuASE für interne Zwecke einen solchen in die Applikation integriert. Dies dient der Vermeidung von Syntax-Fehlern und zur besseren Lesbarkeit während der Implementierung. Abbildung 2-3 zeigt einen Ausschnitt aus diesem sog. „Internal Endpoint“.

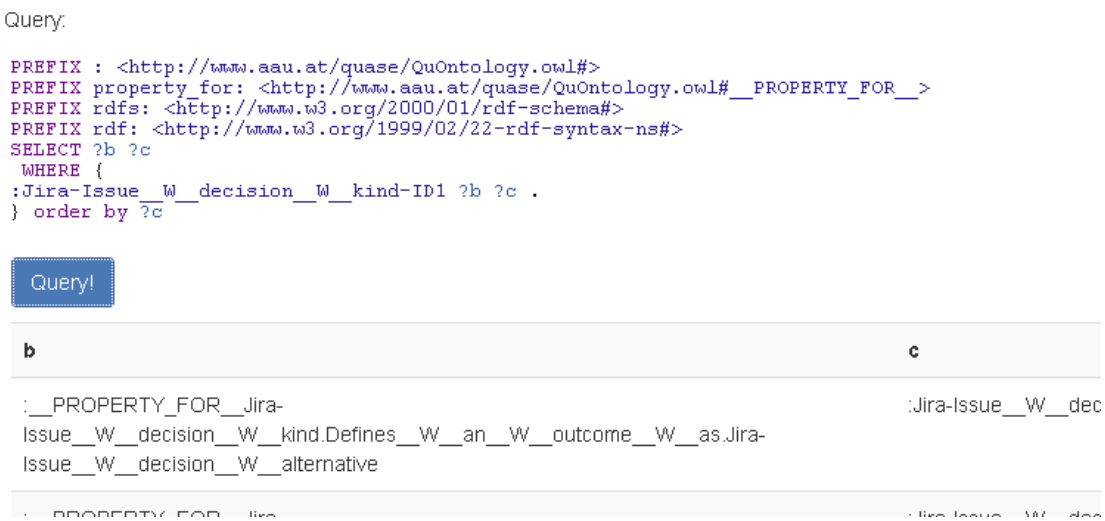

Abbildung 2-3: SPARQL-Editor in QuASE

2.2.5 Weitere Arten der Ontologie-Beschreibung

Weitere Sprachen und Modellierungsarten von Ontologien sind die bereits genannten RDF und RDFS, sowie F-Logic und DAML+OIL[3]. Da diese Ansätze nicht Gegenstand des QuASE-Projekts sind und für diese Arbeit keine besondere Relevanz haben, werden diese nicht näher erläutert. Weiterführende Informationen über F-Logic sind in Kapitel 2 in Handbook on Ontologies [30] und dessen Referenzen nachzulesen. DAML+OIL wird in diesem Buch ebenfalls ausführlicher behandelt, ebenso wie in dem, in der Fußnote angegebenen Link.

2.3 Knowledge Base

Eine Knowledge Base (KB) ist eine Datenstruktur, die auf Wissen und Wissenstransfer aufgebaut ist. In der Regel werden die Daten der Knowledge Base in Datenbanken und/oder Ontologien gespeichert. Um Wissen aus diesen Daten zu extrahieren gibt es mehrere Ansätze, wie z.B. Data-Mining. Nach Felfernig in [31] ist die wachsende Komplexität bei der Entwicklung von KBs ähnlich schwierig zu handhaben wie in regulären Softwaresystemen (vgl. [31], S. 28).

Zum Ende des 20. Jahrhunderts, als das Forschungsgebiet der Knowledge Bases noch weitgehend unbekannt war, wurden die Grundsteine für heutige Erkenntnisse gelegt. Slaughter schreibt in seinem Artikel von 1996 über Knowledge Bases *„called New Thinking for a New Millennium“* [32]. Weiter heißt es im selben Text über KBs *“in 1996 is not what it was in 1986, nor what it will be in 2006 or 2056”* [32]. Guizzardi

[3] DAML+OIL: http://www.daml.org/2001/03/daml+oil-index.html

schreibt in [6], dass erst mit den ontologischen Strukturen in den ersten Jahren des 21. Jahrhunderts die Knowledge Bases verbessert wurden. Zuvor wurden KBs funktional aufgebaut und Sprachen wie XML oder SGML verwendet (vgl. [6, 33]). Nach Guizzardi wurden diese durch Domain-Experten typischerweise in der Lösung für ein gezieltes Problem eingebaut (vgl. [6]).

Mylopoulos schreibt in [34], dass die sog. „Knowledge Representation“, also die Repräsentation von Wissen, und seine Anwendungen mit Datenbanken an der Universität von Toronto seit 1975 erforscht wird. Knowledge Base Management Systeme (KBMS) sollen demnach als repräsentative Frameworks und rechnergestützte Umgebungen zur Entwicklung von Software mit KBs beitragen. Analog zu Datenbank-Managementsystemen (DBMS) werden KBMS zur Bereitstellung von großen und verteilten KBs verwendet (vgl. [34]). Die generellen Unterschiede zwischen KBMS und DBMS sind nach Mylopoulos die Verwendung der Schemas. „Basis-Wissen“ und „generisches Wissen“ werden in KBMS während der Designphase aufgebaut, während dies in Datenbanken eher den späteren BenutzerInnen zugewiesen wird (vgl. [34], S. 4).

Brodie und Mylopoulos schreiben in [35], der Unterschied zwischen Datenbanken und Knowledge Bases besteht darin, dass KBs große Mengen von abstraktem Wissen beinhalten, während DBs meistens für konkretes Wissen designet werden. Außerdem schreiben sie eine DB ist eine Sammlung von Daten, die Fakten repräsentieren. Eine KB beinhaltet Informationen auf einer höheren Abstraktionsebene (vgl. [35]).

Nach Gašević et al. ist die Repräsentation von menschlichem Wissen in der Informatik der AI zugeordnet. Ziel der AI ist die Entwicklung von Computerprogrammen, die Menschen als „intelligent“ bezeichnen (vgl. [12], S. 3). Es gibt weiters auch verschiedene Arten der Repräsentation von Wissen. Diese sind:

- Object-Attribute-Value (O-A-V) Triplets (Objekt-Attribut-Werte Tripel)
- Uncertain Facts (unsichere Fakten)
- Fuzzy Facts (unscharfe Fakten)
- Rules (Regeln)
- Semantic Networks (semantische Netzwerke)
- Frames (Rahmen)

(vgl. [12], S. 15-18).

Die Knowledge Base ist die zentrale Komponente eines sog. *knowledge-based intelligent system*, also eines wissensbasierten intelligenten Systems. Die Knowledge Base enthält eine Menge von Sätzen, die Einheit der Repräsentation von Wissen. (vgl. [12], S. 19).

Zur Repräsentation von Wissen gibt es spezielle Sprachen, die sowohl syntaktisch als auch semantisch eine Repräsentation von Entitäten, Events, Aktionen, Prozessen und Zeit wiedergeben sollen.

Sie werden in die folgenden fünf Kategorien unterteilt:

- Logik basierende Repräsentationssprachen
  (z. B. First-Order Logic, Description Logic)
- Frame basierende Repräsentationssprachen
- Regel basierende Repräsentationssprachen
- Visuelle Sprachen für Knowledge Repräsentation
- Natürliche Sprachen und Knowledge Repräsentation

(vgl. [12], S. 20-35)

Die auf Logik basierenden Sprachen gelten historisch und pragmatisch als einer der wichtigen Zweige der AI (vgl. [12], S. 20). Die beiden bekanntesten Vertreter hierfür dürften die First-Order Logic und DLs sein (vgl. [12], S. 22/25).

Wissen hat nach Gašević et al. drei verschiedene Level:

- Implementation Level
  Level der Knowledge Base eines operationalen intelligenten Systems.
- Logical Level
  Alle Datenstrukturen in der Knowledge Base repräsentieren etwas und man benötigt einen Interpreter zur Repräsentation des Wissens.
- Knowledge Level
  Eindeutiges Level eines Computer Systems über dem logischen Level, charakterisiert durch Wissen als Medium.
  (vgl. [12], S. 41-42)

Gašević et al. definieren auch die Ontologie als einen extrem wichtigen Teil von Knowledge über eine Domain und als die Perspektiven von „computing and knowledge representation“ (vgl. [12], S. 45).

Felfernig verwendet in seiner Dissertation Modelle und Metamodelle um die Problem-Domain zu beschreiben und eine Knowledge Base aufzubauen.

Diese Modelle sind:

- Das **Organization Model** dient zur Modellierung von Business Prozessen, strukturellen Einheiten und korrespondierenden Ressourcen.
- Das **Task Model** beinhaltet eine hierarchische Beschreibung von Aufgaben, die Teil des Business Prozesses im Organization Model sind.
- Das **Agent Model** beschreibt die Einsatzmöglichkeiten von Agenten (z. B. Personen, Software-Einheiten, Computer Hardware), d. h. welche Agenten können welche Tasks übernehmen.
- Das Schlüsselelement im **Communication Model** ist die *transaction*, die die Kooperation zwischen Agenten beschreibt.
- Das Schlüsselmodell von CommonKADS[4] ist das **Expertise Model**, das in drei Ebenen des benötigten Wissens zur Erledigung einer bestimmten Aufgabe (Domain-Ebene, Inferenz-Ebene, Task-Ebene) aufgeteilt ist.
- Das CommonKADS **Design Model** beinhaltet hauptsächlich eine Spezifikation der Systemarchitektur und Details bezüglich der Implementierung des Zielsystems.
  (vgl. [31], S. 28-29)

Nach Felfernig wurden einige ähnliche Ansätze für das klassische Software Engineering entwickelt, um die Implementierung auf Softwaresystemen effektiver gestalten zu können (vgl. [31], S. 29).

[4] CommonKADS: siehe [36]A. T. Schreiber, B. J. Wielinga, R. de Hoog, H. Akkermans, and W. van de Velde, "CommonKADS: A Comprehensive Methodology for KBS Development," *IEEE Transactions on Knowledge and Data Engineering,* vol. 9, pp. 28-37, 1994.

2.4 Software Qualität

In diesem Kapitel wird zunächst der Begriff der Software Qualität anhand von allgemein gültiger Literatur und Definitionen thematisiert. Anschließend wird aus den gegebenen Fakten eine Arbeitsdefinition abgeleitet, unter der in der weiteren Folge dieser Arbeit der Begriff verstanden wird.

2.4.1 Der Begriff in der Literatur

Der Begriff „Software Qualität" lässt sich nicht konkret mit genau einer Definition beschreiben. Das liegt vor allem an den vielen unterschiedlichen Qualitätsmerkmalen eines Software-Produkts. Denn jede/r SoftwareentwicklerIn und auch jede/r EndbenutzerIn hat eine andere Auffassung, welche Kriterien für ihn bzw. sie als Qualitätsmerkmale in einer Software vorhanden sein müssen. Nach der ISO/IEC-Norm 9126[5] wird der Begriff Software Qualität wie folgt definiert:

> *„Software Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen."* ([3], S. 6, vgl. [37])

Der ISO/IEC-Standard ist über den Link in der Fußnote im Original verfügbar. Weitere Definitionen von Software Qualität sind der in [1] erwähnte IEEE-Standard 729-1983 und der ISO 8402:1991 aus [2], die hier aufgelistet sind.

*„**Software quality:***

(1) The totality of features and characteristics of a software product that bear on its ability to satisfy given needs; for example, conform to specifications.
(2) The degree to which software possesses a desired combination of attributes.
(3) The degree to which a customer or user perceives that software meets his or her composite expectations.
(4) The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer"
([1], S. 12-13).

> *„Die Gesamtheit der Features und Charakteristiken eines Produkts oder Service, die die Fähigkeit haben implizite oder explizite Bedürfnisse zu erfüllen"* (vgl. [2], S. 238)

[5] ISO/IEC 9126-2001: http://www.cse.unsw.edu.au/~cs3710/PMmaterials/Resources/9126-1%20Standard.pdf

Diese Definitionen decken sich zwar zum Teil mit dem ISO/IEC 9126, zielen aber teilweise in eine andere Richtung ab, oder konkretisieren einige Details. Neben den standardisierten Formulierungen gibt es in [1, 3] weitere Ansätze und Beschreibungen, wie Qualität von Software vergleichbar wird. Diese und das Zitat von Grady Booch in der Einleitung zeigen die Schwierigkeit, die Bedeutung des Begriffs eindeutig festzulegen, denn um eine genauere Definition des Begriffs zu finden genügt es nicht, sich mit den aktuellen Fragen bezüglich Qualität von Softwareprodukten zu beschäftigen. Vielmehr muss man zum einen die historische Entwicklung betrachten und zum anderen mögliche Szenarien für zukünftige Softwareprojekte berücksichtigen. Letzteres gestaltet sich allerdings in einer schnelllebigen Zeit, in der sich technische Fortschritte und Errungenschaften schon beinahe täglich ändern, mitunter als sehr schwierig.

Die Qualität von Software ist in den letzten Jahren und Jahrzehnten, seit der Softwarekrise in den 1960er-Jahren ein immer wieder zur Diskussion gebrachtes Thema (vgl. [8]). So schrieb Edsger W. Dijkstra 1972 in *The humble programmer*, dass zu dieser Zeit die Softwarekrise voraussehbar war und durch die Verbesserung der Maschinenleistung zwangsläufig zu einer Veränderung der Softwareentwicklung führen würde. Es war zur damaligen Zeit auch vom „turning point“ für die Softwareentwicklung die Rede (vgl. [38]). Diese Veränderungen lassen sich auch auf die heutige Zeit replizieren. Hardware wird immer performanter und kostengünstiger in der Produktion, die Software allerdings bleibt zumeist in der Entwicklung und den Kosten auf dem gleichen Niveau. Seit der Softwarekrise und der anschließenden NATO-Tagung im bayerischen Ort Garmisch (GER) 1968, sowie Dijkstras Werk ([38]) hat sich in der IT-Welt einiges verändert. So wurden beispielsweise Programmiersprachen entwickelt, die eine Wiederverwendbarkeit von Programm-Fragmenten erleichtern und unterstützen, oder auch IT-Management-Prozesse eingeführt und etabliert, um aus vergangenen Softwareprojekten lernen zu können.

Software Qualität und ihre Messbarkeit sind in der heutigen Zeit, in der Programme und Softwaresysteme immer größer und komplexer werden, nicht nur in Expertenrunden ein aktuelles Diskussionsthema. Dabei geht es nicht nur um Benutzerfreundlichkeit (usability) oder Performance der Software, sondern auch wie Fehler zukünftig vermieden oder mit ihnen umgegangen werden kann. So bemängelte beispielsweise Heinrich C. Mayr 1995 in [8] in einer seiner Thesen, dass es keinen „Stand“ für InformatikerInnen und IT-Fachleute gibt, der zur Folge hätte, dass es Standesregeln und ein Standesbewusstsein gäbe. Diese Regeln und das Bewusstsein würden es erleichtern, Qualitätskriterien zu definieren (vgl. [8]). Mayr bemängelt in diesem Artikel auch, *„in kaum einer anderen Disziplin“* sei man *„derart bereit, alles Machbare auch tatsächlich zu machen als in der Informatik“* [8].

Sicher hat dieser Artikel in der heutigen Zeit bei weitem nicht mehr die Aussagekraft, wie in den 1990er-Jahren. Allerdings besitzen die Kernaussagen in [8] nach wie vor ihre Gültigkeit. Dies wird auch durch die Fehler beim Absturz der Ariane 5-Rakete im Juni 1996 deutlich. Eine der Ursachen war die mangelhafte Qualität der Software. Es ist eines von vielen Beispielen, bei denen Software-Fehler Ursachen eines größeren Schadensfalls sind. Nach Meldungen des Internetportals sqs[6] sind in dieser Reihe auch noch Fehler im amerikanischen Gesundheitssystem „Obamacare“ und Buchungsfehler während der Umstellung des europäischen Zahlungsverkehrs SEPA zu nennen. Ausführliche Informationen zu diesen beiden und weiteren Beispielen bietet der Link in der Fußnote.

In vielen anderen Sparten, wie z. B. der Architektur oder im Maschinenbau, gibt es seit Jahrzehnten bzw. Jahrhunderten gültige Qualitätskriterien, die es nicht nur den BenutzerInnen, sondern auch den ProduzentInnen der Produkte erleichtern, diese zu vergleichen. In der IT-Welt ist man hiervon allerdings noch einige Schritte entfernt. Grund dafür ist nicht nur das Fehlen einer normierten Definition von Software Qualität, sondern auch dass die Qualitätskriterien nicht einheitlich und teilweise nicht messbar sind.

Nach Hoffmann gibt es „*nicht **das** eine Kriterium, mit dem sich Software Qualität in direkter Weise und vor allem quantitativ verbinden lässt. Vielmehr verbergen sich hinter dem Begriff eine ganze Reihe vielschichtiger Kriterien, von denen sich einige zu allem Überdruss auch noch gegenseitig ausschließen“* ([3], S. 6). Das bedeutet, dass Software Qualität keine einheitliche Definition über genau eine Ausprägung hat. Die Qualitätskriterien und Eigenschaften einer Software lassen sich nur schwer in einheitliche Kategorien und vor allem Messgrößen einteilen. Die wesentlichen Hauptmerkmale dieser Kategorien von Software Qualität stellt Hoffmann in der Abbildung 2-4 dar.

[6] sqs: https://www.sqs.com/portal/news/de/pressemitteilungen-zehn-spektakulaersten-software-fehler-2013.php

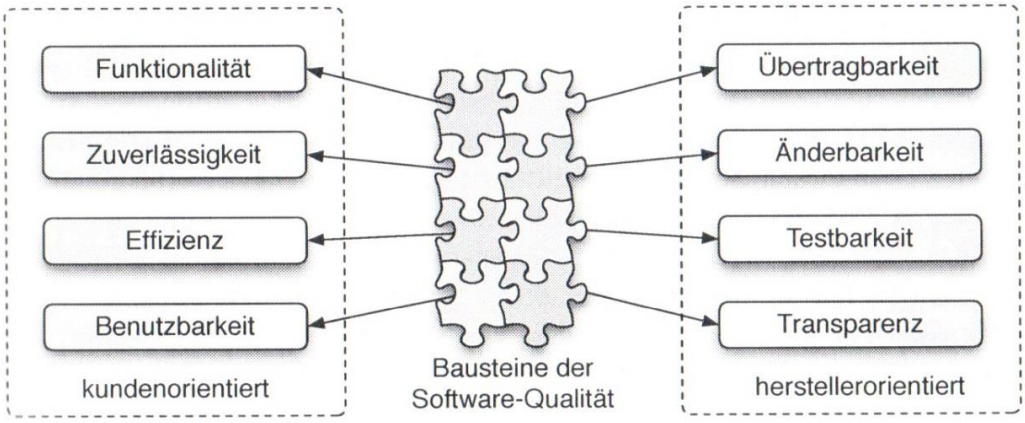

Abbildung 2-4: Qualitätsmerkmale eines Software Produkts (aus [3], S. 7)

Damit die Qualität von Software messbar werden kann, benötigt man Anforderungen, die sog. Requirements. Diese unterteilen sich in funktionale und nicht-funktionale Anforderungen. Nach Chung und do Prado Leite sind die nicht-funktionalen Anforderungen (engl. non-functional requirements (NFRs)), wie Verwendbarkeit, Flexibilität, Performanz, Kompatiblität und Sicherheit (engl. usability, flexibility, performance, interoperability, security) nicht direkt messbare Größen in der Softwareentwicklung, wo hingegen die funktionalen Anforderungen direkt aus den Spezifikationen ablesbar und dadurch auch messbar sind (vgl. [5]). Wallmüller beschreibt in seinem Werk die relevanten Eigenschaften und Qualitätsmerkmale aus der jeweiligen Perspektive des/der BenutzerIn, des Software-Betreibers[7], des/der DesignerIn und des/der ProgrammiererIn, wie die beiden folgenden Abbildungen beispielhaft zeigen.

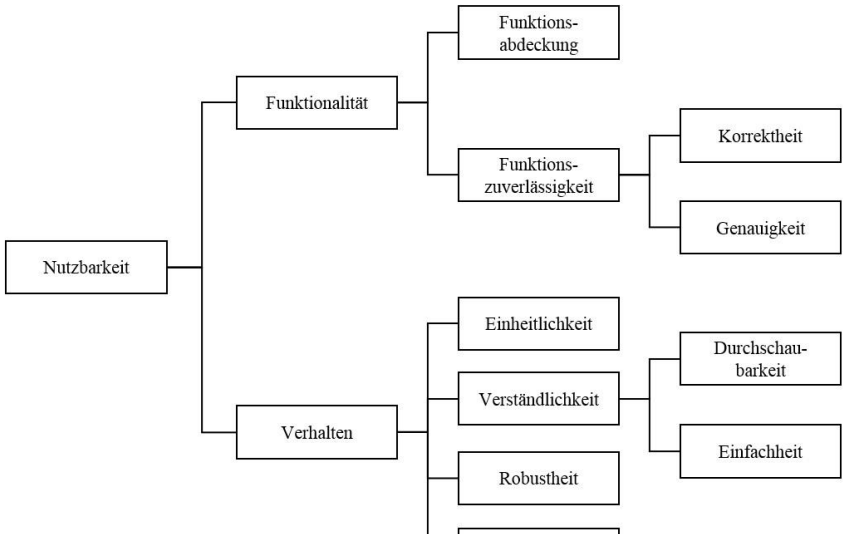

Abbildung 2-5: Relevante Eigenschaften für Benutzer (nach [1], Abb.1.25a))

[7] Software-Betreiber sind in der Regel Unternehmen oder Organisationen (Anm. d. Autors)

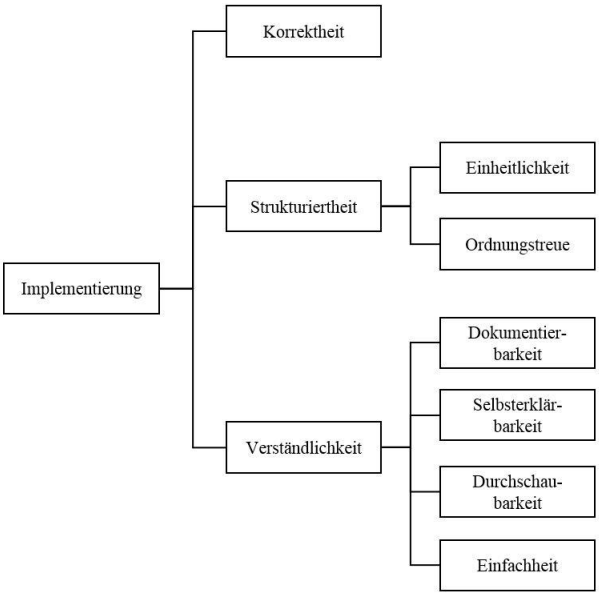

Abbildung 2-6: Relevante Eigenschaften für Programmierer (nach [1], Abb. 1.25d))

Um beide, die funktionalen und nicht-funktionalen Anforderungen, gleichermaßen messbar und vergleichbar zu machen, gibt es nach Chung et al. einige Modelle, wie den Software Quality Tree, das Qualitätsmodell nach McCall (1977), das u.a. in [39] als Vorgängermodell zum Software Quality Tree und dem ISO/IEC-Standard 9126 gilt, und dem NFR Framework, die speziell die Vergleichbarkeit und Messbarkeit der NFRs als Qualitätsmerkmale in der Softwareentwicklung stützen (vgl. [5]). Der Software Quality Tree wird auf Basis der funktionalen und nichtfunktionalen Anforderungen aufgebaut, die sich aus der Anforderungsanalyse eines Software-Projekts ergeben. Diese Anforderungen werden durch Pfeile kategorisiert und miteinander verknüpft. Die Abbildung 2-7 zeigt den Aufbau eines Baums nach Chung und do Prado Leite (vgl. [5]).

Um die Wiederverwendbarkeit von Softwareteilen gewährleisten zu können, bzw. zu ermöglichen wurden einige Standards als Managementprozesse (vgl. [3]) und -modelle (vgl. [1]) umgesetzt. Hierzu zählen vor allem die Vorgehensmodelle, in der Informatik oft auch als Software Engineering Modelle bekannt, Reifegradmodelle, wie Capability Maturity Model (CMM), Capability Maturity Model Integration (CMMI) (vgl. [3], S. 26) und der ISO 15504 Standard, der als Software Process Improvement and Capability dEtermination (SPICE) bekannt ist (vgl. [3], S. 535), Qualitäts- und Risikomanagementprozesse (vgl. [1]), sowie auch der Software

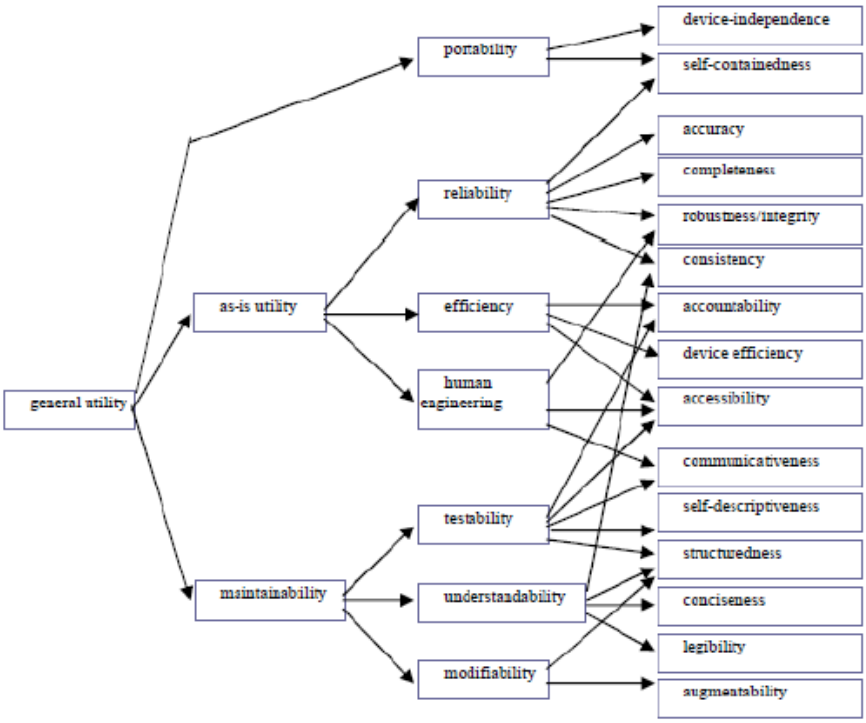

Abbildung 2-7: Der Aufbau eines Software Quality Tree (aus [5])

Engineering Body of Knowledge (SWEBOK)[8]. Der SWEBOK ist dabei allerdings ein Hilfsmittel für das Management von Softwareprojekten. Die Umsetzung wird u.a. durch ein e-Book der IEEE (siehe Fußnote) beschrieben.

Nach Ernest Wallmüller ist Software Qualität nicht nur eine Frage der Entwickler, sondern auch des Software-Managements (vgl. [1], S. 3). Einer der zentralen Ansätze nach Wallmüller ist der Software-Lifecycle-Prozess nach ISO/IEC 12207, der in Abbildung 2-8 dargestellt wird.

Es gibt zahlreiche weitere Ansätze, um die Qualität von Software zu verbessern. Nach Kitchenham und Pfleeger in [39] gibt es nicht nur das von ihnen erklärte Qualitätsmodell nach McCall, sondern auch Modelle um die Sichtweisen der einzelnen Beteiligten und Stakeholdern zu abstrahieren (vgl. [39]). Cadle und Yeates beschreiben die Ansätze zur Qualitätsverbesserung auf der Management-Seite. Mit Value

[8] SWEBOK: http://www.computer.org/portal/web/swebok/htmlformat (IEEE Computer Society)

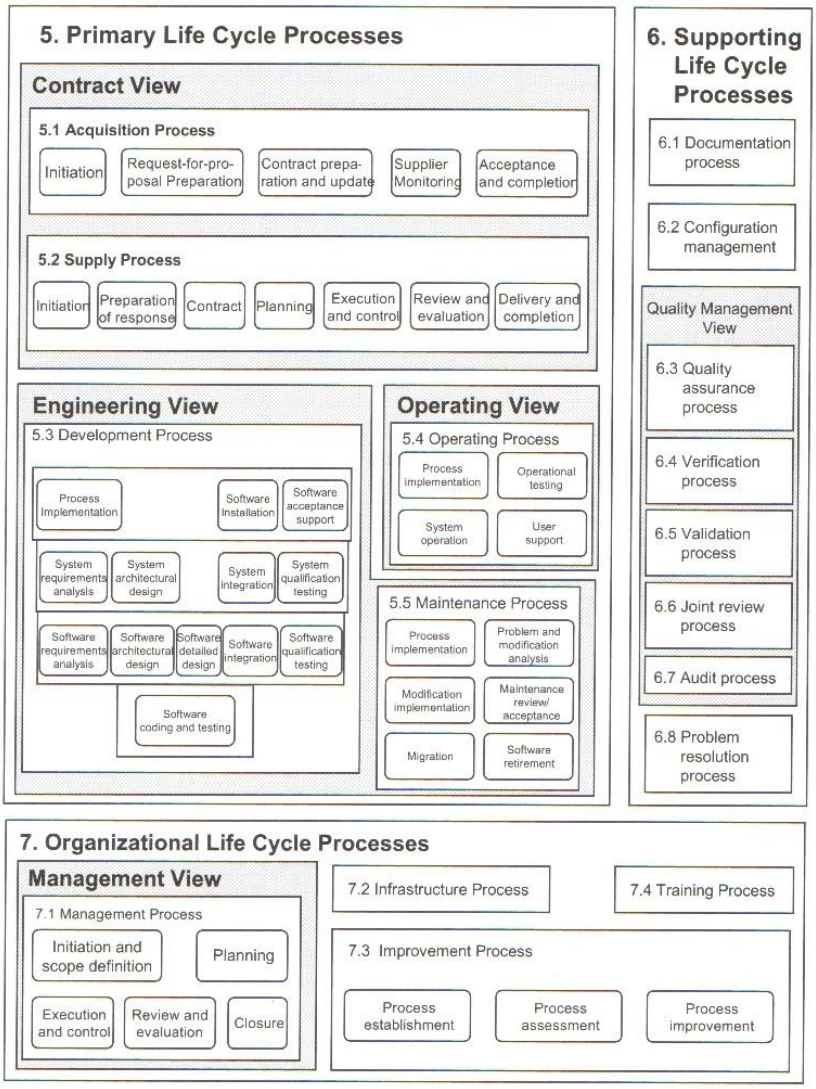

Abbildung 2-8: Software-Lifecycle-Prozesse, Sichten und Aktivitäten nach ISO/IEC 12207 (aus [1], S. 9, Abb. 1.4)

Management und Value Engineering, sowie das Konzept des Value-Trees zeigen sie, dass es bereits vor Beginn eines Projekts Möglichkeiten zur Messbarkeit von Qualität gibt (vgl. [2]).

Mit Sätzen, wie *„Die Idealvorstellung, die Software-Entwicklung rein ingenieurwissenschaftlich zu verstehen und zu betreiben, muss revidiert werden“* und *„Den Phänomenen Projekt und Projekt-Management als kritische Erfolgsfaktoren für das IT- und Software-Geschäft müssen mehr Aufmerksamkeit geschenkt werden“* ([1], S. 381) trifft Wallmüller den Nerv der Zeit. Es ist kaum möglich, alle Qualitätskriterien schon zu Beginn des Softwareentwicklungsprozesses zu definieren und deren Maßeinheiten festzulegen. *„Prozessverbesserung ist auf allen Stufen einer Software-Organisation ein Thema. Manager werden wegen des potenziellen Geschäftsnutzens interessiert sein, Projektleiter und Auftraggeber wegen der besseren Transparenz des*

Projekt- und Produktfortschritts. Endbenutzer werden wegen des besseren Umgangs mit Anforderungen und der Erreichung ihrer (Qualitäts-)Ziele an Verbesserungsmaßnahmen Gefallen finden" ([1], S. 381). Auch aus diesen Gründen schreibt er, dass der Trend wohl zu „*integrierte[n] Managementsysteme[n] und Business Excellence"* gehen und Qualitätsmanagement „*zu einer absoluten Notwendigkeit*" in der IT-Welt werden wird (vgl. [1], S. 381).

Hoffmann bemerkt, dass *„ein Blick in die Open-Source-Szene […] die Skepsis"* verstärkt, *„ob eine ausschließlich produktionsorientierte Sichtweise dem Bereich der Software-Entwicklung voll und ganz gerecht wird"* ([3], S. 545). Er schreibt ebenso, dass es zu wünschen wäre *„beide Sichtweisen in sich zu vereinen und eine ausgewogene Gratwanderung zwischen kreativer Freiheit und organisatorischer Kontrolle"* zu ermöglichen (vgl. [3], S. 545).

Diese ausgewogene Gratwanderung gestaltet sich in der beruflichen Praxis allerdings als schwieriger umzusetzen als gedacht, da viele Softwareprojekte in Unternehmen und Organisationen durchgeführt werden, deren ökonomischer Fokus auf anderen Bereichen als der IT liegen und dadurch die Softwareentwicklung kaum in seiner ursprünglichen Form eingesetzt wird. Außerdem wäre eine ausschließlich produktionsorientierte Sichtweise im Software Engineering durch die Verzweigungen und Einflüsse der IT in beinahe jedem gesellschaftlichen Bereich eine enorm große Änderung, die heutzutage geradezu unmöglich durchzuführen wäre.

2.4.2 Arbeitsdefinition von Software Qualität

Wie zuvor gezeigt, kann der Terminus „Software Qualität“ sehr unterschiedlich interpretiert werden. Deshalb ist es notwendig, eine Arbeitsdefinition festzulegen, aus der in weiterer Folge der Begriff abgeleitet und verstanden wird. Diese könnte in diesem Kontext aus den ISO-Standards[9] abgeleitet werden und wie folgt lauten:

> *Die Qualität von Software wird zum einen durch die Fähigkeit zur Ausführung der gestellten Anforderungen und zum anderen durch die effektive und effiziente Projektdurchführung gekennzeichnet.*

Diese Definition ist keinesfalls allumfassend, auch kann die Software Qualität damit nicht eindeutig beschrieben werden. Allerdings ist es hilfreich zu wissen, wie die Qualität in dem Kontext einer wissenschaftlichen Arbeit definiert wird. Darüber hinaus wird u.a. in den Kapiteln 4 und 6 eine klare Definition benötigt, ohne die das Verständnis der LeserInnen, ohne fachspezifische Kenntnisse nicht mehr gegeben wäre.

Vor allem in der qualitätsbewussten Softwareentwicklung – die in Kapitel 4 ausführlicher behandelt wird – ist eine präzise und für das jeweilige Projekt umfassende Definition der Qualitätskriterien unerlässlich. In der Praxis kann es vorkommen, dass jedes Projektteam in einem Unternehmen oder Organisation seine eigene Definition der Software Qualität definiert. Im wissenschaftlichen und universitären Kontext der angewandten Informatik werden den Studierenden nur die Kernkonzepte vermittelt, um so eine gewisse Flexibilität erhalten zu können.

[9] Folgende ISO- und IEEE- Standards sind die Grundlage dieser Definition:
ISO/IEC 9126, IEEE-Standard 729-1983, ISO 8402:1991, ISO/IEC 12207

2.5 Metamodell

Das Wort Metamodell setzt sich aus *meta* (griech. über) und Modell zusammen. Ein Metamodell ist eine zumeist graphische Beschreibung eines konkreten Modells. Es ist also ein Modell für ein Modell. Dies beschreiben auch Habermann und Leymann in ihrem Buch, wonach das Metamodell ein Modell ist, aus dem wiederum ein Modell hervorgeht (vgl. [4], S. 60). Nach Gašević et al. wurden Modelle und Metamodelle im Model Driven Engineering (MDE) von der Object Management Group (OMG) parallel zum Semantic Web entwickelt (vgl. [12], S. 125).

Modelle haben nach Gašević et al. fünf Schlüssel-Charakteristiken:

- **Abstraction** (Abstraktion)
  Ein Modell ist immer eine reduzierte Visualisierung des repräsentierten Systems
- **Understandability** (Verständlichkeit)
  Es ist nicht ausreichend nur Details weg zu abstrahieren; es muss auch das in einer Form sichtbar bleiben, was die Intuition des Modellierers ist
- **Accuracy** (Präzision)
  Ein Modell muss eine „true-to-life"-Repräsentation des modellierten Systems beinhalten
- **Predictiveness** (Vorhersagegenauigkeit)
  Es soll die Möglichkeit bestehen, ein Modell zur korrekten Vorhersage des modellierten Systems formal zu beschreiben
- **Inexpensiveness** (Billigkeit)
  Modellierung muss signifikant weniger Kosten aufweisen, als das zu modellierende System zu konstruieren und zu analysieren.
  (vgl. [12], S. 126)

Habermann und Leymann beschreiben ebenfalls, dass das *Information Resource Dictionary System* (IRDS) basierend auf den ANSI- und ISO-Standardisierungen definiert wurde. Es lässt eine Schichten-Architektur zu (vgl. [4], S. 61), aus dem sowohl ein Metamodell, als auch ein konkretes Modell hervorgehen. Zu dem Modell existieren reale Daten, auf die dieses zugreift. Im ANSI-Kern-Modul werden diese Zusammenhänge schematisch dargestellt (Abbildung 2-9).

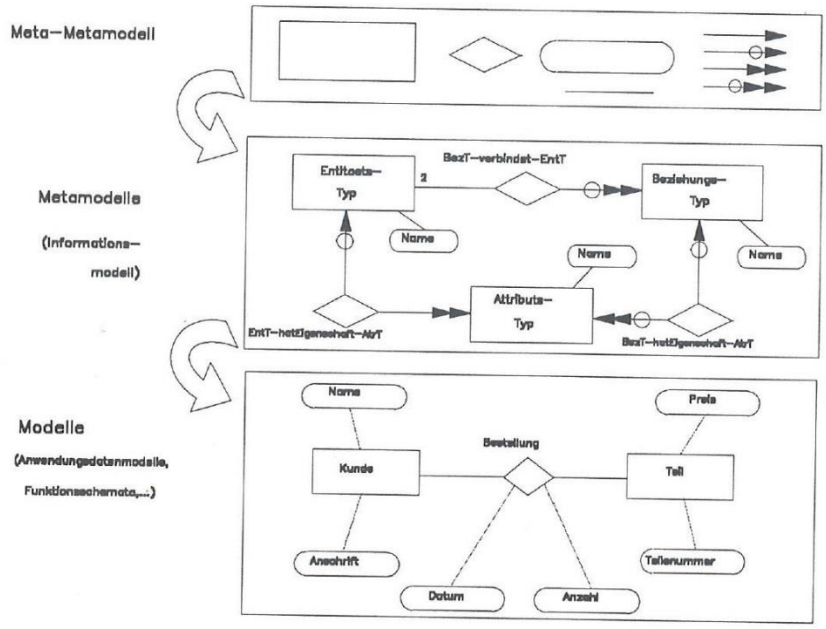

Abbildung 2-10: Modellhierarchien (aus [4])

Metamodelle werden auch Informationsmodelle genannt, die ebenfalls ein Metamodell, das sog. Meta-Metamodell, haben können. Diese Zusammenhänge und der Aufbau von der Schicht L1 zu L3, also vom Meta-Metamodell zum konkreten Modell werden in Abbildung 2-10 gezeigt (vgl. [4]). Der Ansatz *Meta Object Facility (MOF)* von der OMG zeigt den umgekehrten Weg. Dieser Bottom-Up-Ansatz wird von M0 (Daten der „realen Welt“) bis M3 (Meta-Metamodell) durchgeführt und findet im Model Driven Engineering seine Anwendung.

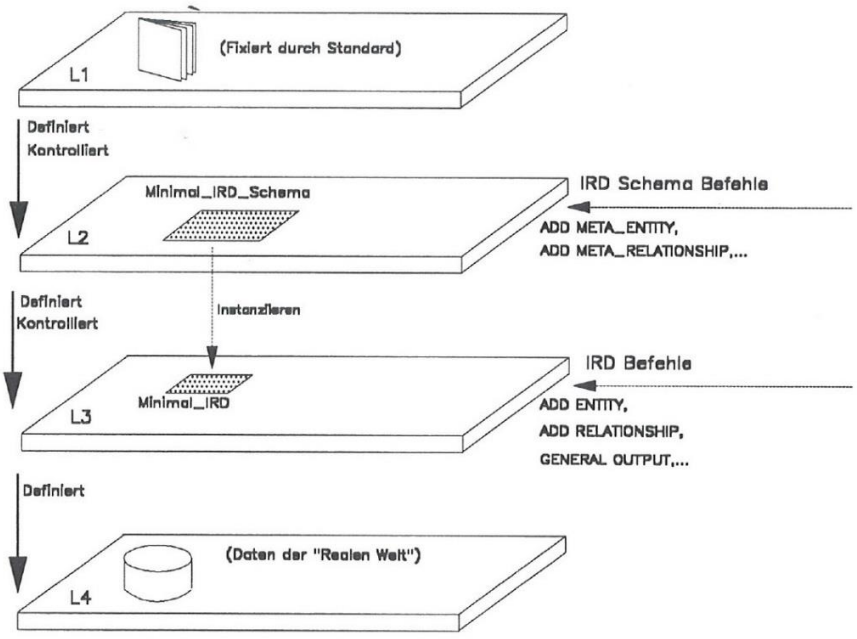

Abbildung 2-9: Die Schichten der Meta-Modellierung (aus [4])

3 Werkzeuge & Software zur Realisierung des Projekts

In diesem Kapitel wird auf die wichtigsten und im Projekt QuASE verwendeten Softwaretools für Issue- und Ticket-Management, Modellierung und einige Frameworks zur Softwareentwicklung von QuASE eingegangen. Bei diesen Softwaretools und Frameworks handelt es sich zu einem großen Teil um Open-Source Software, also Programme und Frameworks, die im Web frei zugänglich sind.

3.1 Ticket- & Issue-Management-Systeme

Ticket- und Issue-Management-Systeme (TIMS) sind in der alltäglichen Projektarbeit verwendete Softwarewerkzeuge, mit denen Planung, Verlauf und Durchführung von verschiedenen Projekten durchgeführt werden können. Im weiteren Verlauf werden Ticket- und Issue-Management-Systeme auch nur als Issue-Managementsysteme (IMS) bezeichnet, die Bedeutung ist allerdings gleich, das bedeutet, diese beiden Bezeichnungen und deren Abkürzungen sind Synonyme.

Zunächst wird die klassische Vorgangsweise bei diesen Werkzeugen kurz erläutert. In der Folge werden die wichtigsten und jene, für die die Applikation des Forschungsprojekts QuASE konzipiert wurde, kurz vorgestellt.

3.1.1 Generelle Funktionsweise von Werkzeugen zur Projektplanung

Projektmanagement-Werkzeuge funktionieren in der Regel mit Geschäftsprozessen. Meistens werden sie als Ticketing-Systeme eingesetzt, das bedeutet, der/die BenutzerIn oder AuftraggeberIn legt ein Ticket oder Issue an und fügt eine Beschreibung hinsichtlich des Problems oder der Änderung des Softwaresystems an und die Projektbeteiligten werden daraufhin benachrichtigt, dass es Änderungen bzw. Änderungswünsche im Projekt gibt. Auf Grund dieser Informationen werden im Normalfall die ProjektmanagerInnen die Kalkulationen über diese Werkzeuge durchführen, um den EntwicklerInnen anschließend eine Freigabe zur Bearbeitung zu erteilen. Nach erfolgreichem Abschluss bekommt der/die AuftraggeberIn eine Benachrichtigung über die Erledigung des Auftrags.

Darüber hinaus kann mit Hilfe dieser Werkzeuge auch eine direkte Kommunikation der einzelnen Stakeholder durchgeführt werden, indem Kommentare oder weitere Beschreibungen an die Tickets/Issues angehängt werden. Jedoch ist die Nutzung der

Issue- und Ticket-Systeme von Organisation zu Organisation ebenso unterschiedlich wie die Bearbeitung der IT-Projekte.

3.1.2 JIRA Issue & Project Tracking Software

JIRA wurde von Atlassian entwickelt und ist aktuell (Stand März 2016) bis zur Version 6.4 verfügbar. Atlassian schreibt über dieses Tool, JIRA sei *„ein Projektverfolgungstool für Teams, die großartige Software erstellen wollen“* [40]. Dieser Satz klingt auf den ersten Blick sehr euphemistisch. Die Erfahrung zeigt jedoch, dass sobald mehrere Personen in den Entscheidungsprozess eingebunden und meherere Personen in den Entwicklungsteams sind, ist ein solches Tool sehr hilfreich und zumeist auch notwendig.

Die erste Version der JIRA-Applikation wurde 2002 auf den Markt gebracht und seitdem stetig weiter entwickelt (vgl. [41]). Atlassian bietet außerdem einige Erweiterungen im Projektmanagement-Sektor an, wie z. B. das Confluence, eine Art unternehmens- oder organisationsinterne Informationssammlung. Darüber hinaus bietet JIRA eine Vielzahl von Konfigurationsmöglichkeiten und Plugins an, um die Individualität der Projektmanagement-Aufgaben für jedes Unternehmen gewährleisten zu können.

Die Funktionsweise von JIRA ist abhängig von den Einstellungen der jeweiligen Unternehmen und kann zu der im Abschnitt 3.1.1 beschriebenen abweichen. So können beispielsweise einige Schritte übersprungen oder mehrfach ausgeführt werden, wenn es die Situation erfordert.

JIRA bietet als interne Datenhaltung die Möglichkeit aus einer Vielzahl von relationalen Datenbankmanagementsystemen (DBMS) zu wählen. Hierzu gehören die SQL-DBMS ORACLE, MySQL, PostgreSQL und die gängigen MS SQL Server Versionen (vgl. [42], Supported Plattforms).

Dieses Projektmanagementsystem wird bei den meisten strategischen Partnern für das Forschungsprojekt verwendet und gilt somit als eine der wichtigsten Grundlagen für QuASE. Weitere Informationen über Atlassian JIRA sind in der Online-Dokumentation [42] zu finden.

3.1.3 @enterprise

Das Projektmanagementsystem @enterprise wurde von Groiss Informatics GmbH, einem in Klagenfurt am Wörthersee (Österreich) ansässigen IT-Dienstleister, entwickelt. Dabei handelt es sich um ein Workflowsystem, das, ausgestattet mit einer Weboberfläche, als Modellierungstool für spezifische Anwendungsszenarien verschiedene Modelle den AnwenderInnen zur Verfügung stellt (vgl. [43]).

Durch die Einschränkung, in erster Linie ein Werkzeug zur Prozessmodellierung zu sein, muss bei @enterprise entweder der Prozess zur Steuerung eines Projekts in diesem System erstellt werden oder ein zusätzliches Plugin entwickelt werden, das die Aufgaben eines Projektmanagementsystems bereit stellt.

Da diese Software allerdings nicht frei verfügbar ist, sollte die Möglichkeit, Erweiterungen zu diesem System zu erstellen, mit erfahrenen EntwicklerInnen oder dem Anbieter besprochen werden. Für weitere Informationen über die Software @enterprise wird empfohlen den IT-Dienstleister direkt zu kontaktieren.

3.1.4 Mantis Bug Tracker

Das Softwaretool Mantis Bug Tracker, abgekürzt MantisBT, zumeist auch nur Mantis genannt, ist ein webbasiertes Bug-Tracking-System, das in der Programmiersprache PHP geschrieben wurde. MantisBT ist ein sogenanntes Open Source System, d. h. der Quellcode ist öffentlich einsehbar. Es wurde erstmals im November 2000 in der Öffentlichkeit präsentiert (vgl. [44]).

Die Funktionsweise ist ähnlich wie im Abschnitt 3.1.1 beschrieben. Allerdings ist durch die chronologische Darstellung der Kommentar-Felder innerhalb eines Issues nicht so übersichtlich, wie beispielsweise bei JIRA, das die wichtigsten Fakten stets im Seitenkopf behält und eine Minimierung der einzelnen Kommentare anbietet.

Für MantisBT besteht, ebenso wie bei JIRA, die Möglichkeit aus verschiedenen DBMS eines zu wählen. Hierbei wird MYSQL als Default, sowie PostgreSQL mit vollständigem Support angeboten. MS SQL Server, ORACLE und IBM DB2 werden als experimentell eingestuft (vgl. [44]).

Das Entwicklerteam von MantisBT bietet eine Vielzahl von Plugins zur Erweiterung der Projektmanagement-Systeme an. Hierzu zählen z. B. agileMantis, eine Erweite-

rung um agile Softwareentwicklungsmethoden wie SCRUM zu realisieren und Gantt-Chart, ein Plugin zur Erstellung von Projektplänen. Die genaue Beschreibung, sowie viele weitere Plugins können auf der Entwickler-Website[10] eingesehen werden.

3.1.5 Weitere Ticket- & Issue-Management-Systeme

Zu den bekannteren bisher noch nicht genannten TIMS gehören Bugzilla[11], ein frei verfügbares Bug-Tracking System und OpenProject[12], ein webbasiertes Collaboration-System.

Verwendung findet Bugzilla in weltweit bekannten Software-Engineering-Projekten. Dazu zählen u. a. diverse Frameworks der Apache Software Foundation, der Linux-Kernel und Linux-Distributionen wie Red Hat und Novell.

OpenProject ist ein im Vergleich zu anderen TIMS noch relativ junges Bug-Tracking Projekt, das von der in Berlin ansässigen OpenProject Foundation e. V. gehostet wird. Zu den bekanntesten Verwendern dieses TIMS gehören die Deutsche Telekom AG und der Westdeutsche Rundfunk (WDR).

Diese beiden und die vorhin in diesem Abschnitt benannten Projekt-Management-Werkzeuge sind nur ein kleiner Ausschnitt von vielen, in der Regel ähnlichen Werkzeugen zur Verwaltung von Software- und IT-Projekten.

[10] MantisBT-Plugins: https://github.com/mantisbt-plugins

[11] Bugzilla: http://www.bugzilla.org/about/

[12] OpenProject: http://community.openproject.org/

3.2 Modellierungswerkzeuge

Um Metamodelle und Ontologien zu erstellen bedarf es einiger Softwarewerkzeuge, die in diesem Abschnitt kurz erläutert werden. Hierzu zählen ADOxx, ein Werkzeug um Modelle und Metamodelle zu erzeugen und das Tool-Set Protégé, zur Erstellung und Bearbeitung von Ontologien. Im Anschluss werden weitere Modellierungswerkzeuge gezeigt, die für verschiedene Anwendungsarten entwickelt wurden.

3.2.1 ADOxx

Die Software ADOxx ist eine von der BOC-Group – einem Spin-off-Unternehmen der Forschungsgruppe Knowledge Engineering an der Universität Wien unter der Leitung von o. Univ.-Prof. Dr. Karagiannis – entwickelte Plattform zur Entwicklung, Konfiguration und Implementierung von Metamodellen und den dazugehörigen Modellen.

Zur Erstellung von Metamodellen können bei diesem Tool nicht nur Modellierungssprachen, sondern auch die Modellierung selbst und korrespondierende Funktionalitäten in Form von Mechanismen und Algorithmen verwendet werden (vgl. [45]). Diese Modellierungssprachen werden im Allgemeinen *Domain Specific Modelling Languages* (DSML) genannt. ADOxx besteht in der Basis-Installation aus einem Development-Toolkit (Abbildung 3-1) und einem Modelling-Toolkit (Abbildung 3-2), wie in folgenden Abbildungen dargestellt, die mit verschiedenen Modulen erweiterbar sind, um spezifische Modelle und Metamodelle erstellen zu können.

Im Allgemeinen erzeugt man im Development-Toolkit die Metamodelle und dazugehörigen Algorithmen, sowie eventuelle Zugriffe auf externe Daten, etc. Im Modelling-Toolkit können dann die konkreten Modelle entwickelt werden. Meist kann man mit Hilfe dieser Modelle die Metamodelle verbessern oder um fehlende Beschreibungen ergänzen. Im Anschluss an die jeweilige Abbildung (3-1 und 3-2) auf den nächsten beiden Seiten folgt eine Übersicht über die Standard-Module und -Funktionalitäten von ADOxx.

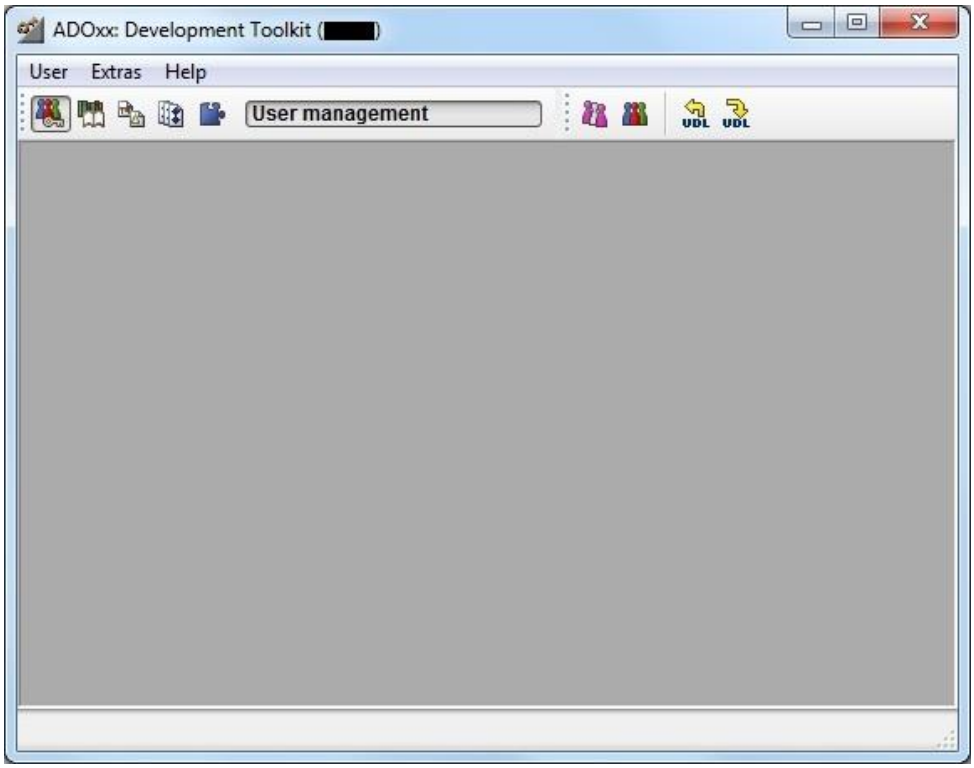

Abbildung 3-1: Screenshot des ADOxx-Development Toolkit Hauptbildschirms

Das Development-Toolkit besteht aus folgenden Teilsystemen:

- **Benutzerverwaltung** – um den jeweiligen Benutzern Berechtigungen und verschiedene Sichten geben zu können
- **Library-Management** – um die Beschreibungssprachen zu verwalten
- **Modell-Verwaltung** – um die einzelnen (Meta-) Modelle angelegen und verändern zu können
- **Attributprofil-Management** – um die Attribute der jeweiligen Modelle und Metamodelle verwalten zu können
- **Component-Management** – um die einzelnen Komponenten eines Metamodells zu verwalten

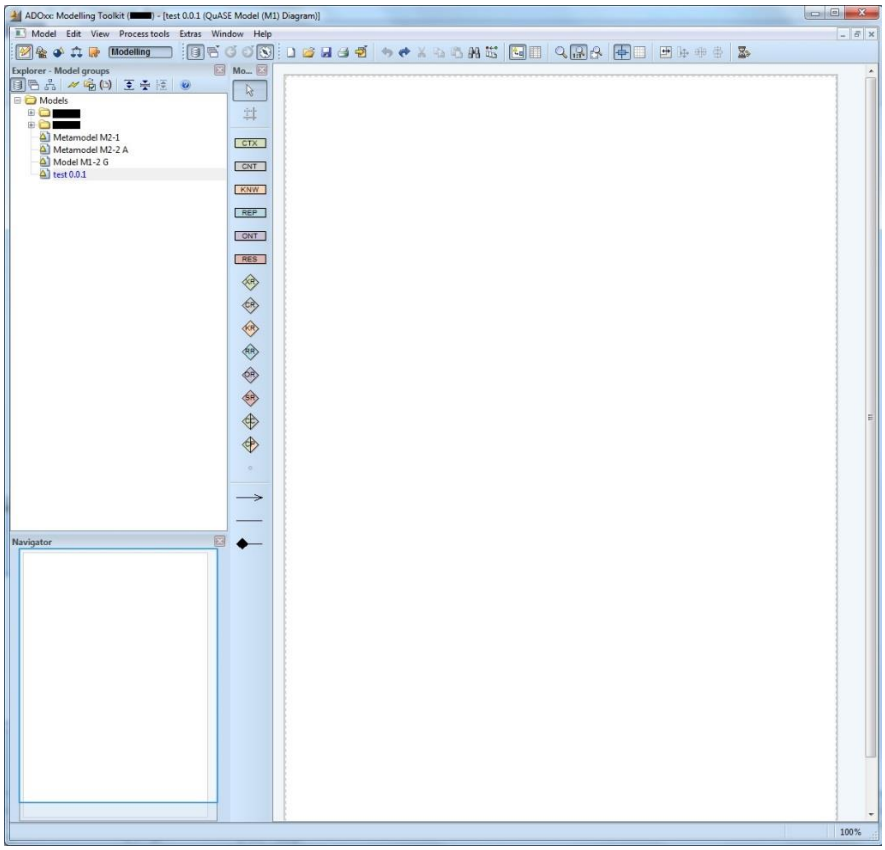

Abbildung 3-2: Screenshot des ADOxx-Modelling Toolkit Hauptbildschirms

Das Modelling-Toolkit besteht aus folgenden Komponenten:

- **Modelling** – um die konkreten Modelle zu konstruieren
- **Analysis** – um spezifische Analyse-Werkzeuge auf die Modelle anwenden zu können
- **Simulation** – um das ausgewählte Modell mit einem konkreten Anwendungsfall simulieren zu können
- **Evaluation** – um das ausgewählte Modell auf Korrektheit überprüfen zu können
- **Import/Export** – um verschiedene (Meta-) Modelle importieren oder exportieren zu können

Weitere Informationen über die Software ADOxx bieten die ADOxx-Website[13], die Forschungsgruppe Knowledge Engineering der Universität Wien[14] und das Open Models Laboratory (OMiLAB)[15].

[13] ADOxx: http://www.adoxx.org

[14] Forschungsgruppe Knowledge Engineering der Universität Wien: http://informatik.univie.ac.at/forschung/forschungsgruppen/knowledge-engineering/

[15] OMiLAB: http://www.omilab.org

3.2.2 Protégé

Die Ontologie-Konstruktionssoftware Protégé wurde von den Universitäten in Stanford und Manchester 2006 entwickelt. Diese Open-Source-Software gibt es derzeit in der Version 4.3.0, welche auch für das QuASE-Projekt verwendet wurde. Außerdem existiert eine Web-Applikation, die Webprotégé genannt wird.

Protégé bietet eine Menge an Werkzeugen und Plug-Ins, um Ontologien zu erstellen und zu konfigurieren. Diese werden in sog. Tabs im Hauptbildschirm – wie in Abbildung 3-3 beispielhaft dargestellt – angeordnet. Eines dieser Werkzeuge ist der SPARQL-Query-Editor, der in Abbildung 2-2 dargestellt wurde. Zudem sind Werkzeuge zur Modellierung von Entitäten, Objekten und deren Daten und Annotationen, sowie graphische Aufbereitungen enthalten. Weitere Informationen über die Software bietet die Protégé-Website[16].

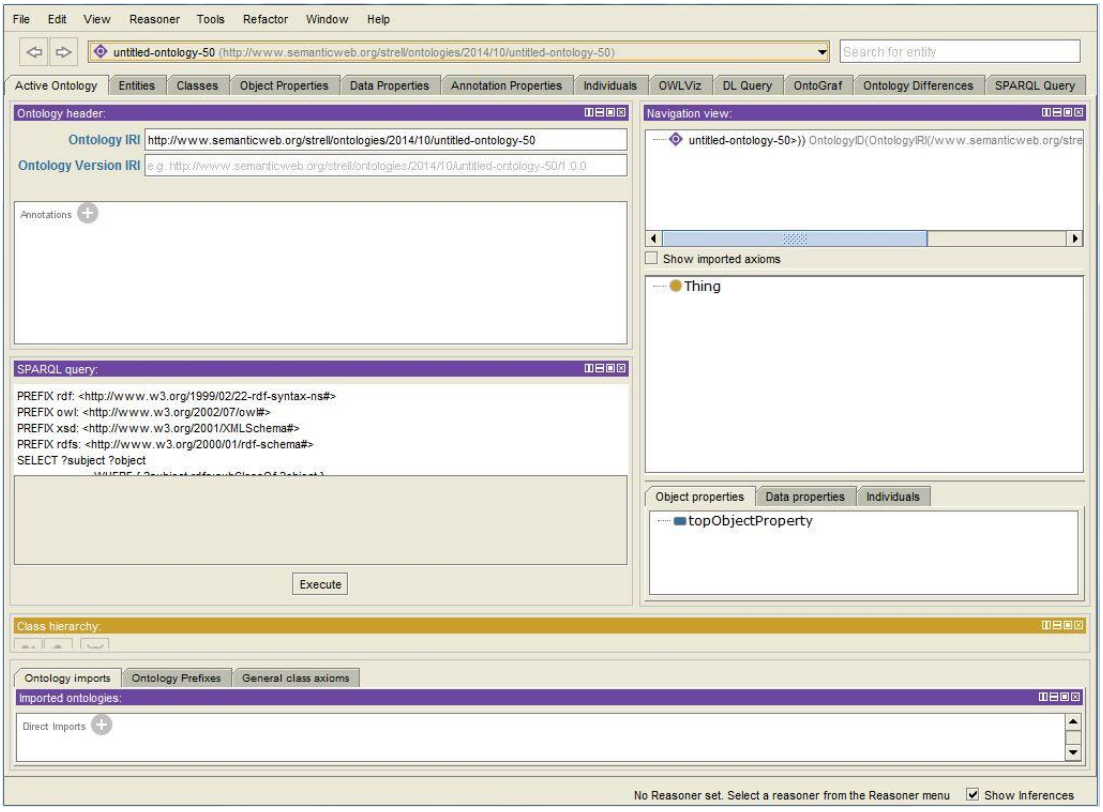

Abbildung 3-3: Screenshot des Protégé Start-Bildschirms

[16] Protégé: http://protege.stanford.edu/

3.2.3 Weitere Modellierungswerkzeuge

Zu den weiteren Modellierungswerkzeugen, die allerdings weder Teil des QuASE-Projekts sind noch dafür verwendet wurden, zählen Programme, wie ADONIS[17] von der BOC-Group, Integranova[18] (früher Olivanova) und Visual Paradigm[19]. Diese und einige weitere Werkzeuge unterstützen zu einem sehr großen Teil die EntwicklerInnen bei der Erstellung spezifischer Modelle, Diagramme und Prozesse.

ADONIS ist ein Werkzeug, das zur Erzeugung von Prozessmodellen und UML-Diagrammen entwickelt wurde. Integranova ist eine Software, dessen Ursprung auf die Universität von Valencia (Spanien) zurückgeht und den Ansatz des Modell Driven Engineering (MDE) verfolgt. Visual Paradigm ist ein Werkzeug zur Modellierung in diversen Notationen, wie UML und Business Process Modelling Notation (BPMN) sowie zur Erstellung von Datenbank-Modellen und Code Engineering. Weitere Informationen über diese Werkzeuge sind über die Verlinkungen in den Fußnoten verfügbar.

[17] ADONIS: http://www.boc-group.com/de/produkte/adonis/
[18] Integranova: http://www.integranova.com/integranova-m-e-s/
[19] Visual Paradigm: http://www.visual-paradigm.com/

3.3 Entwicklungswerkzeuge

Der Proof-of-Concept des Forschungsprojekt QuASE wurde in der Programmiersprache Java in der Version 8 entwickelt. Hierzu wurde das Framework *IntelliJ IDEA* in der lizenzfreien Community Edition verwendet. Weitere Werkzeuge, neben den vorhin bereits genannten Modellierungswerkzeugen und TIMS, sind die Frameworks *Jena*, *Shiro* und *Mahout* von Apache, die Java Plug-Ins *JUnit*, *JMockit*, *Spark*, *AngularJS*, *Jackson* sowie die Versionierungs-Tools *Bitbucket*, *ANT* und *MAVEN*. Diese werden im folgenden Abschnitt etwas näher beschrieben. Die genaue Funktionalität, die in QuASE verwendet wurde findet sich im Abschnitt 4.3.

3.3.1 Die Apache Frameworks

In der Softwareentwicklung des Projekts QuASE wurden diverse Frameworks von Apache verwendet. Mit Jena, Shiro und Mahout wurden für die QuASE-Applikation einige wichtige Module in den Java-Code eingebettet. Diese drei Frameworks werden in der Folge kurz vorgestellt. QuASE nutzt außerdem noch zwei andere Tools der Apache Software Foundation, ANT und MAVEN, die im letzten Abschnitt dieses Kapitels kurz vorgestellt werden.

Apache Jena

Apache Jena ist ein Open Source Framework zur Erstellung von Semantic Web-Inhalten. Dazu werden verschiedene APIs zur Bearbeitung u.a. von RDF-Daten und SPARQL-Queries verwendet (vgl. [46]). Mit diesem Framework lassen sich diese in einen Java-Code einbinden. Weitere Informationen und Einsatzmöglichkeiten sind in [46] nachzulesen.

Apache Shiro

Apache Shiro ist ein Security-Framework, das verschiedene APIs zur Authentifizierung, Autorisierung, Kryptographie und Session-Management bereitstellt. Diese werden u. a. für Login-Seiten und Datensicherheit während des Aufrufs eines Webinhalts oder einer Applikation verwendet (vgl. [47]). Da es sich bei Shiro ebenfalls um eine Open Source Software handelt, können diese Module für den individuellen Gebrauch angepasst werden. Weitere Informationen und Einsatzmöglichkeiten zur Verwendung in weiteren Programmiersprachen können auf der Hilfeseite im Web ([47]) nachgeschlagen werden.

Apache Mahout

Apache Mahout ist, wie Jena und Shiro, ein Open Source Framework der Apache Software Foundation. Mit Mahout können skalierbare Bibliotheken für maschinelles Lernen (machine learning) erstellt werden (vgl. [48]). Maschinelles Lernen wird verwendet, wenn eine Software unter verschiedenen Terminologien die im Kontext passende auswählen und darstellen soll. Für diesen Einsatz wurde Mahout auch in der QuASE-Applikation verwendet. Für weitere Informationen rund um Apache Mahout ist es sinnvoll, in [48] nachzuschlagen.

3.3.2 Java Plug-Ins

Die Programmiersprache Java bietet neben den Basis-APIs verschiedene Erweiterungen, die sog. Plug-Ins, die es den SoftwareentwicklerInnen ermöglichen Applikationen für die verschiedenen Anwendungsgebiete anzupassen und einsatzfähig zu machen. In der QuASE-Applikation werden die Plug-Ins Spark, Jackson, AngularJS, OWL API, TreeTagger for Java (TT4J), JUnit und JMockit verwendet. Diese werden in den folgenden Abschnitten näher erläutert.

Spark

Das Java Plug-In Spark ist ein einfaches Framework, dass eine schnelle Entwicklung von Web-UIs ermöglicht. Die Intention hinter diesem Framework ist es, eine einfache, auf Java basierende Weboberfläche zu erstellen, um jene Informationen bereitstellen zu können, die von den EntwicklerInnen benötigt werden [49]. Spark unterstützt im QuASE-Projekt den Ansatz eines Proof-of-Concepts, da die QuASE-Applikation zwar funktional, aber nicht visuell, vollständig sein muss. Weitere Informationen über Spark bietet die zugehörige Website des Entwicklers ([49]).

Jackson

Jackson ist ein Open Source Java JSON Parser und Data Binder, der von Tatu Saloranta entwickelt wurde. Das bedeutet mit Jackson ist die Software in der Lage das Datenformat JSON sowohl in Stream-APIs, Plain Old Java Objects (POJOs) oder in eine Baumstruktur zu konvertieren, als auch in umgekehrter Richtung agieren zu können (vgl. [50]). Weitere Informationen über Jackson[20] sind in den Links in der Fußnote nachlesbar.

[20] Jackson: http://jackson.codehaus.org/ bzw. https://github.com/FasterXML/jackson

AngularJS

AngularJS ist ein Struktur-Framework von google zur einfacheren Einbindung von HTML in eine dynamische Web-Applikation. Es ist erweiterbar und mit anderen APIs kombinierbar. Hierbei ist HTML die Template-Sprache, auf der die Web-Applikation aufbaut und mit dynamischen Inhalten kombiniert wird (vgl. [51, 52]).

OWL API

Die OWL API ist ein Plug-In für Java um Ontologien einzubinden, aufzurufen oder zu erstellen. Zentraler Punkt hierbei ist der *OWLOntologyManager*, der diese Funktionen in den Java-Code einbettet. Ein weiterer Teil dieser API ist der *OWL-Reasoner*, mit dem man die Klassenhierarchie berechnen kann, um damit durch die Struktur zu navigieren (vgl. [53]). Weiterführende Informationen über die OWL API und Code-Beispiele werden in [53] bereitgestellt.

TreeTagger for Java

Der TreeTagger for Java (TT4J) ist ein Java Wrapper um den originalen TreeTagger von Helmut Schmid. TT4J wurde entwickelt, um den TreeTagger zum einen plattformunabhängig und zum anderen einfach in Applikationen integrierbar zu machen (vgl. [54, 55]).

Der TreeTagger ist ein Tool, das Kommentare zu Texten mit „Part-of-Speech“- und Lemma-Informationen, basierend auf einem Markov Modell, zulässt. Der Algorithmus des TreeTaggers baut auf binären Entscheidungsbäumen auf (vgl. [56, 57]).

JUnit

JUnit ist ein Open-Source Framework für Java um wiederverwendbare Testklassen zu erstellen. Es ist eine Instanz der xUnit-Architektur um sog. Unit-Tests durchzuführen [58]. In einigen Entwicklungsumgebungen, wie z. B. Eclipse, gibt es dieses Framework als Plug-In-Feature. Ziel ist es, Unit-Tests schneller und einfacher zu wiederholen, ohne dass bei jedem Test-Durchlauf diverse Parameter neu gesetzt werden müssen. In Fachkreisen wird diese Art zu testen auch „Whitebox-Testing“ genannt.

JMockit

JMockit ist ein Open Source Framework für Mock-up-Tests, das in Zusammenhang mit Unit-Test-Frameworks, wie z. B. das zuvor beschriebene JUnit, funktioniert (vgl. [59]). Hierzu werden verschiedene APIs zur Verfügung gestellt, um innerhalb einer Testklasse die sog. Mock-ups zu generieren. Ein Mock-up ist ein Modell, mit dem

man z. B. bisher noch nicht implementierte Schnittstellen oder Code-Fragmente in einem bestehenden Modul nachbilden kann. Diese Tests werden in der IT auch als „Blackbox-Test“ bezeichnet.

Mit JUnit und JMockit lassen sich spezifische Tests und Testzyklen einfacher realisieren und eventuelle Design-Fehler früher bemerken, da diese APIs und Testklassen schon zu Beginn der Softwareentwicklung eingebunden werden können und so die Tests schon an einem sehr frühen Zeitpunkt im Softwareentwicklungsprozess durchlaufen werden können.

3.3.3 Weitere Frameworks in QuASE

Die weiteren Tools und Frameworks, die zur Erstellung der QuASE Software verwendet wurden, sind Bitbucket, ANT und MAVEN. Diese wurden zur Online-Speicherung des Quellcodes und zur Versionierung der Applikation, sowie Steuerung der Zugriffe verwendet.

Bitbucket

Bitbucket von Atlassian ist ein webbasierter Hosting-Service zur Speicherung von Quellcode. Bitbucket[21] nutzt Funktionen des Mercurial und Git Versionierungssystem. Die einzelnen EntwicklerInnen bzw. das Entwickler-Team können dabei selbst entscheiden, ob und für wen der Quellcode von außerhalb zugänglich gemacht wird. Weitere Informationen bietet die Informationsseite von Atlassian, deren Link in der Fußnote zu finden ist.

ANT und MAVEN

Die Frameworks ANT und MAVEN wurden wie Shiro, Mahout und Jena von der Apache Software Foundation entwickelt. ANT[22] und MAVEN[23] sind allerdings nicht, wie die vorhin beschriebenen Apache Frameworks, direkter Bestandteil der Softwareentwicklung, sondern in Verbindung mit Git und Bitbucket Teil der Versionierung und Online-Speicherung. Hierbei ist ANT für den „Build“ der Applikation zuständig und MAVEN für das Management der Software als Projekt. Weitere Informationen über diese Tools bieten die Links in den Fußnoten.

21 Atlassian Bitbucket: https://de.atlassian.com/software/bitbucket/overview
22 Apache ANT: http://ant.apache.org/
23 Apache MAVEN: http://maven.apache.org/

4 Quality-Aware Software Engineering

In diesem Kapitel wird mit den Zielsetzungen des Technologiefeldes „Quality-Aware Software Engineering“, also der qualitätsbewussten Softwareentwicklung, die thematische Einordnung der Forschungsfrage und des Forschungsprojekts QuASE, sowie deren konkreter Applikation der projektspezifische Teil der Arbeit eingeleitet. Das bedeutet dieses und die folgenden Kapitel sind enger an das Forschungsprojekt QuASE und der Beantwortung der Forschungsfrage geknüpft.

4.1 Zielsetzungen der qualitätsbewussten Softwareentwicklung

Die qualitätsbewusste Softwareentwicklung unterscheidet sich vom klassischen Software Engineering vor allem dadurch, dass die Software Qualität und die nichtfunktionalen Anforderungen im Vordergrund stehen. Wie in Kapitel 2.4 bereits erläutert hängt die Definition der Software Qualität stark von den benötigten Anforderungen und Zielsetzungen ab.

Eine der bekanntesten Methoden, die zur qualitätsbewussten Softwareentwicklung zählt, ist das testgetriebene Software Engineering (engl. Test-Driven Development[24], TDD). Hierbei wird im Gegensatz zu den klassischen Entwicklungsmethoden zunächst die Testumgebung aufgebaut, anhand der anschließend die Software entwickelt wird.

Die qualitätsbewusste Softwareentwicklung wird vermehrt auch in der Entwicklung von Automatisierungs- und modellbasierten Systemen eingesetzt. Dies geht u. a. aus Abufouda in [72] hervor. In diesem Artikel geht es um den qualitätsbewussten Ansatz für selbstanpassende Software Systeme, um die Kosten bei der durch ExpertInnen durchgeführten Wartung zu reduzieren (vgl. [72]). Dieser Ansatz – der an der Technischen Universität Kaiserslautern entwickelt wurde – verfolgt ähnliche Ziele wie die QuASE-Applikation der Alpen-Adria-Universität Klagenfurt, die Grundlage dieser Masterarbeit war. So geht es nach Abufouda in dessen Artikel und die darauf bezogene Applikation darum aus der Vergangenheit zu lernen und die Unsicherheit (tech. Uncertainty) in Software Systemen zu reduzieren (vgl. [72]).

Die Uncertainty entsteht in der Softwareentwicklung hauptsächlich durch Abläufe, deren Auswirkungen nicht zuverlässig in einer Formel berechnet werden können. So ist zum Beispiel die Auswirkung auf eine Ausnahmebehandlung (tech. Exception-

[24] Weitere Informationen zu Test-Driven Development: http://www.frankwestphal.de/ftp/Einleitung.pdf

Handling) durch die Software bei einem Fehler oder anfangs nicht berücksichtigten Abläufen erfahrungsgemäß eine der größten Unsicherheiten in der Softwareentwicklung. Niemand kann zuverlässig vorhersagen, ob die technischen Gegebenheiten nach dieser Ausnahmebehandlung genau so sind wie erwartet oder wie die EndbenutzerInnen diese Ausnahmebehandlung auffassen und das Programm eventuell in einen anderen Zustand überführen (z.B. in einen Debug-Modus).

Die qualitätsbewusste Softwareentwicklung zielt auf eine detailliertere Einhaltung der wichtigsten Qualitätsstandards einer Software ab. Hierbei werden vor allem die ISO-Standards 9000 und 9001 (u.a. in [61]), sowie eine klare Arbeitsdefinition für Software Qualität als Grundlage verwendet. Vor allem den nicht-funktionalen Anforderungen, wie z.B. Wartbarkeit und Benutzerfreundlichkeit wird in der qualitätsbewussten Softwareentwicklung ein hoher Stellenwert beigemessen. Erfahrungsgemäß werden Kosten und Zeit als eher weniger wichtig erachtet, dafür mehr auf Funktionalität und Einfachheit in der Benutzung der Software geachtet. Was allerdings nicht bedeutet, dass man unendlich Zeit und einen übermäßig hohen Kostenrahmen für diese Projekte zur Verfügung stehen hat.

Die qualitativ höherwertigen Softwareentwicklungen erkennt man außerdem an einem klaren Konzept und einer leicht verständlichen Architektur. Zudem zeichnen sich die Entwicklungsteams durch eine gute Kommunikation untereinander und mit den Stakeholdern aus. Das bedeutet nicht, dass diese Software perfekt ist bzw. keine Fehler enthält. Jedoch werden diese im Laufe des Projekts schneller erkannt und deren Auswirkungen minimiert. Ein weiterer Trugschluss in der qualitätsbewussten Softwareentwicklung ist, dass ein abgeschlossenes Projekt keine Wartungsarbeiten mehr erfordert. Gerade in diesem Umfeld ist es wichtig schnell und flexibel auf Änderungen zu reagieren, um die negativen Auswirkungen zu reduzieren. Hierfür sind eine gute Softwarearchitektur und ein modularer Aufbau wichtig, damit die Reaktionszeit des Entwicklerteams gering gehalten wird. In der Regel sind Projekte mit dieser Softwareentwicklungsmethode umfangreicher und dauern meist etwas länger als Projekte mit klassischen Methoden. Über einen Software-Lebenszyklus gesehen relativiert sich jedoch dieser zeitliche Nachteil durch die eben erwähnten Vorteile bei Wartungs- und Änderungsarbeiten, weshalb vor allem bei sog. Early-Bird-Entwicklungen noch immer die klassischen oder agilen Varianten mehr Vorteile haben.

Um diese Entwicklung in der Technologie des Software Engineering weiter voran zu treiben, werden Techniken wie Knowledge Bases, Projektdatenbanken und Metriken verwendet, die dem Projektteam helfen sollen den Softwareentwicklungsprozess zu

optimieren. In welchem Rahmen dies passieren kann, wird im nächsten Abschnitt dargestellt.

4.2 Thematische Einordnung der Forschungsfrage

Die Forschungsfrage „*Wie können Ontologien und Wissensdatenbanken die Arbeit der Akteure im Softwareentwicklungsprozess erleichtern und helfen Probleme und Missverständnisse zu reduzieren?“*, die in der Einleitung gestellt wurde, wird in diesem Abschnitt näher erläutert. Hierzu sollte zunächst jeweils ein Teilaspekt der Forschungsfrage für sich betrachtet werden.

Einen guten Einblick in diese Thematik bietet auch das Werk *Spielräume* [62] von Tom DeMarco, das eine Einschätzung des Projektmanagement-Wesens im Hinblick auf Qualität und Führungsverantwortung bietet. Vor allem der Abschnitt „Risiko und Risikomanagement“ zeigt Synergien zu der hier gestellten Forschungsfrage.

4.2.1 Wie können Ontologien und Wissensdatenbanken die Arbeit der Akteure im Softwareentwicklungsprozess erleichtern?

Es ist in der Fachwelt unbestritten, dass zusätzliche Werkzeuge sinnvoll eingesetzt die tägliche Projektarbeit erleichtern und verbessern können. Der Einsatz von Hilfsmitteln, basierend auf Ontologien und Knowledge Bases im Bereich der Softwareentwicklung ist jedoch gering, weshalb es hier nur sehr wenige Erfahrungswerte gibt. Zumeist werden von den IT-Unternehmen bzw. IT-Abteilungen in Betrieben Werkzeuge verwendet, die bereits besser bekannt und sich als Standard herauskristallisiert haben. Hierzu zählen Recommender-Systeme und die in Kapitel 3 erläuterten Projekt- und Ticket- & Issue-Management Systeme.

Über Knowledge Bases und Ontologien für sich gesehen gibt es allerdings bereits – wie in Kapitel 2.1 gezeigt – einige Forschungsbeiträge und Erfahrungswerte, so dass diese schon lange hätten mit den herkörmlichen Werkzeugen kombiniert werden können. Der Einsatz von Unbekanntem gilt allgemeinhin immer als schwieriger, da ein Scheitern fatale Auswirkungen haben könnte. Dies thematisiert auch Tom DeMarco in [62] ab Seite 164.

Generell gilt aber, um die Arbeit der Akteure zu erleichtern gibt es bereits sehr viele nützliche Werkzeuge, die in der Fachwelt anerkannt sind. Wenn man diese Werkzeuge mit Entscheidungshilfen und Visualisierungen mittels Wissensaufnahme und Wissensverarbeitung – wie es Knowledge Bases und Ontologien zur Verfügung stellen – weiterentwickelt, kann man zum Einen die Vorteile der bisherigen Tools weiterhin verwenden und zugleich das gesammelte Wissen so einsetzen, dass diese

Tools noch besser werden. In Kapitel 6.2 und den folgenden Abschnitten werden diese und weitere Ansätze noch konkreter erläutert und die Antwort auf das „Wie“ gegeben.

4.2.2 Wie können Ontologien und Wissensdatenbanken helfen Probleme und Missverständnisse zu reduzieren?

Die Qualität in der Softwareentwicklung kann zusätzlich verringert werden, wenn die einzelnen Teammitglieder und die Stakeholder nicht eine Sprache sprechen. Missverständnisse und Mehraufwand sind oft ein Resultat schlechter Kommunikation oder falsch verstandener Schlagwörter und Fachbegriffe. Um hier entgegen zu steuern bedarf es Synonymen in den beteiligten Fachwelten, denen sich alle mitarbeitenden Personen bedienen können. Mit aktuellen Hilfswerkzeugen lässt sich dies nicht immer garantieren.

Diesen Ansatz verfolgt das Forschungsprojekt QuASE mit seinem Proof-of-Concept. Aus dem bisherigen Wissen der einzelnen Personengruppen werden ebensolche Schlagwörter und Fachbegriffe extrahiert und in speziellen Wissensdatenbanken gespeichert. In Kapitel 5 werden die Techniken und Metriken dargestellt und erläutert, die einer Knowlege Base zugrunde liegen, um gespeichertes Wissen in den benötigten Kontext zu bringen.

Der nächste Abschnitt beschäftigt sich ebenfalls intensiver mit dem Forschungsprojekt, dessen Zielen und dem Aufbau der Applikation. Auch hier werden bereits einige wichtige Technologien und Methoden zur Verbesserung des Softwareentwicklungsprozesses erläutert.

4.3 Das Projekt QuASE

Zunächst wird in diesem Abschnitt ein Überblick über das Forschungsprojekt gegeben. Anschließend werden die verwendete Architektur und die Systemkomponenten näher erläutert.

4.3.1 Überblick über das Projekt

Das Forschungsprojekt *Quality-Aware Software Engineering* (QuASE) der Forschungsgruppe Application Engineering an der Universität Klagenfurt beschäftigt sich mit der der qualitätsbewussten Softwareentwicklung, also der Verbesserung der Qualität in bestehenden und zukünftigen Softwareprojekten. Das Projekt wurde im März 2013 begonnen und endete mit einem Proof-of-Concept im Februar 2015. Das Projektteam bestand in dieser Entwicklungsphase aus zwei Forschern, die hauptamtlich an der Universität angestellt sind, und vier Softwareentwicklern, die als studentische Projektmitarbeiter hauptsächlich für die Entwicklung der Applikation verantwortlich waren. Zudem wurde das Projekt durch vier strategische Partner aus der Wirtschaft, die zum Teil IT-Unternehmen und Unternehmen mit IT-Abteilungen sind, u. a. mit Knowledge Support gefördert.

Der Fokus in QuASE richtet sich auf Entscheidungshilfen und Wissenstransfer einzelner Akteure und Berufsgruppen innerhalb eines Projektteams. Dies geschieht mit Hilfe einer sogenannten Wissensdatenbank und Kontext-Modellen, die die Grundlage für das bereitgestellte System darstellen (vgl. [10, 14]). Abbildung 4-1 zeigt einen Überblick über das System und wie es eingesetzt wird.

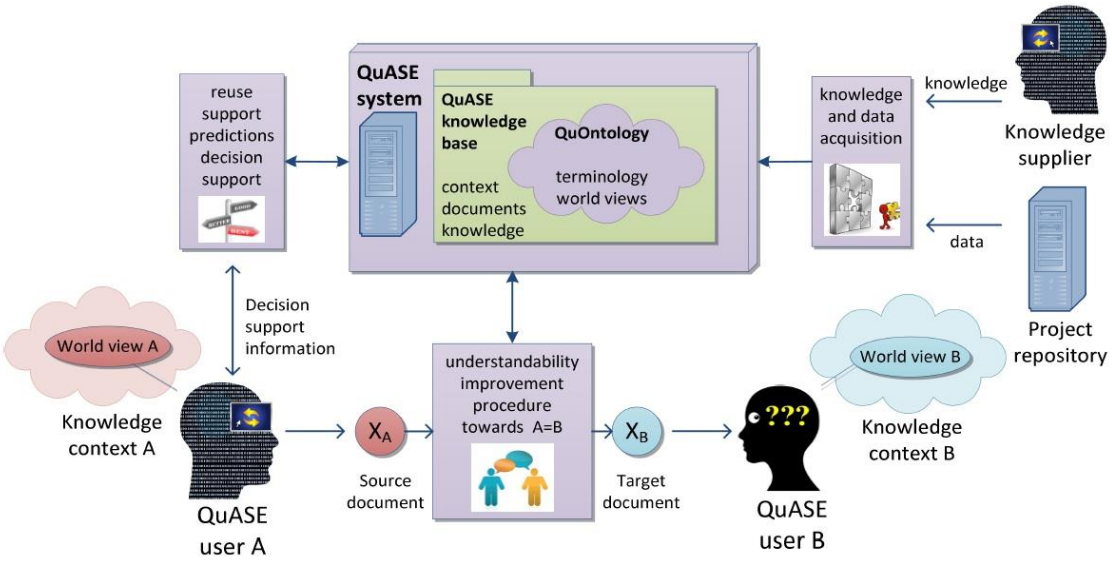

Abbildung 4-1: Overview of QuASE (aus [9, 65])

Kernbestandteile der QuASE-Applikation sind dabei die *Knowledge Base* und die Ontologien *QuOntology* und *QuASE site ontology*. Um diese beiden Komponenten wurde die Software mit verschiedenen APIs und Systemkomponenten erweitert, die sicherstellen sollen, dass die Applikation für seine Einsatzzwecke verwendbar ist. Ein weiterer nicht unerheblicher Teil des Systems sind die externen Daten, die es dem System ermöglichen sollen den jeweiligen *world view* aus einer anderen Perspektive, z. B. eines bzw. einer anderen Projektbeteiligten, zu übersetzen oder zu erklären. Hierzu sind verschiedene Daten notwendig, die in speziellen Datenbanken oder Dokumenten vorliegen.

Zunächst werden die allgemeinen Projektziele und Grundlagen zu Understandability Management, Decision Making Support und Knowledge-oriented Access Interface gegeben. Diese wurden aus [9], der Informationsseite[25] zum QuASE-Projekt, entnommen. Im darauf folgenden Abschnitt werden dann die verwendeten Technologien zur Softwareentwicklung beschrieben und anschließend die Architektur und der Aufbau dieses Software-Tools näher erläutert.

Projektziele

Für das QuASE-Projekt wurden folgende Projektziele formuliert:

- Definition einer theoretischen Basis; Ausarbeitung von Entwicklungsverfahren; Proof-of-Concept Tool Support für
  - G-1

    Erlangung und Formalisierung von Domänenwissen für Qualitätsaufgaben innerhalb eines Softwareentwicklungsprozesses (SEP) zwischen verschiedenen Akteuren und Beteiligten
  - G-2
    - Sammeln der Rohinformationen über qualitätsbezogene Aufgaben in einem SEP der verschiedenen Beteiligten
    - Konvertierung in operationales Wissen
    - Verwendung von verfügbarem Domänenwissen zur Gewährleistung der Korrektheit einer Konversation

[25] Full information about QuASE: http://quase-ainf.aau.at/

- G-3

  Verwendung des gesammelten Wissens im SEP zur Erstellung einer qualitätsbezogenen Kommunikationsbasis für verschiedene Projektbeteiligte im SEP, speziell für Softwareentwickler und Business-Akteuren

- G-4

  Unterstützung von getroffenen Entscheidungen im SEP, Wiederverwendbarkeit von qualitätsbezogenen Erfahrungen und Vorhersage des zukünftigen qualitätsbezogenen Verhaltens der involvierten Beteiligten, basierend auf dem gesammelten Wissen

- Proof-of-Concept Tool soll einfach integrierbar in einen existierenden SEP der Unternehmen sein (G-5).

(vgl. [9])

Diese Projektziele konnten im Februar 2015 als erreicht erklärt und mit der Vorbereitung zur Installation der Applikation bei den Partner-Unternehmen abgeschlossen werden. Natürlich ließen sich weitere Konzepte und Sekundärziele in die Software integrieren, oder die vorhandenen Ergebnisse weiter aufbereiten, allerdings ist das kein Teil eines Forschungsprojekts, sondern einer Weiterführung.

Understandability Management

Um das Projektziel G-3 zu erreichen unterstützt die QuASE-Applikation die Verwaltung zur Verständlichkeit der Informationen aus einer qualitätsbezogenen Kommunikation in einem Softwareentwicklungsprozess (im weiteren auch kommunizierte Information genannt) durch die Veranschaulichung der Tätigkeiten hinsichtlich Verständlichkeitsbewertung und -verbesserung. Diese Aktivitäten sollen durch Anpassung aktueller und historischer Daten auf die Sichten der verschiedenen Beteiligten unter Berücksichtigung bestehender Erfahrung in qualitätsbezogener Kommunikation unterstützt werden (vgl. [9]).

Ein Beispiel für diese Art von qualitätsbezogener Kommunikation kann ein Austausch von E-Mails über Testergebnisse oder ein Änderungsauftrag, bzw. Change-Request einer bestehenden Software zwischen einem Softwareentwickler bzw. einer -entwicklerin und einem Projektmanager bzw. einer Projektmanagerin oder KundInnen sein. Auch sog. Tickets, also die Meldung eines Problems bei Verwendung einer Software, werden als solche qualitätsbezogene Kommunikation angesehen. Abbildung 4-2 zeigt, wie Understandability Assessment, ein Teilgebiet des Understandability Management in QuASE aussieht. Im Anschluss wird noch ein Algorithmus zur Understandability Verbesserung gezeigt.

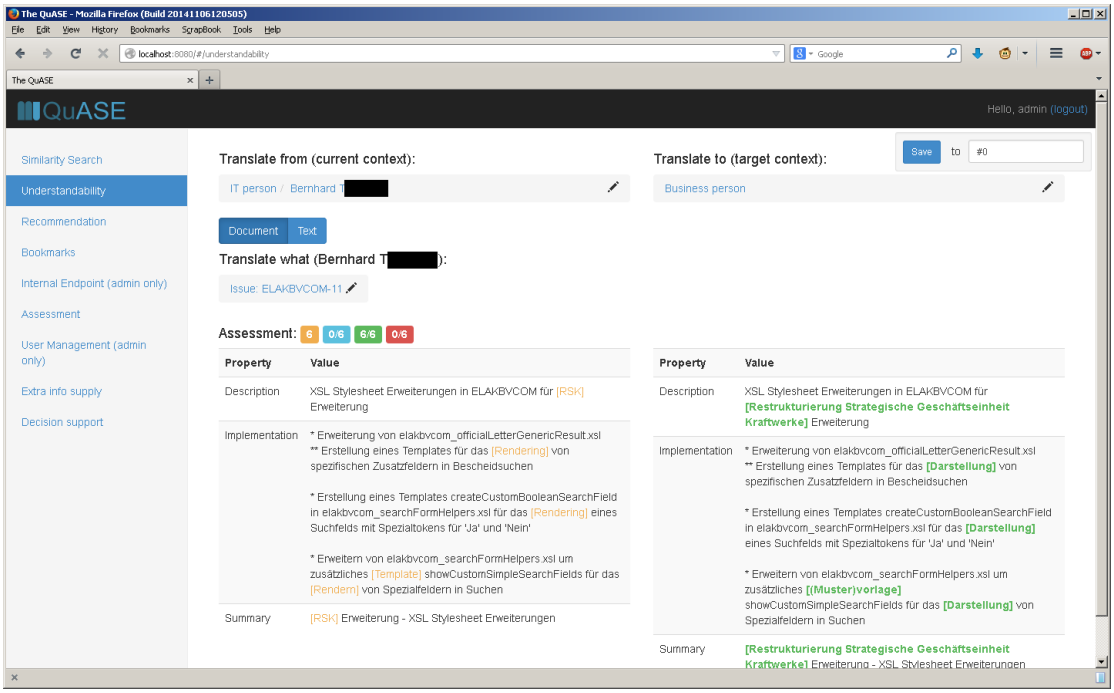

Abbildung 4-2: Beispiel für Understandability Assessment in QuASE

Um die Verbesserung der Understandability durch die Übersetzung der Terminologien durchzuführen, ist es notwendig ein Umschalten zwischen den Kontexten der Knowledge, die zu den verstandenen Termen gehören, zu implementieren; das erlaubt eine Übersetzung zwischen World Views von verschiedenen kommunizierenden Personen und –gruppen. Die Prozedur der Understandability Verbesserung im Fall eines Übersetzungseinführungskonflikts wird in zwei Ebenen durchgeführt (siehe Abbildung 4-3):

1 Auflösung (durch Anwendung des *ontological reasoner*) der korrespondierenden Konzepte, die in den Context Ontologies definiert sind, in die generischen Konzepte, die in der QuOntology core oder den Domain Ontologies definert sind;

2 Wechseln zum Ziel-Kontext, Nachschlagen der Konzepte im Ziel-Kontext, die mit den generischen Konzepten korrespondieren; diese Ziel-Konzepte formen die Resultate der Übersetzung.

(vgl. [68])

Dabei sind sowohl die Context und Domain Ontologies, als auch die QuOntology core ein Teil der in Kapitel 5.2.2 beschriebenen QuOntology (siehe auch Abbildung 5-6).

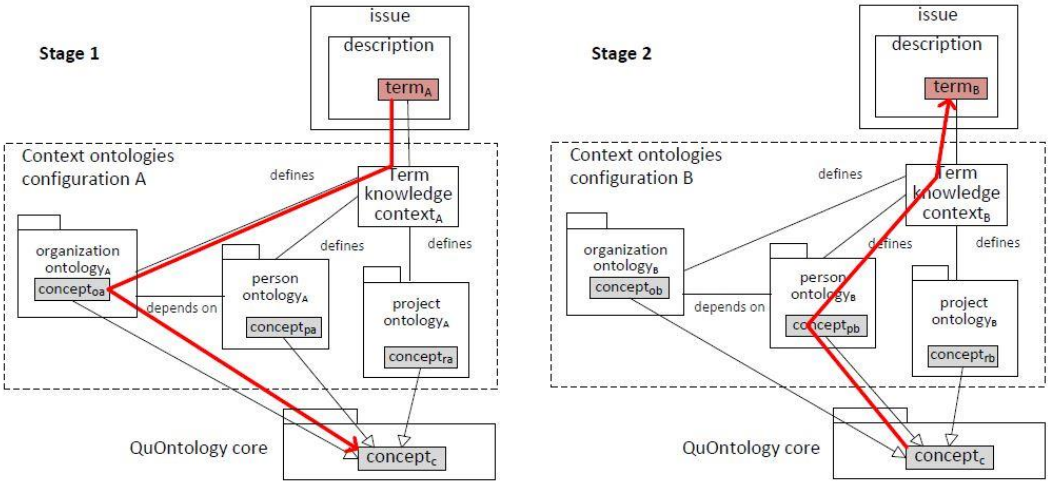

Abbildung 4-3: Understandability Verbesserung im Fall von Terminologie-Konflikten (aus [68])

Decision Making Support

Das Projektziel G-4 umfasst die Verbesserung der Qualität von Entscheidungen basierend auf kommunizierter Information durch die QuASE-Applikation. Diese Charakteristik soll im speziellen durch folgende Punkte erreicht werden:

- durch Ausgabe von Empfehlungen für die Gestaltung der Durchführung qualitätsbezogener Informationen basierend auf der Analyse der bisherigen Erfahrungen auf die Ordnung solcher Informationen;
- es soll auch diese Entscheidungen durch Prognosen von Kommunikationsparametern und zukünftigen Ergebnissen unterstützen.

Ziel dieser Decision-Support-Aktivitäten ist es, die Erkenntnis der Beteiligten vom Kontext der Kommunikation und die möglichen Handlungen vor und während der Kommunikation zu steigern und den Aufwand zu einer (richtigen) Entscheidung zu kommen, bzgl. der Implementierung der Informationen, zu reduzieren (vgl. [9]).

Man kann beispielsweise durch Wiederverwendung von Wissen oder Analyse diverser Metriken mögliche Entscheidungen beeinflussen. Die folgende Abbildung zeigt, wie das sog. Knowledge Reuse in der Applikation von QuASE aussieht.

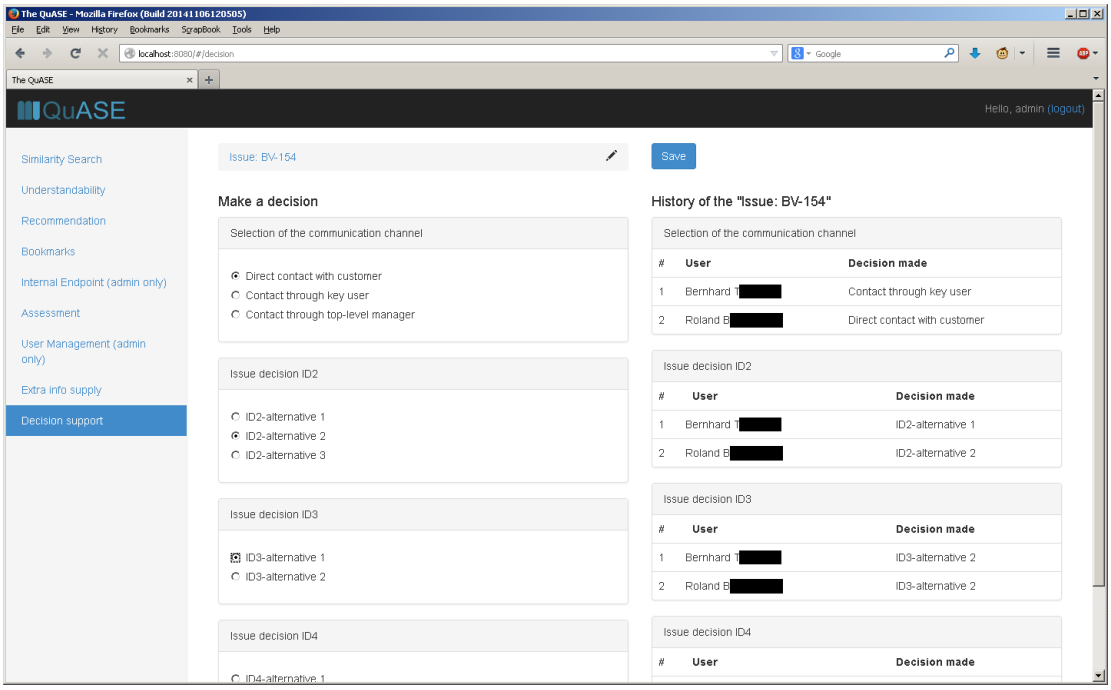

Abbildung 4-4: Beispiel für Knowledge Reuse in der QuASE-Applikation

Knowledge-oriented Access Interface

Verständlichkeit und Qualität von Entscheidungen sollen basierend auf der wissensorientierten Zugriffsschnittstelle zu kommunikationsbezogenen Daten (kommunizierte Informationen) implementiert werden. Diese Daten werden aus gewerblichen Softwareentwicklungsprojekten gesammelt und sollen als Teil der QuASE-Applikation implementiert werden. Diese Schnittstelle kann folgendermaßen veranschaulicht werden:

- Kommunikationsbezogene Daten in gewerblichen Softwareentwicklungsprojekten werden in Projekt-Datenbanken gehalten und durch Issue-Managementsysteme (IMS), wie z. B. JIRA, kontrolliert. Um eine wissensorientierte Zugriffsschnittstelle zu implementieren, wird vorgeschlagen, einen Zugang zu den Daten in solchen Datenbanken durch eine Ontologie (z. B. QuOntology) zu ermöglichen, die sich auf das Erfassen der Konzepte im Bereich der qualitätsbezogenen Kommunikation zwischen den Beteiligten im SEP beziehen; diese Ontologie soll nicht nur die generischen Domain-Konzepte, sondern auch die Konzepte enthalten, die spezifisch für die jeweiligen Kontexte der Softwareentwicklung sind.

- Die bereitgestellte Schnittstelle umfasst eine Reihe von Konzepten und deren Attribute, die sich an der Struktur der verfügbaren Daten in den Projektdatenbanken (sog. primäre Projektinformationen) mit semantischen Annotationen (repräsentiert durch zusätzliche Konzepte), Konzeptattributen und Beziehungen zwischen den Konzepten (die notwendig zur Adressierung über Qualitätsmerkmale sind) orientiert. Insbesondere solche Erläuterungsinformationen können als Einflussfaktoren gelten, die eine Haltung der Beteiligten gegenüber der Software Qualität im SEP aufweisen, und als Ergebnis zu dem Prozess der Entscheidungsfindung im Zuge der Kommunikation zwischen den Beteiligten beitragen.

(vgl. [9])

4.3.2 Technologie in QuASE

Im Wesentlichen stützt sich die verwendete Technologie auf Java und den Konzepten der Unified Foundational Ontologies (UFO), welche als Standard in der Ontologie-Beschreibung angesehen werden kann. Zusammen mit der Ontology Web Language (OWL), dem RDF/RDFS als Schema und der Description Logic (DL) bildet die UFO den Knowledge Base- und Ontologie-Bereich ab. Wichtig für das Forschungsprojekt war es, das Zusammenspiel der doch sehr unterschiedlichen Komponenten auf Basis der Qualitätsverbesserung in Software-Projekten zu erforschen. Die Herausforderung für das Entwicklerteam war, diese unterschiedlichen Komponenten effektiv und effizient in ein Softwarepaket einzubinden, ohne unerwünschte Seiteneffekte zu haben.

Die QuASE-Applikation ist ein webbasiertes System, das in den Programmiersprachen Java – in der zu dieser Zeit aktuellsten Version 8 – für die Basis-APIs und JavaScript für die UI geschrieben wurde. Zudem wurden SQL für die Datenbank-Zugriffe und SPARQL für die Zugriffe auf die Ontologie verwendet.

Außerdem wurden die im Kapitel 3 beschriebenen Frameworks für Zugriffe außerhalb der regulären Java-APIs mit deren verschiedenen Funktionalitäten verwendet. Die meisten dieser APIs und Frameworks werden in der Anwendung u. a. für Berechnungen und zur Durchführung bestimmter Algorithmen verwendet. Apache Jena wird für die Anbindung der Knowledge Base, der QuASE site ontology und der QuOntology mit Hilfe einer *Trivial-Database* an den Java Code verwendet, Shiro ist Grundlage für Login und Berechtigungen in der QuASE-Applikation und Mahout ist für das selbständige Lernen der Ontologien zuständig. Die Java-APIs und Plug-Ins sind unterstützend für die Programm-Logik der QuASE-Anwendung eingebunden.

Die Modellierungswerkzeuge ADOxx und Protégé wurden zur Erstellung der Ontologien und Metamodelle verwendet. ADOxx soll auf der Server-Seite zum Upload neuer Metamodelle bzw. Änderungen in diesen weiterhin zur Verfügung stehen, wohingegen Protégé nach der Fertigstellung der QuASE-Applikation keine Relevanz für die Änderungen haben sollte. Dies wurde deshalb von den Projektbeteiligten entschieden, da es zur Verwendung von Protégé notwendig ist detaillierte Fachkenntnisse in Ontologien und deren Erstellung zu haben und ein für AnwenderInnen ohne Expertenwissen einfach zu handhabendes Plug-In nicht gegeben ist.

In einer möglichen zweiten Projektphase sollte evaluiert werden, ob es möglich ist, auch ohne ADOxx die kontext-spezifischen Modelle in das System zu integrieren. Hierfür soll geprüft werden, ob es mögliche lizenzfreie Software-Tools gibt, mit denen ADOxx ersetzt werden kann. Das wird in weiterer Folge notwendig sein, da ein geplanter kommerzieller Einsatz ohne eine Modularisierung und Ersetzung von einfacheren Werkzeugen nicht möglich sein würde.

Weitere Änderungen in der Programmierung und den eingesetzten Werkzeugen bedürfen allerdings eine detaillierte Planung und Expertenwissen im jeweiligen Software-Teil. So ist es auch denkbar, dass der Weboberfläche vor einem größeren Einsatz des Systems in anderen Bereichen einem sog. „Facelift“ unterzogen wird. Das bedeutet, dass die Oberfläche ein neues Design erhält, ohne die Inhalte zu verändern. Zudem könnten, wie in einem Softwareentwicklungsprozess üblich, einzelne Module im Zuge neuer wissenschaftlicher Erkenntnisse oder mit neuen Richtlinien zur Softwareentwicklung verändert werden.

Detailliertere Beschreibungen dieser Veränderungen sind auch im Kapitel 5.4 beschrieben. Zunächst wird jedoch noch auf die genauen Software-Teile mit der System-Architektur im nächsten Abschnitt und der Ontologie und Knowledge Base in Kapitel 5 eingegangen.

4.3.3 QuASE-Architektur

Die QuASE-Applikation beinhaltet im Wesentlichen fünf Kernkomponenten, um den Benutzern die Möglichkeit der Interaktion zu den verschiedenen Aufgabenbereichen der Software bieten zu können. Diese Komponenten sind:

1. QuASE site modeling tool
   Ein auf Meta-Modellen basierte Komponente, aufbauend auf ADOxx (welches in Kapitel 3.2.1 beschrieben wurde), die eine graphische Domain Specific Language (DSL) zur Beschreibung der Site-spezifischen Kommunikationsumgebung (Kommunikationskontext, Inhaber der kommunizierten Informationen, kommuniziertes Wissen) und das Mapping zwischen dem Modell und dem speziellen Projekt Repository (z.B. Jira DB) bietet;
2. QuASE ontology builder utility
   Ein Konverter zum transformieren der Modelle im QuASE DSL in eine OWL2-Repräsentation der QuASE site ontology;
3. QuASE Knowledge Base Builder Utility
   Die sog. Knowledge Acquisition Engine konvertiert die Daten aus den Projekt Repositorys in Individuals entsprechend den bestehenden Ontologien basierend auf der mit Hilfe des *QuASE site modeling tool*s beschriebenen Mappings;
4. QuASE terminology editor
   Das interaktive webbasierte Tool erfasst die Informationen im Zusammenhang mit dem kommunizierten Wissen von den WissenslieferantInnen[26] und konvertiert diese Informationen in die QuOntology-basierende Repräsentation;
5. QuASE Kernmodul
   Das Kernmodul, z. T. auch QuASE tool genannt, ist eine interaktive webbasierte Komponente, die u.a. die EndbenutzerInnen-Support Szenarien, sowie die Übernahme der zusätzlichen Informationen der WissenslieferantInnen, die nicht 1:1 in den Projekt Repositorys zu finden sind bietet (z.B. Benutzer-spezifische Attribut-Werte oder Informationen über Entscheidungen die mit Elementen aus der Kommunikationsumgebung zusammenhängen).

   (vgl. [64])

[26] Anm. d. Autors: Als WissenslieferantInnen werden Personen genannt, die Wissen in irgendeiner Form (kommuniziert, etc.) in das Projekt einbringen.

Die Applikation unterteilt sich in eine server-seitige und eine client-seitige Anwendung, dabei werden intern drei Schichten verwendet. Im server-seitigen Teil befinden sich die Datenschicht und die Schicht für die Applikationslogik. In dieser werden Algorithmen und Berechnungen, sowie die Zugriffe auf Knowledge Base und Ontologien durchgeführt. Die Client-Anwendung beschränkt sich auf die Darstellung der Inhalte auf der Benutzeroberfläche (engl. User Interface, UI).

Bei der Architektur der einzelnen Schichten in der QuASE-Applikation handelt es sich im Wesentlichen um APIs, die die Funktionalität der einzelnen Module bereitstellen. Dazu gehören die Ontologien QuOntology und QuASE site ontology, deren Aufbau und Funktionen in Kapitel 5 näher erläutert werden, der Data Access Layer (DAL), die Core-API, die die Aufrufe des Clients auf der Server-Anwendung steuert, und die UI als Weboberfläche. Außerdem wurde für die Datenhaltung eine MySQL-Datenbank verwendet. Die Abbildungen 4-5 und 4-6 geben einen schemenhaften Überblick über die Architektur der Software.

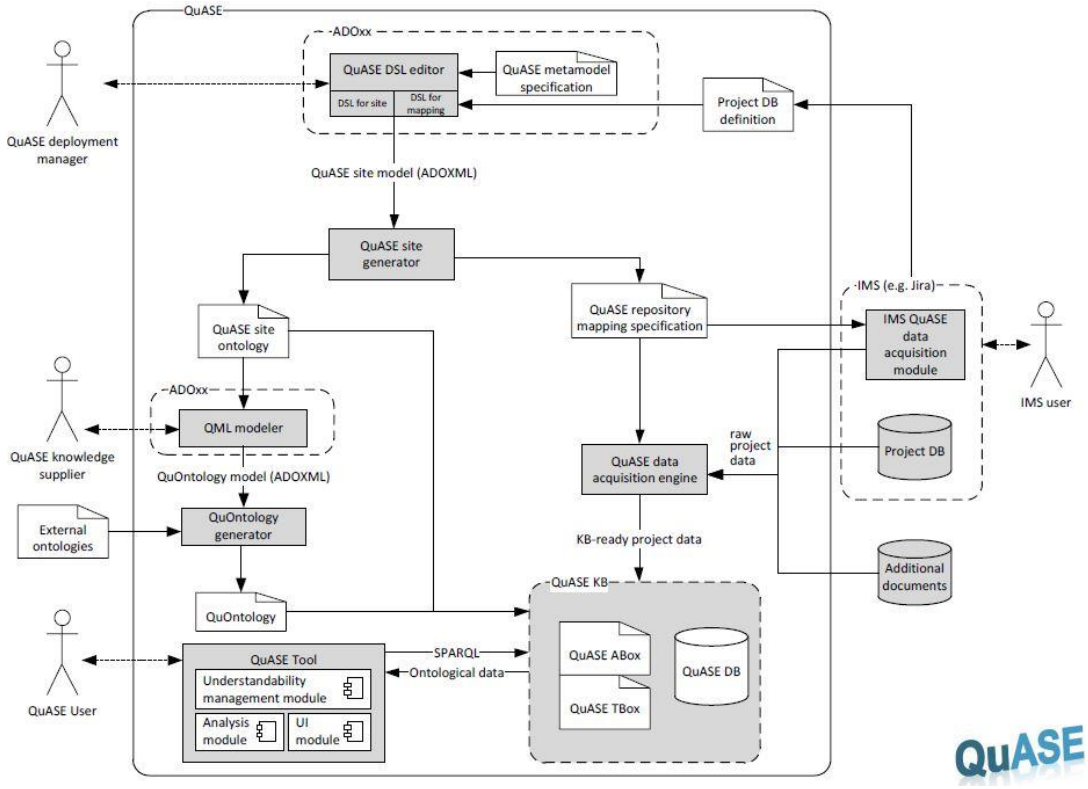

Abbildung 4-5: QuASE Architektur (vgl. [65])

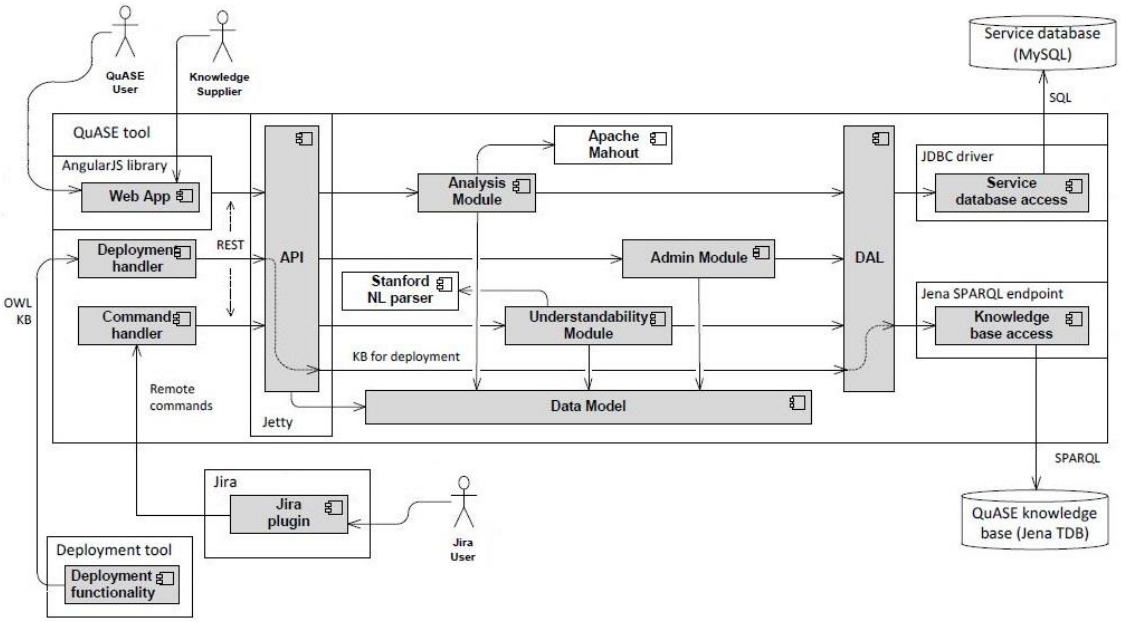

Abbildung 4-6: QuASE Tool Architecture (vgl. [64])

Um eine klare Struktur in der Software zu erhalten, wurden von den Entwicklern einzelne Pakete und Unterpakete erstellt, die dann in der Gesamtheit das Backend der Applikation darstellen. Diese Pakete wurden auf Basis ihrer Funktion innerhalb der Applikation definiert und benannt. Hierbei wurde auch auf die Client-Server-Struktur geachtet. *understandability*, *model*, *dal*, *app* und *application* sind die Software-Pakete (engl. packages), die innerhalb der gesamten Applikation zusammenwirken. *app* ist der Arbeitsname für die server-seitige *Core-API* und *application* bezeichnet hier alle Code-Fragmente innerhalb der Client-Anwendung zur Steuerung der Benutzeroberfläche. Teile des Source Codes sind im Anhang verfügbar. Für weitere Informationen sollten die Entwickler bzw. Projekt-Owner direkt angefragt werden.

Neben den bereits genannten Paketen existiert im Entwicklungsstadium ein weiteres Paket namens *quontology* zur Definition und Erstellung der Metamodelle und Ontologien in der Server-Anwendung. Der Teil im Proof-of-Concept und dieser Arbeit, der ebenfalls als QuOntology bezeichnet wird, wurde mit diesem Paket zur Applikation hinzugefügt. Dies wird im Detail im nächsten Kapitel erläutert.

QuASE-Core

Einer der Hauptbestandteile ist die Schicht der Applikationslogik, die auch als Server-Anwendung der QuASE-Applikation bezeichnet wird. Sie beinhaltet verschiedene Schnittstellen und Bibliotheken, die zum einen auf Datenbanken, Knowledge Base und Ontologien zugreifen, und zum anderen die Informationen aus diesen für die Ausgabe auf der Benutzeroberfläche vorbereiten. So sind *Security*, *Common*, *Core*, *Data* und *Management* die Unterpakete der Core-API.

Der Kern der Applikation wurde hauptsächlich in Java implementiert und dabei zum Teil auf bereits bestehende APIs und Frameworks zurückgegriffen, die dann auf die jeweiligen Bedingungen angepasst wurden. Im Anhang stehen die jeweiligen Hauptklassen der serverseitigen Applikation und der clientseitigen Web-Applikation als Quellcode zur Verfügung.

Die Benutzeroberfläche

Die Benutzeroberfläche wird als Client-Anwendung der QuASE-Applikation bezeichnet. Hier werden die Informationen, die server-seitig berechnet und zur Wiedergabe vorbereitet wurden, dargestellt. Die Abbildung 4-7 zeigt den Startbildschirm der Applikation. Der zur Abbildung korrespondierende Quellcode befindet sich, wie die beiden Hauptklassen von Frontend und Backend, in Anhang C.

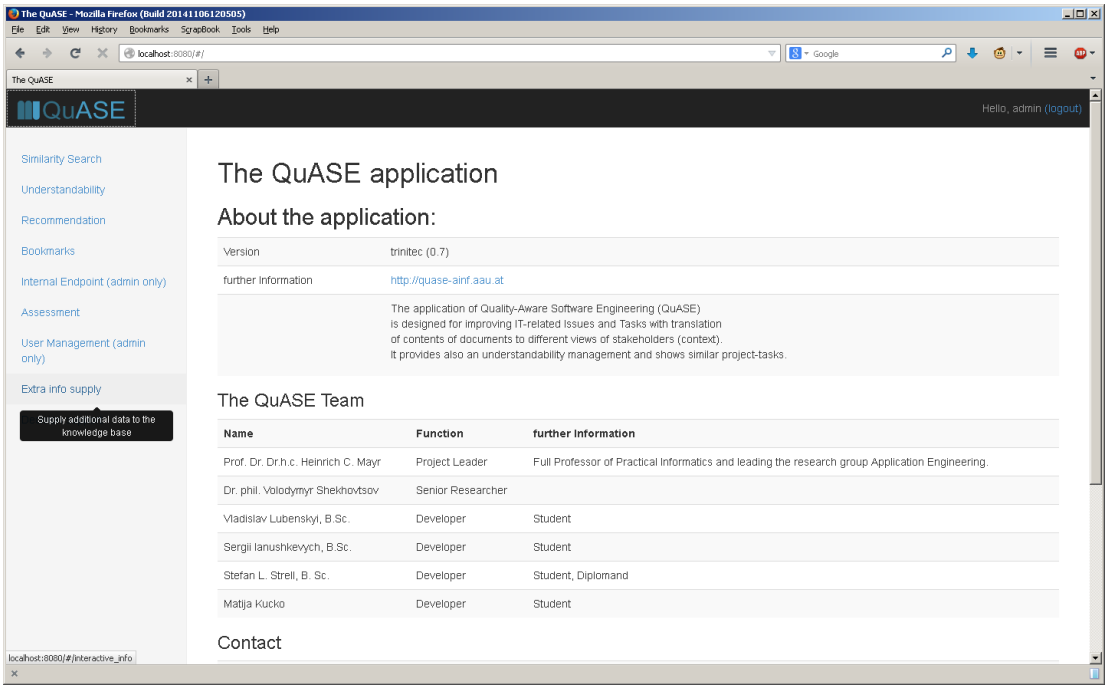

Abbildung 4-7: Die QuASE-Weboberfläche

Das Web-Interface wurde hauptsächlich in JavaScript und mit Hilfe von Cascading Stylesheets (CSS) implementiert. Außerdem wurden die in Kapitel 3.3 erwähnten APIs und Frameworks zur korrekten Darstellung und zur Weitergabe von Informationen zu Funktionalitäten an die Core-API eingebunden. So wird beispielsweise ein Klick auf den „Save to New Bookmark“-Button über ein AngularJS-Modul über das zuständige DAO-Objekt an die Datenbank im Hintergrund der Software weitergeleitet. Um den Inhalt oder Teile des Inhalts eines Tickets oder Issues zu übersetzen oder zu erklären muss dieser wie in Abbildung 4-8 gezeigt ausgewählt werden. Die Auswahl ist wie die „Save to Bookmark“-Funktionalität über das entsprechende

AngularJS-Modul und das DAO-Objekt mit der Applikationslogik auf der Server-Seite verbunden.

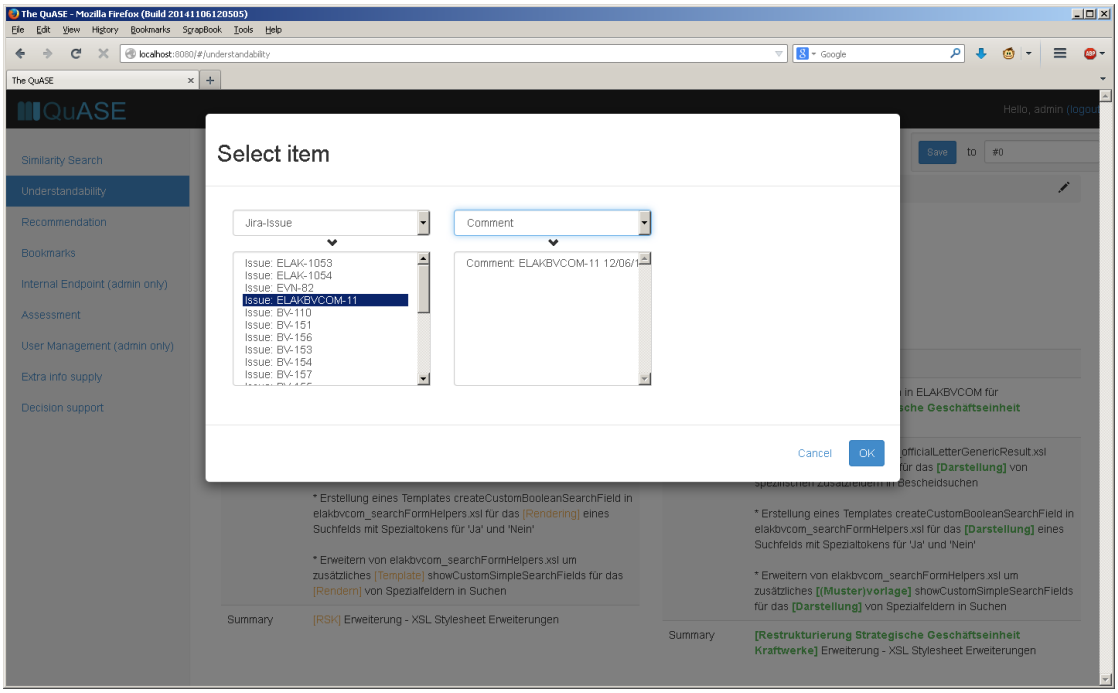

Abbildung 4-8: Dokumenten-Auswahlfeld für Translation/Explanation

Datenbanken & Datenbankmanagementsysteme

Dieser Abschnitt beschäftigt sich zum einen mit der Integration der Datenbanken aus den verschiedenen Ticketing- und Issue-Management-Systemen (TIMS), die im zweiten Kapitel näher beschrieben wurden. Die QuASE-Applikation wurde zudem mit einer MySQL-Datenbank ausgestattet, die die Metriken, Bookmarks, die spezifischen User der Applikation und die benutzerspezifischen Einstellungen beinhaltet. Der Zugriff auf dieses relationale Datenbank-Management-System (RDBMS) erfolgt über den DAL, der zur Applikationslogik gehört.

Die dritte und unterste Schicht der Applikation bildet die Datenschicht. In dieser Schicht sind die sog. Trivial Database (TDB), eine auf Apache Jena basierender Triple-Store und das RDBMS. Die TDB wird dabei als Speicher für RDF und SPARQL-Queries verwendet und bildet zusammen mit dem Apache Jena Plug-In die Zugriffsebene auf die Knowledge Base.

Die Anbindung der TIMS wird über die Site-Modelle aus ADOxx an die Applikation durchgeführt. Da jedes dieser Systeme eine eigene Datenstruktur mit sich bringt, die zum Teil für BenutzerInnen und EntwicklerInnen außerhalb der Anbieter nicht einsehbar sind, muss sowohl für jedes Unternehmen bzw. Organisation, als auch für die einzelnen TIMS ein spezielles Modell erstellt werden. Diese Site-Modelle fungieren dabei als Schnittstelle zwischen der QuASE site ontology und den TIMS-Datenbanken.

Dabei handelt es sich aber nicht um einen physischen Import, sondern um die Einbindung von Referenzen auf die einzelnen Daten. Da die Systeme wie eingangs erwähnt sehr unterschiedlich und auf unterschiedliche Architekturen und Technologien aufgebaut sind, kann nicht durch einfaches Kopieren die Datenstruktur übernommen werden. Die QuASE Ontologien greifen auf die Daten innerhalb dieser DBs über Jena lesend zu, ohne dabei irgendwelche Daten zu verändern. Wie im Abschnitt 4.1 dargestellt, werden diese Daten u. a. für Understandability Management, Decision Support und Knowledge-oriented Access verwendet.

Modelle und Metamodelle in QuASE

Zudem wurden für die Nutzung der QuASE Knowledge Base und der Ontologien Metamodelle und Modelle generiert, die für die weitere Verwendung der Software wichtig sind. Diese Modelle, wie beispielsweise das in Abbildung 4-9 gezeigte Metamodell für QuASE site DSL aus [10, 66], werden hier kurz erklärt. Die Abbildung 4-10 zeigt einen Ausschnitt aus einem konkreten QuASE site model, das auf dem Metamodell basiert.

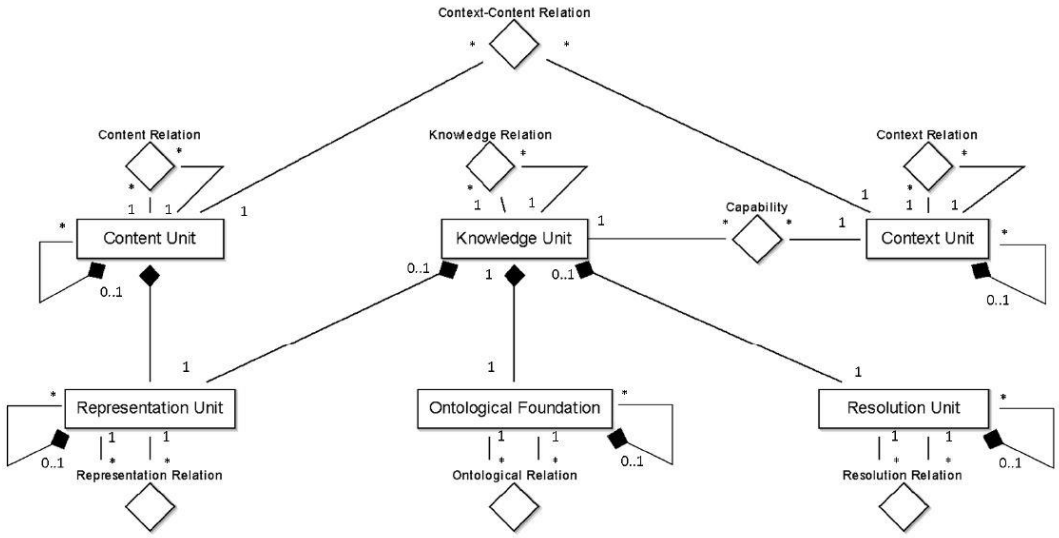

Abbildung 4-9: Metamodel für die QuASE site DSL (aus [10, 66])

Die in QuASE verwendeten Modelle und Metamodelle wurden mit dem Modellierungswerkzeug ADOxx erstellt und in die Struktur der Software, sowie der QuASE KB und deren Ontologien zumeist als XML oder XML-ähnliche Datei eingebunden. ADOxx bietet dafür einen eigenen Export an, der die Modelle im Dateiformat ADOXML zwischenspeichert. Dieses Format ist aus der Sprachfamilie der XML und bietet einige weitere Features.

Die QuASE site DSL wird nach [66] für die Implementierung des QuASE Analysis Support verwendet. Hierbei werden die grundlegenden Konzepte als Metaklassen instanziiert. Separate Entity-Metaklassen wurden zum spezifizieren von Modellierungskonstrukten zu den entsprechenden Context Units und Content Units spezifiziert. Die zugehörigen Metaklassen für *„relation to itself"*, also die Relation auf sich selbst, können für die *Connect Modeling Constructs*, die zur gleichen Metaklasse gehören (z. B. die Instanzen der Content-Unit-Relation Metaklasse können mit den Instantzen der Content Unit verbunden werden), instanziiert werden. Auch Relation-Metaklassen, wie Context-Content-Relation können instanziiert werden um Instanzen von verschiedenen Entity-Metaklassen zu verbinden (vgl. [66]).

Weiterführende Informationen über den Aufbau von Modellen und Metamodellen in QuASE sind u.a im Artikel [66] zu finden.

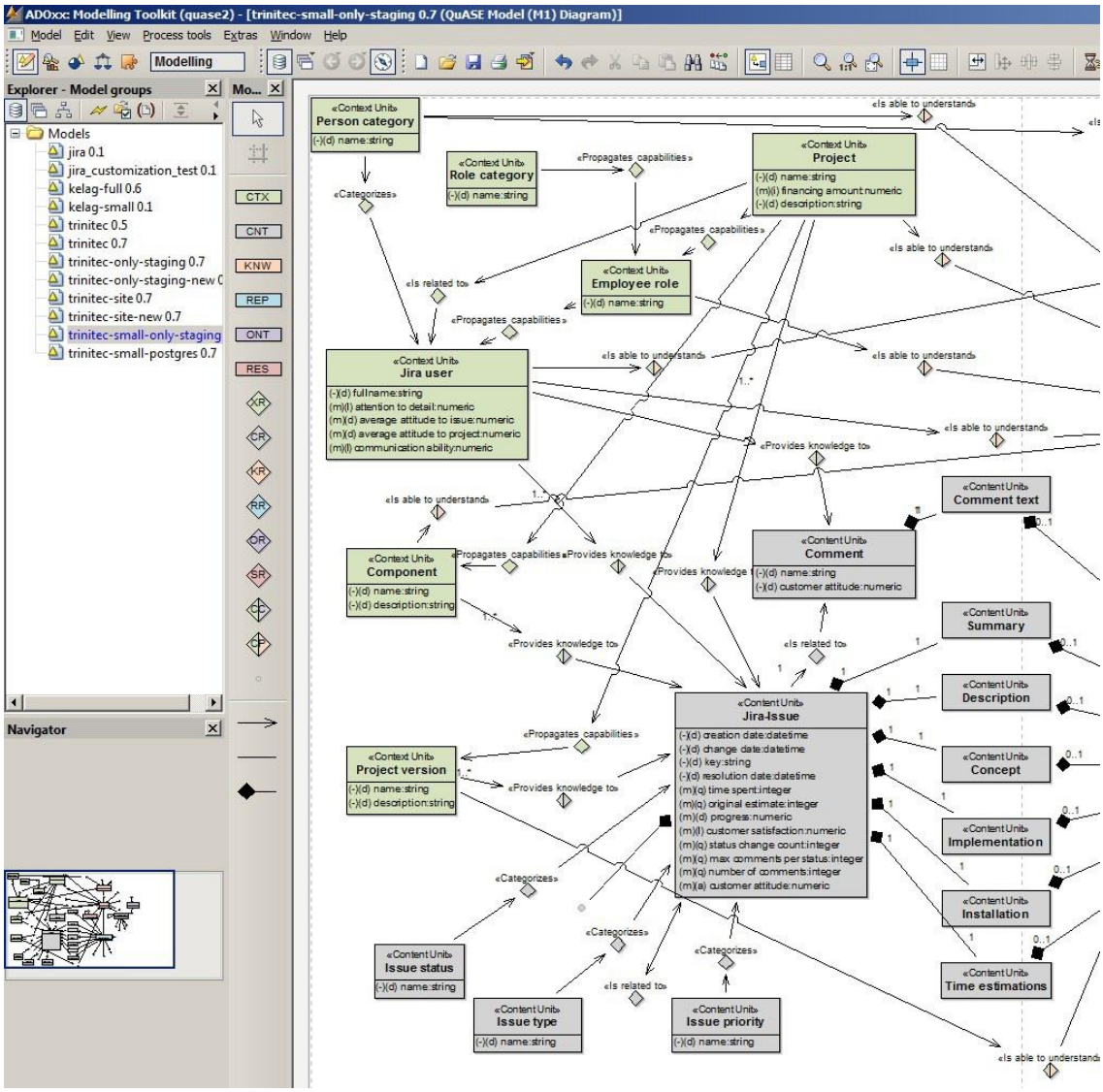

Abbildung 4-10: Ausschnitt aus einem konkreten QuASE site model in ADOxx (aus [66])

5 Ontologien & Knowledge Base in QuASE

In diesem Kapitel wird zunächst die QuASE Knowledge Base (QuASE KB) als Inhaltscontainer der Ontologie veranschaulicht. Anschließend wird näher auf die Umsetzung und Aufbau der Kernbestandteile der QuASE KB, die Ontologien, sowie auf mögliche Erweiterungsszenarien eingegangen.

Eine Knowledge Base, wie die in QuASE verwendete, wird zumeist wie in Kapitel 2 gezeigt zu einem Teil mit Hilfe einer Ontologie aufgebaut. U.a. in Russell und Norvig [69] werden weitere Möglichkeiten zum Aufbau von Knowlede Bases im Rahmen der Artificial Intelligence (AI) gezeigt. Für die Kategorie von KBs, die mit Hilfe von Ontologien aufgebaut werden, gibt es zahlreiche unterschiedliche Anwendungsgebiete, wie z. B. [23, 31]. Die Knowledge Base von QuASE unterscheidet sich von diesen allerdings in einem wesentlichen Merkmal. Im Fall von QuASE werden dafür zwei in ihrem Inhalt unabhängige Ontologien verwendet. Die beiden Ontologien sind die QuOntology, die die Wissensbasis darstellt, und die QuASE site ontology, die eine kontext-spezifische Ontologie darstellt. Die folgende Abbildung zeigt einen systematischen Aufbau der beiden Kernbestandteile dieses Systems. Die QuOntology und die QuASE site ontology werden in Kapitel 5.2 näher erläutert. Im folgenden Abschnitt wird zunächst die Knowledge Base (in der Abbildung das Zusammenwirken beider Teile) näher beleuchtet.

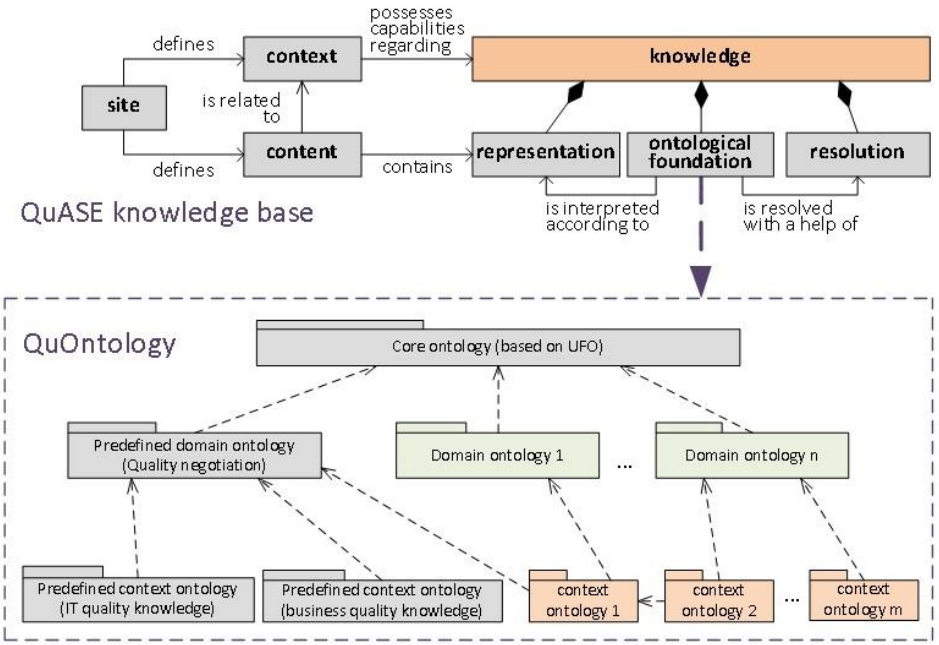

Abbildung 5-1: Konzept der QuASE Knowledge Base, site ontology & QuOntology (aus [65])

5.1 Die QuASE Knowledge Base

In diesem Abschnitt werden zunächst der Aufbau und die Modellierung von Knowledge Bases und im konkreten Fall der QuASE KB dargestellt. Anschließend werden noch Metriken zur Berechnung und Darstellung von Werten gezeigt, um den Überblick über die Knowledge Base abzurunden.

5.1.1 Modellierung und Aufbau der Knowledge Base

Eine Knowledge Base (KB) repräsentiert die Schnittstelle zwischen der Ontologie und der jeweiligen Anwendung, wobei in den meisten Fällen die Ontologie ein Bestandteil der KB ist. Die QuASE Knowledge Base (QuASE KB) unterteilt sich in sog. ABoxen und TBoxen. Eine TBox (terminological box) beinhaltet kontext- und domänenspezifische Komponenten aus der Ontologie, eine ABox (assertional box) hingegen beinhaltet Instanzen und Entitäten dieser Komponenten. Dabei wird die TBox wie eine Klasse in einer objekt-orientierten Programmiersprache verwendet, wohingegen die ABox ähnlich zu Instanzen von Klassen fungieren. Abbildung 5-2 zeigt eine schematische Darstellung der QuASE KB und einen Teil der Einbindung in die Software-Struktur.

Der architektonische Ansatz, der in Abbildung 5-2 gezeigt und in [65] näher beschrieben wird, basiert auf einem transparenten Ontologie-Speicher: die ABox wird dabei von der *QuASE data aquisition engine* als Menge von individuals, korrespondierend zu den definierten Klassen in der TBox, geformt. ABox und TBox zusammen bilden eine OWL-2-Ontologie, welche dann im sog. Triple-Store gespeichert wird (vgl. [65]).

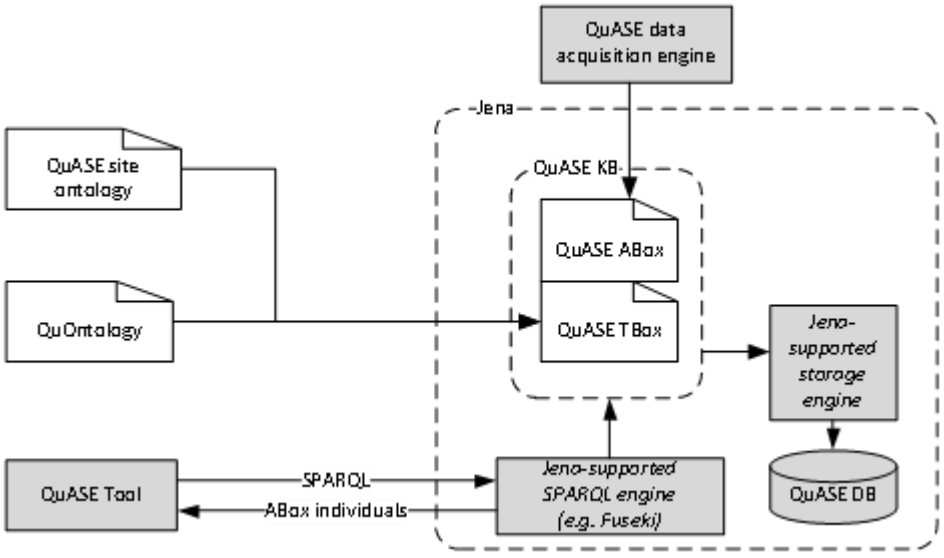

Abbildung 5-2: Schematische Darstellung der Knowledge Base in QuASE (aus [65])

Die QuASE KB wird dabei wie in den Abbildungen 4-5 und 4-9 im vorherigen Kapitel gezeigt durch verschiedene Modelle und Metamodelle beschrieben. Diese bilden in der Gesamtheit die Domain Specific Language (DSL) der Knowledge Base. Zusätzlich wurden Bibliotheken eingebunden, die die QuASE KB lernfähig machen, damit bei längerer Nutzung bestimmte Wörter im Kontext schneller übersetzt oder erklärt werden können. Ein weiterer Teil der QuASE KB sind die im nächsten Abschnitt beschriebenen Ontologien, QuOntology und QuASE site ontology, und die dazugehörige Datenbank, deren Anbindung mit dem Framework Apache Jena realisiert wurde. Die Zusammenhänge zwischen Ontologien und der QuASE KB werden in einem späteren Abschnitt (Kap. 5.3 Integration in das System) etwas näher behandelt.

5.1.2 Metriken in der Knowledge Base

Um Wissen aus einer Knowledge Base für computerbasierte Systeme verwendbar zu machen, benötigt man Metriken. Diese Metriken werden verwendet um dem User konkrete Ergebnisse und Vergleiche anbieten zu können. Aus diesem Grund und damit Entscheidungshilfen generiert werden können, werden die Inhalte der QuASE KB mit Metriken in der Applikation hinterlegt. Ein Beispiel für solche Metriken sind die Zeiten in einem Projekt. In diesem konkreten Fall werden einerseits die zu Beginn festgelegte Projektdauer, die aktuell aufgewendete Arbeitszeit und die voraussichtliche Restdauer und andererseits die aus diesen Messungen resultierenden, berechneten Werte verwendet.

Aus diesen Metriken werden anschließend mittels verschiedener Algorithmen Entscheidungshilfen für den Decision Support und Erklärungen für das Understandability Management generiert. Die Metriken in QuASE unterteilen sich in (1) *QuASE Metrics in site model* und (2) *QuASE metrics in site ontology and knowledge base*. Diese Klassifikation der Metriken und die unterschiedliche Anwendung ist aus dem Journal Paper von Shekhovtsov, Mayr und Kucko ([66]) entnommen.

QuASE Metrics in site model

Metriken im QuASE site model sind als spezielle Art von Attributen definiert, die auf Context und Content Units, den sog. Owner Units, übertragbar sind. Die Metriken sind als numerische Werte definiert und so direkt für Analyse und Berechnungen verwendbar.

Diese Metriken treten als Tupel in Form von <*name, data type, mapping mode, mapping data*> auf. *Name* und *data type* sind dabei typische Programmierkonstrukte,

die üblicherweise in der Softwareentwicklung verwendet werden. Der *mapping mode* [14] definiert die Methode, wie die Werte der Metriken während der Erstellung der KB gesammelt werden. *Mapping data* enthält zusätzliche Informationen um das Sammeln der Werte durchzuführen. Die Semantik dabei stützt sich auf den mapping mode.

Die QuASE site DSL erlaubt direkte, query-basierende, berechnete und interaktive mapping modes (vgl. [66]). Weiterführende Informationen über die verschiedenen Modi sind in [66] nachzulesen.

QuASE metrics in site ontology and knowledge base

In der QuASE site ontology basiert die Unterstützung von Metriken auf der speziellen Weiterentwicklung des ontologischen Konzeptes der Attribute zu Metriken: *Jira-Issue.customer_attitude ⊑ IntegerMetric ⊑ Metric ⊑ Attribute ⊑ ModelingElement*. Metriken sind mit deren Ownern mit der „*hasAttribute*"-Objekt-Property verbunden.

Zum Aufbau der Unterstützung für analytische Aktivitäten während der Erstellung der KB wird jedes Metrik-Individual mit den Eigenschaften hinzugefügt und sowohl mit den absoluten, als auch mit den normalisierten Werten verbunden und mit einem eindeutigen Identifier der Metrik-Klasse versehen: *<id, value> : [Individual properties]*. Dabei gibt es einige vordefinierte Werte, die von 1 := „very high" in Schritten der Größe 0.25 bis 0 := „very low" gehen. Berechnete oder numerische Werte sind davon ausgeschlossen, diese haben bereits messbare Werte. In einem konkreten Beispiel könnte dies wie folgt aussehen:

<Jira-Issue-4513.customer_attitude, 0.75> : Jira-Issue
.customer_attitude.hasValue.

Die Werte für die Metriken mit direktem mapping mode kommen, durch die Ausführung der Entity-Query, aus dem JDBC Record Set. Die Daten für query-basierende Metriken werden, durch die Ausführung von Metrik-definierenden Queries, aus den separaten Record Sets bezogen. Die Daten für Metriken mit dem berechneten mapping mode werden durch den Code des QuASE-KB-Builder Tools geformt. Die Werte der interaktiven Metriken, gesammelt im QuASE Tool, werden von deren State Provider-Komponente bezogen (vgl. [66]).

5.2 Ontologien in QuASE

Für das Forschungsprojekt QuASE war es notwendig, die Ontologie in zwei wesentliche Teile aufzuteilen, da die Abbildungen der jeweiligen Organisationen wegen ihrer unterschiedlichen internen Strukturen nicht fester Bestandteil der Ontologie sein können. Eine Integration der Organisationsstrukturen in einer einzelnen Ontologie hätte zur Folge, dass die Ontologie je nach Einsatzzweck und Organisation neu aufgebaut werden müsste. Das hätte einen erheblichen Mehraufwand in der Entwicklung der Applikation durch die Änderungen in den Schnittstellen und einen größeren Wartungsaufwand zur Folge. In den nächsten Abschnitten werden deshalb die beiden (Teil-) Ontologien näher betrachtet.

5.2.1 Die QuASE site ontology

Die QuASE site ontology ist jener Teil der Ontologien, der für jede Organisation individuell erstellt wird. Hierbei spielen vor allem die verwendeten TIMS, sowie Aufbau und Struktur der Organisation eine Rolle. Eine zentrale Rolle in der site ontology spielen die site und der Context, sowie deren Repräsentationen.

Zusammen mit der QuOntology bildet die QuASE site ontology das Herzstück der QuASE KB und der Applikation. In der folgenden Abbildung 5-3 wird die Grundstruktur der QuASE site ontology abgebildet.

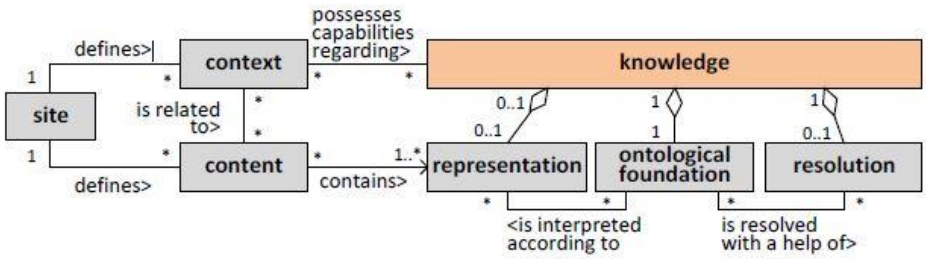

Abbildung 5-3: Grundstruktur der QuASE Site Ontology (aus [65])

Die Kernbestandteile der QuASE site ontology sind nach [65]:

site:

Eigentümer der aktuellen QuASE-Installation, z. B. ein Software-Provider.

context:

Einheiten, die spezielle Sichten auf kommunizierte Informationen haben, z. B. Projekte, Organisationen und deren Abteilungen, eingebundene Personen (Stakeholder), etc. Context Units werden durch Context-Attribute charakterisiert und können mit anderen Units verbunden sein.

Eine Context Konfiguration kann z.B. eine Repräsentation der gesamten Organisationshierarchie oder das gesamte Projekt-Portfolio eines IT-Unternehmens beinhalten. Zusätzlich kann eine Menge an Context Units die Kategorien dieser Einheiten enthalten (z.B. „IT-Unternehmen“, „Business Stakeholder“), wenn es eine Möglichkeit zur Sicht auf kommunizierte Information zu diesen Kategorien gibt.

content:

Einheiten, die kommunizierte Informationen aus den Projekt-Repositories darstellen: sie sind gleichzeitig Container für solche Informationen oder Organisationselemente für diese Container. Beispiele für diese Content Units sind Issues oder Tickets und deren Menge an Attributen, Werten und Requirement-Spezifikationen. Sie können auch als Unterelement eines Issues mit den einzelnen Inhalten auftreten.

knowledge:

Einheiten, die Qualität und Domänenwissen, welche Gegenstand der Kommunikation und Harmonisierung sind, voneinander abkapseln. Jede QuASE Knowledge Unit ist ein Triple (o, v, r) das folgende Komponenten kombiniert:

a) *ontological foundation*:
   eine Referenz zur Konzeptualisierung der speziellen Teile von Qualität oder Domänenwissen durch die ontologischen Mittel: Solche Konzeptualisierungen formen eine modulare Ontologie (*QuOntology*) und stellen ein Framework zur Übersetzung zwischen den Sichtweisen bereit.

b) *representation*:
   die Repräsentation einer Knowledge Unit in einem Format, das von den Kommunikationspartnern wahrgenommen werden kann (z. B. normaler Text). Representation Units sind in Content Units enthalten und können miteinander verbunden werden.

c) *resolution means*:
   die Mittel der Auflösung von Verständnis-Konflikten, die zu speziellen Knowledge Units gehören (z.B. Erläuterungen oder externe Referenzen)

Context Units besitzen *capabilities* (deutsch: Funktionen) um mit Knowledge Units umgehen zu können; insbesondere diese Funktionen können sich auf die Fähigkeit des Verstehens einer gegebenen Knowledge Unit (z.B. seine Repräsentation) beziehen; oder eine Knowledge Unit mit einer resolution means erklären (vgl. [65]).

Die beschriebenen Bestandteile wurden mittels Protégé zusammengefügt. Die Abbildungen 5-4 und 5-5 zeigen ein konkretes Beispiel des Aufbaus der QuASE site ontology aus der Implementierungsphase und den hierarchischen Aufbau einer QuASE site ontology.

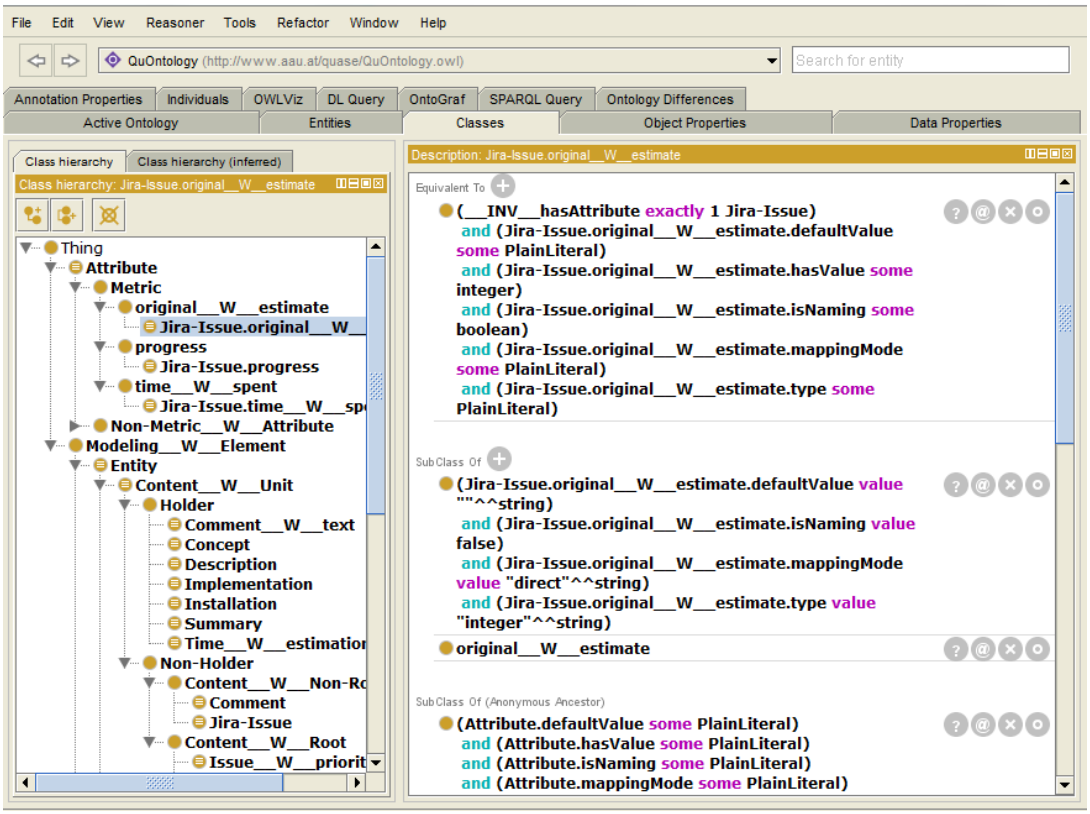

Abbildung 5-4: Die QuASE site ontology (Screenshot der Protégé-Anwendung)

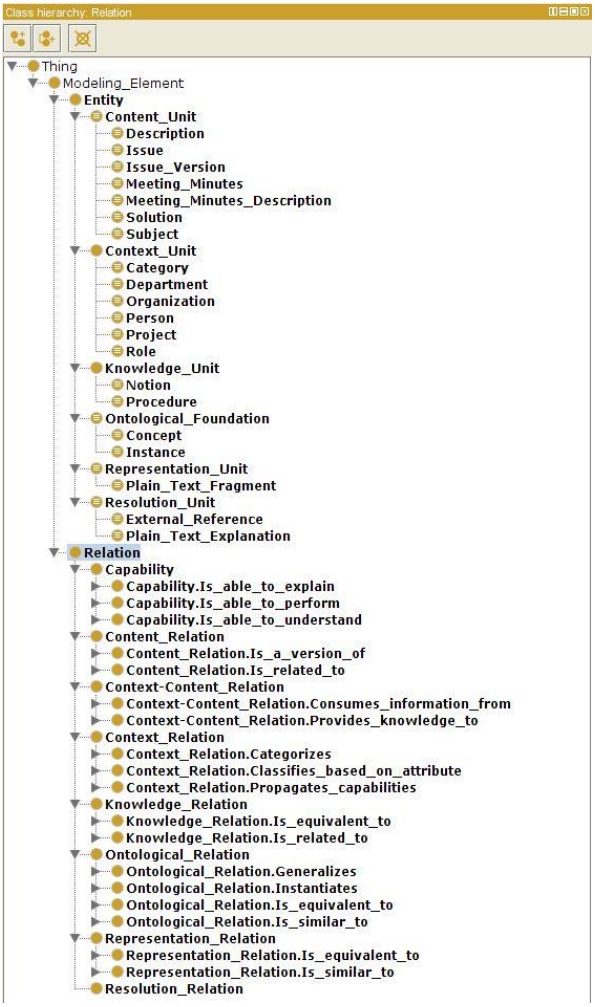

Abbildung 5-5: Hierarchie der QuASE site Ontology (Screenshot der Protégé-Anwendung)

Um das in Kapitel 4.3.4 und [66] beschriebene QuASE site model in die Struktur der QuASE KB zu übernehmen, muss es in eine rechnergestützte Ontologie-Darstellung basierend auf OWL 2 unter Verwendung des QuASE Ontology Builder Utility umgewandelt werden. Hierfür gilt eine Reihe von Regeln zur Erzeugung von OWL-2-Konstrukten aus dem XML-serialisierten QuASE site model. Diese sind insbesondere:

1. Die Menge von auf Metamodellen basierten OWL Klassen ist in der Ontologie vor der Verarbeitung der Modellelemente enthalten: *ContextUnit* $\sqsubseteq$ *Entity* $\sqsubseteq$ *ModelingElement*;
2. Ein Entity-Modellelement wird in eine OWL-Klasse als Subklasse der Klasse, die die Meta-Klasse der Entity repräsentiert, umgewandelt: *JiraUser* $\sqsubseteq$ *ContextUnit* $\sqsubseteq$ *Entity* $\sqsubseteq$ *ModelingElement*,
3. Ein Relation-Element wird in eine OWL-Klasse für das Element umgewandelt: *PersonCategory.Categorizes.JiraUser* $\sqsubseteq$ *Categorizes* $\sqsubseteq$ *ContextRelation* $\sqsubseteq$

Relation ⊑ *ModelingElement*, eine Menge von Objekt-Eigenschaften für Perspektiven und für die Relation selbst: *[source_for]PersonCategory.Categorizes.JiraUser* ⊑ *[source_for]ContextRelation* ⊑ *[source_for]* Relation und eine Menge von Axiomen, die mit der resultierenden Klasse und seiner Ziel-Entity-Klasse durch die Perspektiv-Eigenschaften verbunden ist;

4. Attribute werden in Daten-Properties umgewandelt und die korrespondierenden Axiome verbinden diese Properties zu den besitzenden Klassen.

(vgl. [66])

Diese logischen Formeln werden im Anschluss in OWL und RDF/RDFS übersetzt und im XML-ähnlichen Format gespeichert. Dies sieht wie im folgenden Codebeispiel für den Header und ein Object-Property aus:

```
<?xml version="1.0"?>
   <rdf:RDF xmlns="http://www.aau.at/quase/QuOntology.owl#"
      xml:base="http://www.aau.at/quase/QuOntology.owl"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:owl="http://www.w3.org/2002/07/owl#"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:QuOntology="http://www.aau.at/quase/QuOntology.owl#">
<owl:Ontology     rdf:about="http://www.aau.at/quase/QuOntology.owl"/>

<!--
////////////////////////////////////////////////////////////////////////
//
// Object Properties
//
////////////////////////////////////////////////////////////////////////
-->

<!-- http://www.aau.at/quase/QuOntology.owl#Comment.Contains.Comment__W__text -->

   <owl:ObjectProperty
          rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.Contains.Comment
          __W__text">
      <rdfs:domain rdf:resource=
          "http://www.aau.at/quase/QuOntology.owl#Comment"/>
      <rdfs:range rdf:resource=
          "http://www.aau.at/quase/QuOntology.owl#Comment__W__text"/>
      <rdfs:subPropertyOf rdf:resource=
          "http://www.aau.at/quase/QuOntology.owl#Content__W__Unit.Contains.Content_
          _W__Unit"/>
      <owl:inverseOf rdf:resource=
          "http://www.aau.at/quase/QuOntology.owl#__INV__Comment.Contains.Comment__W
          __text"/>
</owl:ObjectProperty>
```

(Quelle: QuOntology13.3.owl (RDFS-Source), Stand: 22.04.2015; Bearbeiter: V. Shekhovtsov, S. Strell & das Entwicklungsteam)

Einen detaillierten Einblick in den Aufbau der Ontologien bieten die Auszüge aus der kompletten Datei in Anhang B. Dort werden zum einen größere Teile aus der DL und zum anderen die korrespondierenden OWL-Fragmente gezeigt. Eine Gesamt-Auflistung der Ontologie-Daten ist aufgrund der Größe nicht möglich.

5.2.2 Die QuOntology

Die QuOntology ist das generische Modell der verwendeten Ontologie in der QuASE-Applikation. Hier werden u. a. die kontext-spezifischen Daten für die Wissensdatenbanken aufgebaut. Die QuOntology enthält alles Wissen, das im jeweiligen Umfeld kommuniziert werden kann. Die detaillierte Beschreibung der initialen Forschung an der QuOntology ist in [67] zu finden. Die Ontologie ist in drei Ebenen organisiert, wie in Abbildung 5-6 dargestellt wird.

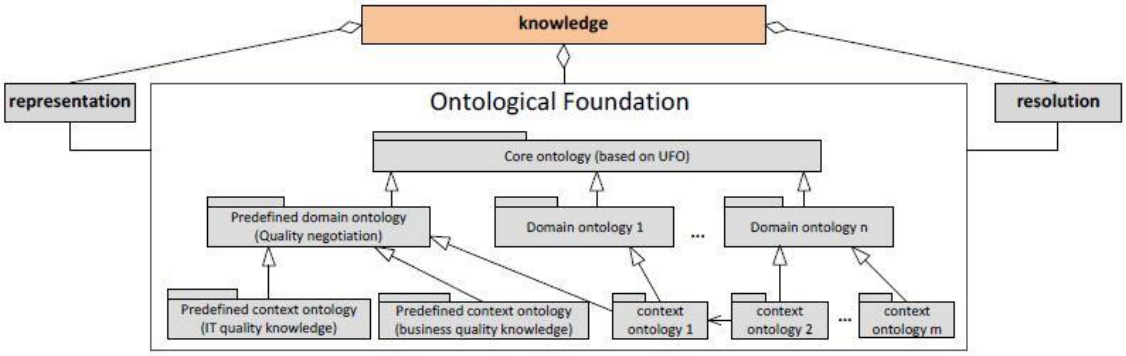

Abbildung 5-6: Organisation der QuOntology (aus [67])

1. *QuOntology core* repräsentiert ein stabiles Subset des verfügbaren Wissens aus der Wissenschaft und der industriellen Praxis; es ist nicht abhängig von der speziellen Problemdomäne. Als Basis für QuOntology core wurde die Unified Foundational Ontology (UFO) verwendet.
2. *Domain ontologies* repräsentieren die Besonderheiten des jeweiligen Problembereichs, der durch die gegebene Software während der Entwicklung angegangen wird (Finanzen, Öl und Gas, etc.); im Projekt QuASE wird für diese Ebene eine Domain Ontologie für Software Qualität eingesetzt.
3. *Context ontologies* stellen das Wissen bezogen auf spezielle Komponenten des QuASE Kontext dar: sie enthalten u.a. organisations- und projektspezifische Konzepte. In QuASE wurden für diese Ebene Ontologien für unternehmens- und IT-spezifische Sichten der Qualität verwendet.

   (vgl. [65])

Um die Zusammenhänge der einzelnen Objekte in der Knowledge Base darstellen zu können, wurden mit ADOxx und Protégé zum einen Metamodelle und Modelle erstellt, und zum anderen diese Modelle in eine OWL-Struktur übergeführt. Die Abbildung 5-7 zeigt einen Teil der graphischen Repräsentation der Ontologie aus dem Modellierungstool Protégé.

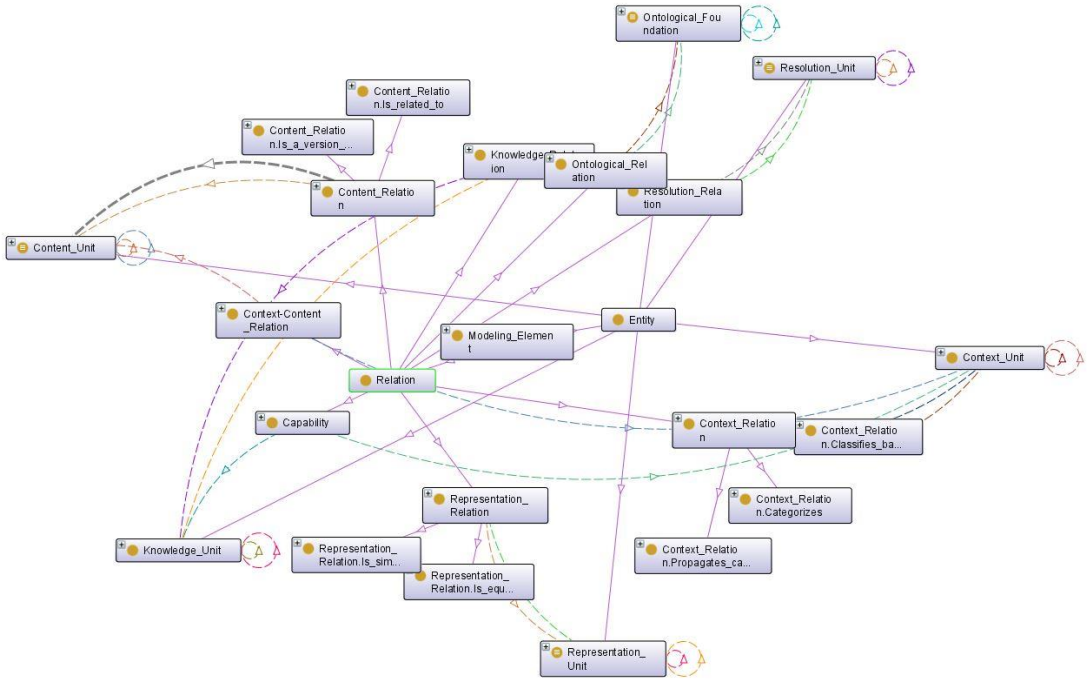

Abbildung 5-7: Graphische Repräsentation der QuOntology (Screenshot der Protégé-Anwendung)

5.3 Integration in das System

Die Ontologie wurde mit Hilfe von verschiedenen APIs von der OWL in ein von Java ausführbares Format umgewandelt. Zuvor mussten jedoch die Schnittstellen zwischen Knowledge Base und Ontologien und der Applikation definiert werden. Außerdem war es notwendig die Datentypen, sowie deren Inhalte innerhalb der Ontologien zu spezifizieren und modellieren. Dies wird in diesem Abschnitt näher behandelt. Um die Modellierung und die Erstellung von Spezifikationen effizienter und effektiver zu gestalten ist es notwendig zunächst die Metamodelle zu erstellen. Die Prinzipien der Modelle und Metamodelle wurden bereits im Abschnitt 4.3.4 beschrieben. Im nächsten Abschnitt werden an einem Beispiel die Zusammenhänge kurz erläutert.

5.3.1 Spezifikationen in QuASE und QuOntology

In Bezug auf die Ontologien QuOntology und QuASE site ontology, sowie der Knowledge Base, war ein Teil der Entwicklung des Proof-of-Concepts eine Spezifikation der sog. Site-Models zu erstellen. Diese wurden zum größten Teil in ADOxx modelliert. Ein weiterer Teil war die damit verbundene Auseinandersetzung mit Qualitätsanforderungen der verschiedenen Stakeholder, bis hin zum Verstehen der Inhalte der einzelnen TIMS-Datenbanken.

Die Site-Models bilden dabei die Grundlage für die QuASE site ontology, die für jede Organisation unterschiedlich und ein Grundgerüst für die Context-Selection, also der Kontext-Selektion, in der Applikation ist. Um die Spezifikation so detailliert und genau wie möglich zu gestalten wurden hierfür zum einen die strukturellen Daten aus den jeweiligen TIMS-DBs und zum anderen Interviews, die im Vorfeld der Implementierung durchgeführt wurden, verwendet.

Somit konnte, nachdem die Metamodelle für die Site-Models verfügbar waren, beide Spezifikationen daraus abgeleitet werden. In der folgenden Abbildung 5-8 wird das Metamodell zu dieser Context-Specification dargestellt.

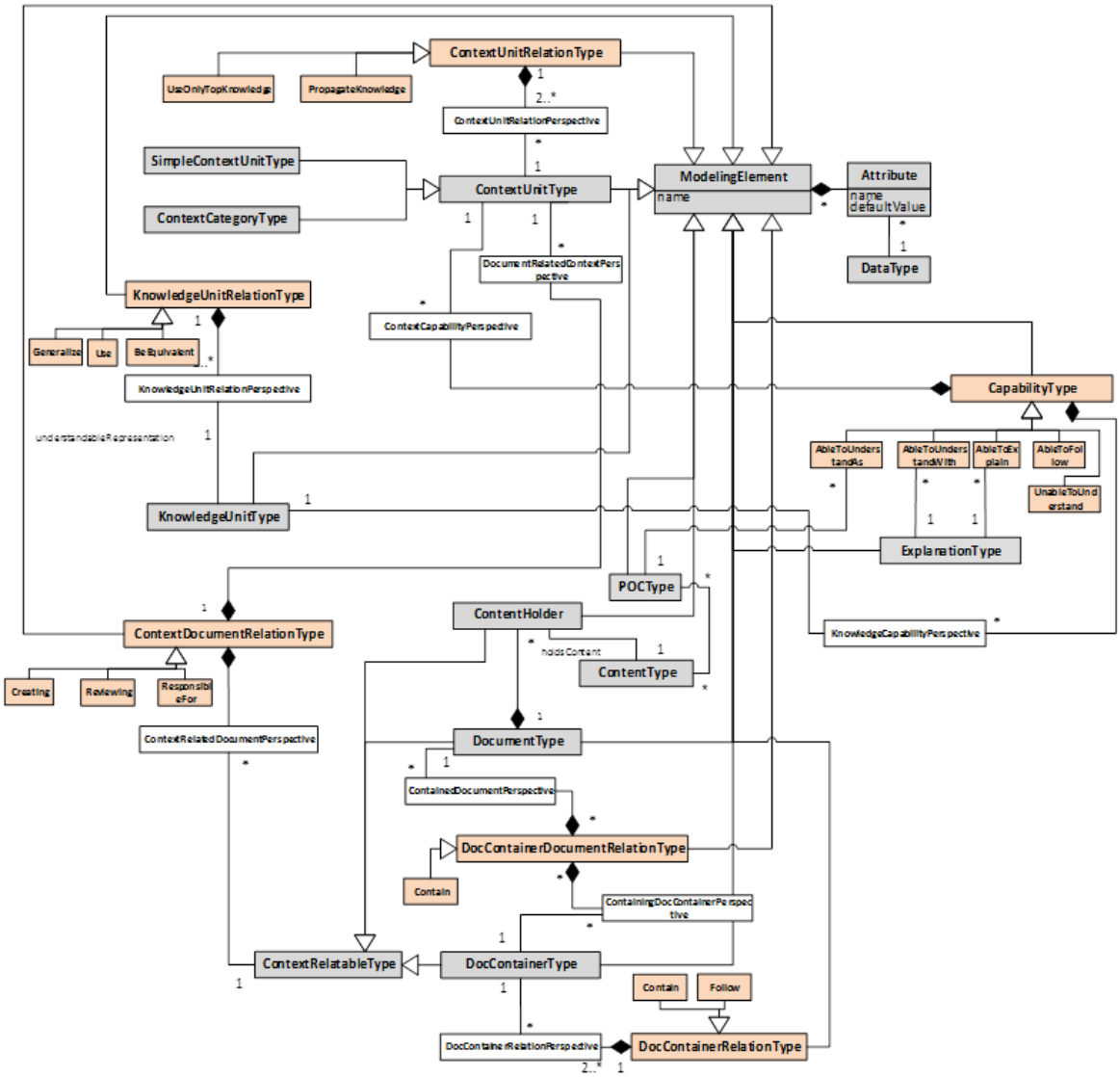

Abbildung 5-8: Metamodell der Context-Specification (aus: interne Dokumentation)

Ein weiterer Teil der Spezifikationen umfasste die Knowledge Base selbst. Da die QuASE KB den Hauptbestandteil der QuASE-Applikation bildet, ist diese Spezifikation als die wichtigste im Rahmen des Forschungsprojekts einzuordnen.

Wie in anderen Softwareentwicklungsprojekten auch, waren auch im Projekt QuASE Änderungen der Anforderungen und dadurch auch in den Spezifikationen notwendig. Dies diente allerdings überwiegend der korrekten Datenübernahme aus den einzelnen Organisationen, den sog. „Sites“. Erst nachdem die gesamte Datenstruktur in den einzelnen TIMS-Datenbanken analysiert worden war, konnten die einzelnen Site-Models, sowie der „Default user context“ modelliert und für die QuASE-Applikation zugänglich gemacht werden.

Der „Default user context“ richtet sich nach den im System angemeldeten Benutzern, deren jeweilige Sichtweise und Hintergrundwissen im Generellen und in den spezifischen Projekten. Auf dieser Basis können dann der Desicision Support, Knowledge Reuse, Translation und Explanation für die jeweiligen BenutzerInnen zur Verfügung gestellt werden.

5.3.2 Anbindung der QuASE KB an die Applikation

Die Modelle und Metamodelle der Knowledge Base und der Ontologien wurden, wie zuvor bereits erläutert, im XML-Format erstellt. Um allerdings aus einem für den Computer leicht und schnell interpretierbaren Dokument ein für den menschlichen Benutzer lesbaren Text zu erstellen sind meist mehrere Zwischenschritte mit Übersetzern und Interpretern notwendig.

Zunächst werden die Modelle und Metamodelle wie im vorherigen Abschnitt gezeigt erstellt und in ein XML-Format übertragen. Die Ontologien nutzen diese Modelle und eine TDB um daraus die Grundlage einer Knowledge Base zu formen. Dies geschieht oft in einer Description Logic (DL) oder der Ontologie-Sprache OWL. Diese nutzen eine standardisierte Syntax, wie z. B. die Manchester-Syntax, welche logisch aufgebaut, und deshalb für EntwicklerInnen ohne Spezialkenntnisse schwer lesbar sein kann. Im nächsten Schritt werden die Ontologien mit Internationalized Ressource Identifier (IRI) versehen, die dann durch die KB verwendbar sind.

Die Informationen aus den (Meta-)Modellen werden bei der QuASE KB zur Abbildung des Repositorys und der externen Daten in die KB verwendet. Um diese externen Daten, z. B. aus TIMS-DBs, in die KB einlesen zu können, wurde das Werkzeug *QuASE knowledge base builder* entwickelt und eingesetzt. Dieses Werkzeug dient zur Übernahme von organisationsspezifischen Vorgaben in die QuASE KB. Die QuASE KB kann auch inkrementell aufgebaut werden (KB synchronization). Allerdings muss dafür eine KB zur Verfügung stehen, die zumindest die Struktur der späteren QuASE KB enthält. Hierfür werden die spezifizierten Attribute jeder Entity aus dem Site Model mit dem Namen aus der Spalte *last update timestamp* der Entity Query verwendet (vgl. [66]).

Die Bestandteile einer KB werden in der Regel als *Entity Individuals* aus einer IRI geformt. Dabei werden die Werte der Spalte *id-designated column* aus einer verarbeiteten SPARQL-Query als *entity query*-Attribut (der Name dieser Spalte ist ebenfalls als *entity attribute* spezifiziert) und dem Entity-Namen verwendet:

Jira-Issue-4312 : Jira-Issue ⊑ *ContentUnit* ⊑ *Entity*.

Verbindungen zwischen Daten-Eigenschaften solcher individuals zu literals (deutsch: Literalen) ist (1) die Eigenschaft enthält den Namen der Entity aus dem Repository (z. B. den Issue- oder Ticket-Namen):

Jira-Issue-4312.__name ⊑ *Entity.__name*, *<Jira-Issue-4312, "Installation problem"> : Jira-Issue-4312.__name*

und (2) die Eigenschaft für sich als eindeutiger Identifier (basierend auf *Entity.__id*) (vgl. [66]).

Im Anschluss wird ein Auszug aus dem Quellcode des QuASE knowledge base builder gezeigt. Dabei handelt es sich um die Methode `reloadKB`, die die QuASE KB neu in die Applikation lädt, um z.B. Änderungen wirksam zu machen. Die bereits zuvor auskommentierten Zeilen wurden hierbei weggelassen, da diese aus einem frühen Entwicklungsstadium stammen und eventuell für LeserInnen ohne spezifische Fachkenntnisse irritierend sein könnten.

```
void reloadKB() throws QuASEException {

   FileHandler.INCREMENTAL_MODE = true;

   Connection conn = KBBuilderHandler.getConnection();

   try {

       ResultSet res = conn.prepareStatement("select model, ontology
           from models_and_ontologies where isactive=true")
           .executeQuery();

       if (res.next()) {
           String modelContents = res.getString("model");
           String ontologyContents = res.getString("ontology");

           File modelXMLFile = File.createTempFile("model", ".xml");
           FileUtils.writeStringToFile(modelXMLFile, modelContents);

           File ontologyFile = File.createTempFile("ontology",
               ".owl");
           FileUtils.writeStringToFile(ontologyFile,
               ontologyContents);

           File kbOutputFile = File.createTempFile("init", ".owl");

           String incrementalModeFlag =
               FileHandler.INCREMENTAL_MODE ? " -a " : "";

           String knowledgeBaseBuilderPath =
               System.getenv("JAVA_HOME") + "/bin/java -Xmx2000m -
               DCONVERTER_EXE_PATH=" +
               System.getenv("QUASE_CONVERTER_EXE_PATH")
                + " -jar " +
```

```
                System.getenv("QUASE_CONVERTER_EXE_PATH") +
                "/QuKnowledgeBaseBuilder.jar" +
                incrementalModeFlag + " -x " +
                modelXMLFile.getAbsolutePath() + " -i " +
                ontologyFile.getAbsolutePath() + " -p " +
                System.getenv("QUASE_CONVERTER_EXE_PATH") +
                " -o " + kbOutputFile.getAbsolutePath() + " -r jdbc";

            if (!XMLUploadFunctionality.runProcess
                (knowledgeBaseBuilderPath, "kb.lock"))
                throw new QuASEException("Synchronization error");

            OWLUploadFunctionality
                .updateKBFromOWLPath(kbOutputFile.getAbsolutePath(),
                FileHandler.INCREMENTAL_MODE);

            conn.prepareStatement("update models_and_ontologies set
                last_kb_update_time=current_timestamp where
                isactive=true").executeUpdate();

            modelXMLFile.delete();
            ontologyFile.delete();

        } else
            throw new SQLException();

    } catch (Exception e) {
        e.printStackTrace();
    } finally { }
}
```

(Quelle: QuASE-Application Source Code, Stand: 22.04.2015, Entwickler: M. Kucko, S. Strell, V. Lubenskyi, S. Ianushkevych)

Die *reloadKB*-Methode ist ein Teil der Klasse KBBuilderRunner, die den bereits zuvor benannten inkrementellen Ansatz realisiert. Hierzu werden zunächst die veränderten Daten als temporäre Dateien angelegt, die mit den bereits in der Applikation verwendeten Dateien der QuASE KB verglichen und bei Bedarf übernommen werden. Dadurch kann unnötiger Speicher- und Zeitverlust aufgefangen werden, da nicht jedes Zeichen neu eingelesen und verarbeitet werden muss.
Die Einbindung der Knowledge Base mit den Ontologien in die Applikation wurde im serverseitigen Teil realisiert. Dazu gehören nicht nur der bereits gezeigte *QuASE knowledge base builder*, sondern auch weitere Methoden, die die KB mit der Applikation verknüpfen. Dafür wurden zumeist spezielle Pakete innerhalb der einzelnen Klassen, in denen die Inhalte der QuASE KB verwendet werden, eingeführt und spezielle Verarbeitungsalgorithmen verwendet. Konkret wird dies z.B. im Understandability Management mit dem folgenden Code-Beispiel realisiert.

```
private List<MEOverview> createOntologicalDomainSelector
        (List<OutputOntologicalDomain> units) {
    DataProvider dataProvider = new OntologicalDomainProvider();
    List<MEOverview> selector = new ArrayList<>();
    for (OutputOntologicalDomain unit : units) {
        MEOverview overview = null;
        try {
            overview = dataProvider
                  .getOverview(SPARQLHelper.fragment(unit.getIri()))
                  .query();
        } catch (QuASEException e) {
            e.printStackTrace();
        }
        selector.add(overview);
    }
    return selector;
}
```

(Quelle: QuASE-Application Source Code, Stand: 22.04.2015)

Das oben angeführte Code-Beispiel ist ein Auszug aus der Klasse *UnderstandabilityFunctionality* im Unterpaket *understandability* des serverseitigen *application*-Pakets und zeigt die Methode `createOntologicalDomainSelector`. Die Methode wird im Understandability-Management zur Erzeugung eines Datenselektors verwendet, dessen Inhalte aus der ontologischen Domain kommen.

Um die speziellen Fragmente aus der Ontologie in die Applikation zu integrieren und damit verwendbar zu machen, benötigt man zum einen die in Kapitel 2.2.4 erläuterten IRIs, sowie Teile des *SPARQL-Helper*. Diese API ist eine öffentlich verfügbare Programm-Bibliothek, die die Verarbeitung von SPARQL-Queries in einem Java-Programm unterstützt. Der SPARQL-Helper wird von der Klasse *dataProvider* verwendet, der die Daten aus der Ontologie der Applikation zur Verfügung stellt.

Einige Teile der Applikationslogik, die mit der QuASE KB verknüpft sind, werden in der Regel mehrfach verwendet, um zum einen die Komplexität niedrig zu halten und um zum anderen so wenig wie möglich ähnliche Methoden zu erhalten.

So wird beispielsweise das in Kapitel 4.1.2 beschriebene Understandability Management durch die Verwendung der QuASE KB unterstützt. Hierbei wird durch einen Parser der Issue, bzw. dessen Beschreibung textuell erfasst und zum Kontext A hinzugefügt. Um das Understandability-Improvement für einen anderen Kontext zu schaffen wird der Text in Phrasen und Terme, in der Fachsprache auch „tokens“ genannt, unterteilt, die anschließend über die KB und Ontologien übersetzt oder erklärt werden sollen. Die folgende Abbildung 5-9 illustriert diesen Vorgang. Falls keine Übersetzung durch Synonyme oder eine Erklärung des tokens vorliegt, wird dieser als „derzeit nicht erklärbar“ markiert und es wird mit verschiedenen Funktionen u.a. mit denen des machine learnings versucht ihn in die KB mit aufzunehmen. Viele der

Terme und Synonyme werden bereits durch die Erstellung und Installation der Ontologien und der QuASE KB in das laufende System gespeichert. Je präziser dabei die Erstellung durchgeführt wird, desto größer die Trefferquote des Moduls Understandability Management. Das bedeutet, je mehr Phrasen und tokens bereits vor der Einführung des Systems in einem spezifischen Bereich vorhanden sind, desto größer ist die Anzahl der Synonyme und Erklärungen eines Terms. Diese Quantität spiegelt auch eine gewisse Qualität wider, speziell wenn die Synonyme für verschiedene Stakeholder gelten.

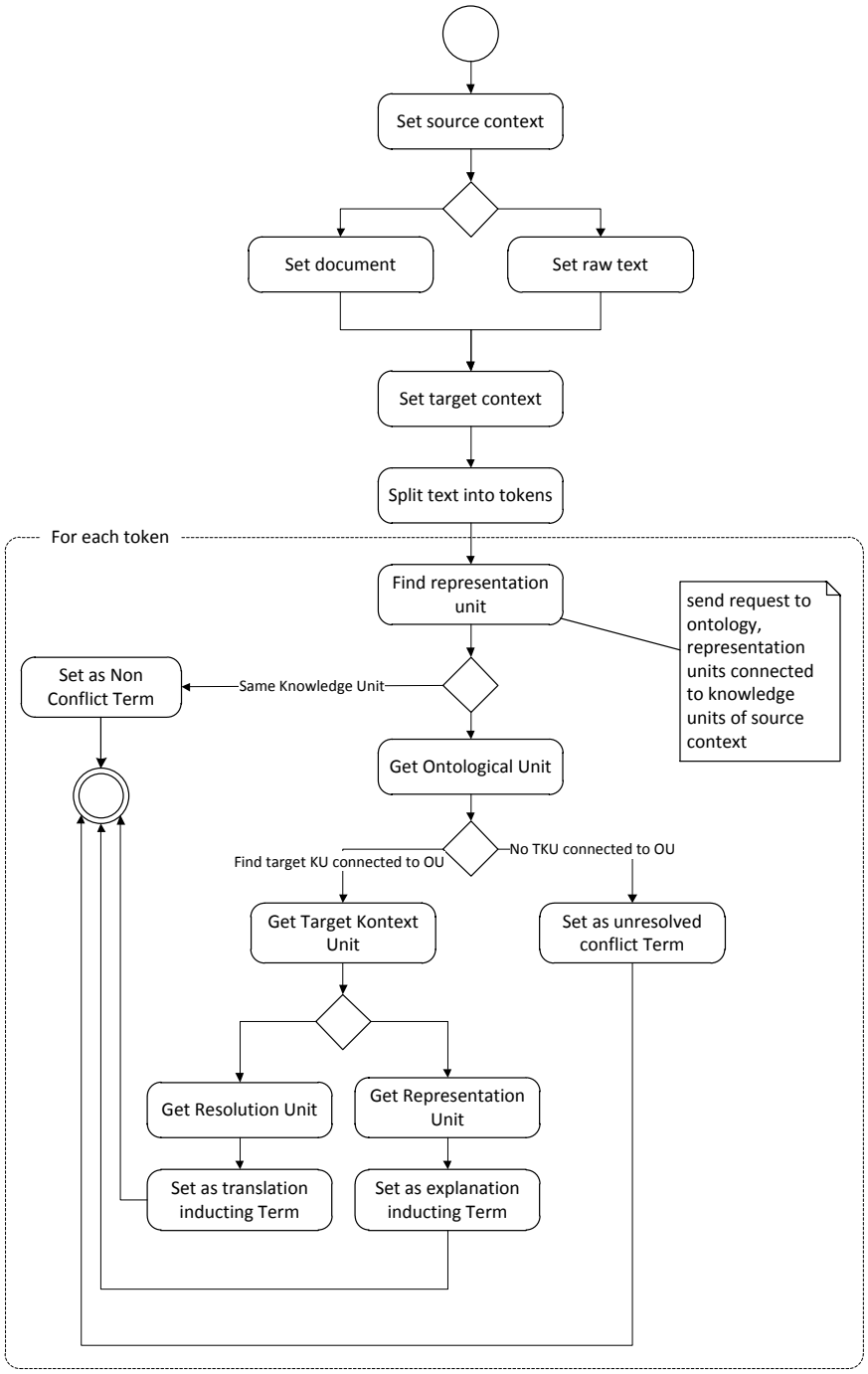

Abbildung 5-9: Flussdiagramm des Understandability Improvement (aus: interne Dokumentationen)

Weitere Einsatzgebiete der QuASE KB sind auch die anderen in Kapitel 4.1 beschriebenen Use Cases. Hierbei wird zum größten Teil allerdings differenziert auf die Daten zugegriffen. So ist das Knowledge-oriented Access Interface ein wichtiger Bestandteil um die Applikation mit den ontologischen und wissensbasierten Teilen der Applikation zu verknüpfen. Im Decision Making Support, also der Entscheidungshilfe, wird die Knowledge Base zur Bereitstellung und Berechnung der verschiedenen Metriken verwendet.

Die Liste der Applikationsmodule kann mit verschiedenen Funktionaliäten erweitert werden. Hierfür sind nicht nur Softwareteile, die in Verbindung mit Ontologien, KBs oder logischen Elementen arbeiten vorgesehen, sondern auch klassische Elemente der Softwareentwicklung, Datenhaltung, -nutzung und -erweiterung möglich. Einige Möglichkeiten zur Erweiterung und Verbesserung der aktuellen Applikation werden im nächsten Abschnitt dargestellt.

5.4 Mögliche Erweiterungen

Es gibt eine Vielzahl von Erweiterungen oder Verwendungen des Forschungsprojekts QuASE und deren Applikation. Vom Austausch einzelner Module bis hin zu gänzlich anderen Betätigungsfeldern oder Erweiterungsszenarien gibt es Möglichkeiten zur Veränderung und Verbesserung. Einige dieser Möglichkeiten werden nun in diesem Abschnitt dargestellt.

Die Ansätze von Ontologien und Knowledge Bases finden sich mittlerweile in vielen technischen Bereichen wieder, wie auch in [23, 31] gezeigt wurde. Dies ist vor allem, wie in Kapitel 2 beschrieben, durch die Errungenschaften im Forschungsfeld des Semantic Web möglich. Auf diese Bereiche zielt auch das Forschungsprojekt QuASE durch die Einführung von generischen Modellen und veränderbaren Inhalten in der Knowledge Base ab.

Die Ontologien des Projekts QuASE stellen zusammen mit der Knowledge Base, wie zuvor dargestellt, das Herzstück der zugehörigen Software dar. Zu den Erweiterungsszenarien könnten in diesem Spektrum nicht nur andere Wirtschaftsbereiche, sondern auch Forschung und Lehre zählen. Prinzipiell gilt, überall wo Wissen in andere Bereiche übertragen oder genutzt wird, können Betätigungsfelder der QuASE KB und deren Ontologien sein.

Da die Applikation des Forschungsprojekts QuASE modular aufgebaut wurde, ist es auch möglich eine andere Form einer Ontologie oder Knowledge Base zu verwenden. Hierzu sollte allerdings beachtet werden, dass Änderungen an Schnittstellen zwischen der Applikation und den neu zu verwendeten Teilen nötig werden könnten.

Denkbar wäre im Kontext von Softwareprojekten auch, dass die QuASE-Applikation als Modul in einem Projektmanagement-System als sog. Plug-In zur Erleichterung der Planung und Aufgabenverteilung angeboten wird. Eine Erweiterung von QuASE als Framework hätte den Vorteil, für eine Vielzahl von interessierten Unternehmen und Organisationen zur einfachen Verwendung zugänglich zu machen.

Die Forschungsgruppe Application Engineering, die dieses Projekt durchführte, arbeitet ebenso an sogenannten Mensch-Maschinen-Schnittstellen, wie dem Human Behavior Monitoring and Support (HBMS)[27]. HBMS ist ein Forschungsgebiet innerhalb der sogenannten Ambient Assisted Living (AAL), bei dem es um die technische Unterstützung von gesundheitlich beeinträchtigten Menschen geht, damit diese weiterhin in gewohnter Umgebung mit möglichst wenig Hilfe von außerhalb leben können. Hier könnten Synergien aus den Ontologien, Metamodellen und dem Software-Prototyp genutzt werden.

Da sich QuASE auch mit Übersetzungen und Erklärungen von Begriffen in einen anderen Kontext beschäftigt, wäre es auch denkbar, dass man die Ontologien und KBs mit Computer-Linguistik-Ansätzen verknüpft und für Übersetzungsprogramme optimiert und einsetzt. Dies hätte zur Folge, dass die „Verluste“ durch Übersetzungen gering gehalten werden könnten. Damit wäre ein Einsatz auch in international operierenden Unternehmen und Organisationen denkbar und die Akzeptanz der Applikation weltweit könnte gesteigert werden.

Im Zuge der aktuellen Forschungsthemen in Bezug auf Internet of Things (IoT) und die sog. „Smart Factory“ (intelligente Fabrik), ein aktueller Teil des Konzepts zu Industrie 4.0[28] (siehe auch [71]) könnten ebenfalls Teile aus dem QuASE-Projekt genutzt werden. Ein Beispiel hierfür wäre die Ontologie und das maschinelle Lernen, da dies mit ein wenig Aufwand von den „klassischen“ Methoden der Mensch-zu-Mensch-Kommunikation und Mensch-Maschine-Schnittstellen (engl. Human-Computer Interface, HCI) auf die dort genutzten Maschine-Maschine-Schnittstellen übertragen werden könnte. Auch weitere Aspekte der Wissensaufbereitung und Wissensverteilung sind in diesen Systemen denkbar und möglich.

Weitere Szenarien, in denen Teile der QuASE-Applikation eine Rolle spielen könnten sind die bereits vorhandenen TIMS. Hier könnte nicht nur die Ontologie, sondern die gesamte Knowledge Base zur Verbesserung von Entscheidungshilfen (Decision

[27] HBMS: http://hbms-ainf.aau.at

[28] Projekt-Definition des deutschen Bundesministeriums für Bildung und Forschung: https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html

Support) und dadurch zur Verbesserung der Software Qualität bei allen Nutzern führen. Für diesen Erweiterungsbereich gibt es bereits einige Forschungsansätze an der Alpen-Adria-Universität Klagenfurt.

Um in den am meisten genutzten Ticketing- und Issue-Managementsystemen eine Knowledge Base einzubauen ist es allerdings nötig, das QuASE-Projekt nicht nur zu einem erfolgreichen Abschluss zu bringen, sondern auch zu zeigen, dass durch die Nutzung von Ontologien und Knowledge Bases Verbesserungen erzielt werden können. Damit beschäftigt sich das nächste Kapitel.

6 Verbesserung der Software Qualität

Dieses Kapitel bildet mit der Beantwortung der eingangs gestellten und im Kapitel 4 erläuterten Forschungsfrage den inhaltlichen Abschluss der Masterarbeit. Hierzu wird zuerst auf generelle Möglichkeiten zur Verbesserung der Software Qualität eingegangen und im Anschluss daran wird gezeigt, wie Ontologien und KBs eine Verbesserung bringen können. Den Abschluss bildet die Einordnung der Verbesserungsmöglichkeiten.

Zur Beantwortung der Frage „*Wie können Ontologien und Wissensdatenbanken die Arbeit der Akteure im Softwareentwicklungsprozess erleichtern und helfen Probleme und Missverständnisse zu reduzieren?"* sollte man zunächst wissen, wo die Probleme und Missverständnisse in einem Softwareentwicklungsprozess (SEP) vorkommen können und welche Ansätze zur Qualitätsverbesserung bereits zur Verfügung stehen.

Zunächst sollte dafür die Arbeitsdefinition[29] von Software Qualität aus Kapitel 2.4.2 nochmal in Erinnerung gerufen werden, da dieser Terminus sehr unterschiedlich interpretiert werden kann.

Unbestritten ist der Einsatz von Knowledge Bases und Ontologien in der heutigen Technik. Jedoch wurde bisher darauf Wert gelegt, dass bestehende Systeme durch gezielten Einsatz von Ontologien, wie in Fuchs' Dissertation [23], verbessert wurden oder gänzlich neue Bereiche erschlossen werden können. Die Qualität beim Entwickeln von Software selbst zu verbessern ist in diesem Zusammenhang jedoch sozusagen noch „Neuland".

Seitdem es Software Engineering und deren Teilbereiche gibt, wird nach immer effizienteren und effektiveren Methoden gesucht, um die Qualität angesichts des immer weiter steigenden Erwartungsdrucks anzupassen. Ansätze hierzu bieten zum einen die bereits weit verbreitete computer-unterstützte Softwareentwicklung (engl. Computer Aided Software Engineering, CAiSE), zum anderen bessere Softwareentwicklungswerkzeuge und die Weiterentwicklungen im Projektmanagement der Softwareentwicklung.

Um das Zitat von Grady Booch nochmals aufzugreifen, es ist bisher nicht möglich gewesen die Entwicklung von Software trivial und dadurch fehlerfrei zu gestalten. Dies wird wohl auch in Zukunft nicht möglich sein, denn nach Edsger W. Dijkstra können Tests zwar die Existenz von Fehlern aufzeigen, allerdings nicht ihre Abwesenheit (vgl. [60, 61], Original in [38]). Um die potenzielle Fehlerzahl in

[29] die Arbeitsdefinition von Software Qualität in Verbindung mit KBs und Ontologien aus S. 27 (Anm. d. Autors)

Softwaresystemen zu verringern gibt es zahlreiche Möglichkeiten, die zu Beginn eines Projekts in Betracht gezogen werden können.

6.1 Möglichkeiten zur Verbesserung der Software Qualität

Wie bereits in Kapitel 2.4 gezeigt gibt es mehrere Möglichkeiten um die Qualität von Software messbar und vergleichbar zu machen. Daraus folgt ebenfalls, dass es mindestens ebenso viele Möglichkeiten gibt die Qualität zu verbessern. Viele dieser Möglichkeiten finden sich bereits in der Fachliteratur, wie [1, 3], aber auch Vorlesungen an Universitäten und Hochschulen beschäftigen sich damit, bereits bestehende Software zu verbessern und Fehler, die in vorherigen Softwareprojekten gemacht wurden nicht zu wiederholen.

In der Theorie hört sich das natürlich sehr einfach an, allerdings ist es in der Praxis meist deutlich schwieriger, da Budget und Termine eingehalten werden müssen und nicht jede/r SoftwareentwicklerIn oder IT-ExpertIn das gleiche Arbeitstempo und die gleiche Ausbildung hat. Nach Tom DeMarco's Buch „Spielräume" [62] sollte man die Verbesserungsmöglichkeiten auch im Projektmanagement suchen. Dabei sollte vor allem die Kommunikation „im weißen Raum", also nicht über die Hierarchie-Ebenen, vorangetrieben werden. Außerdem sollten nach DeMarco immer genügend Puffer zur Verfügung stehen und Personalentscheidungen möglichst früh im Projekt getroffen werden (vgl. [62]). Diese und weitere Ansätze nach [62] können zwar nicht direkt die Qualität einzelner Softwareprojekte merkbar verbessern, aber zumindest bei Fehlentwicklungen in der Organisationsstruktur ein Alarmsignal setzen. Einen ähnlichen Ansatz verfolgen auch Cadle und Yeates in [2] u.a. mit dem Value Engineering und dem daraus folgenden Value Tree, der in Abbildung 6-1 dargestellt wird.

Verbesserung der Qualität von Software ist auch das Ziel des SWEBOK[30], dessen Inhalte sich vor allem auf die Veränderung im Softwareentwicklungsprozess stützen. So wird darin eine Menge an Verbesserungsvorschlägen geboten, die das Managen dieser Prozesse beeinflussen. Ähnlich verhalten sich die CMMI- und ISO-Standards, die für Software Engineering definiert und umgesetzt wurden.

[30] SWEBOK: http://www.computer.org/portal/web/swebok/htmlformat (IEEE Computer Society)

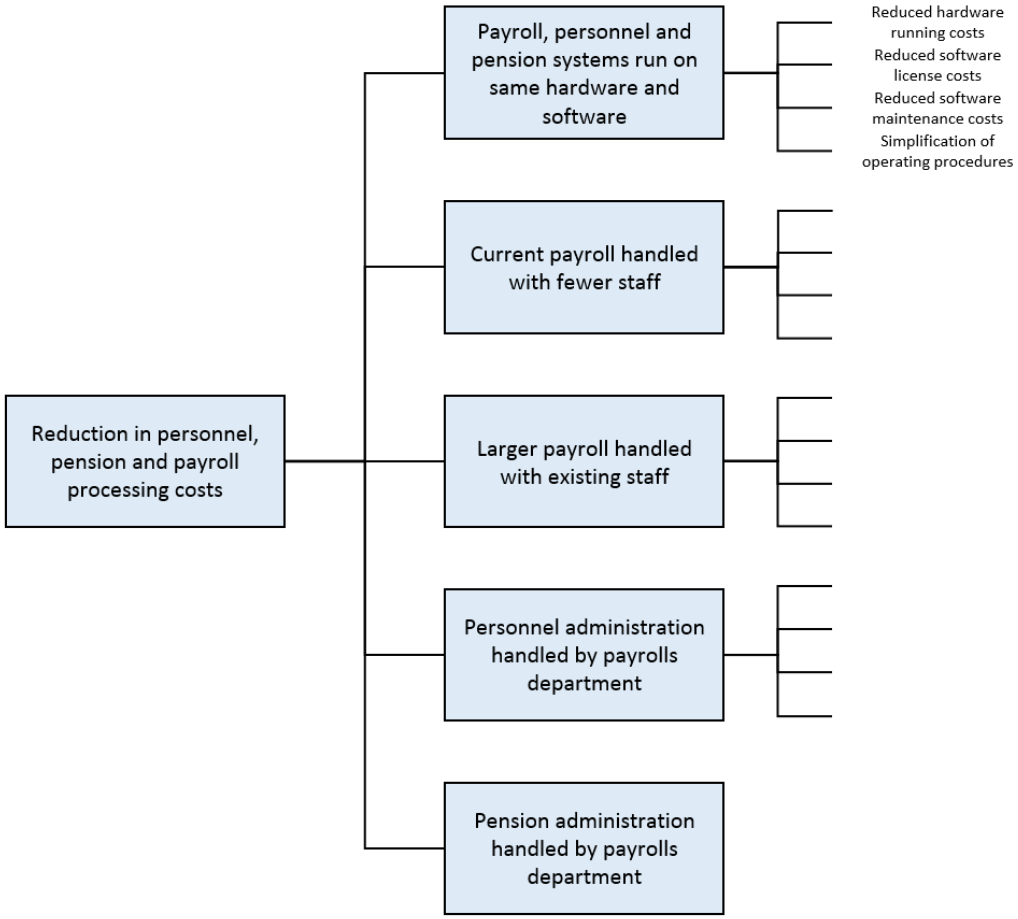

Abbildung 6-1: Beispiel eines Value Trees (nach [2], Figure 16.3)

Das größte Problem jedoch, warum die Qualität der Software nicht auf dem Level liegt, auf dem man es erwartet, ist der Faktor Mensch. Nicht nur das individuelle Wissen der einzelnen Personen, sondern auch wie mit Informationen umgegangen wird, spielen dabei eine sehr große Rolle. So kann beispielsweise ein Benutzer/eine Benutzerin andere persönliche Voraussetzungen an eine Software stellen, als andere BenutzerInnen. Auch bei den SoftwareentwicklerInnen spielen diese Unterschiede eine große Rolle. Was ein Entwickler als sinnvoll und notwendig erachtet kann bei einem anderen eine eher untergeordnete Rolle spielen. Um diesen Faktor zu minimieren und so gering wie möglich zu halten wurden seit der Softwarekrise in den 1960er-Jahren immer wieder Forschungen und Weiterentwicklungen in der Art der Softwareentwicklung betrieben.

Nach Sommerville in [61] und einigen weiteren namhaften AutorInnen, gibt es beispielsweise Modelle, die es den ProjektplanerInnen erleichtern, die Entwicklung in Phasen einzuteilen und diese in weiterer Folge zu verfeinern (vgl. u.a. [60, 61]). Diese Modelle und Mechanismen erleichtern zwar die Planung, jedoch kann die Durchführung durch Änderungen oder Änderungswünsche, die sich häufig erst während

den ersten Entwicklungsphasen abzeichnen, immer wieder zu Problemen oder Verzögerungen führen. In einer früheren wissenschaftlichen Arbeit des Autors dieser Thesis wurde festgestellt, dass die Software-Engineering-Modelle und -Patterns *„es dem Softwareentwickler einfacher machen ein Problem in einer vernünftigen Zeit und mit adäquaten Mitteln lösen zu können“* ([60], S. 40). Allerdings sei hier erwähnt, dass diese Modelle und Patterns kein „Allheilmittel“ sind, sondern lediglich eine Hilfestellung für den Softwareentwicklungsprozess anbieten.

Jedes dieser Softwareentwicklungsmodelle, ob agil wie SCRUM und XP, oder klassisch wie Wasserfallmodell und das Spiralmodell nach Boehm, stellt zumeist ein großes Wissen im Aufbau und Organisation von Softwareentwicklungsprojekten dar. Zumeist werden auch Mischformen dieser Modelle verwendet um die eigenen Ziele und Lösungsansätze hervorheben oder um das Projekt auf die Struktur des Unternehmens anpassen zu können.

Es gibt viele „klassische“ Möglichkeiten, die Qualität von Softwareprojekten und deren Produkte zu steigern, die meisten davon sind bereits Teil der „best practices“ in den Organisationen und Unternehmen. Und gerade deshalb ist es für die Einen notwendig und für Andere erstrebenswert neue, vielleicht effektivere, Methoden zur Qualitätsverbesserung zu finden. Ontologien und Knowledge Bases, die im nächsten Abschnitt behandelt werden, sind hierbei nur ein Spektrum der möglichen Ansätze.

6.2 Ontologien & Knowledge Bases zur Verbesserung der Software Qualität

In den vorherigen Kapiteln wurde die Theorie zu Software Qualität und Ontologien und Wissensdatenbanken erläutert, sowie ein praktisches Beispiel zur Verknüpfung dieser Themen gegeben. Das Forschungsprojekt QuASE, dass qualitätsbewusste Softwareentwicklung zum Ziel hat, ist nur eine mögliche Anwendungsform dieser Themen. Wissensdatenbanken können in der Regel aus allen Bereichen stammen, unabhängig ob diese mit Software Qualität in Verbindung stehen oder nicht. Auch Ontologien gibt es mittlerweile in anderen Bereichen der angewandten Informatik, also in konkreten Computer-Anwendungen, und es werden bereits weitere Anwendungsgebiete im Rahmen des Semantic Web erforscht. Einige dieser Anwendungsgebiete fallen in den Bereich des sog. „Next Generation Enterprise Modelling“ (NEMO) und Business Information (BI) Systems. Diese Anwendungsgebiete beschäftigen sich mit Software-Lösungen, die den späteren AnwenderInnen mit Konzepten zur einfacheren Bedienung oder Visualisierung von komplexen Systemen unterstützen sollen.

Hintergrund dafür sind spezielle Algorithmen, die in den Systemen Lernprozesse ermöglichen. Dieses sogenannte „Machine Learning“ ist eines der grundlegenden Konzepte für Semantic Web und Ontologien. Dabei werden z.B. Schlagworte oder Webseiten, die vom User in der Vergangenheit oft verwendet wurden, in speziellen Systemen zwischengespeichert, um daraus Metriken zu generieren, die das Verhalten der BenutzerInnen wiedergeben. Daraus lassen sich, mit einer bestimmten Wahrscheinlichkeit, die nächsten Abläufe der User errechnen, die das Computersystem dem Benutzer vorschlägt.

Bisher wurde in diesem Kapitel ein Überblick über die Verbesserung von Software Qualität mit klassischen Methoden behandelt. Um sich nun der Beantwortung der RQ weiter nähern zu können wird nun kurz ein Überblick über die Probleme und Missverständnisse in einem SEP gegeben.

Um im Kontext des Projekts QuASE feststellen zu können wo diese Probleme sind, wurden im Vorfeld der Implementierungsarbeiten Interviews mit MitarbeiterInnen der Kooperationspartner aus der Wirtschaft geführt. Diese Interviews stellten gleichzeitig eine Grundlage der Knowledge Base und der Basisimplementierung der Ontologien QuOntology und QuASE site ontology dar. Allerdings ist es aufgrund von Geheimhaltungsvereinbarungen und zur Wahrung des Wettbewerbs nicht möglich Teile dieser Interviews direkt in dieser Arbeit einzubringen. Deshalb werden hier nur die Themengebiete und einige als allgemein bekannte Qualitätskriterien aufgeführt. Einige der Themengebiete dieser Interviews waren, wie die Qualität der Software durch Missverständnisse der einzelnen Akteure beeinflusst wird und die Definition von nichtfunktionalen Anforderungen, wie Benutzerfreundlichkeit und Sicherheit aus Sicht der einzelnen Rollen.

Eine Lösungsstrategie zur Reduktion von Missverständnissen eines der Kooperationspartner ist, zum einen genau einen Ansprechpartner im Unternehmen für das jeweilige Projekt zu haben und mit diesem durch gezielte Kommunikation in regelmäßigen Abständen auf beiden Seiten eine vertrauensvolle Basis zur Zusammenarbeit zu errichten. Zum anderen werden im eigenen Unternehmen die Rollen im Projekt genau definiert und die Kommunikation und Zusammenarbeit innerhalb des Teams gestärkt.

Um diese Strategie möglichst optimal umzusetzen bedarf es nicht nur einer gezielten Kommunikation und Auswahl von potenziellen MitarbeiterInnen, sondern auch teilweise technischer Unterstützung. Denn ob die Zusammenarbeit funktioniert zeigt in der Regel nicht nur der Arbeitsalltag, sondern auch, ob die Ziele im Projekt erreicht oder übertroffen wurden. So kann es z. B. trotzdem vorkommen, dass die Kommunikation im Team hervorragend funktioniert, allerdings das Projekt zum festgelegten

Zeitpunkt nicht fertig ist. Mit einem TIMS (Beschreibungen: siehe Kapitel 3.1) lassen sich zwar durchaus solche Missstände aufdecken, eine Begründung jedoch kann ein solches System nicht gezielt liefern.

Ein weiterer Aspekt um die Unternehmensphilosophie umzusetzen und Softwareentwicklungsprojekte gezielt zum Erfolg zu führen, ist aus vergangenen Projekten zu lernen. Hierzu können nicht nur die Fehler und negativen Aspekte, sondern auch positive Ergebnisse helfen. Allerdings ist es mit den klassischen Hilfsmitteln mitunter schwierig, diese Projektergebnisse und -erfahrungen so zu speichern, dass man diese ohne großen Aufwand wieder verwenden kann. Denn die Zusammenhänge zwischen den einzelnen Projekten sind nicht immer auf den ersten Blick feststellbar.

Nachdem einige der Probleme und Missverständnisse benannt und definiert wurden, kann man diesen auf den Grund gehen und versuchen die eigentliche Antwort auf die Forschungsfrage zu finden. Kernpunkt der Aussage ist hierbei allerdings nicht, ob dies möglich ist, sondern wie und in welchem Rahmen. Dabei geht es außerdem nicht darum eine spezielle Antwort zu finden, sondern aus einer Menge an Entscheidungshilfen die optimale Lösung finden zu können. Denn vor allem in der Forschung ist eine Problemlösungsstrategie nur eine Sichtweise oder ein Teil eines konkreten Problems.

Durch Ontologien und Knowledge Bases kann informelles Wissen formal ausgedrückt werden, das dadurch für Mensch und Maschine lesbar und vor allem für Personen leichter zu erfassen ist. Hierzu ist vor allem bereits erlangtes Wissen, das in diversen Dokumenten oder Quellcodes vorliegt, eine gute Grundlage um solche Ontologien und Wissensdatenbanken aufzubauen und für darauf folgende Begebenheiten strukturierte Vorgehensweisen als Entscheidungshilfen parat zu haben.

Dabei ist allerdings auch wichtig, dass man auch die Sichtweisen der einzelnen Akteure und Projektbeteiligten dokumentiert und eventuell übersetzt. Denn das gesammelte Wissen aus Dokumenten ist nahezu nutzlos, wenn man die unterschiedlichen Kontexte nicht vereinen kann. Sollte eine einzelne Sichtweise nicht in der Knowledge Base beinhaltet sein, so kann man zumeist aus ähnlichen Kontexten auf eben diese schließen und versuchen das ursprüngliche Problem mit anderen Worten zu erklären.

Was aktuell bei persönlichen Gesprächen zwischen den Stakeholdern passiert soll nun maschinen-unterstützt fortgeführt und dadurch vereinfacht werden. Dadurch kann man zum einen die Kommunikation der Projektbeteiligten untereinander fördern und in weiterer Folge die Qualität der Software verbessern. Ein weiterer Vorteil dabei ist, dass u. U. die Zeit, die aufgewendet werden müsste um die

verschiedenen Sichtweisen der Beteiligten auf eine Linie zu bringen, drastisch reduziert werden kann.

6.3 Bewertung des Ergebnisses

In diesem Abschnitt wird zunächst auf einschlägige Fachliteratur, die bereits Gegenstand dieser Arbeit war, den wissenschaftlichen Kontext und anschließend mit einer subjektiven Bewertung anhand eigener Erfahrungen vor, während und nach Durchführung des Forschungsprojekts eingegangen. Ziel dieses Kapitels ist, eine qualitative Einschätzung der QuASE Applikation zu vermitteln, um der eingangs gestellten Forschungsfrage eine Bewertung bzw. Einschätzung geben zu können.

6.3.1 Bewertung anhand von Fachliteratur und Forschung

Unabhängig von einer Befragung kann allerdings festgestellt werden, dass die Nutzung der Decision Support Systeme und Machine Learning zweifellos zu einem besseren Ergebnis beitragen kann. Wie im vorherigen Kapitel 6.2 gezeigt, sind diese Komponenten bereits Teil der Business Information Systeme und in der Wirtschaft fest verankert. Ontologien und Knowledge Bases sind hierzu eine Weiterentwicklung, um die Vorteile von CAiSE mit dem bereits erlangten Wissen zu verknüpfen.

Wie in Kapitel 2 gezeigt, bietet auch Guizzardi eine solche Sichtweise an (vgl. [6], S. 385). Ein ähnlicher Ansatz gilt auch für die QuASE-Applikation, die für einen durchschlagenden Erfolg zu einem Framework weiterentwickelt werden sollte, um die Konzepte der Knowledge Base besser in der qualitätsbewussten Softwareentwicklung zu verankern und für die Verwendung in Unternehmen einfacher handhabbar wird.

Stuart Russell und Peter Norvig, zwei Experten im Bereich der Artificial Intelligence (AI), schreiben in [69] über *knowledge representation*, also der übergeordneten Klasse von KBs und Ontologien, dass zwar die Aussicht alles in der Welt zu repräsentieren sei entmutigend, aber das so heißt es hier weiter sei auch gar nicht das Ziel. Vielmehr sollten Platzhalter für neues Wissen, das in irgendeiner Domain passt, frei gelassen werden (vgl. [69], S. 444). Hierzu gehört auch, dass man nicht nur die technischen Aspekte, sondern auch philosophische und gesellschaftliche Werte in Betracht zieht. Denn auch wenn es z.B. technisch möglich ist, wird ein Automobil nicht vollständig ohne menschlicher Interaktion fahren, ohne dass auch die gesetzlichen Rahmenbedingungen und die Akzeptanz in der Gesellschaft vorhanden sind.

Diese Ansätze zeigen auch die Anwendungen von Knowledge Bases in den Dissertationen [23, 31], die vor allem in klassischen Industriezweigen mit Knowledge Bases arbeiten. Dass sich Konzepte aus anderen technischen Bereichen auf die Software Entwicklung übertragen lassen, zeigen die verschiedenen Softwareentwicklungsmodelle, die bereits in Kapitel 6.1 kurz vorgestellt wurden.

Aus wissenschaftlicher Sicht kann das Konzept, das in QuASE verfolgt wird, ein voller Erfolg werden, wenn sich die Grundidee in der Wirtschaft verankern lässt. Die in Kapitel 5.4 gezeigten Erweiterungsmöglichkeiten können hierfür ebenfalls einen guten Beitrag leisten. Es ist fachlich unbestreitbar, dass die Speicherung, Wiederverwendung und intelligente Verarbeitung von Wissen und vergangenen Abläufen helfen können, die Qualität von Software während der Entwicklung zu steigern.

In diversen Fachgesprächen des Autors kam jedoch bei einem Großteil zum Vorschein, dass die möglichen Erweiterungen und der Ausbau des Proof-of-Concept zu einem Framework diese Vorteile bringen können um QuASE zum Erfolg führen zu können. Im gegenwärtigen System ist es allerdings nur für die direkt beteiligten Organisationen und Unternehmen mit Vorteilen verbunden. Die Nachteile – wie Performanz und Einarbeitung zur Nutzung der ontologischen und wissensverarbeitenden Strukturen – zur Einführung in weitere Unternehmen überwiegen derzeit noch.

Kernpunkt der Forschungsfrage ist allerdings das „Wie“ und „wie gut“ können die Konzepte dabei helfen. Das „wie“ wurde in dieser Masterarbeit bereits vor allem in den Kapiteln 4 und 5 ausführlich erläutert und ist auch in der Fachliteratur (u.a. in [6, 10, 11, 12]) sehr gut beschrieben. Um das „wie gut“ zu analysieren bedürfte es auch einer statistischen Erhebung, die allerdings aus unterschiedlichen Gründen nicht vor Abschluss dieser Masterarbeit vorlag. Auch hier könnten nachgelagerte wissenschaftliche Arbeiten ansetzen, um auf diese Frage nicht nur eine subjektive Meinung, sondern auch objektiv eine Antwort finden zu können.

6.3.2 Bewertung anhand einer eigenen Reflexion

In diesem Abschnitt wird der Autor selbst, als Mitentwickler und mit der mehrjährigen Erfahrung als Softwareentwickler und IT-Mitarbeiter in mehreren Unternehmen, Stellung zu den Themengebieten „Software Qualität“ und „Tool-Unterstützung durch QuASE“ nehmen. Die Bewertung soll zusätzlich zum vorherigen Abschnitt die Beantwortung der Forschungsfrage aus Sicht von Experten in diesem Umfeld geben.

Allein die Einführung des Proof-of-Concepts in den beteiligten Unternehmen zeigt bereits eine gewisse Akzeptanz und Notwendigkeit des Forschungsgebiets der Ontologien und Knowledge Bases in den modernen Softwareentwicklungsprozessen. Auch die Aufnahme von weiteren Studien- und Abschlussarbeiten anderer Studierender und Forschender im Umfeld von QuASE und dessen Weiterentwicklung ist ein Hinweis auf diese Akzeptanz.

Es ist allerdings auch notwendig einen ganzheitlichen Blick auf die Softwareentwicklung und deren Methoden zu werfen, da das Forschungsprojekt und die Applikation um QuASE ebenfalls so entstanden sind. Die Wahl der richtigen Sofwareentwicklungsmethode (agil, klassisch oder eine Mischform) kann zu einer gleichmäßigeren Aufteilung der Aufgabengebiete, aber auch zu besserer Identifikation der einzelnen Teammitgliedern mit dem Projekt führen. So wurde beispielsweise QuASE mit der agilen Software Engineering Methode SCRUM entwickelt und dadurch eine sehr große Flexibilität in der Zuteilung der einzelnen Aufgaben, aber auch eine gewisse Identifikation des Teams mit der späteren Software erzielt.

Als IT-Fachmann und Softwareentwickler erlebt man auch in den ersten Jahren der beruflichen Praxis Rückschläge und teils gescheiterte Projekte. Oft ist der Grund nicht bei der mangelnden Motivation oder Expertise der einzelnen Team-Mitglieder zu suchen, sondern in der von DeMarco erwähnten und kritisierten „Kommunikation im weißen Raum eines Organigramms“ (vgl. [62]). Es ist ein Phänomen, das vor allem in der Kommunikation zwischen Personen und Gruppen ohne IT-Hintergrund mit den IT-Experten stattfindet. In der Ausbildung eines Informatikers bzw. in diversen Studiengängen der (angewandten) Informatik wird hierfür gerne das in Abbildung 6-2 dargestellte Bild vermittelt, um den IT-Nachwuchs bereits zu Beginn der Aus- und Weiterbildung auf dieses Problem hinzuweisen.

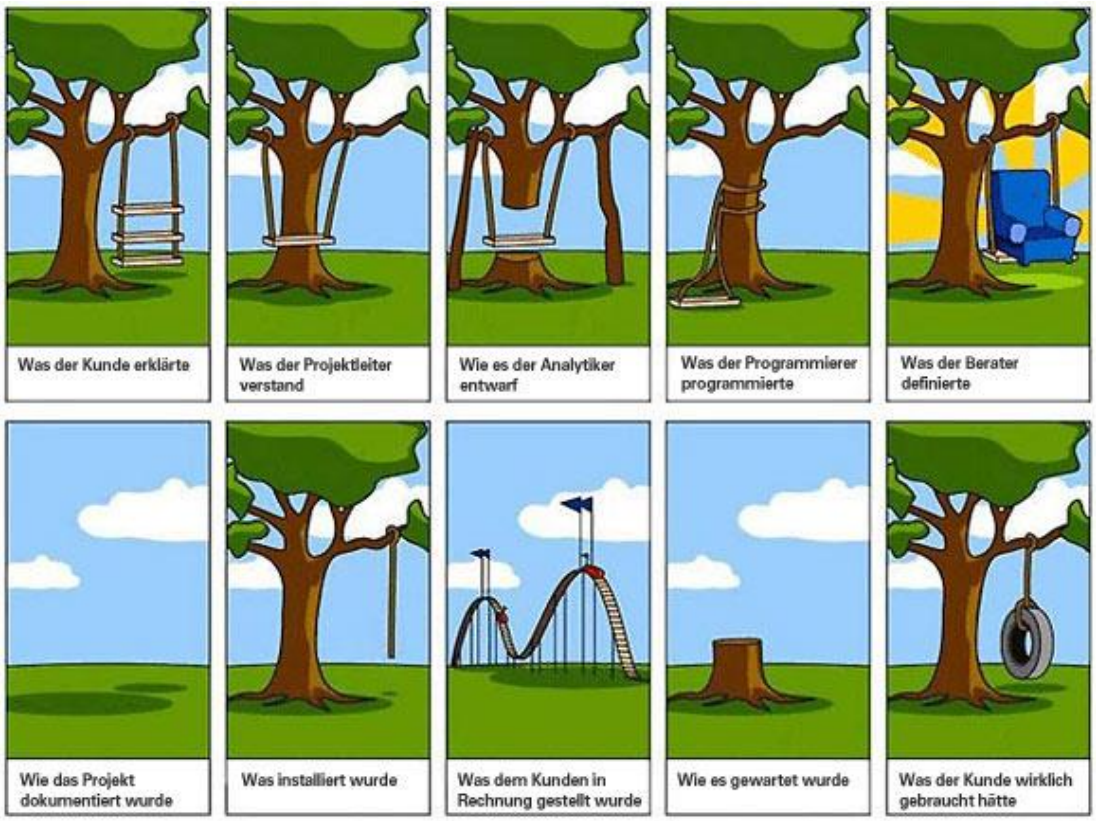

Abbildung 6-2: Kommunikationsprobleme im Projekt (aus: Vortragsfolien „Software Engineering I“ 2009 FH Landshut)

In dieser Abbildung kann man auch erkennen, welche Maßnahmen die IT-Abteilungen und Fachexperten ergreifen könnten um diesem Phänomen entgegenzuwirken. So könnten beispielsweise die Dokumentation, Installation und Wartung bereits von einer einzigen Person besser gemacht werden. Doch die Realität sieht anders aus. Knappe Budgets, zeitliche Vorgaben und Reduktion von Projektmitarbeitern sind vor allem in kleineren IT-Unternehmen oder Produktionsbetrieben mit kleinen IT-Abteilungen und in Ausnahmefällen auch in größeren Organisationen ohne IT-Hintergrund oft die Regel. Jedoch gibt es auch ohne QuASE bereits heute wirksame Maßnahmen (siehe Kapitel 6.2), die dem entgegenwirken können.

Bei den bisherigen Anstellungen und Projekten des Autors musste teilweise festgestellt werden, dass Releases und Abgaben durch die unterschiedliche Auffassung von Zielen, Anforderungen und Umfang scheiterten oder verzögert wurden. So wurden beispielsweise oft Änderungswünsche oder -aufträge mit von den EntwicklerInnen missverstandenen Anforderungen verwechselt, die Dauer von Entwicklungsphasen bei Budget- oder Projekt-Planungen drastisch reduziert oder die Ressourcen-Planung zu spät diskutiert. Diese und weitere Negativ-Beispiele sind in der Softwareentwicklung leider oft die Regel, da die IT-Mitarbeiter oft kein oder nur wenig Mitspracherecht haben, allerdings, auch im Zuge der neuen Erkenntnisse rund

um Industrie 4.0 und IoT, einen immer größer werdenden Anteil an dem Erfolg oder Misserfolg eines Projektes einnehmen.

Der größte und wichtigste Erfolgsfaktor in der heutigen Softwareentwicklung und dadurch auch der Software Qualität sind die Benutzbarkeit und Bedienbarkeit einer Software. Die genialsten Algorithmen und Applikationen bringen sehr wenig, wenn beispielsweise der spätere Benutzer bzw. die spätere Benutzerin einen erheblichen Mehraufwand in der Bedienung hat. Hierfür kann mit Applikationen wie QuASE und mit erfahrenen EntwicklerInnen, die ebenso Erfahrung als BenutzerInnen einer Software haben bzw. mit diesen gut interagieren können, Vorteile und höhere Akzeptanz in der Softwareentwicklung geschaffen werden.

6.3.3 Zusammenfassung der Bewertung

Die Beantwortung der Forschungsfrage würde sich auch mit der Erhebung von Daten aus einem Fragebogen oder Interview als nicht ganz trivial gestalten. Zum einen bedingt durch die immer komplexer werdenden Software-Systeme, die eine einfache Klassifikation von Qualität kaum zulassen, und zum anderen ist eine statistische Erhebung immer auch von dem Fachwissen der Personen und von der Quantität der Daten abhängig. Ein Hinzuziehen von Fachliteratur und Expertenwissen birgt zusätzlich die Gefahr, dass Thesen falsch interpretiert oder subjektiv behandelt werden. Dennoch könnten diese Methoden mit großer Sicherheit zusammen mit einer subjektiven Einschätzung eines Einzelnen einen großen Schritt bei der Beantwortung sein. Denn, wie in Kapitel 2 gezeigt, ist die Definition von Qualität einer Software nicht nur abhängig von der Meinung der Allgemeinheit oder mehreren Experten, sondern auch der Einzelne und dessen subjektive Einschätzung spielen hierbei eine große Rolle.

Auch dadurch, dass die Qualität von Software in Fachkreisen immer wieder thematisiert wird und Fehler in der Entwicklung passieren, sind die Fortschritte in der Entwicklung und deren Prozesse deutlich besser als noch vor einigen Jahrzehnten. Das Software Engineering muss sich jedoch stets an die neuen Gegebenheiten anpassen, denn mit immer neuer Technik und weiterführender Automatisierung wird es immer neue Entwicklungsansätze und -techniken geben, die zum einen neue Herausforderungen stellen und zum anderen ältere Standards revidieren werden. Das Wissen, also das Know-how, über diese Vorgänge und der Softwareentwicklung selbst kann über die Jahre gesehen ein wichtiger Faktor werden, um die Fehler in der Vergangenheit nicht als Last weiterzutragen, sondern aus diesen effektiv und effizient zu lernen und die richtigen Schlüsse daraus ziehen zu können.

Ein Proof-of-Concept aus einem Forschungsprojekt und wissenschaftliche Arbeiten können hierfür zumindest die Beteiligten und Leser sensibilisieren, jedoch müssten die Ansätze aus heutiger Sicht weltweit und für alle Experten standardisiert werden, um alle diejenigen zu erreichen, die sich diesen Problemen stellen müssen.

Das „Wie“ in der Forschungsfrage kann nur beantwortet werden, indem QuASE aus dem Status eines Proof-of-Concept in eine für Experten zugängliche Anwendung übergeführt wird. Dieser Übergang muss zum einen fließend und zum anderen durch mehr Relevanz und dem Erkennen der Notwendigkeit in der Fachwelt erfolgen. Hierfür wäre denkbar die Software nicht nur mit den in Kapitel 3 aufgeführten TIMS, sondern auch mit weiteren Systemen zu koppeln.

Unbestritten aus heutiger Sicht dürfte sein, dass eine Applikation wie QuASE helfen kann Missverständnisse und Probleme bei der Umsetzung eines Softwareprojekts zu lösen bzw. zu reduzieren. Hier sollte an den weisen Einsatz der Projektverantwortlichen appeliert werden, denn eine Tool-Unterstützung in einem Softwareentwicklungsprojekt kann nicht alle Probleme lösen und die Kommunikation zwischen den einzelnen Beteiligten übernehmen. Tom DeMarco hat hierzu in [62] dem Satz *„Wir haben uns verirrt, kommen aber gut voran“* ein ganzes Kapitel gewidmet. Dies zeigt, dass zunächst ein Umdenken in den Köpfen der Beteiligten stattfinden muss, bevor über die Strategien zur Lösung von Problemen gesprochen werden kann. Missverständnisse sollten am Besten sofort, wenn sie bemerkt werden angesprochen und beseitigt werden, um gemeinsam gesteckte Ziele effektiv und effizient erreichen zu können. Auch die Ziele müssen noch vor dem Start transparent formuliert werden, damit sich das komplette Team und die Projektbeteiligten darauf einstellen können. Diese und weitere Faktoren können das Risiko, dass das Softwareentwicklungsprojekt scheitern lassen könnte, reduzieren.

Eine weitere Reduktion von Fehlern und Problemen in einem SEP kann durch effektive und effiziente Entwickler-Teams, die ein Ineinandergreifen der einzelnen Tasks, ohne über Zuständigkeiten zu diskutieren, zulassen. Auch dies birgt Probleme, wenn zum Beispiel Analyse und Design oder Entwicklung und Test nicht als Gemeinschaftsaufgabe gesehen werden.

Zusammenfassend kann gesagt werden: Wenn man auch ohne Tool-Unterstützung und spezieller Software auskommen will, gibt es bereits jetzt die richtigen Methoden und Werkzeuge um ein Softwareprojekt zum Erfolg zu führen. Eine weitere Verbesserung mit Entscheidungshilfen können jedoch nur auf Knowledge-Bases basierende Softwaresysteme wie QuASE bieten. Die Vorteile dieser Systeme sind hauptsächlich der Faktor Zeit, denn wenn man ohne Tool-Unterstützung arbeitet, sollte

zumindest ein Mitarbeiter die Prozesse überwachen, während dies in den Softwaresystemen erleichtert wird und Fehlentwicklungen früher visualisiert werden können.

7 Zusammenfassung, Fazit, Ausblick

Zum Abschluss dieser Masterarbeit werden noch eine Zusammenfassung des Themas, sowie ein Ausblick für das Forschungsprojekt und ein persönliches Resümee des Autors gegeben. Außerdem werden im nächsten Abschnitt auch die Ergebnisse aus Kapitel 6 nochmal kritisch betrachtet.

7.1 Zusammenfassung

Die Qualität von Software ist seit der Softwarekrise in den 1960er-Jahren oft ein Streitpunkt unter den ExpertInnen. Da die Hardware immer kostengünstiger und Software immer komplexer wird, sollte der Fokus auf die Steigerung der Software Qualität und vor allem die qualitätsbewusste Softwareentwicklung gelegt werden. Die Schwierigkeiten dabei sind zum Einen, dass es für diesen Begriff keine eindeutige und allgemein gültige Definition gibt und zum Anderen die zum Teil sehr unterschiedlichen Sichtweisen der Stakeholder in einem Softwareentwicklungsprozess.

Die Wissenschaft der künstlichen Intelligenz und des Semantic Web erfährt vor allem in der IT-Branche immer größeren Zuspruch. Hintergrund dafür sind nicht nur die KBs und Ontologien, die immer besser entwickelt werden, sondern auch die Theorie, dass man – u.a. nach Russell und Norvig in [69] und Guizzardi in [6] – das menschliche Wissen an technische Gegebenheiten anpassen kann. Ziel dieser Forschungen ist es allerdings nicht, die komplette reale Welt in Computern darstellen zu können, sondern die Teilaspekte in einen Zusammenhang zu bringen, die es ermöglichen eine Maschine rational denken zu lassen und die Handlungen darauf anzupassen.

Das Forschungsprojekt Quality-Aware Software Engineering (QuASE) sollte die Unterschiede in den verschiedenen world views in einem SEP minimieren. Im gemeinsamen Artikel der Projektgruppe [64] wird die Begründung zu dem Forschungsprojekt beschrieben. Der originale Wortlaut lautet: *„The motivation for the project is derived from the fact that software development processes require a continuous involvement of all the affected parties including business stakeholders"* [64]. Übersetzt bedeutet das, dass das Projekt aus dem Fakt, dass Softwareentwicklungsprozesse eine kontinuierliche Eingebundenheit aller betroffenen Parteien inklusive Business Stakeholder benötigen, abgeleitet wird. Für diesen Ansatz wurden die Nutzung von Ontologien und Knowledge Bases als Kernbereich der QuASE-Applikation verwendet. Die Vorteile gegenüber den klassischen Ticketing- und Issue-Managementsystemen sind vor allem die bessere Nutzung von bereits erlangtem Wissen und die Entscheidungshilfen auf Basis von Wissensstrukturen. Ein Nachteil, vor allem für

SoftwareentwicklerInnen ist, dass dieses Wissen zunächst in spezielle Datenbanken und Datenstrukturen übergeführt werden muss, um Nutzen aus der Applikation ziehen zu können.

Kann man jedoch mit diesem Trade-Off leben und plant man eine längerfristige Verwendung des Tools, so kann vor allem die Verbesserung der Software Qualität und damit einhergehend niedrigere Kosten für die Softwareentwicklung als Gewinn für die Gesellschaft verbucht werden.

Ziel sollte es auch für alle Softwareprojekte sein, einen guten Mittelweg zwischen Qualitätsverbesserung und Kostenreduktion zu finden, da im Allgemeinen bei einer Verbesserung von Software Qualität höhere Kosten entstehen können. Eine Kostenreduktion ist ebenfalls sinnvoll, solange die grundlegenden Anforderungen in einem angemessenen Zeitrahmen durchgeführt werden können. In diesem Maße kann eine Applikation wie QuASE hilfreich sein, dass beides möglich wird und die Projektbeteiligten sich besser auf ihren Aufgabenbereich konzentrieren können, ohne mit zusätzlichem Overhead belastet zu werden. Aber auch andere *Recommender Systeme* und *Decision Support Systeme*, die nicht auf Basis einer Ontologie aufgebaut sind können hier wirksam zum Erfolg in Projekten beitragen.

7.2 Ausblick für das Projekt QuASE

Das Forschungsprojekt QuASE, bei dem ich als Autor dieser Arbeit zehn Monate als Softwareentwickler mitwirken durfte, stellt einen neuen Ansatz zur Verbesserung der Software Qualität dar. Wie bereits mehrfach erwähnt, sind die einzelnen Ansätze nicht neu, allerdings die Kombination von Modellierung, Ontologien, Knowledge Bases und Decision Support Systeme sind in dieser Form bisher noch Neuland.

Der Proof-of-Concept, also die QuASE-Applikation, könnte nicht nur für die am Projekt beteiligten Organisationen und Unternehmen, sondern für die Fachwelt eine gute Grundlage zur Verbesserung der Software Qualität werden. Hierbei sollte allerdings auch beachtet werden, dass die Applikation lediglich die Basis dieses Ansatzes ist. Um das Konzept zukunftsfähig zu machen, sollten meiner Meinung nach auch die Techniken, Technologien und Ansätze überdacht und eventuell auch mit anderen Systemen kombiniert und getestet werden. Außerdem könnten Verbesserungen und die Weiterentwicklung in andere Anwendungsgebiete einen interessanten Ansatz bieten, bei dem es sich lohnt über die Einführung der QuASE-Applikation als Framework oder Softwareprodukt nachzudenken.

Die direkt am Projekt mitarbeitenden Entwickler haben, auch wegen der zum Teil knapp bemessenen Zeit, nicht alle Ansätze, die möglich gewesen wären, bearbeiten

können. Jedoch wurde meiner Meinung nach ein sehr guter Grundstein gelegt, der zukünftigen EntwicklerInnen und WissenschaftlerInnen die Möglichkeit bietet weiter an diesen Ansätzen zu arbeiten.

Direkt nach dem Ende der ersten Projektphase im Februar 2015 wurde bereits weiter an diesem Konzept gearbeitet. Dazu wurden nach [64] bereits weitere Ziele und mögliche Weiterentwicklungen erfasst. Diese sind neben der Erweiterung auf andere TIMS als Jira auch eine inkrementelle Repository Synchronisation und weitere Verbesserungen, die QuASE stabiler machen und zur Marktreife führen sollen. Des Weiteren sollen auch Forschungen zur Verbesserung der theoretischen Konzepte und Implementierungen, die diese in die Praxis umsetzen, geprüft und vorangetrieben werden.

Diese Masterarbeit könnte dabei eine Hilfestellung zur Einarbeitung in das Thema „Ontologien und KBs im Softwareentwicklungsprozess“ sein und zur Verbesserung der Grundlagen herangezogen werden. Dies und auch den in Kapitel 5.4 gezeigten möglichen Weiterentwicklungen von QuASE sollte dringend angedacht und durchgeführt werden. Im Zuge der Entwicklungen um „Industrie 4.0“ und „Internet of Things“ könnten die Bausteine Software Qualität und Knowledge Bases wichtige Eckpfeiler werden. Eine Verschmelzung der beiden Subthemen ist meiner Meinung nach nicht nur im Softwareentwicklungsprozess in Vorbereitung, sondern auch in den Bereichen der Mensch-Maschine-Schnittstellen (engl. Human-Computer-Interface, HCI) und Maschine-Maschine-Schnittstellen, die auch als Internet der Dinge bezeichnet werden, bereits Gegenstand von Forschungsarbeiten, die im Zuge der 4. Industriellen Revolution zwar nicht direkt involviert sein wird, sondern sich eher als „Nebenprodukt“ herauskristallisiert.

7.3 Fazit des Autors

Am Ende dieser Arbeit möchte ich mich zunächst nochmals bei allen beteiligten Personen, Organisationen und Unternehmen am Forschungsprojekt für die gute Zusammenarbeit bedanken. Ich konnte während der Entwicklung an der Applikation und dem Schreiben dieser Arbeit einige sehr gute Erfahrungen machen, die für mich persönlich eine Bereicherung darstellen.

Die Mitarbeit an einem Forschungsprojekt an einer Universität war für mich persönlich eine neue Erfahrung, die es mir ermöglicht, das Zusammenspiel zwischen Wissenschaft, Technik und Wirtschaft in einem anderen Blickwinkel zu sehen und mir außerdem in sehr guter Erinnerung bleiben wird. Eine weitere Bereicherung stellte für mich das internationale Team der Forschungsgruppe dar.

Schlußendlich komme ich zu der Meinung, dass sowohl die Mitarbeit in einem Forschungsprojekt, als auch die persönlichen Erfahrungswerte mir in meinem weiteren Berufsweg eine gute Hilfe und wertvolle Anregungen für zukünftige Projekte sein werden. Allerdings möchte ich auch darauf verweisen, dass nicht nur innovative Lösungen und der Tool-unterstützte Support, sondern das Know-How generell zum Erfolg in einem Softwareentwicklungsprozess führen.

Ich denke, mit der QuASE-Applikation sollte ein neuer Abschnitt der Software Qualität und vor allem für Knowledge Bases und Modellierungen beginnen. Die Grundlagen hierfür sind geschaffen und eine Weiterführung sollte damit machbar sein. Es ist allerdings kein Garant dafür, dass dieses Technologie-Spektrum in der Gesellschaft verankert wird. Hier liegt es meiner Meinung nach auch an den ExpertInnen daraus etwas zu machen.

Für mich persönlich stellt der Ansatz, den QuASE verfolgt, eine zukunftsweisende Technologie dar, die es wert ist zumindest für den Einsatz eingehend geprüft und getestet zu werden. Denn in der Wissenschaft werden manchmal gute Ansätze in Schreibtischschubladen gesteckt um Jahre oder Jahrzehnte später neu entdeckt zu werden und in der Gesellschaft ein „Revival“ zu erleben. Dies ist allerdings manchmal problematisch, da das erworbene Know-How der anfänglichen Beteiligten teilweise oder komplett für die Wiederaufnahme von Forschung und Entwicklung verloren gehen könnte. Auch um dies zu verhindern habe ich mich entschieden, das in diesem Projekt angesammelte Wissen als Abschlussarbeit zu verfassen.

Ein weiterer Grund für mich diese Masterarbeit zu schreiben war es, auch meine bisherigen Erfahrungen in der Praxis bestmöglich in eine wissenschaftliche Arbeit einfließen zu lassen und das Wissen mit all denjenigen zu teilen, die daraus profitieren

könnten. Aus eigener Erfahrung weiß ich, dass auch in der IT zum Zwecke der Selbsterhaltung und der Erhaltung des eigenen Postens ungern das Wissen in dem Maße weitergegeben wird, das notwendig wäre um ein reibungsloses Arbeiten zu ermöglichen. Der Trend geht zwar mittlerweile in die richtige Richtung, jedoch wird in meinen Augen das Wissen noch zu wenig geteilt. Es wäre für junge Ingenieure und IT-Fachkräfte viel einfacher sich in einer neuen Umgebung zurecht zu finden, wenn man nicht immer wieder alles neu lernen muss, weil jeder bestimmte Abläufe anders aufnimmt, für mich der eigentliche Kernpunkt für Qualität. Um effektiv und effizient arbeiten zu können braucht es mehr als nur das theoretische Wissen und die Werkzeuge um seiner Arbeit nachzugehen. Man braucht Grundlagen und Hintergrundwissen, das man allerdings erst mit Eintritt in ein Unternehmen nachhaltig erweitern kann.

Appendix

A. QuOntology

Die QuOntology ist in der Description Logic (DL) aufgebaut. Auf den folgenden Seiten ist ein Auszug aus der Ontologie in QuASE dargestellt. Dadurch, dass Ontologien und KB in QuASE auch in der kleinsten Ausprägung sehr groß (33 MB als LaTeX-Datei) sind, ist es leider nicht möglich die QuOntology als kompletten Anhang zur Verfügung zu stellen. Auch die OWL-Datei ist in dieser Version mit über 700 Seiten zu groß für einen Anhang. Die vollständigen Dokumente werden auf der offiziellen Projekt-Webseite http://quase-ainf.aau.at/ zur Verfügung gestellt bzw. sind hier die Kontaktdaten der Projekt-Verantwortlichen ersichtlich, um die Einsichtnahme zu erfragen.

In der Description Logic werden einige Beschreibungen aus ausgewählten Bereichen dargestellt. Diese Bereiche sind *Classes*, *Attribute* und das korrespondierende *calculated Attribute*, *Capability* und die korrespondierende *"is able to understand" Capability*. Zusätzlich wurde die etwas kleinere Beschreibungslogik des *Integer Attribute* eingefügt.

Die OWL über QuOntology erstreckt sich in diesem Anhang über die *Object Properties*, *Individuals* und die letzten drei Beschreibungen der *general axioms*.

Auszug aus QuOntology in Description Logic (DL)-Format

Classes

Assessor-Decision__W__Relation

Assessor-Decision__W__Relation ⊑ Relation
bedeutet: alle Instanzen der Klasse *Assessor-Decision__W__Relation* sind vom Typ *Relation*

Content-Decision__W__Relation ⊑ ¬ Capability
bedeutet: alle Instanzen der Klasse *Content-Decision__W__Relation* sind **nicht** vom Typ *Capatility*

[…]

Attribute

Attribute ≡ = __INV__hasAttribute Modeling__W__Element ⊓ ∃
Attribute.hasValue Datatype http://www.w3.org/1999/02/22-rdf-syntax-nsPlainLiteral
bedeutet: *Attribute* ist äquivalent zu *__INV__hasAttribute Modeling__W__Element* und es existiert ein Wert vom Datentyp nsPlainLiteral

Calculated__W__Attribute

Calculated__W__Attribute ⊑ Attribute
bedeutet: alle Instanzen von *Calculated__W__Attribute* sind vom Typ *Attribute*

Calculated__W__Attribute ⊑ ¬ Metric
bedeutet: alle Instanzen von *Calculated__W__Attribute* sind **nicht** vom Typ *Metric*

[…]

Capability

Capability ⊑ Relation
bedeutet: alle Instanzen von *Capability* sind vom Typ *Relation*

Content-Decision__W__Relation ⊑ ¬ Assessor-Decision__W__Relation
bedeutet: alle Instanzen von *Content-Decision__W__Relation* sind nicht vom Typ *Assessor-Decision__W__Relation*

[…]

Auszug aus QuOntology in OWL-Format

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.aau.at/quase/QuOntology.owl#"
     xml:base="http://www.aau.at/quase/QuOntology.owl"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:QuOntology="http://www.aau.at/quase/QuOntology.owl#">
   <owl:Ontology rdf:about="http://www.aau.at/quase/QuOntology.owl"/>

<!--
///////////////////////////////////////////////////////////////
//
// Object Properties
//
///////////////////////////////////////////////////////////////
-->

<!--
http://www.aau.at/quase/QuOntology.owl#Comment.Contains.Comment__W__text --
>

<owl:ObjectProperty
rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.Contains.Comment_
_W__text">
      <rdfs:domain
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment"/>
      <rdfs:range
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment__W__
          text"/>
      <rdfs:subPropertyOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Content__W__
          Unit.Contains.Content__W__Unit"/>
      <owl:inverseOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#__INV__Comme
          nt.Contains.Comment__W__text"/>
</owl:ObjectProperty>

<!--
http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comment.average
__W__level__W__of__W__conflicts__W__resolved__W__with__W__explanation -->

<owl:ObjectProperty
rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comm
ent.average__W__level__W__of__W__conflicts__W__resolved__W__with__W__explan
ation">
      <rdfs:domain
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment"/>
      <rdfs:range
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment.aver
          age__W__level__W__of__W__conflicts__W__resolved__W__with__W__expl
          anation"/>
      <owl:inverseOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#__INV__Comme
          nt.hasAttribute.Comment.average__W__level__W__of__W__conflicts__W
          __resolved__W__with__W__explanation"/>
      <rdfs:subPropertyOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#hasAttribute
          "/>
</owl:ObjectProperty>
<!--
http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comment.average
__W__level__W__of__W__conflicts__W__resolved__W__with__W__translation -->
```

```
<owl:ObjectProperty
rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comm
ent.average__W__level__W__of__W__conflicts__W__resolved__W__with__W__transl
ation">
      <rdfs:domain
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment"/>
      <rdfs:range
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment.aver
          age__W__level__W__of__W__conflicts__W__resolved__W__with__W__tran
          slation"/>
      <owl:inverseOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#__INV__Comme
          nt.hasAttribute.Comment.average__W__level__W__of__W__conflicts__W
          __resolved__W__with__W__translation"/>
      <rdfs:subPropertyOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#hasAttribute
          "/>
</owl:ObjectProperty>

<!--
http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comment.average
__W__level__W__of__W__non-conflicting__W__terms -->

<owl:ObjectProperty
rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comm
ent.average__W__level__W__of__W__non-conflicting__W__terms">
      <rdfs:domain
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment"/>
      <rdfs:range
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment.aver
          age__W__level__W__of__W__non-conflicting__W__terms"/>
      <owl:inverseOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#__INV__Comme
          nt.hasAttribute.Comment.average__W__level__W__of__W__non-
          conflicting__W__terms"/>
      <rdfs:subPropertyOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#hasAttribute
          "/>
</owl:ObjectProperty>

<!--
http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comment.average
__W__level__W__of__W__unresolved__W__conflicts -->

<owl:ObjectProperty
rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comm
ent.average__W__level__W__of__W__unresolved__W__conflicts">
      <rdfs:domain
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment"/>
      <rdfs:range
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment.aver
          age__W__level__W__of__W__unresolved__W__conflicts"/>
      <owl:inverseOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#__INV__Comme
          nt.hasAttribute.Comment.average__W__level__W__of__W__unresolved__
          W__conflicts"/>
      <rdfs:subPropertyOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#hasAttribute
          "/>
</owl:ObjectProperty>

<!--
http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comment.custome
r__W__attitude -->
```

```
<owl:ObjectProperty
rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comm
ent.customer__W__attitude">
      <rdfs:domain
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment"/>
      <rdfs:range
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment.cust
          omer__W__attitude"/>
      <owl:inverseOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#__INV__Comme
          nt.hasAttribute.Comment.customer__W__attitude"/>
      <rdfs:subPropertyOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#hasAttribute
          "/>
</owl:ObjectProperty>

<!--
http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comment.max__W_
_level__W__of__W__conflicts__W__resolved__W__with__W__explanation -->

<owl:ObjectProperty
rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comm
ent.max__W__level__W__of__W__conflicts__W__resolved__W__with__W__explanatio
n">
      <rdfs:domain
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment"/>
      <rdfs:range
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment.max_
          _W__level__W__of__W__conflicts__W__resolved__W__with__W__explanat
          ion"/>
      <owl:inverseOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#__INV__Comme
          nt.hasAttribute.Comment.max__W__level__W__of__W__conflicts__W__re
          solved__W__with__W__explanation"/>
      <rdfs:subPropertyOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#hasAttribute
          "/>
</owl:ObjectProperty>

<!--
http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comment.max__W_
_level__W__of__W__conflicts__W__resolved__W__with__W__translation -->

<owl:ObjectProperty
rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comm
ent.max__W__level__W__of__W__conflicts__W__resolved__W__with__W__translatio
n">
      <rdfs:domain
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment"/>
      <rdfs:range
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment.max_
          _W__level__W__of__W__conflicts__W__resolved__W__with__W__translat
          ion"/>
      <owl:inverseOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#__INV__Comme
          nt.hasAttribute.Comment.max__W__level__W__of__W__conflicts__W__re
          solved__W__with__W__translation"/>
      <rdfs:subPropertyOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#hasAttribute
          "/>
</owl:ObjectProperty>

<!--
http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comment.max__W_
_level__W__of__W__non-conflicting__W__terms -->
```

```
<owl:ObjectProperty
rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comm
ent.max__W__level__W__of__W__non-conflicting__W__terms">
      <rdfs:domain
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment"/>
      <rdfs:range
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment.max_
          _W__level__W__of__W__non-conflicting__W__terms"/>
      <owl:inverseOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#__INV__Comme
          nt.hasAttribute.Comment.max__W__level__W__of__W__non-
          conflicting__W__terms"/>
      <rdfs:subPropertyOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#hasAttribute
          "/>
</owl:ObjectProperty>

<!--
http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comment.max__W_
_level__W__of__W__unresolved__W__conflicts -->

<owl:ObjectProperty
rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.hasAttribute.Comm
ent.max__W__level__W__of__W__unresolved__W__conflicts">
      <rdfs:domain
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment"/>
      <rdfs:range
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#Comment.max_
          _W__level__W__of__W__unresolved__W__conflicts"/>
      <owl:inverseOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#__INV__Comme
          nt.hasAttribute.Comment.max__W__level__W__of__W__unresolved__W__c
          onflicts"/>
      <rdfs:subPropertyOf
          rdf:resource="http://www.aau.at/quase/QuOntology.owl#hasAttribute
          "/>
</owl:ObjectProperty>
```

[…]

(skipped until last three descriptions of general axioms)

```
<rdf:Description>
    <rdf:type
        rdf:resource="http://www.w3.org/2002/07/owl#AllDisjointClasses"/>
    <owl:members rdf:parseType="Collection">
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.max__
           W__level__W__of__W__conflicts__W__resolved__W__with__W__transla
           tion"/>
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-
           Issue.max__W__level__W__of__W__conflicts__W__resolved__W__with_
           _W__translation"/>
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
           .max__W__level__W__of__W__conflicts__W__resolved__W__with__W__t
           ranslation"/>
    </owl:members>
</rdf:Description>

<rdf:Description>
    <rdf:type
        rdf:resource="http://www.w3.org/2002/07/owl#AllDisjointClasses"/>
    <owl:members rdf:parseType="Collection">
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment"/>
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-Issue"/>
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
           "/>
    </owl:members>
</rdf:Description>

<rdf:Description>
    <rdf:type
        rdf:resource="http://www.w3.org/2002/07/owl#AllDisjointClasses"/>
    <owl:members rdf:parseType="Collection">
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.avera
           ge__W__level__W__of__W__conflicts__W__resolved__W__with__W__exp
           lanation"/>
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.avera
           ge__W__level__W__of__W__conflicts__W__resolved__W__with__W__tra
           nslation"/>
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.avera
           ge__W__level__W__of__W__non-conflicting__W__terms"/>
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.avera
           ge__W__level__W__of__W__unresolved__W__conflicts"/>
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.custo
           mer__W__attitude"/>
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.max__
           W__level__W__of__W__conflicts__W__resolved__W__with__W__explana
           tion"/>
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.max__
           W__level__W__of__W__conflicts__W__resolved__W__with__W__transla
           tion"/>
        <rdf:Description
           rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.max__
           W__level__W__of__W__non-conflicting__W__terms"/>
```

```
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Comment.max__
   W__level__W__of__W__unresolved__W__conflicts"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-
   Issue.average__W__level__W__of__W__conflicts__W__resolved__W__w
   ith__W__explanation"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-
   Issue.average__W__level__W__of__W__conflicts__W__resolved__W__w
   ith__W__translation"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-
   Issue.average__W__level__W__of__W__non-conflicting__W__terms"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-
   Issue.average__W__level__W__of__W__unresolved__W__conflicts"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-
   Issue.customer__W__attitude"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-
   Issue.customer__W__satisfaction"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-
   Issue.max__W__level__W__of__W__conflicts__W__resolved__W__with_
   _W__explanation"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-
   Issue.max__W__level__W__of__W__conflicts__W__resolved__W__with_
   _W__translation"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-
   Issue.max__W__level__W__of__W__non-conflicting__W__terms"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-
   Issue.max__W__level__W__of__W__unresolved__W__conflicts"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira-
   Issue.progress"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
   .Provides__W__knowledge__W__to.Jira-
   Issue.user__W__attitude__W__to__W__issue"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
   .attention__W__to__W__detail"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
   .average__W__attitude__W__to__W__issue"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
   .average__W__attitude__W__to__W__project"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
   .average__W__level__W__of__W__conflicts__W__resolved__W__with__
   W__explanation"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
   .average__W__level__W__of__W__conflicts__W__resolved__W__with__
   W__translation"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
   .average__W__level__W__of__W__non-conflicting__W__terms"/>
<rdf:Description
   rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
   .average__W__level__W__of__W__unresolved__W__conflicts"/>
```

```
         <rdf:Description
            rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
            .communication__W__ability"/>
         <rdf:Description
            rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
            .max__W__level__W__of__W__conflicts__W__resolved__W__with__W__e
            xplanation"/>
         <rdf:Description
            rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
            .max__W__level__W__of__W__conflicts__W__resolved__W__with__W__t
            ranslation"/>
         <rdf:Description
            rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
            .max__W__level__W__of__W__non-conflicting__W__terms"/>
         <rdf:Description
            rdf:about="http://www.aau.at/quase/QuOntology.owl#Jira__W__user
            .max__W__level__W__of__W__unresolved__W__conflicts"/>
         <rdf:Description
            rdf:about="http://www.aau.at/quase/QuOntology.owl#Project.Is__W
            __related__W__to.Jira__W__user.user__W__attitude__W__to__W__pro
            ject"/>
         <rdf:Description
            rdf:about="http://www.aau.at/quase/QuOntology.owl#Project.finan
            cing__W__amount"/>
      </owl:members>
   </rdf:Description>
</rdf:RDF>

<!-- Generated by the OWL API (version 3.5.0) http://owlapi.sourceforge.net
-->
```

B. Source Code QuASE Applikation

In diesem Abschnitt werden die Java-Hauptklasse des Backends, die JavaScript-Hauptklasse des Web-Frontends und die Startseite als Quellcode angeführt. Technisch wäre es zwar möglich und denkbar, den kompletten Code als Anhang zur Verfügung zu stellen, allerdings würde es den Rahmen sprengen. Bei Bedarf kann der komplette Quellcode der Applikation bei einem der Betreuer der Masterarbeit angefragt werden.

Klasse QuASEApplication.java

Es folgt hier der Quellcode der Hauptklasse mit der Start-Methode für die Server-Applikation (Stand: 20.04.2015):

```
package org.aau.quase.application;

   import org.aau.quase.application.context.
            ContextSelectorHandler;
   import org.aau.quase.application.core.*;
   import org.aau.quase.application.knowledge.
            OntologicalDomainSelectorHandler;
   import org.aau.quase.application.security.
            AuthenticationHandler;
   import org.aau.quase.application.security.
            AuthorizationHandler;
   import org.aau.quase.application.tasks.TaskHandler;
   import org.aau.quase.application.security.LoginHandler;
   import org.aau.quase.application.security.LogoutHandler;
   import org.aau.quase.api.core.CoreAPI;
   import org.aau.quase.api.management.ManagementAPI;
   import org.aau.quase.api.security.SecurityAPI;
   import org.aau.quase.application.understandability.
            UnderstandabilityHandler;
   import org.aau.quase.application.users.UserHandler;
   import org.aau.quase.application.util.HttpMethod;
   import org.aau.quase.application.util.
            InternalEndPointHandler;
   import org.aau.quase.application.util.
            documents.InteractiveInfoHandler;
   import org.aau.quase.dal.utils.Pair;
   import org.aau.quase.application.documents.
            DocumentHandler;
   import org.quartz.Scheduler;
   import org.quartz.TriggerBuilder;
   import org.quartz.impl.StdSchedulerFactory;
   import spark.ModelAndView;
   import spark.Route;
   import spark.Spark;
   import spark.template.velocity.VelocityTemplateEngine;

   import java.util.*;
```

```
import static org.quartz.CronScheduleBuilder.
        cronSchedule;
import static org.quartz.JobBuilder.newJob;
import static org.quartz.core.jmx.
        CronTriggerSupport.newTrigger;
import static spark.Spark.get;
import static spark.Spark.staticFileLocation;
import static spark.SparkBase.setPort;

/**
 * Main class of the application, ran on the embedded Spark
        (Jetty) server.
 * Routes HTTP requests to CoreAPI and ManagementAPI,
        defines available HTTP methods.
 *
 * @author Vladyslav Lubenskyi
 * @see org.aau.quase.application.
        ProtocolHandler
 */

public class QuASEApplication {
  final static private String MAIN_TEMPLATE = "index.wm";
  final static private String STATIC_LOCATION = "/static";
  final static private Map<HttpMethod, SparkMethod>
        methodMap =
          new EnumMap<HttpMethod,
        SparkMethod>(HttpMethod.class);
   static {
          methodMap.put(HttpMethod.GET, Spark::get);
          methodMap.put(HttpMethod.POST, Spark::post);
          methodMap.put(HttpMethod.PUT, Spark::put);
          methodMap.put(HttpMethod.DELETE, Spark::delete);
          methodMap.put(HttpMethod.HEAD, Spark::head);
          methodMap.put(HttpMethod.TRACE, Spark::trace);
          methodMap.put(HttpMethod.CONNECT,
        Spark::connect);
          methodMap.put(HttpMethod.OPTIONS,
        Spark::options);
   }

   static private String syncSchedule =
        System.getenv("QUASE_SYNC_SCHEDULE")!=null ?
        System.getenv("QUASE_SYNC_SCHEDULE") : "0 0 0/3
        1/1 * ? *";

   static private Lock lock = Lock.NONE;
   // Add your route handlers here.
   final private Map<String, Pair<Route, Set<HttpMethod>>>
        routes =
          new HashMap<String, Pair<Route,
        Set<HttpMethod>>>() {{
                  Set<HttpMethod> supportedMethods = new
        HashSet<>();
                  supportedMethods.add(HttpMethod.POST);
                  supportedMethods.add(HttpMethod.GET);

                 put("security/login", new Pair<>(new
                 LoggingProtocolHandler(new
                 LoginHandler()), supportedMethods));
```

```
        put("security/logout", new Pair<>(new
        LoggingProtocolHandler(new
        LogoutHandler()), supportedMethods));
        put("security/authentication", new
        Pair<>(new LoggingProtocolHandler(new
        AuthenticationHandler()),
        supportedMethods));
        put("security/authorization", new
        Pair<>(new LoggingProtocolHandler(new
        AuthorizationHandler()),
        supportedMethods));
        put("ping", new Pair<>(new
        LoggingProtocolHandler(new PingHandler()),
        supportedMethods));

        put("neighborhood", new Pair<>(new
        LoggingProtocolHandler(new
        NeighborhoodHandler()),
        supportedMethods));
        put("classification", new Pair<>(new
        LoggingProtocolHandler(new
        ClassificationHandler()),
        supportedMethods));
        put("prediction", new Pair<>(new
        LoggingProtocolHandler(new
        PredictionHandler()), supportedMethods));

        put("task", new Pair<>(new
        LoggingProtocolHandler(new TaskHandler()),
        supportedMethods));

        put("user", new Pair<>(new
        LoggingProtocolHandler(new UserHandler()),
        supportedMethods));
        put("context", new Pair<>(new
        LoggingProtocolHandler(new
        ContextSelectorHandler()),
        supportedMethods));
        put("ep", new Pair<>(new
        LoggingProtocolHandler(new
        InternalEndPointHandler()),
        supportedMethods));
        put("documents", new Pair<>(new
        LoggingProtocolHandler(new
        DocumentHandler()), supportedMethods));
        put("understandability", new Pair<>(new
        LoggingProtocolHandler(new
        UnderstandabilityHandler()),
        supportedMethods));
        put("interactive_info", new Pair<>(new
        LoggingProtocolHandler(new
        InteractiveInfoHandler()),
        supportedMethods));
//      put("tabbed_view", new Pair<>(new
        LoggingProtocolHandler(new
        TabbedViewHandler()), supportedMethods));
        put("decision", new Pair<>(new
        LoggingProtocolHandler(new
        InteractiveInfoHandler()),
        supportedMethods));
```

```
                put("upload", new Pair<>(new
                OWLFileHandler(), supportedMethods));
                put("modelupload", new Pair<>(new
                XMLFileHandler(), supportedMethods));
                put("overview", new Pair<>(new
                OverviewHandler(), supportedMethods));
                put("dump", new Pair<>(new DumpHandler(),
                supportedMethods));
                put("issuecheck", new Pair<>(new
                IssueCheckHandler(), supportedMethods));
                put("buildkb", new Pair<>(new
                KBBuilderHandler(), supportedMethods));
                put("ontologicaldomains", new Pair<>(new
                LoggingProtocolHandler(new
                OntologicalDomainSelectorHandler()),
                supportedMethods));
                // TODO import the ContextAPIHandler
                //put("api/get_context", new Pair<>(new
                ContextAPIHandler(), supportedMethods));
                }};

    public static Lock getLock() {
          return lock;
    }

    public static void setLock(Lock lock) {
          QuASEApplication.lock = lock;
    }

    public static String getSyncSchedule() {
          return syncSchedule;
    }

    public static void setSyncSchedule(String syncSchedule){
          QuASEApplication.syncSchedule = syncSchedule;
    }

    /**
     * Initializes application and starts Spark
     */
    public void run() {
          String port = System.getenv("QUASE_PORT");
          if (port != null) {
            setPort(Integer.parseInt(port));
          } else {
            setPort(8080);
          }

          init();
          registerControllers();
    }
```

```
/**
 * Register routes from this.routes in Spark.
 */
private void registerControllers() {
      routes.forEach((path, pair) -> {
        Route route = pair.getKey();
               for (HttpMethod method : pair.getValue())
               methodMap.get(method).handle(path,
               route);
      });
}

/**
 * Initialize application.
 *
 * First of all defines static file location (need to be
     done before routing).
 * Then initialize API's and Root Controller.
 */
private void init() {
      // Remove this, if application is deployed to the
      servlet container
      // In case of servlets -- put you static resources
      under .../src/main/webapp
      // or configure container (web.xml) to use any
      other location.
      staticFileLocation(STATIC_LOCATION);
      SecurityAPI.init();
      CoreAPI.init();
      ManagementAPI.init();

      scheduleKBSynchronization();
      initRootController();
}

private void scheduleKBSynchronization() {

      try {
        //schedule it
        Scheduler scheduler = new
               StdSchedulerFactory().getScheduler();
        scheduler.start();
        scheduler.scheduleJob(
               newJob(KBBuilderRunner.class)
                .withIdentity("KBBuilderJob", "group1")
                .build(),
               TriggerBuilder.newTrigger()
                .withIdentity("trigger1", "group1")

      .withSchedule(cronSchedule(getSyncSchedule()))
                .build());
      } catch (Exception e) {
               e.printStackTrace();
      }
}
```

```
    /**
     * Initialize controller for / url.
     * Special case of controller, which the only one who
         returns html page.
     */
    private void initRootController() {
          get("/", (request, response) -> {
            Map<String, Object> model = new HashMap<>();
                  model.put("static", STATIC_LOCATION);
                  return new ModelAndView(model,
                   MAIN_TEMPLATE);
            }, new VelocityTemplateEngine());
    }

    public static enum Lock {
          NONE(""), MAINTENANCE("MAINTENANCE");

          private final String label;

          Lock(String label) {
            this.label = label;
          }

          public String getLabel() {
            return label;
          }
    }

    // HTTP method to Spark route method
    private interface SparkMethod {
          public void handle(String path, Route route);
    }
}
```

Klasse app.js

Es folgt hier der Quellcode der JavaScript-Hauptklasse für die Web-Anwendung (Stand: 21.04.2015):

```
/**
 * QuASE web application. This is the central angular module for
 * the whole application. It defines dependencies on every other
 * modules and URL route for the root.
 *
 */

/* Dependencies */
var app = angular.module('app', [ 'ngRoute', 'ngCookies',
    'ngSanitize', , 'ui.bootstrap', 'ur.file', 'APIModule',
    'SecurityModule', 'TaskModule', 'SectionSelector',
    'MenuModule', 'NeighborhoodModule', 'UnderstandabilityModule',
    'UserList', 'UtilModule', "ClassificationModule",
    'InternalEndPoint', 'InteractiveInfoModule', 'DecisionModule',
    'AssessmentModule', 'RecommendationSupplyModule',
    "PredictionModule"]);

/* URL routes */
app.config(function($routeProvider) {
    "use strict";
    $routeProvider.when('/', {
          templateUrl : '/static/partials/home.html',
          controller: "HomePageController"
    });
}).run(function(Session, API, $interval, $rootScope, $location) {
    "use strict";

// support for unitid in query strings
    $rootScope.externalUnitID =
         $location.search().hasOwnProperty("unitid") ?
         $location.search()["unitid"] : null;
    $rootScope.externalUnitClass =
         $location.search().hasOwnProperty("unitclass") ?
         $location.search()["unitclass"] : null;

    $rootScope.isEmbeddedMode = function () {
          return $rootScope.externalUnitID != null;
    };

    if ($rootScope.isEmbeddedMode()) {
          API.call("/documents", {command: "check_by_id", id:
    $rootScope.externalUnitID}, function (result) {
                $rootScope.unitExists = result.exists;
          });

    }

    $rootScope.isUnitExists = function () {
          return $rootScope.unitExists;
    };
```

```
    if ($rootScope.isUnitExists()) {
      API.call("/documents", {command: "by_iri", iri:
          "http://www.aau.at/quase/QuOntology.owl#" +
          $rootScope.externalUnitClass + "-" +
          $rootScope.externalUnitID}, function (result) {
                $rootScope.selectedName = result.name;
          });
    }

    $rootScope.getSelectedName = function () {
          return $rootScope.selectedName;
    };

    function validateSession() {

          API.call("/ping", {unitId : $rootScope.externalUnitID,
              unitClass : $rootScope.externalUnitClass },
              function(result) {
          if (result.sessionIsValid === false && Session.exists())
          {
                Session.destroy();
          }
          $rootScope.KB_NAME = result.kbName;
          $rootScope.KB_VERSION = result.kbVersion;
          $rootScope.KB_UPDATE_TIME = result.kbUpdateTime;
          $rootScope.EDITOR_HOST = result.editorHost;
          $rootScope.EDITOR_PORT = result.editorPort;
          $rootScope.unitExists = result.unitExists;
          $rootScope.unitName = result.unitName;
          });
    }
    validateSession();
    $interval(validateSession, 100 * 60 * 1);
    // once per 10 minutes

    $rootScope.loginLink = function() {
          return "#/login?return_to=" + $location.path();
    };

    $rootScope.lock = false;
    $rootScope.isLocked = function() {
          return $rootScope.lock;
    }
    $rootScope.iframeMode = !!$location.search()['iframe'];
    // "existence" operator

});

app.controller("HomePageController", function($scope, $rootScope,
      API) {
    $scope.file = null;
    $scope.incrementalOWL = "false";
    $scope.incrementalXML = "false";

    $scope.uploadOWL = function(uploadUrl){
          API.callFD("/upload", {"file": $scope.file,
              "incrementalOWL": $scope.incrementalOWL},
              function(result) {
                $rootScope.lock = false;
          });
```

```
    }

    $scope.uploadXML = function(uploadUrl){
        API.callFD("/modelupload", {"file": $scope.file,
            "incrementalXML": $scope.incrementalXML},
            function(result) {
                $rootScope.lock = false;
        });
    }

});

/*
 * Help tooltip for any visible dom element. To use it, first add
 * to the element class or attribute "helpable".
 * <a ... helpable>...</a> or <a class="helpable">...</a>
 * and then specify tooltip text and position (top | bottom | left
 * | right | auto)
 * <a class="helpable" tooltip-position="top" tooltip-text="Help
 * text">Link</a>
 * If tooltip-position is not specified, then it applies default
 * value "auto".
 */

app.directive("helpable", function() {
    "use strict";
    return {
        restrict: "AC",
        link: function(scope, element, attrs) {
            var el = element[0];
            if (!!attrs.tooltipText) {
                el.setAttribute("data-original-title",
                    attrs.tooltipText);
            } else {
                return;
            }
            if (!!attrs.tooltipPosition) {
                el.setAttribute("data-placement",
                    attrs.tooltipPosition);
            } else {
                el.setAttribute("data-placement", "auto");
            }
            el.setAttribute("data-toggle", "tooltip");
            $("[data-toggle='tooltip']").tooltip();
        }
    };
});

app.directive("helpableBig", function() {
    "use strict";
    return {
        restrict: "AC",
        link: function(scope, element, attrs) {
            var el = element[0];
            if (!!attrs.popoverText) {
                el.setAttribute("data-content",
                    attrs.popoverText);
            } else {
                return;
            }
```

```
                if (!!attrs.popoverFocus) {
                    el.setAttribute("data-
                        trigger",attrs.popoverFocus);
                    el.setAttribute("role","button");
                    el.setAttribute("tabindex","0");
                }
                if (!!attrs.popoverPosition) {
                    el.setAttribute("data-placement",
                        attrs.popoverPosition);
                } else {
                    el.setAttribute("data-placement", "auto");
                }
                if (!!attrs.popoverHead) {
                    el.setAttribute("title", attrs.popoverHead);
                } else {
                    return;
                }
                el.setAttribute("data-toggle", "popover");
                $("[data-toggle='popover']").popover();
            }
        };
    });

    /*
     * Directive for the application-scope error dialog.
     */

    app.directive('qalert', function() {
        "use strict";
        return {
            restrict : 'E',
            templateUrl : '/static/partials/alert_directive.html',
            scope : true,
            controller : function($scope, $rootScope, $interval) {
                $scope.show = false;
                var stop;
                $rootScope.showAlertDialog = function(text, title) {
                    $scope.show = true;
                    if (!title) {
                        $scope.title = "Error occurred";
                    } else {
                        $scope.title = title;
                    }
                    $scope.text = text;
                    stop = $interval(close, 5000, 1);
                };
                function close() {
                    $scope.show = false;
                }

                $rootScope.hideAlertDialog = close;
                $scope.close = close;
            }
        };
    });
```

Startseite home.html

Es folgt hier der Quellcode der Startseite der Applikation (Stand: 21.04.2015):

```
[...] Some developer-internal programming description is hidden!
<div>
    <div ng-if="!AuthService.isAuthenticated()&& !
        isEmbeddedMode()">
    <h1>The QuASE application</h1>

    <br/>
    </div>
    <h2 ng-if="!AuthService.isAuthenticated() ">About the
        application:</h2>
    <h3 ng-if="AuthService.isAuthenticated()">QuASE status
        information</h3>
    <table class="table table-striped">
         <tr ng-if="AuthService.isAuthenticated()">
              <td>QuASE site model</td>
              <td>{{KB_NAME}}</td>
              <td width="50%"></td>
         </tr>
         <tr ng-if="AuthService.isAuthenticated()">
              <td>Version</td>
              <td>{{KB_VERSION}}</td>
              <td></td>
         </tr>
          <tr ng-if="AuthService.isAuthenticated() &&
              isEmbeddedMode()">
              <td>Current document</td>
              <td style="color:green" ng-
                  if="isUnitExists()">{{unitName}}</td>
              <td style="color:red" ng-if="! isUnitExists()">not
                  synchronized with QuASE</td>
              <td></td>
         </tr>
         <tr ng-if="AuthService.isAuthenticated()">
              <td>Last knowledge base synchronization</td>
              <td>{{KB_UPDATE_TIME}} <button name="updatekb"
                  onclick="var ifr=document.
                      getElementById('buildkb');
                      ifr.src='/buildkb?reload'">
                      Synchronize knowledge base</button>
                       <iframe id="buildkb" src="/buildkb?init"
                           style="visibility:hidden" width="0"
                           height="0"></iframe>
              </td>
              <td></td>
         </tr>
         <tr ng-if="!AuthService.isAuthenticated()">
              <td>Further Information</td>
              <td><a href="http://quase-ainf.aau.at">http://quase-
                  ainf.aau.at</a></td>
         </tr>
```

```
        <tr ng-if="!AuthService.isAuthenticated()">
            <td></td>
            <td><p>The application of Quality-Aware Software
                Engineering (QuASE)<br>
                is designed for improving IT-related Issues and
                Tasks with translation<br>
                of contents of documents to different views of
                stakeholders (context).<br>
                It provides also an understandability management
                and shows similar project-tasks.</p>
            </td>
        </tr>
    </table>
    <div ng-if="!AuthService.isAuthenticated()">
    <h3>The QuASE Team</h3>
    <table class="table table-striped">
        <tr>
            <th>Name</th>
            <th>Function</th>
            <th>further Information</th>
        </tr>
        <tr>
            <td>Prof. Dr. Dr.h.c. Heinrich C. Mayr</td>
            <td>Project Leader</td>
            <td>Full Professor of Practical Informatics and
                leading the research group Application
                Engineering.
            </td>
        </tr>
        <tr>
            <td>Dr. phil. Volodymyr Shekhovtsov</td>
            <td>Senior Researcher</td>
            <td></td>
        </tr>
        <tr>
            <td>Vladislav Lubenskyi, B.Sc.</td>
            <td>Developer</td>
            <td>Student</td>
        </tr>
        <tr>
            <td>Sergii Ianushkevych, B.Sc.</td>
            <td>Developer</td>
            <td>Student</td>
        </tr>
        <tr>
            <td>Stefan L. Strell, B. Sc.</td>
            <td>Developer</td>
            <td>Student, Diplomand</td>
        </tr>
        <tr>
            <td>Matija Kucko</td>
            <td>Developer</td>
            <td>Student</td>
        </tr>
    </table>
```

```
    <h3>Contact</h3>
    <table class="table table-striped">
        <tr>
            <td>Address</td>
            <td>
                Alpen-Adria-Universität Klagenfurt<br>
                Institute of Applied Incormatics / AE<br>
                Universitätsstraße 65-67<br>
                A-9020 Klagenfurt am Wörthersee<br>
            </td>
        </tr>
        <tr>
            <td>Mail</td>
            <td><a href="mailto:office-ae@aau.at">office-
                ae@aau.at</a></td>
        </tr>
        <tr>
            <td>Phone</td>
            <td>+43(0) 463 2700 3703</td>
        </tr>
        <tr>
            <td>Fax</td>
            <td>+43(0) 463 2700 993703</td>
        </tr>
    </table>
    </div>
    <div style="{{!AuthService.isAuthenticated() ||
        isEmbeddedMode() ? 'visibility:hidden':''}}">
        <h3>Upload new site model XML file</h3>
        <div>The ontology will be generated automatically and the
            knowledge base will be synchronized
            <input type="file" ng-model="file"
                change="uploadXML()" />
    <!-- <input type="checkbox" ng-model="incrementalXML" />
        incremental -->
        </div>
        <h3>Reload knowledge base from OWL file</h3>
        <div>Please make sure that the current version of the site
            model in the system corresponds to the KB version you
            are going to upload!
            <input type="file" ng-model="file"
            change="uploadOWL()" />
    <!-- <input type="checkbox" ng-model="incrementalOWL" />
        incremental -->
        </div>

        <div ng-if="AuthService.isAuthenticated() && !
            isEmbeddedMode()">
            <h3>Other QuASE components</h3>
            <a target="new"
                href="http://{{EDITOR_HOST}}:{{EDITOR_PORT}}/admi
                n/editor">Terminological Knowledge Editor</a>
[...] Some commented out code is hidden (unused)!!
        </div>

    </div>
</div>
```

Glossar

Ambient Assisted Living

Ambient Assisted Living (AAL), das so viel wie „altersgerechte Assistenzsysteme für ein selbstbestimmtes Leben“ bedeutet, ist ein Forschungsgebiet mit dem Ziel, technische Unterstützung von älteren und gesundheitlich beeinträchtigten Menschen zu gewähren, damit diese weiterhin in gewohnter Umgebung möglichst autonom leben können. AAL ist eines der Forschungsgebiete der Forschungsgruppe Application Engineering[31] an der AAU Klagenfurt. Weitere Informationen über dieses und andere Forschungsgebiete und -projekte, bietet der Link in der Fußnote.

Application Program Interface (API)

Eine API ist eine Programm-Schnittstelle, die direkt in einen Java-Code eingebunden werden kann. Umgangssprachlich werden diese APIs auch Programm-Bibliotheken genannt, deren Inhalte den EntwicklerInnen nach der Einbindung zur Verfügung stehen. Manche APIs stehen als ausführbare Komponente den Endnutzern als sogenannte Plug-Ins zur Verfügung.

Applikation

Als Applikation wird die Gesamtheit der ausführbaren Programmkonstrukte bezeichnet. Hierbei wird in der Regel zwischen Client/Server-Applikation und Standalone-Applikation unterschieden. Als Client/Server-Applikation werden verteilte Systeme, deren Applikationsteile über ein Netzwerk miteinander kommunizieren, bezeichnet, wohingegen Standalone-Applikationen auf einer physischen Maschine ausgeführt werden.

Artificial Intelligence

Artificial Intelligence (AI), im Deutschen als künstliche Intelligenz (KI) genannt, bezeichnet ein Forschungsgebiet der Informatik, in dem Maschinen und Computer das menschliche Wesen nachempfinden sollen. Eines der Teilgebiete darin ist Wissen in Form von Knowledge Bases für Maschinen lesbar zu machen, um dieses in weiterer Folge verarbeiten zu können.

[31] Application Engineering: http://www.uni-klu.ac.at/tewi/inf/ainf/ae/2504.htm

Computer Aided Software Engineering

Computer Aided Software Engineering (CAiSE) beschreibt den Ansatz der Software Entwicklung mit computer-unterstützten Hilfsmittel zur Entscheidungsfindung und bestimmten Tool-Support, wie Modellierung von Software-Systemen. Sommerville beschreibt CAiSE als *„ein weites Gebiet von verschiedenen Programmtypen, die benutzt werden, um Softwareprozessaktivitäten […] zu unterstützen“* [61].

Content

Als Content werden im Sinne von QuASE alle projektspezifischen Dokumente bezeichnet. Hierzu zählen z.B. die Kommunikation und Spezifikationen oder Change Requests.

Content Unit

Die Content Unit ist eine Einheit zur Darstellung spezieller Inhalte einer Knowledge Base. Sie bezeichnet in QuASE eine Einheit mit speziellen Inhalten für ein Projekt. So kann der Content, in einem Projekt klassifiziert und mit Metriken ausgestattet, verarbeitet werden.

Context

Als Kontext (engl. context) wird im ontologischen Sinne die domänenspezifische Komponente verstanden. Diese kann, abhängig von der Verwendung, als Organisationseinheit, Personengruppe oder Person verstanden werden.

Context Unit

Die Context Unit stellt in QuASE eine Einheit einer domänenspezifischen Komponente dar. Diese wird verwendet um in der QuASE KB Interaktionen zwischen den einzelnen Organisationseinheiten und Personen zulassen und vor allem aufzeigen zu können.

Data Access Layer (DAL)

Als Data Access Layer werden die Module und Prozeduren einer Software bezeichnet, die als Schnittstelle und Zugriffsebene zwischen der Applikationslogik und den Rohdaten, z.B. in einer Datenbank, verwendet werden.

Data-Mining

Data-Mining ist ein Teilbereich der Knowledge Bases und Artificial Intelligence, bei dem aus einer großen Menge von Daten (Big-Data) mit speziellen Analyse-Algorithmen Wissen extrahiert und teilweise aufbereitet wird. Hierbei werden keine neuen Daten erhoben oder erzeugt, sondern auf Basis der sog. Big-Data Wissen und

Querverbindungen erstellt, die wiederum Auskunft auf bestimmte Vorgehensweisen oder Metriken geben können und zur weiteren Verwendung im „Machine Learning“ dienen.

Datenbank

Datenbanken, im technischen Sinne als Datenbanksysteme (DBS) bekannt, sind Speichersysteme für alle Art von elektronischen Daten. Hierbei gibt es für die verschiedenen Verwendungsarten unterschiedliche Grundformen. Die am meisten verwendete sind die relationalen Datenbanken (RDBs). Weitere Arten sind triviale Datenbanken (TDB) und sog. NoSQL-Datenbanken, wie MongoDB. Zu einem DBS gehört in der Regel auch ein Datenbankmanagementsystem (DBMS).

Datenbankmanagementsystem

Ein Datenbankmanagementsystem (DBMS) ist die Verwaltungseinheit eines DBS. Hier werden Anfragen an die Datenbank, sowie Speicher- und Benutzermanagement verwaltet. Ein DBMS enthält typischerweise eine Administrationsschnittstelle, die es den AnwenderInnen bzw. ProgrammiererInnen erlaubt auf die Management-Aufgaben zuzugreifen.

Decision Support

Als Decision Support versteht man computer-unterstützte Entscheidungshilfen für bestimmte Tätigkeiten. Im Allgemeinen werden hierzu bestimmte Möglichkeiten aufgezeigt, oder eine Reaktion mit einem bestimmten Prozentwert auf eine Handlung berechnet. Im Kapitel 4.1.3 wird eine konkrete Anwendung dieses Algorithmus gezeigt.

Domain Specific Language (DSL)

Eine Domain Specific Language (deutsch: domänenspezifische Sprache) ist eine formale Sammlung von sprachähnlichen oder modellartigen Konstrukten, um ein bestimmtes Problemfeld oder eine Domäne zu beschreiben. In der Informatik werden diese häufig zur Beschreibung von Problemen aus der realen Welt und zur Übersetzung dieser für die computer-unterstützte Bearbeitung verwendet.

Domain Specific Modelling Language (DSML)

Als Domain Specific Modelling Languages (deutsch: domänenspezifische Modellierungssprachen) wird jene Teilmenge von DSLs bezeichnet, die ein bestimmtes Problemfeld mit Modellen und Meta-Modellen beschreiben.

Early-Bird-Entwicklung

Eine Early-Bird-Entwicklung bezeichnet in der angewandten Informatik ein Softwareentwicklungsprojekt, das in wenigen Wochen oder Monaten nach der Freigabe zur Entwicklung zumeist in einer reduzierteren Fassung den AuftraggeberInnen zur Verfügung gestellt wird. Zumeist werden hierfür die SWE-Konzepte des *Rapid Prototyping* oder agile Softwareentwicklungsmethoden verwendet.

Framework

Als Framework bezeichnet man in der Informatik ein Software-Tool, mit dem entweder die Softwareentwicklung einfacher wird, oder verschiedene Funktionalitäten in einem Paket für die verschiedensten Anwendungsbereiche zur Verfügung gestellt werden. Als Beispiel für diese Art von Software-Tools dienen Eclipse, IntelliJ IDE, Protégé, Visual Studio von Microsoft u.v.a.

Human Behavior Monitoring and Support

Human Behavior Monitoring and Support (HBMS) ist ein Forschungsprojekt des Instituts für Angewandte Informatik an der AAU Klagenfurt innerhalb des Forschungsgebiets des Ambient Assisted Living (AAL). HBMS soll in diesem Fall eine kognitive Unterstützung sein, um z.B. die Bedienung einer Kaffeemaschine oder eines Fernsehgerätes zu erleichtern.

Industrie 4.0

Der Begriff „Industrie 4.0“ wurde im Zusammenhang mit der Automatisierung von Produktionslinien und -anlagen vom deutschen Bundesministerium für Bildung und Forschung (BMBF) geprägt. Die beiden wichtigsten Bausteine für dieses „Zukunftsprojekt“ sind zum einen die Errungenschaften im Internet der Dinge und zum anderen die Forschung im Bereich „Smart Factory“, also der intelligenten Fabrik. (vgl. [71])[32]

Information Technology Infrastructure Library

Die Information Technology Infrastructure Library (ITIL) ist eine Sammlung von Best Practices zur Verbesserung der Geschäftsprozesse rund um die IT-Infrastruktur. Der Vorläufer aus den 1980er Jahren war das „Government Information Technology Infrastructure Management“, eine Standardisierung der IT-Infrastruktur in der britischen Regierung. Seit Beginn des 21. Jahrhunderts gibt es eine Reihe großer

[32] Projekt-Definition des deutschen Bundesministeriums für Bildung und Forschung: https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html

Unternehmen, die ITIL verwenden, wie z.B. der Software-Hersteller Microsoft oder der Hardware-Produzent Hewlett Packard (HP). (vgl. [70])

Internet der Dinge

Das Internet der Dinge (engl. Internet of things, IoT) bezeichnet die Technologie der eingebetteten Computersysteme in alltäglichen Gegenständen, die über ein Netzwerk miteinander verbunden werden und dadurch kommunizieren können. Diese Form des Internets wird auch als *Web 4.0* bezeichnet.

Machine Learning

Der Begriff Machine Learning umfasst die Algorithmen, die es einem Computersystem ermöglichen, selbstständig aus Verhaltensmustern weitere Vorgehensweisen und Prozesse zu erlernen. Dieser Begriff wird durch die Konzepte von Ontologien, Knowledge Bases und Semantic Web geprägt und ist Teil der künstlichen Intelligenz und von Decision Support Systemen.

Plug-In

Als Plug-In versteht man eine Software, die innerhalb eines Frameworks oder einer Applikation zum Einsatz kommt. Diese Plug-Ins bieten in der Regel spezielle Werkzeuge oder Methoden an, um die Bedienung der Applikation zu erleichtern oder zu beschleunigen.

Proof-of-Concept

Als Proof-of-Concept (deutsch: Beweis des Konzepts oder auch Konzeptbeweis) bezeichnet man in der Softwareentwicklung einen Prototyp oder ein Konzept aus der wissenschaftlichen universitären Forschung. In der Regel ist ein Proof-of-Concept vor der Marktreife eines möglichen Produkts einzuordnen.

Qualitätsbezogene Kommunikation

Unter der qualitätsbezogenen Kommunikation werden im Kontext dieser Arbeit die Gespräche und schriftliche Kommunikation verstanden, die zur Qualitätsüberprüfung von Projekten und Projektaufgaben dienen.

Recommender Systeme

Als Recommender System versteht man im Fachbereich der Informatik ein Softwaresystem, das Entscheidungen auf Basis von Empfehlungen erleichtert und mit bestehenden Daten eine Quantifizierung vornimmt. Zu solchen Technologien zählen auch Data Mining und anwendungsspezfische Applikationen auf Basis großer Datenbanksysteme.

Semantic Web

Das Semantic Web ist eine Erweiterung des World Wide Web aus dem Ende des 20. Jahrhunderts. Nach Berners-Lee, Hendler und Lassila ist es eine neue Form der Web-Inhalte, die für Computer aussagekräftig sind und eine Revolution von neuen Möglichkeiten entfesseln (vgl. [63]). In Fachkreisen wird das Semantic Web auch als Synonym zu *Web 3.0* verwendet und als Vorgänger zum sog. Internet der Dinge bezeichnet. Kerngebiet des Semantic Web ist die sog. Knowledge Representation, also die Darstellung von Wissen.

Softwareentwicklungsprozess

Als Softwareentwicklungsprozess (SEP), in mancher Fachliteratur auch Softwareprozess genannt (u.a. [61]), versteht man nach Sommerville „eine Menge von Tätigkeiten […] zur Produktion eines Softwareprodukts“ [61]. Das bedeutet nicht immer eine komplette Neuentwicklung, sondern schließt auch Software Reengineering, sowie Wartung und Weiterentwicklung ein.

Softwaresystem

Als Softwaresystem wird die Gesamtheit der ausführbaren und nichtausführbaren Teile einer Software bezeichnet. Hierzu gehören in der Regel die Applikation, Datenbanken und weitere für die Ausführung notwendige Teile, die für die BenutzerInnen nicht direkt sichtbar sind.

Spezifikation

Als Spezifikation in der IT versteht man ein oder mehrere Dokumente, die sich aus den Anforderungen der AuftraggeberInnen und den Machbarkeitsanalysen des Entwicklungsteams ergeben. Die Spezifikation ist ein für den gesamten Softwareentwicklungsprozess gültiges Dokument und wird im wirtschaftlichen Bereich auch als Vertragsgrundlage verwendet.

Stakeholder

Als Stakeholder werden alle Personen und -gruppen bezeichnet, die in irgendeiner Beziehung zu einem Projekt stehen. Im Allgemeinen sind dies alle Personen und Projektbeteiligten, die entweder zur Durchführung benötigt werden oder im Projektmanagement bzw. als Kunden oder Kundenbeauftragte an der Fertigstellung des Projektes interessiert sind. In dieser Arbeit steht Stakeholder auch als Synonym für die aktiven und passiven Akteure im Softwareentwicklungsprozess.

World view

Als world view wird in dieser Arbeit die Sichtweise aus einer bestimmten Perspektive verstanden. Der world view ist abhängig von Ausbildung, Erfahrung und persönlichen Aspekten der jeweiligen Person oder Personengruppe.

Abkürzungsverzeichnis

AAL	**A**mbient **A**ssisted **L**iving
AAU	**A**lpen-**A**dria-**U**niversität Klagenfurt
AI	**A**rtificial **I**ntelligence (= künstliche Intelligenz)
AINF	Institut für **A**ngewandte **Inf**ormatik
API	**A**pplication **P**rogramming **I**nterface (= Programmierschnittstelle)
BNF	**B**ackus-**N**aur-**F**orm
BPMN	**B**usiness **P**rocess **M**odelling **N**otation
CAiSE	**C**omputer **Ai**ded **S**oftware **E**ngineering (= computer-unterstützte Softwareentwicklung)
CSS	**C**ascading **S**tyle **S**heet
DAL	**D**ata **A**ccess **L**ayer
DB	**D**ata**b**ase (= Datenbank)
DBMS	**D**ata**b**ase **M**anagement **S**ystem
DBS	**D**ata**b**ase **S**ystem
DL	**D**escription **L**ogic (= Beschreibungslogik)
DSL	**D**omain **S**pecific **L**anguage
DSML	**D**omain **S**pecific **M**odelling **L**anguage
FFG	Österreichische **F**orschungs**f**örderungs**g**esellschaft
GUI	**G**raphical **U**ser **I**nterface
HCI	**H**uman-**C**omputer **I**nterface (= Mensch-Maschine-Schnittstelle)
HBMS	**H**uman **B**ehavior **M**onitoring and **S**upport
IEC	**I**nternational **E**lectrotechnical **C**ommission
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers

IMS	**I**ssue **M**anagement **S**ystems
IRDS	**I**nformation **R**esource **D**ictionary **S**ystem
IRI	**I**nternationalized **R**esource **I**dentifier
ISO	**I**nternational **O**rganization for **S**tandardization
ITIL	**I**nformation **T**echnology **I**nfrastrucure **L**ibrary
KB	**K**nowledge **B**ase
KBMS	**K**nowledge **B**ase **M**anagement **S**ystem
MDE	**M**odel **D**riven **E**ngineering
NEMO	**N**ext Generation **E**nterprise **Mo**delling
NFR	**N**on-**F**unctional **R**equirement (= nicht-funktionale Anforderung)
OMG	**O**bject **M**anagement **G**roup
OWL	**O**ntology **W**eb **L**anguage
QuASE	**Qu**ality-**A**ware **S**oftware **E**ngineering
RDB	**r**elationale **D**aten**b**ank
RDBMS	**R**elationales **DBMS** (i.d.R. SQL-Datenbank-Systeme, wie ORACLE, MySQL, etc.)
RDF	**R**esource **D**escription **F**ramework
RDFS	**RDF S**chema
RQ	**R**esearch **Q**uestion (Forschungsfrage)
SEP	**S**oftware**e**ntwicklungs**p**rozess
SPARQL	**S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage
SPICE	**S**oftware **P**rocess **I**mprovement and **C**apability d**E**termination
SQL	**S**tructured **Q**uery **L**anguage
SWEBOK	**S**oft**w**are **E**ngineering **B**ody **o**f **K**nowledge
SWRL	**S**emantic **W**eb **R**ule **L**anguage
TDB	**T**rivial **D**ata**b**ase (auch als Triple-Store bezeichnet)

TDD	**T**est-**D**riven **D**evelopment (testgetriebene Softwareentwicklung)
TIMS	**T**icketing- & **I**ssue-**M**anagement-**S**ystem
TT4J	**T**ree**T**agger **for J**ava
UFO	**U**nified **F**oundational **O**ntology
UI	**U**ser **I**nterface (= Benutzeroberfläche)
UML	**U**nified **M**odeling **L**anguage
URI	**U**niform **R**esource **I**dentifier
W3C	**W**orld **W**ide **W**eb **C**onsortium
XML	E**x**tensible **M**arkup **L**anguage

Literaturverzeichnis

[1] E. Wallmüller, *Software-Qualitätsmanagement in der Praxis*. München (GER): Carl Hanser, 2001.

[2] J. Cadle and D. Yeates, *Project Management for Information Systems*, 5th Edition ed.: Pearson/Prentice Hall, 2008.

[3] D. W. Hoffmann, *Software-Qualität*, 2nd Edition ed. Berlin, Heidelberg (GER): Springer, 2013.

[4] H.-J. Habermann and F. Leymann, *Repository - Eine Einführung*. München (GER), Wien (AUT): R. Oldenbourg, 1993.

[5] L. Chung and J. C. S. do Prado Leite, "On Non-Functional Requirements in Software Engineering," *Mylopoulus Festschrift, LNCS 5600*, p. 16, 2009.

[6] G. Guizzardi, "Ontological foundations for structural conceptual models," PhD Dissertation, Centre for Telematics and Information Technology, University of Twente, Enschede (NED), 2005.

[7] B. Brügge and A. H. Dutoit, *Objektorientierte Softwaretechnik, Deutsche Übersetzung*. München (GER): Pearson Studium, 2004.

[8] H. C. Mayr, "Software Qualität: Nur eine Frage des Software Engineering?," in *ISO 9000 - Softwareentwicklung ; Ethik, Analysen, Tools ; Beiträge vom adi QM/IT Expertentreffen 1994*, ed. Münster (GER) [u.a.]: Norbert Ruppenthal, 1995, p. 49 ff.

[9] H. C. Mayr and V. A. Shekhovtsov. (2014, 10.10.2014). *QuASE - About the Project* [Website]. Available: http://quase-ainf.aau.at/

[10] V. A. Shekhovtsov and H. C. Mayr, "Towards Managing Understandability of Quality-Related Information in Software Development Processes," 2014.

[11] S. Staab and R. Studer, *Handbook on Ontologies*. Berlin, Heidelberg (GER): Springer, 2004.

[12] D. Gašević, D. Djurić, and V. Devedžić, *Model Driven Engineering and Ontology Development*, 2nd Edition ed. Berlin, Heidelberg (GER): Springer, 2009.

[13] S. Brockmans, J. Jung, and Y. Sure, *Meta-Modelling and Ontologies*. Karlsruhe, Bonn (GER): Gesellschaft für Informatik (GI), 2006.

[14] V. A. Shekhovtsov and H. C. Mayr, "Managing Quality Related Information in Software Development Processes," 2014.

[15] Webster. (2014, 29.09.2014). *Ontology in Merriam Webster Dictionary* [Website, Dictionary]. Available: http://www.merriam-webster.com/dictionary/ontology

[16] I. Kant, *Kritik der reinen Vernunft*, B-1 ed. Hamburg (GER): Felix Meiner.

[17] I. Kant. *Kritik der reinen Vernunft*. Available: http://www.sapientia.ch/E-Buecher/Philosophie/Kant%20-%20Kritik%20der%20reinen%20Vernunft.pdf

[18] G. H. Mealy, "Another Look at Data," *Proceedings of the Fall Joint Computer Conference, November 14-16, Anaheim, California (AFIPS Conference Proceedings)*, vol. 31, 1967.

[19] C. Atkinson, M. Gutheil, and K. Kiko, "On the Relationship of Ontologies and Models," in *Meta-Modelling and Ontologies*, S. Brockmans, J. Jung, and Y. Sure, Eds., ed Karlsruhe, Bonn (GER): Gesellschaft für Informatik (GI), 2006.

[20] T. R. Gruber, "A translation approach to portable ontology specifications," Knowledge System Laboratory, Stanford University, 1993.
[21] F. Baader, I. Horrocks, and U. Sattler, "Description Logics," in *Handbook on Ontologies*, S. Staab and R. Studer, Eds., ed Berlin, Heidelberg (GER): Springer, 2004.
[22] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins, "What are Ontologies, and why do we need them?," *Intelligent Systems and their Applications, IEEE (Volume:14 , Issue: 1)* 1999.
[23] S. Fuchs, "A comprehensive Knowledge Base for Context-Aware Tactical Driver Assistance Systems," PhD Dissertation, Fakultät für Technische Wissenschaften, Alpen-Adria-Universität Klagenfurt, Klagenfurt am Wörthersee (AUT), 2008.
[24] G. Klyne and J. J. Carroll. (2004, 07.10.2014). *Resource Description Framework (RDF) Concepts and Abstract Syntax* [Website]. Available: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
[25] B. McBride, "The Resource Description Framework (RDF) in its Vocabulary Description Language RDFS," in *Handbook on Ontologies*, S. Staab and R. Studer, Eds., ed Berlin, Heidelberg (GER), Bristol (UK): Springer, 2004.
[26] G. Antoniou and F. van Harmelen, "Web Ontology Language: OWL," in *Handbook on Ontologies*, S. Staab and R. Studer, Eds., ed Berlin, Heidelberg (GER): Springer, 2004.
[27] M. Horridge and P. F. Patel-Schneider. (2012, 04.11.2014). *OWL 2 Web Ontology Language - Manchester Syntax (Second Edition)* [Website]. Available: http://www.w3.org/TR/owl2-manchester-syntax/
[28] W3C. (30.09.2014). *SPARQL 1.1 Overview* [Website]. Available: http://www.w3.org/TR/sparql11-overview/
[29] W3C. (06.10.2014). *About World Wide Web Consortium (W3C)* [Website]. Available: http://www.w3.org/
[30] J. Angele and G. Lausen, "Ontologies in F-logic," in *Handbook on Ontologies*, S. Staab and R. Studer, Eds., ed Berlin, Heidelberg (GER): Springer, 2004.
[31] A. Felfernig, "Improving the Configuration Knowledge Base Development Process," PhD Dissertation, Fakultät für Witschaftswissenschaften und Informatik, Alpen-Adria-Universität Klagenfurt, Klagenfurt am Wörthersee (AUT), 2001.
[32] R. A. Slaughter, "The knowledge base of futures studies as an evolving process," *Futures, 1996 Nov*, vol. 28, pp. 799-812, 1996.
[33] G. Frank, A. Farquhar, and R. Fikes, "Building a Large Knowledge Base from a Structured Source," *Intelligent Systems and their Applications, IEEE*, vol. 14, pp. 47-54, 6. Aug. 2002 1999.
[34] J. Mylopoulos, "On Knowledge Base Management Systems," in *On Knowledge Base Management Systems - Integrating Artificial Intelligence and Database Technologies*, M. L. Brodie and J. Mylopoulos, Eds., ed New York (USA) [u.a.]: Springer, 1986, pp. 3-8.
[35] M. L. Brodie and J. Mylopoulos, "Knowledge Bases vs Databases," in *On Knowledge Base Management Systems - Integrating Artificial Intelligence and Database Technologies*, M. L. Brodie and J. Mylopoulos, Eds., ed New York (USA) [u.a.]: Springer, 1986, pp. 83-86.

[36] A. T. Schreiber, B. J. Wielinga, R. de Hoog, H. Akkermans, and W. van de Velde, "CommonKADS: A Comprehensive Methodology for KBS Development," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, pp. 28-37, 1994.
[37] *Information technology – Software product quality – Part 1: Quality Model*, ISO/IEC ISO/IEC 9126-1:2001, 2000/2001.
[38] E. W. Dijkstra. (1972, 14.10.2014). The humble programmer. Available: http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD340.PDF
[39] B. Kitchenham and S. L. Pfleeger, "Software Quality: The elusive target," *Software, IEEE*, vol. 13, pp. 12 - 21, 1996.
[40] Atlassian. (30.09.2014). *About Atlassian Jira* [Website]. Available: www.atlassian.com/de/software/jira
[41] Atlassian. (30.09.2014). *Atlassian Company Information* [Website]. Available: www.atlassian.com/de/company
[42] Atlassian. (30.09.2014). *JIRA Online Documentation* [Online Documentation (Confluence Page)]. Available: http://confluence.atlassian.com/display/JIRA/JIRA+Documentation
[43] Groiss Informatics GmbH. (2014, 26.09.2014). *@enterprise Business Process Management* [pdf-File]. Available: http://www.groiss.com/products/enterprise_de.pdf
[44] The MantisBT Team. (02.10.2014). *Mantis Bug Tracker Administration Guide* [Webpage]. Available: http://www.mantisbt.org/docs/master-1.2.x/en/administration_guide/
[45] BOC-Group. (2014, 21.10.2014). *ADOxx Documentation* [Online Documentation]. Available: http://www.adoxx.org/live/adoxx-documentation
[46] The Apache Software Foundation. (2011, 18.11.2014). *Getting started with Apache Jena* [Website]. Available: https://jena.apache.org/getting_started/index.html
[47] The Apache Software Foundation. (20.11.2014). *Getting started with Apache Shiro* [Website]. Available: http://shiro.apache.org/get-started.html
[48] The Apache Software Foundation. (20.11.2014). *Apache Mahout* [Website]. Available: https://mahout.apache.org/
[49] P. Wendel. (24.11.2014). *About Spark* [Website]. Available: http://sparkjava.com/
[50] T. Saloranta. (2009, 01.12.2014). *Jackson JSON Processor Wiki* [Website]. Available: http://wiki.fasterxml.com/JacksonHome
[51] Google Inc. (2010, 01.12.2014). *Introduction in AngularJS* [Website]. Available: https://docs.angularjs.org/guide/introduction
[52] Google Inc. (2010, 01.12.2014). *AngularJS* [Website]. Available: https://angularjs.org
[53] unknown. (04.12.2014). *Documentation of OWL API*. Available: https://github.com/owlcs/owlapi/wiki/Documentation
[54] R. Eckart. (02.12.2014). *About TreeTagger for Java* [Website]. Available: https://code.google.com/p/tt4j/
[55] R. Eckart. (02.12.2014). *Documentation of TreeTagger for Java* [Website]. Available: https://code.google.com/p/tt4j/wiki/Usage
[56] H. Schmid. (02.12.2014). *TreeTagger - a language independent part-of-speech tagger*. Available: http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

[57] H. Schmid, "Probabilistic Part-of-Speech Tagging Using Decision Trees," *Proceedings of International Conference on New Methods in Language Processing, Manchester (UK),* 1994.
[58] D. Saff, K. Cooney, S. Birkner, and M. Philipp. (24.11.2014). *About JUnit* [Website]. Available: http://junit.org/index.html
[59] unknown. (26.11.2014). *Getting started with the JMockit toolkit* [Website]. Available: http://jmockit.github.io/gettingStarted.html
[60] S. Strell, "Software Engineering nach der „Extreme Programming"- Methode zum Zwecke der Verbesserung der Anlagenauslastung mit Hilfe der linearen Optimierung," Bachelor of Science Bachelor thesis, Fakultät Informatik, Hochschule für Angewandte Wissenschaften Landshut, Landshut (GER), Villach (AUT), 2011.
[61] I. Sommerville, *Software Engineering, Deutsche Übersetzung* vol. 8. Auflage. München (GER): Addison-Wesley, Pearson Studium, 2007.
[62] T. DeMarco, *Spielräume*. München (GER), Wien (AUT): Carl Hanser, 2001.
[63] T. Berners-Lee, J. Hendler, and O. Lassila. (2001, 05.03.2015). *The Semantic Web*.
[64] V. A. Shekhovtsov, H. C. Mayr, S. Ianushkevych, M. Kucko, V. Lubenskyi, and S. Strell, "Implementing Tool Support for Effective Stakeholder Communication in Software Development – A Project Report", unpublished.
[65] V. A. Shekhovtsov, H. C. Mayr, and C. Kop, "Facilitating Effective Stakeholder Communication in Software Development Processes" unpublished.
[66] V. A. Shekhovtsov, H. C. Mayr, and M. Kucko, "Implementing Tool Support for Analyzing Stakeholder Communications in Software Development," IEEE Eight International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2015.
[67] V. A. Shekhovtsov, H. C. Mayr, and C. Kop, "Towards Conceptualizing Quality-Related Stakeholder Interactions in Software Development," in Information Systems: Methods, Models, and Applications, H. C. Mayr, C. Kop, S. Liddle, and A. Ginige, Eds., ed Berlin, Heidelberg (GER): Springer, 2013, pp. 73-86.
[68] V. A. Shekhovtsov, "Understandability management for quality-related communicated information in the software process: a generic description," Institute of Applied Informatics, Alpen-Adria-Universität Klagenfurt, http://quase-ainf.aau.at/ January 30, 2014.
[69] S. Russell and P. Norvig, Artificial Intelligence - A Modern Approach, 3 ed. Essex (GB): Pearson Education, 2014.
[70] U. Schultze, "Finding the process edge: ITIL at Celanese," Journal of Information Technology Teaching Cases, p. 18, 2010.
[71] Projektträger im Deutschen Zentrum für Luft- und Raumfahrt e.V. Softwaresysteme und Wissenstechnologien, "Zukunftsbild "Industrie 4.0"," Deutsches Zentrum für Luft- und Raumfahrt e.V. Softwaresysteme und Wissenstechnologien, Ed., ed. https://www.bmbf.de/pub/Zukunftsbild_Industrie_40.pdf: Bundesministerium für Bildung und Forschung, 2013, p. 35.
[72] M. Abufouda, "Quality-aware Approach for Engineering Self-adaptive Software Systems," 2014.