EINSATZ SEMANTISCHER TECHNOLOGIEN FÜR DIE ANFORDERUNGSANALYSE

I n a u g u r a l d i s s e r t a t i o n

zur Erlangung des akademischen Grades eines Doktors der Wirtschaftswissenschaften der Universität Mannheim

vorgelegt

von

Simone Krug

im HWS 2013/2014

Dekan: Dr. Jürgen M. Schneider

Erstgutachter: Prof. Dr. Martin Schader

Zweitgutachter: Prof. Dr. Christian Becker

Tag der mündlichen Prüfung: 02.04.2014

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
BPMN	Business Process Model and Notation
CLEoS	Collaborative Lightweight Extension for Software Engineering
CMS	Content Management Systemen
EMF	Eclipse Modeling Framework
GPL	GNU General Public License
GUI	Graphical User Interface
GWT	Google Web Toolkit
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IS	Information Systems
JAR	Java Archive
JDBC	Java Database Connectivity
JRE	Java Runtime Environment
JVM	Java Virtual Machine
MOF	Meta Object Facility
NER	Named Entity recognition
NLP	Natural Language Processing
OCL	Object Constraint Language
OMG	Object Management Group
OWL	Web Ontology Language
POS	Part-of-Speech
RDF	Resource Description Framework
RE	Requirements Engineering
RMI	Remote Method Invocation

RPC	Remote Procedure Call
SLIM	Synchronous Lightweight Modeling
SOA	Service-orientierte Architekturen
SQL	Structured Query Language
UML	Unified Modeling Language
WAR	Web Application Archive
WSD	Word Sense Disambiguation
XMI	XML Metadata Interchange
XML	Extensible Markup Language

1 Einleitung

In der Softwareentwicklung spielt die Verbindung verschiedener Artefakte untereinander, auch Nachverfolgbarkeit (engl. tracebility) genannt, einen herausragende Rolle. Watkins und Neal (1994) formulieren treffend:

> *„You can't manage what you can't trace."* (S. 104)

Bezogen auf betriebliche Anwendungssysteme bedeutet dies, dass es unumgänglich ist, die Beziehungen zwischen verschiedenen Artefakten wie schriftlichen Anforderungsspezifikationen und Entwurfsdokumenten wie Klassendiagrammen nachvollziehen zu können Die Unterstützung der Nachverfolgbarkeit durch semantische Technologien steht im Fokus dieser Arbeit.

1.1 Problemstellung

Softwareentwicklung im betrieblichen Kontext erfordert neben dem Zusammenspiel verschiedener Technologien auch die Zusammenarbeit mehrerer Beteiligter. Dank der Einführung von Standards und der damit einhergehenden Vereinheitlichung von Notationen und Schnittstellen wurde in den letzten Jahren das Zusammenspiel der verschiedenen Technologien an vielen Stellen entscheidend vereinfacht. So hat sich beispielsweise UML (Unified Modeling Language) als Standard für den Softwareentwurf etabliert. Ebenfalls wurde eine große Zahl von Methoden und Werkzeugen entwickelt, die zur Unterstützung der Softwareentwicklung im Ganzen verwendet werden können oder speziell auf die Unterstützung der Zusammenarbeit der Beteiligten ausgerichtet sind. Trotz dieser Verbesserungen bleiben einige Probleme im Spannungsumfeld der Softwareentwicklung bestehen. So wird während des Softwareentwicklungsprozesses eine Vielzahl an Artefakten erstellt, ohne dass die Nachverfolgbarkeit zwischen diesen Artefakten dauerhaft sichergestellt wird. Insbesondere zwischen schriftlichen Anforderungsspezifikationen und den Dokumenten des Entwurfs besteht dabei häufig eine Lücke, die sich noch vergrößert, wenn die Modelle komplexer werden (Berenbach 2003).

Die Ursache für diese Lücke ist in der Art und Weise zu suchen, in der schriftliche Anforderungsspezifikationen in Entwurfsdokumente überführt werden. In der Regel handelt es sich dabei um einen von Menschen ausgeführten Prozess, der fehleranfällig ist und viele Ressourcen bindet (Dori et al. 2004). Grundlegend ist dabei, dass die Beteiligten die verschiedenen Artefakte überhaupt miteinander in Zusammenhang bringen können, d. h. die Nachverfolgbarkeit sicherstellen können. Konzeptionell ist Nachverfolgbarkeit als bedeutender Faktor der Softwareentwicklung anerkannt (Spanoudakis und Zisman 2005). Durch Ressourcenbeschränkungen in Entwicklungsprojekten ist eine vollständige Erfassung aller denkbaren Informationen aber so gut wie nie realisierbar (Pohl und Rupp 2011). Insbesondere ist die Erstellung und Pflege von Nachverfolgbarkeitsbeziehungen zwischen den Artefakten – besonders, wenn sie manuell betrieben werden – mit hohen Kosten verbunden (Finkelstein 2012), sodass fraglich bleibt, ob die zur Verfügung stehenden Methoden und Werkzeuge zufriedenstellend für die Praxis sind und auch eingesetzt werden. Ein Grund dafür kann sein, dass An-

forderungsspezifikationen häufig als Textdokument, E-Mail etc. vorliegen, während Entwurfsdokumente, wie beispielsweise Klassendiagramme, vorwiegend grafischer Natur sind und mit entsprechenden Anwendungen bearbeitet und gespeichert werden. Im einfachsten Fall können für die Erfassung der Informationen zur Nachverfolgbarkeit selbst weitere Dokumente angelegt werden. Alternativ dazu können Werkzeuge eingesetzt werden, welche diese Aufgaben unterstützen.

Die verschiedenen Artefakttypen werden oft als zu verbindende gleichwertige Objekte behandelt, und es bleibt unberücksichtigt, dass schriftlichen Anforderungsspezifikationen eine besondere Bedeutung zukommt. Viele Probleme in der Softwareentwicklung liegen in der Ungenauigkeit schriftlicher Anforderungsspezifikationen begründet, welche zu Änderungen und damit verbundenen Verzögerungen und Kosten führt (Sommerville 2012). Anforderungsspezifikationen sollen folglich präzise und konsistent formuliert sein und darüber hinaus auch die gewünschten Funktionalitäten vollständig beschreiben. Im weiteren Fortschreiten eines Entwicklungsprojekts werden sich darüber hinaus weitere Änderungen in den Anforderungsspezifikationen ergeben, die durch die anderen Artefakte reflektiert werden sollten. Nicht selten werden dem System nicht dokumentierte Funktionalitäten hinzugefügt, was zu einem Verlust der Beziehung zwischen den schriftlichen Anforderungsspezifikationen und dem Code führt (Piprani et al. 2008). Vor allem in Hinblick auf die Wartung eines Softwaresystems sollte Wert auf eine Konsistenz zwischen allen Artefakten gelegt werden. Das Ziel ist es, dass alle Beteiligten ein einheitliches Verständnis für die schriftlichen Anforderungsspezifikationen und die darauf basierenden weiteren Artefakte entwickeln können, damit das Softwareprojekt erfolgreich durchführen zu können.

Schriftliche Anforderungsspezifikationen sind nicht nur aufgrund ihrer Bedeutung hervorzuheben, sondern sie besitzen mehrheitlich auch eine Form, die ein Alleinstellungsmerkmal darstellt: 79 % der Anforderungen liegen in natürlicher Sprache vor, die weiteren sind in strukturierter oder formalisierter Sprache verfasst (Luisa et al. 2004). Natürliche Sprache besitzt als einzige Notation die Eigenschaft, dass sie von allen Beteiligten ohne zusätzliche Ausbildung verstanden wird. Auf der anderen Seite ist natürliche Sprache inhärent uneindeutig und unpräzise, sodass zu erwarten ist, dass die schriftlichen Anforderungsspezifikationen unvollständig, aber gleichzeitig auch redundant beschrieben werden (Ambriola und Gervasi 1997). Die gewünschten Eigenschaften der schriftlichen Anforderungsspezifikationen und die Eigenschaften der Form der natürlichen Sprache stehen sich damit unvereinbar gegenüber. Trotzdem ist die natürliche Sprache den restriktierten Notationen vorzuziehen, da jede Art von Formalisierung eine Einstiegshürde darstellt, insbesondere für neue Benutzer[1] eines Systems (Dalianis 1992).

Die Lücke zwischen Anforderungsspezifikationen und den weiteren Artefakten muss nicht nur durch die Erstellung von Nachverfolgbarkeitsbeziehungen unterstützt werden, sondern es muss auch die zeitliche Trennung überwunden werden. Mit zeitlicher Trennung ist in diesem Zusammenhang gemeint, dass mit der Erstellung der Softwareartefakte in der Regel erst begonnen wird, wenn die schriftlichen Anforderungsspezifikationen bereits vollständig vorlie-

[1] In dieser Arbeit schließen die jeweiligen Terme sowohl die weibliche als auch die männliche Form ein.

gen. Insbesondere wenn eine (teil-)automatisierte Analyse durchgeführt werden soll, ist dies für alle existierenden Ansätze eine Grundvoraussetzung. Die Analyse findet dann nur einmalig statt, und die Ergebnisse werden als weitgehend feststehend präsentiert; eine Änderung der Anforderungsspezifikationen nach dem Analysezeitpunkt ist nicht vorgesehen. Dies steht im Widerspruch zur Auffassung der Softwareentwicklung als dynamischer und iterativer Prozess, dessen Schritte über Rückkopplungen miteinander verbunden sind. Es ist in der Praxis nicht zu erwarten, dass sich die Erstellung und Weiterentwicklung der schriftlichen Anforderungsspezifikationen zeitlich vollständig von den weiteren Artefakten trennen lassen. Selbst wenn dies der Fall ist, so werden doch mit hoher Wahrscheinlichkeit Änderungen an den schriftlichen Anforderungsspezifikationen zu einem späteren Zeitpunkt auftreten, die dann mit hohem Aufwand für die anderen Artefakte umgesetzt werden müssen, damit die Konsistenz gewahrt bleibt. Statt die schriftlichen Anforderungsspezifikationen und die Erstellung der weitern Artefakte durch einen künstlichen Zeitpunkt der Analyse zu trennen, sollte vielmehr ein zeitlich unabhängiger Zusammenhang zwischen den schriftlichen Anforderungsspezifikationen und den Analyseergebnissen hergestellt werden können. Die Ergebnisse sind dann nicht als feststehend zu betrachten, sondern erlauben vielmehr eine direkte Einarbeitung der Erkenntnisse aus der Bearbeitung der nachgelagerten Artefakte in die schriftlichen Anforderungsspezifikationen und vice versa.

1.2 Forschungsfrage

Aufbauend auf der aufgezeigten Problemstellung, werden im Folgenden die Forschungsfrage und die daraus abgeleiteten konkreten Fragestellungen vorgestellt. Das übergeordnete Ziel ist es, durch diese Unterstützung die Qualität der erstellten Software zu verbessern, indem bereits in den frühen Phasen der Entwicklung die Kommunikation auf Basis einer gemeinsamen Verständigungsgrundlage erfolgen kann.

Eine zentrale Herausforderung ist die Überführung der informal vorliegenden Anforderungsspezifikationen in eine formalisierte Form (Jackson 1995). Eine Möglichkeit ist es, dieses Vorgehen durch die Analyse der Anforderungsspezifikationen mithilfe von Technologien des Natural Language Processing (NLP) zu unterstützen. Diese Technologien erlauben es, natürliche Sprache auf Basis von Algorithmen zu analysieren. Dies kann mit sehr unterschiedlichen Zielen erfolgen. Beispielsweise kann es das Ziel sein, herauszufinden, welcher Wortart ein bestimmtes Wort angehört, oder Ambiguitäten in der Bedeutung aufzulösen. Auch komplexere Fragen, beispielsweise zu den Bedeutungszusammenhängen zwischen Wörtern oder Referenzen von Wörtern untereinander, können algorithmisch untersucht werden. Die Idee, natürlichsprachliche Anforderungsspezifikationen mithilfe von NLP-Technologien zu analysieren, ist keinesfalls neu (beispielsweise Fantechi et al. 1994). Mit der Weiterentwicklung der einzelnen Technologien der semantischen Analyse stehen für die Umsetzung im Bereich von schriftlichen Anforderungsspezifikationen zunehmend effektivere Ansätze und Werkzeuge zur Verfügung. Bisher werden diese fortgeschrittenen semantischen Technologien für die Analyse von Anforderungsspezifikationen allerdings nicht eingesetzt; die existierenden Ansätze beschränken sich vielmehr auf die Verwendung basaler Funktionalitäten.

Aus der Überlegung heraus, dass Anforderungsspezifikationen bereits Informationen enthalten, die für die weiteren Artefakte von Bedeutung sind, sollen semantische Technologien des-

halb für die Identifikation der relevanten Informationen eingesetzt werden. Es ergibt sich die grundlegende Forschungsfrage:

> Wie können im Rahmen der Nachverfolgbarkeit semantische Technologien dazu eingesetzt werden, die Trennung von schriftlichen Anforderungsspezifikationen und weiteren Artefakten zu überwinden?

Um diese Forschungsfrage zu beantworten, müssen die grundlegenden Bestandteile der Nachverfolgbarkeit spezifiziert werden. Es leiten sich daher folgende Fragestellungen ab:

- Welche **Typen von Artefakten** sollen einbezogen werden?
- Wie sollen **Nachverfolgbarkeitsbeziehungen** gestaltet sein?

Bezogen auf die erste Fragestellung, ist zu hinterfragen, ob die in der Forschung betrachteten Artefakte angemessen sind, die Bedürfnisse der Benutzer des entwickelten Systems zu erfüllen. Für den Benutzer steht beim Einsatz einer Software im Vordergrund, wie diese die eigenen Geschäftsprozesse unterstützt. Trotzdem werden Geschäftsprozesse bei der Betrachtung der Artefakte bisher nicht einbezogen. Diese fehlende Sicht führt dazu, dass die Beziehungen zwischen Geschäftsprozessen, schriftlichen Anforderungsspezifikationen und Softwarekomponenten nicht mehr nachvollzogen werden können. Erst kürzlich wurde, basierend auf dieser Kritik, ein Ansatz von Nordheimer et al. (2012) vorgestellt, der sich dieses Problems annimmt. Es ist wichtig, diese Idee weiterzuentwickeln, da die an der Softwareentwicklung Beteiligten in der Regel in zwei (oder mehr) Gruppen eingeteilt werden. Auf der einen Seite stehen die Entwickler, die das Projekt umsetzen, auf der anderen Seite die Kunden, die die Entwicklung in Auftrag geben. Da an der Erstellung und Verbesserung der schriftlichen Anforderungsspezifikationen aber beide Gruppen in Form eines iterativen und essenziell kollaborativen Arbeitsprozesses beteiligt sind, muss es ein System geben, welches von allen Beteiligten verwendet werden kann und welches Geschäftsprozesse einbindet.

Auf die semantische Analyse selbst bezogen, muss die Art und Weise spezifiziert werden, in der die semantische Analyse eingesetzt werden kann und wie mit den Ergebnissen der Analyse umzugehen ist. Daraus lassen sich folgende Fragestellungen ableiten:

- Inwieweit ist die **semantische Analyse** automatisierbar?
- Auf welche Weise ist die **Ergebnispräsentation der Analyse** umzusetzen?

Ein wichtiger Aspekt bei diesen beiden Fragestellungen ist, dass der Benutzer bestmöglich unterstützt wird. Eine weitgehend automatische Identifizierung relevanter Informationen innerhalb der natürlichsprachlichen Anforderungsspezifikationen ermöglicht die direkte Nutzung dieser Informationen für die Erstellung der weiteren Artefakte. Eine Automatisierung entlastet den Benutzer, da bereits eingegebene Informationen nicht erneut eingegeben werden müssen und die Gefahr, relevante Informationen in den Anforderungsspezifikationen zu übersehen, verringert wird. Da in der Softwareentwicklung Diagramme als Artefakte im Vordergrund stehen, kann die Erstellung eines Diagramms in der Art unterstützt werden, dass ein Grundgerüst als Vorschlag generiert wird. Auch die Ergebnispräsentation darf nicht vernachlässigt werden, da die Informationen für den Benutzer einfach zugänglich und verständlich sein sollen.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in acht Kapitel. Einen Überblick gibt Abbildung 1. In diesem Kapitel, Kapitel 1, wurden bereits die Problemstellung und die Forschungsfragen dargestellt. Das Kapitel erläutert darüber hinaus das gewählte Forschungsparadigma.

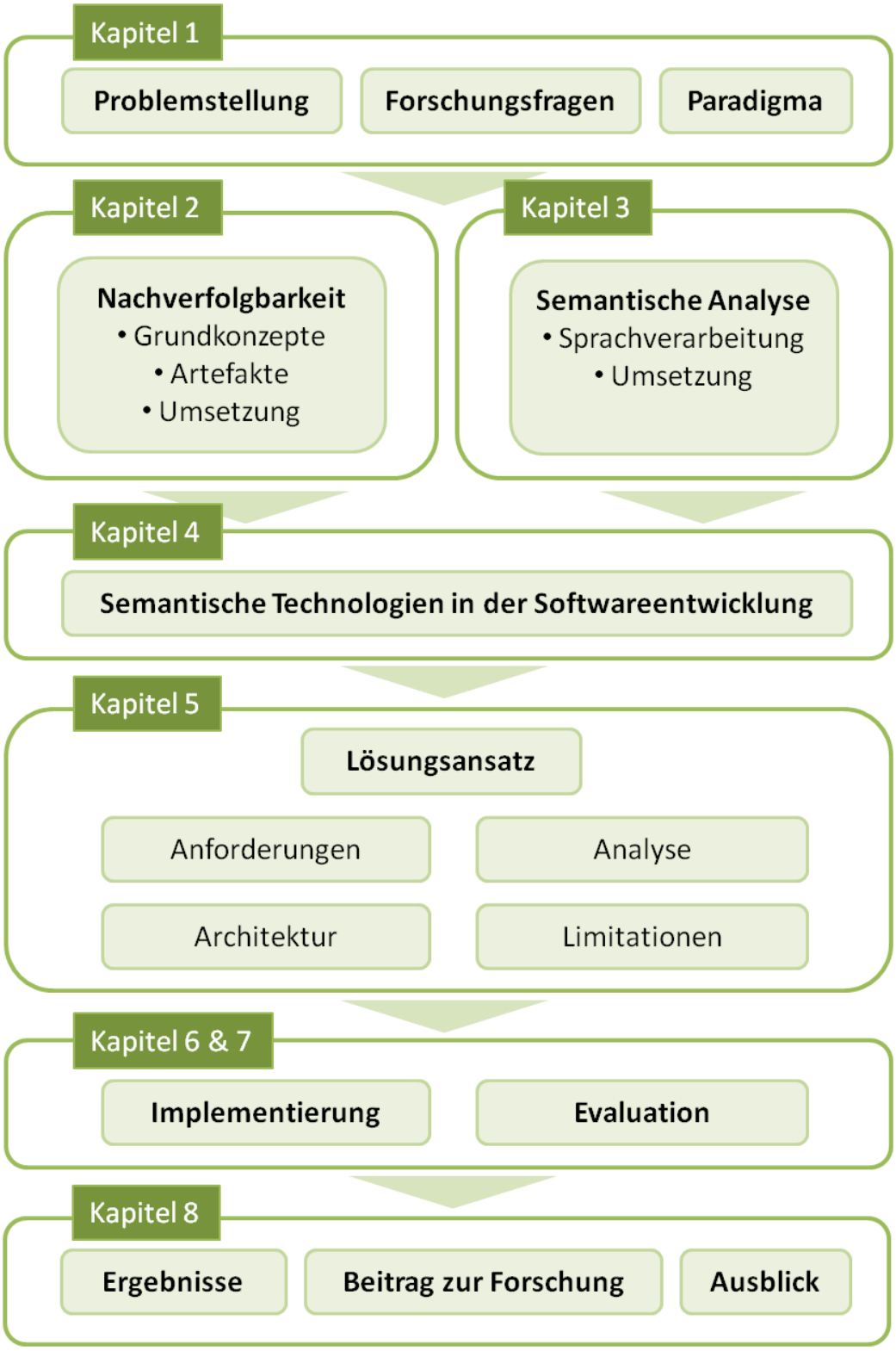


Abbildung 1: Aufbau der Arbeit

Kapitel 2 und Kapitel 3 sind als theoretische Grundlagenkapitel konzipiert, die sich der Nachverfolgbarkeit beziehungsweise den semantischen Technologien widmen. Beide Kapitel legen für die jeweiligen Themengebiete die notwendigen Grundvorrausetzungen für die Inhalte der nachfolgenden Kapitel. Innerhalb der Kapitel werden zunächst die wichtigsten Konzepte besprochen und anschließend die verschiedenen Möglichkeiten der Umsetzung dargestellt. Die Reihenfolge der beiden Themengebiete impliziert nicht deren Bedeutung; beide Themengebiete werden als unabhängig und gleichwertig betrachtet.

Die Überschneidungen der Themengebiete werden getrennt in Kapitel 4 behandelt, welches damit den aktuellen Forschungsstand in Bezug auf den Einsatz semantischer Technologien in der Softwareentwicklung und insbesondere im Rahmen der Nachverfolgbarkeit darstellt. Auch in diesem Kapitel wird die konzeptuelle von der umsetzungsbezogenen Darstellung getrennt, wobei die Möglichkeiten der Umsetzung besonders ausführlich beschrieben werden, da diese Arbeiten die Grundlage für die Abgrenzung des anschließend vorgestellten Lösungsansatzes bilden.

Kapitel 5 stellt den Lösungsansatz vor. In der vorliegenden Arbeit wird dieser bewusst getrennt von den grundlegenden und den verwandten Arbeiten dargestellt, damit der eigene Beitrag besser herausgearbeitet werden kann. Zunächst werden dazu die Anforderungen sowie die Architektur des Lösungsansatzes erläutert. Anschließend wird die Analyse im Detail beschrieben. Abschließend werden die Limitationen des Lösungsansatzes diskutiert.

In Kapitel 6 und Kapitel 7 werden die prototypische Implementierung und Evaluation des Lösungsansatzes aufgezeigt. Die beiden Kapitel bilden damit den praktischen Teil der vorliegenden Arbeit.

Kapitel 8 rundet die Arbeit ab. Dazu wird der Beitrag zur Forschung dargestellt sowie ein Ausblick auf zukünftige Forschungsmöglichkeiten gegeben.

1.4 Forschungsparadigma

Diese Arbeit folgt dem Ansatz der Design Science (Gregor und Hevner 2013; Iivari 2007; Vaishnavi und Kuechler 2007). Dieses von Hevner et al. (2004) vorgelegte Forschungsparadigma steht im Gegensatz zum verhaltenswissenschaftlichen Paradigma, welches darauf abzielt, Theorien zur Vorhersage oder Erklärung von Verhalten eines Individuums oder einer Organisation zu entwickeln und zu verifizieren. Der Design-Science-Ansatz fokussiert sich demgegenüber auf die Erweiterung der Grenzen individueller und organisationaler Leistungsfähigkeit. Dies wird durch die Erstellung neuer Artefakte erreicht. Das von Hevner et al. bereitgestellte Framework bietet dabei klare Richtlinien, die Forschung auszuführen und zu evaluieren.

Das übergeordnete Ziel der Forschung im Bereich der IS (Information Systems) ist dabei immer der Erkenntnisgewinn als Beitrag zur Verbesserung der Effektivität und Effizienz eines Unternehmens. Der Fokus liegt darauf, ein gegebenes Problem zu lösen. Das Design und die Erstellung eines hierfür nützlichen Artefakts sind dabei eine komplexe Aufgabe, da bereits bestehende Theorien oft unzureichend sind. Es gibt verschiedene Arten von Artefakten; diese können sein: (1) Konstrukte, wie beispielsweise ein Vokabular oder Symbole, (2) Modelle,

wie beispielsweise Abstraktionen oder Repräsentationen, (3) Methoden, wie beispielsweise Algorithmen und Praktiken, oder (4) Instantiierungen (engl. instantiations), wie beispielsweise Implementierungen und prototypische Systeme. Diese Artefakte können, sobald sie in einem organisatorischen Kontext implementiert wurden, wiederum Gegenstand einer Untersuchung sein, die auf dem verhaltenswissenschaftlichen Paradigma beruht. Um die Relevanz und Effektivität der IS-Forschung sicherzustellen, werden beide Paradigmen benötigt.

In dieser Arbeit wird ein Artefakt des vierten Typs, der Instantiierungen, erarbeitet und vorgestellt. Dazu wurde bereits eingangs das vorliegende Problem aufgezeigt. Im theoretischen Teil dieser Arbeit wird gezeigt werden, dass keine existierenden Ansätze zur Lösung dieses Problems vorliegen, während der praktische Teil dieser Arbeit einen Lösungsansatz vorstellt, der genau das beschriebene Problem löst.

2 Nachverfolgbarkeit in der Softwareentwicklung

Techniken der Softwareentwicklung unterliegen einer stetigen Weiterentwicklung. Eines der Ziele der Softwareentwicklung ist die Verbesserung der Qualität von Softwaresystemen. Dieses Ziel soll auch durch die Nachverfolgbarkeit erreicht werden (Spanoudakis und Zisman 2005), deren Idee ungefähr in der Mitte der 1980er-Jahre Eingang in die Softwareentwicklung gefunden hat. Zunächst wird der Begriff Nachverfolgbarkeit abgegrenzt, und verschiedene Klassifikationsmöglichkeiten werden vorgestellt. Des Weiteren werden die Bedeutung und Einordnung der Nachverfolgbarkeit im Softwarelebenszyklus aufgezeigt.

Das zweite Unterkapitel beschreibt die verschiedenen Artefakte. Dabei werden zu Anfang die relevanten Artefakttypen kurz vorgestellt und auch mehrere Möglichkeiten für die Betrachtung der Verbindungen beschrieben. Anschließend werden Metamodelle der Nachverfolgbarkeit dargestellt.

Im dritten Unterkapitel wird auf die Umsetzung, Visualisierung und auf Werkzeuge zur Nachverfolgbarkeit eingegangen. Die verschiedenen Repräsentationsformen bilden dabei die Grundlage für die Umsetzung. Eine Übersicht über die Einsatzmöglichkeiten und Anwendungsfelder sowie eine Zusammenfassung bilden den Abschluss dieses Kapitels.

2.1 Grundkonzepte der Nachverfolgbarkeit

2.1.1 Begriffsabgrenzung

Der Begriff Nachverfolgbarkeit (engl. traceability) wird zunächst definiert. Dabei ist anzumerken, dass der Begriff auch außerhalb der Softwareentwicklung verwendet wird, beispielsweise in der Warenwirtschaft. Innerhalb der Softwareentwicklung gibt es eine Vielzahl an verschiedenen Definitionen für Nachverfolgbarkeit, wobei eine weit verbreitete die von Gotel und Finkelstein (1994) ist:

> „Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases).*"* (Gotel und Finkelstein, 1994, S. 94)

Diese Definition bezieht sich auf die Nachverfolgbarkeit von Anforderungen (engl. requirements) über alle Phasen der Softwareentwicklung und -verwendung hinweg. Außerdem wird darauf eingegangen, dass Nachverfolgbarkeitsbeziehungen sowohl vorwärts als auch rückwärts gerichtet, also bidirektional, sind. Für eine genauere Begriffsabgrenzung soll diese Definition derart erweitert werden, dass ein übergeordnetes Verständnis für die Konzepte entsteht, auf die sich Nachverfolgbarkeit in dieser Arbeit bezieht.

Hierzu bietet die Arbeit von Spanoudakis und Zisman (2005) einen passenden Ansatz. Nachverfolgbarkeit wird von den Autoren definiert als:

> „the ability to relate artefacts created during the development of a software system to describe the system from different perspectives and levels of abstraction with each other, the stakeholders that have contributed to the creation of the artefacts, and the rationale that explains the form of the artefacts.*"* (Spanoudakis und Zisman, 2005, S. 2)

Im Gegensatz zur ersten Definition, die Bezug auf den Softwareentwicklungszyklus nimmt, wird dies zwar in der zweiten Definition nicht explizit aufgeführt, die Autoren erklären aber an anderer Stelle, dass ihre Arbeit sich ebenfalls auf alle Phasen der Softwareentwicklung bezieht. Ein wichtiger Unterschied zwischen den beiden Definitionen ist das Verständnis für die Gegenstände, zwischen denen Nachverfolgbarkeitsbeziehungen bestehen. Während Gotel und Finkelstein sich auf die Nachverfolgbarkeit zwischen Anforderungen beziehen, erweitern Spanoudakis und Zisman die Sichtweise auf alle Arten von Artefakten und beachten zudem die unterschiedlichen Abstraktionslevel.

Der Begriff Artefakt (lat. arte = mit Geschick, factum = das Gemachte) bezeichnet dabei alles, was im Laufe des Softwareentwicklungsprozesses erstellt wird, also beispielsweise Dokumente der Anforderungsspezifikation oder des Entwurfs, Quellcode, Modelle, Fehlermeldungen oder Tests (Asuncion et al. 2010; Hildenbrand et al. 2009; Winkler und Pilgrim 2009). Die Definition des IEEE (Institute of Electrical and Electronics Engineers) verwendet statt Artefakt den Term Produkt; demnach ist die Nachverfolgbarkeit definiert als das Ausmaß, in dem zwei oder mehr Produkte des Softwareentwicklungsprozesses miteinander in Verbindung gebracht werden können (IEEE 1990). Es findet sich ebenfalls der englische Term Item (dt. Objekt oder Gegenstand) für die Bezeichnung des gleichen Begriffsfeldes (zum Beispiel Lindvall und Sandahl 1996). In der relevanten Literatur ist jedoch der Begriff Artefakt am gebräuchlichsten und wird daher auch in dieser Arbeit verwendet.

Zusammengefasst wird in dieser Arbeit von einer Bedeutung des Begriffs Nachverfolgbarkeit ausgegangen, die die oben aufgeführten Definitionen vereint. Demnach beschreibt Nachverfolgbarkeit die Fähigkeit, Verbindungen zwischen Artefakten sowie Beteiligten (engl. stakeholders) des Softwareentwicklungsprozesses über dessen sämtliche Phasen hinweg herzustellen. Ausgehend von dieser Definition ergibt sich eine Reihe von Implikationen für die Klassifikation von Nachverfolgbarkeit, welche im folgenden Unterkapitel vertieft behandelt werden.

2.1.2 Klassifikation

Um Nachverfolgbarkeit in verschiedene Klassen zu unterscheiden, wurde bereits eine Vielzahl an Ansätzen vorgeschlagen, von denen die einflussreichsten im Folgenden vorstellt werden. Wie in der oben zitierten Definition von Gotel und Finkelstein (1994) ersichtlich ist, ist eine basale Form der Unterscheidung die Richtung, in der die Artefakte miteinander verknüpft sind: vorwärts und/oder rückwärts. Dabei würde die Erfassung einer ausschließlich einseitigen Verknüpfung, d. h. in nur eine der beiden Richtungen, wichtige Informationen vernachlässigen beziehungsweise Mehraufwand bei der Nachverfolgung von Artefakten erzeugen, sodass eine bidirektionale Verbindung vorzuziehen ist.

Eine weitere Klassifikation von Gotel und Finkelstein (1994) ist die, die zwischen Pre- und Post-Anforderungsspezifikation differenziert. Der Unterschied liegt dabei in dem Zeitpunkt, zu dem eine Anforderung betrachtet wird: vor oder respektive nach ihrer Aufnahme in die

Anforderungsspezifikation. Pre-Anforderungsspezifikationen beziehen sich auf die Nachverfolgbarkeit von Anforderungen zu den Aussagen verschiedener Quellen, die über einen Verfeinerungsprozess in eine einzelne Anforderung überführt werden. Post-Anforderungsspezifikationen beschreiben dagegen die Möglichkeit, Anforderungen in und aus Dokumenten und Produkten nachzuverfolgen (Gotel und Finkelstein 1994).

Eine Übersicht, die mehrere Perspektiven auf die Nachverfolgbarkeit vereint, wird von Pinheiro (2003) vorgelegt. Hier werden neben den Richtungen vorwärts und rückwärts ebenfalls die Pre- und die Post-Anforderungsspezifikationsnachverfolgbarkeit dargestellt. Des Weiteren werden die Konzepte der Inter- und Extra-Anforderungsnachverfolgbarkeit (engl. inter-requirements traceability beziehungsweise extra-requirements traceability) vorgestellt. Die Inter-Anforderungsnachverfolgbarkeit bezieht sich dabei auf die Verbindungen zwischen Anforderungen, die Extra-Anforderungsnachverfolgbarkeit auf die Verbindungen zwischen Anforderungen und anderen Artefakten. Abbildung 2 zeigt die verschiedenen genannten Perspektiven.

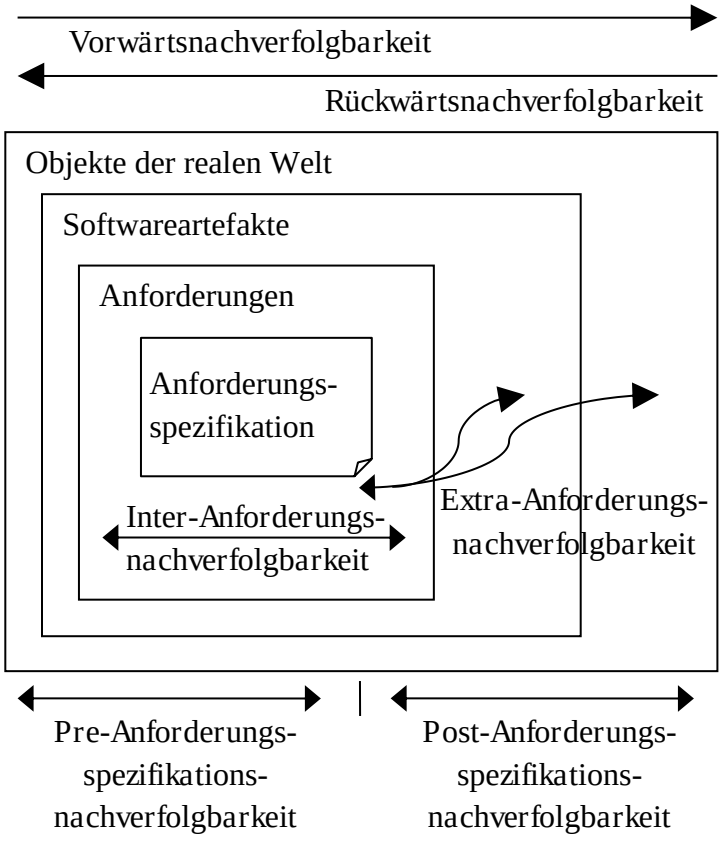


Abbildung 2: Verschiedene Perspektiven der Nachverfolgbarkeit (in Anlehnung an Pinheiro 2003)

Neben dieser temporalen Klassifikation kann zwischen horizontaler und vertikaler Nachverfolgbarkeit unterschieden werden (Lindvall und Sandahl 1996). Dabei beinhaltet die horizontale Nachverfolgbarkeit die Möglichkeit, Artefakte innerhalb eines Modells untereinander zu verbinden. Dagegen erlaubt die vertikale Nachverfolgbarkeit die Verbindung zwischen Artefakten verschiedener Modelle. Abbildung 3 stellt die horizontale und die vertikale Nachverfolgbarkeit bildlich dar.

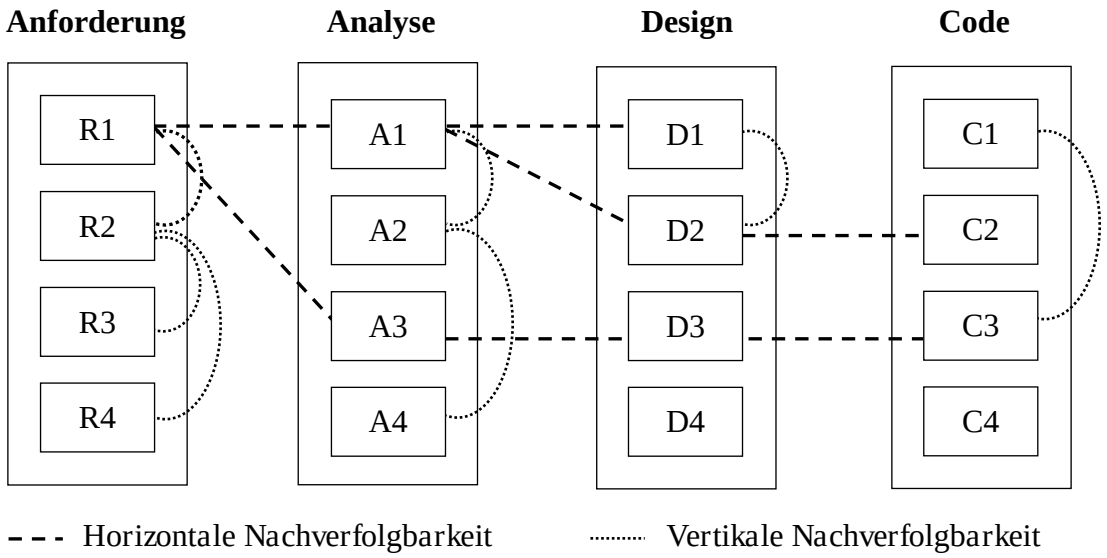


Abbildung 3: Horizontale und vertikale Nachverfolgbarkeit (in Anlehnung an Lindvall und Sandahl 1996)

Eine Unterscheidung zwischen horizontalen und vertikalen Verbindungen ist in der Anwendung ebenfalls nur von untergeordneter Bedeutung, da die benötigten Mechanismen sich für diese beiden Arten der Nachverfolgbarkeit in ihrer technischen Umsetzung nicht unterscheiden.

Es findet sich eine Vielzahl an weiteren Ansätzen in der Literatur, zum Beispiel Kotonya und Sommerville (2004) oder Pohl (2008). Die hier Gezeigten stellen eine Auswahl der wichtigsten und häufigsten Klassifikationen dar.

2.1.3 Bedeutung

Das Hauptmotiv und übergeordnete Ziel der Nachverfolgbarkeit ist die Verbesserung der Qualität eines Softwaresystems (Spanoudakis und Zisman 2005). Dabei wird von manchen Ansätzen (vgl. beispielsweise Lindvall und Sandahl 1996) die Bedeutung für die Dokumentation und Wartbarkeit besonders in den Vordergrund gestellt, von anderen die Bedeutung für die Anforderungsanalyse (vgl. beispielsweise Pinheiro 2000). Die Beschränkung auf wenige Abschnitte im Softwarelebenszyklus greift allerdings zu kurz, sodass im Folgenden die Bedeutung der Nachverfolgbarkeit über alle Phasen der Softwareentwicklung hinweg diskutiert wird.

Pohl und Rupp (2011) benennen als weiteren Nutzen der Nachverfolgbarkeit außerdem die Wiederverwendbarkeit, die Zurechenbarkeit und die Nachweisbarkeit, d. h. den Nachweis dafür, dass eine Anforderung realisiert wurde. Weiterhin betonen sie die Bedeutung für die Identifikation von sogenannten Goldrandlösungen (engl. gold plating) in den Anforderungen und im System. Goldrandlösungen bezeichnen dabei Eigenschaften, die nicht zu einem Systemziel beitragen und oder nicht einer Quelle zuordenbar sind. Diese Eigenschaften besitzen keine Existenzberechtigung und würden somit unnötigen Aufwand verursachen, falls es nicht gelingt, sie zu identifizieren und eliminieren. Weitere Vorteile der Nachverfolgbarkeit umfassen die erhöhte Akzeptanz durch den Endbenutzer (Spanoudakis und Zisman 2005), ein möglicherweise vereinfachtes Systemverständnis (Lindvall und Sandahl 1996) sowie die Möglichkeit zur Auswirkungsanalyse und zur Wiederbenutzung (engl. reuse) existierender Syste-

me (Antoniol et al. 2002). Fehlt die Nachverfolgbarkeit, steigt die Fehleranfälligkeit bei Entscheidungen (Egyed 2006), und das Projekt kann Effizienzeinbußen erleiden (Hildenbrand et al. 2009).

Auf der anderen Seite verursacht die Nachverfolgbarkeit einen Mehraufwand sowie höhere Kosten bei der Erstellung und Wartung (Asuncion et al. 2010; Kotonya und Sommerville 2004). Trotzdem ist in Softwareprojekten der Einbezug von Techniken zur Nachverfolgbarkeit von großer Bedeutung und wird unter anderem durch formale Qualitätskriterien und Standards gefordert (Asuncion et al. 2007), wie beispielsweise durch den IEEE-Standard 830 (Pohl und Rupp 2011), durch ISO 15504 oder DOD Std 2167A (Egyed 2006).

Zusammengefasst kann gesagt werden, dass eine Abwägung zwischen den Vorteilen und den Kosten der Nachverfolgbarkeit getroffen werden muss. Technologien und Werkzeuge sind noch nicht vollkommen ausgereift, daher ist die Nachverfolgbarkeit noch nicht vollständig in der Praxis angekommen (Winkler und Pilgrim 2009). Mit zunehmender Reife der Technologien ist zu erwarten, dass Anwender durch die Nachverfolgbarkeit eine Steigerung des Mehrwerts erfahren und die Technologien auch dank der Erweiterung von Richtlinien und Standards größere Verbreitung finden wird.

2.1.4 Einordnung im Softwarelebenszyklus

Es stellt sich die Frage, wo in der Literatur die Nachverfolgbarkeit im Softwarelebenszyklus anzusiedeln ist. Forscher im Bereich der Anforderungen haben dabei in der Vergangenheit den größten Beitrag zur Weiterentwicklung der Nachverfolgbarkeit geleistet (Winkler und Pilgrim 2009). Dies ist nachvollziehbar, da eine genaue Analyse der Anforderungen die Grundlage für die Verbindungen zwischen Artefakten bildet. Insbesondere wird innerhalb der Anforderungsanalyse und -entwicklung eine spezielle Unterphase betont, die Erhebung (engl. elicitation) genannt wird. Die besondere Herausforderung in dieser Phase ist es, Anforderungen zu verstehen und in eine konsistente Form zu bringen. Dies ist schwierig, da Kunden ihre Anforderungen nicht unbedingt verbalisieren können, verschiedene Kunden unterschiedliche Meinungen zu einer bestimmten Anforderung haben oder auch sonstige Rahmenbedingungen in die Entscheidungen mit einbezogen werden müssen (Sommerville 2007).

Eine ausschließliche Fokussierung auf die Phasen der Anforderungsanalyse oder der Wartung ist nicht ausreichend. Es soll daher geklärt werden, welche Aktivitäten die Nachverfolgbarkeit umfasst. Pinheiro (2003) beschreibt in seiner Arbeit vier Schritte, die zur Erstellung und Benutzung einer Nachverfolgbarkeitsumgebung ausgeführt werden: (1) Im ersten Schritt wird ein Modell für mögliche Verbindungen (engl. traces) definiert. (2) Im zweiten Schritt werden auftretende Verbindungen zeitnah in der Nachverfolgbarkeitsumgebung registriert. (3) Im dritten Schritt tritt der Bedarf auf, Informationen zu suchen. (4) Im vierten und letzten Schritt können mithilfe von Abruftechniken die bereits registrierten Verbindungen nachverfolgt werden. Auf diesem Ansatz basierend, unterscheiden Winkler und Pilgrim (2009) die vier Aktivitäten Planung und Vorbereitung, Erstellung, Benutzung sowie Wartung.

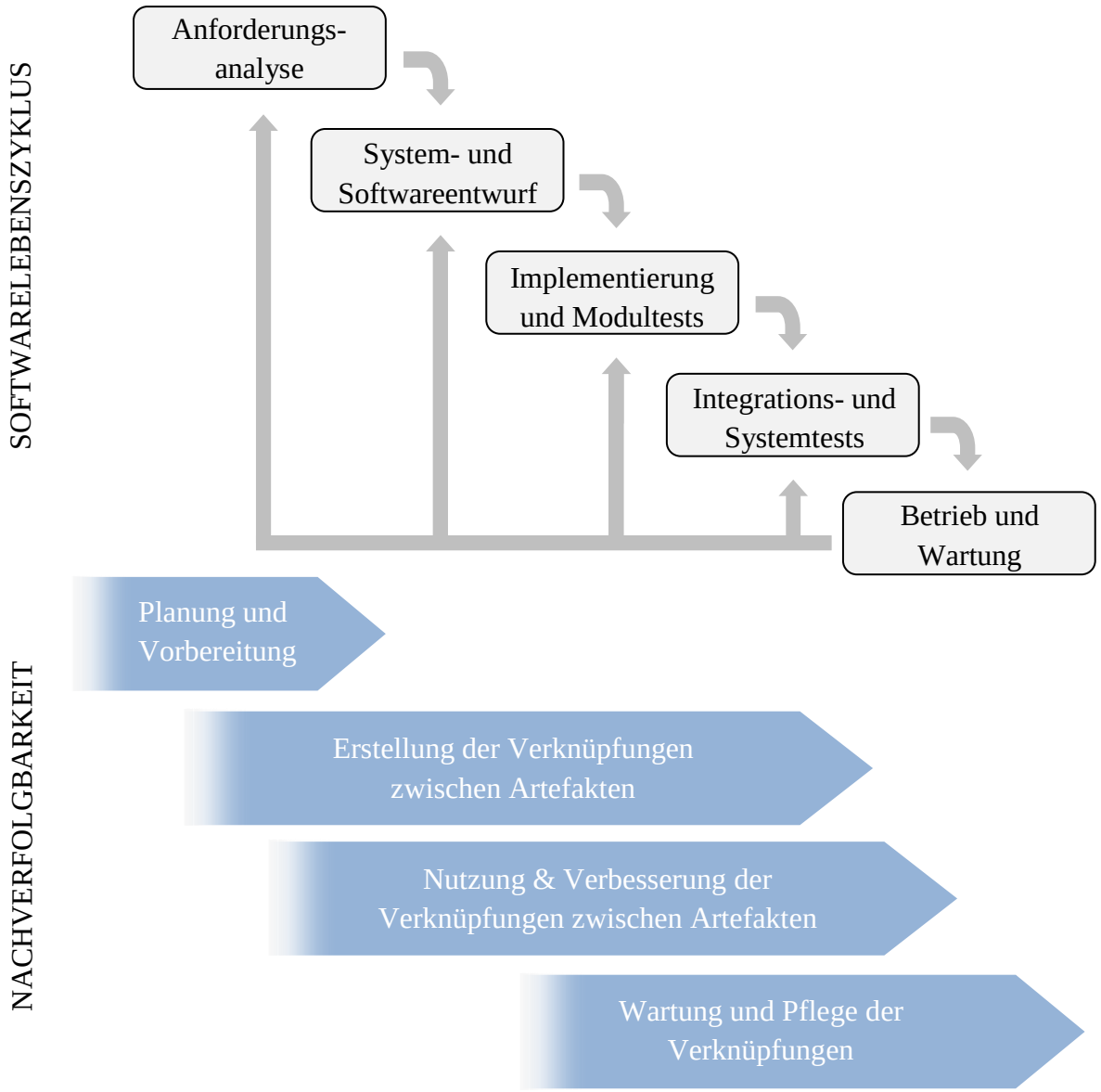


Abbildung 4: Softwarelebenszyklus und Nachverfolgbarkeit (in Anlehnung an Sommerville 2012 sowie Winkler und Pilgrim 2009)

Abbildung 4 stellt den Softwarelebenszyklus nach Sommerville (2012) und die oben genannten Aktivitäten der Nachverfolgbarkeit nebeneinander. Die einzelnen Aktivitäten der Nachverfolgbarkeit laufen dabei nicht nacheinander, sondern teilweise gleichzeitig und wiederholt ab. In der Regel werden sie deshalb in Form eines Kreises oder eines Ablaufmodells mit Rückkopplungen dargestellt, auf deren Darstellung hier aus Gründen der Übersichtlichkeit verzichtet wird. Die Aktivitäten laufen damit nicht ausschließlich, aber schwerpunktmäßig in einer bestimmten Phase des Softwarelebenszyklus ab. So wird die Nachverfolgbarkeit insbesondere vor und während der Anforderungsanalyse geplant und vorbereitet. Die Erstellung, Nutzung und Verbesserung der Verknüpfungen sind vor allem den mittleren Phasen des Softwarelebenszyklus zuzuordnen, also dem Systementwurf, der Implementierung und dem Testen. In den späteren Phasen, in denen das Softwaresystem selbst betrieben und gewartet wird, sindauch die Wartung und Pflege der Verknüpfungen anzusiedeln. Insgesamt laufen die Phasen des Softwarelebenszyklus und die Aktivitäten der Nachverfolgbarkeit gleichzeitig ab und sind miteinander verwoben.

2.2 Artefakte

Bereits in der Begriffsabgrenzung der Nachverfolgbarkeit (vgl. Abschnitt 2.1.1) wurde der Term Artefakt eingeführt. Er bezeichnet in dieser Arbeit alle Produkte oder Objekte, die vor oder während des Softwareentwicklungsprozesses erstellt werden oder mit diesem in Zusammenhang stehen. Insbesondere umfasst der Term alle Anforderungsspezifikationen und Diagramme. Diese Typen werden im Folgenden vertieft dargestellt. Anschließend werden die verschiedenen Möglichkeiten erläutert, die für die Verbindungen zwischen Artefakten existieren. Das Unterkapitel schließt mit einer Betrachtung der Metamodelle der Nachverfolgbarkeit.

2.2.1 Typen

2.2.1.1 Schriftliche Anforderungsspezifikationen

Nach Kotonya und Sommerville (2004) ist eine Anforderung „a statement of a system service or constraint“ (S. 6), während ein Anforderungsdokument die Formalisierung einer Anforderung enthält. Die Terme Anforderungsdokument und Anforderungsspezifikation werden in dieser Arbeit synonym verwendet. Teilweise werden auch weitere Begriffe wie beispielsweise Anforderungskatalog oder Anforderungsdefinition in der gleichen Bedeutung verwendet (Partsch 2010).

Eine weitere Einteilung ist die von Sommerville (2012) in funktionale und nichtfunktionale Anforderungen. Funktionale Anforderungen sind Aussagen, die beschreiben, was das System leisten soll oder was es explizit nicht leisten soll. Sie können sowohl sehr allgemein auf die Systemebene, aber auch sehr spezifisch auf interne Arbeitsmechanismen abzielen. Dabei sollten funktionale Anforderungen vollständig und konsistent sein, was für große und komplexe Systeme allerdings fast unmöglich zu erreichen ist. Nichtfunktionale Anforderungen beschreiben alle Anforderungen, die nicht die spezifischen Dienste betreffen, sondern Eigenschaften wie Zuverlässigkeit, Antwortzeit oder Speicherbedarf. Sie beziehen sich eher auf die Gesamtstruktur als auf einzelne Komponenten des Systems und entstehen beispielsweise auf Basis von Nutzerbedürfnissen, Budgetrestriktionen oder Entscheidungen der Unternehmenspolitik (Sommerville 2012).

Eine besondere Herausforderung der Softwareentwicklung ist dabei die Überführung von informalen Anforderungen in eine formale Notation (Jackson 1995). Informale Anforderungsspezifikationen bezeichnen in dieser Arbeit alle Artefakte, die in natürlicher Sprache vorliegen. Synonym wird hier der Begriff schriftliche Anforderungsspezifikation verwendet. In anderen Publikationen wird auch der Begriff narrativ verwendet (Pohl 2008). Formale Anforderungen umfassen alle Anforderungen, die in einer genau festgelegten Notation vorliegen, also beispielsweise Modelle oder Diagramme. Formale Sprachen stellen eine Untermenge der natürlichen Sprache dar (Luisa et al. 2004). Die wichtigste und weitverbreitetste formale Sprache in der Softwareentwicklung ist die Unified Modeling Language (dt. Vereinheitlichte Modellierungssprache), kurz UML; sie wird gesondert im folgenden Unterabschnitt 2.2.1.2 behandelt.

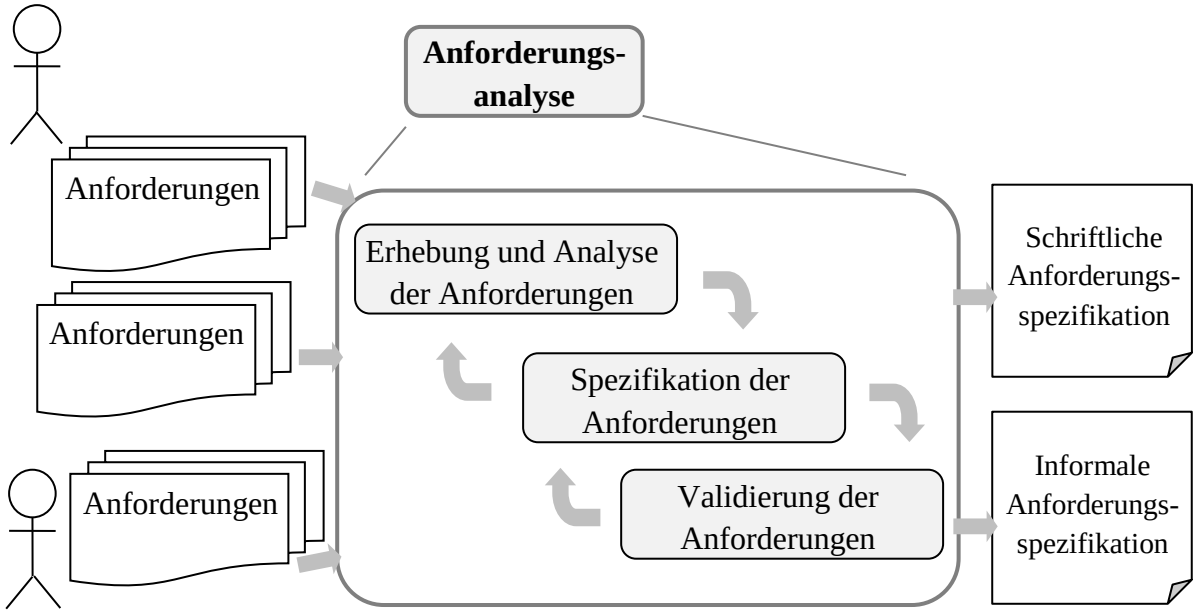


Abbildung 5: Anforderungsanalyse (in Anlehnung an Sommerville 2012)

Sowohl schriftliche als auch formale Anforderungsspezifikationen sind Ergebnisse der Anforderungsanalyse. Diese frühe Phase des bereits oben beschriebenen Softwarelebenszyklus (vgl. Abschnitt 2.1.4) kann selbst in mehrere Unterschritte gegliedert werden (vgl. Abbildung 5). In der Anforderungsanalyse werden über die miteinander rückgekoppelten Schritte der Erhebung und Analyse, der Spezifikation und der Validierung (Sommerville 2012) die Anforderungen der Nutzer in schriftliche und formale Anforderungsspezifikationen transformiert.

Der große Vorteil schriftlicher Anforderungsspezifikationen ist, dass es sich um die einzige Notation handelt, die von allen potenziellen Lesern verstanden werden kann (Kotonya und Sommerville 2004). Allerdings bringt die Formulierung in natürlicher Sprache auch eine ganze Reihe von Nachteilen mit sich. Meyer (1985) beschreibt in seiner Arbeit folgende Nachteile: höherer Umfang, Widersprüche, Mehrdeutigkeiten, Überspezifikationen und Rauschen (engl. noise). Er sieht Rauschen dabei nicht als an sich schlecht an, sondern vergleicht es mit der Funktion von Kommentaren im Quellcode. Allerdings verschleiert Rauschen oft relevante Information im Text. Überspezifikation in Anforderungen bezieht sich auf die übermäßige Beschreibung der Lösung zu einem Zeitpunkt, an dem das Problem noch unzureichend verstanden ist. Als ein Beispiel führt Meyer an, dass Programmierer in manchen Fällen bereits Konzepte auf der Ebene der Implementierung beschreiben, auch wenn die Konzeptualisierung noch nicht ausreichend abgeschlossen ist. In dieser Situation könnten sich frühe Entscheidungen hinsichtlich der Implementierung später als falsch herausstellen, und wichtige Systemeigenschaften könnten fehlen (Meyer 1985).

Um diesen Nachteilen zu begegnen, kann und sollte eine Formalisierung der Anforderungsspezifikationen stattfinden (Jackson 1995). In welcher Art diese Formalisierung durchgeführt wird, ist dabei projekt- und unternehmensabhängig. Aufgrund der Komplexität der natürlichen Sprache dürfte eine vollständige Formalisierung allerdings schwer zu erreichen sein, sodass die Nachverfolgbarkeit zwischen schriftlichen und formalen Anforderungsspezifikationen von essenzieller Bedeutung ist.

2.2.1.2 Softwaremodellierung

Die formalen Anforderungsspezifikationen werden in der Softwareentwicklung häufig durch UML-Diagramme erfasst, welche inzwischen als Industriestandard für objektorientierte Modellierung gelten (Pohl 2008). Die Unified Modeling Language wird von der Object Management Group (OMG), einer gemeinnützigen Organisation, weiterentwickelt und liegt aktuell in der Version 2.4.1 vor. Die UML kann für die Visualisierung, Spezifikation, Konstruktion und Dokumentation eines Softwaresystems eingesetzt werden (Booch et al. 2005). Die Sprache stellt eine abstrakte Syntax, Regeln zur Wohlgeformtheit (engl. well-formedness) und eine dynamische Semantik bereit (Page-Jones 1999).

Es existiert eine große Bandbreite an Möglichkeiten dafür, wie UML für die Softwaremodellierung eingesetzt werden kann. Insgesamt stehen 14 verschiedene Diagrammtypen zur Verfügung. Die aktuelle Spezifikation (OMG 2012a) gliedert sich grob in drei Teile: Konzepte zur Modellierung von Strukturen, Konzepte zur Modellierung von Verhalten und weitere Konzepte. Die ersten beiden Konzeptgruppen beinhalten jeweils die Struktur- beziehungsweise Verhaltensdiagramme. Strukturdiagramme verbildlichen alle statischen Elemente, also all solche, die zeitunabhängig sind. Sie zeigen zwar nicht die Details dynamischen Verhaltens, können aber Beziehungen zu diesen aufweisen. Bekannte Beispiele für Strukturdiagramme sind Klassendiagramme oder Paketdiagramme. Dynamisches Verhalten, also die Serie von Veränderungen des Systems über die Zeit, wird in Verhaltensdiagrammen abgebildet. Als bekannte Beispiele können hier Anwendungsfalldiagramme (engl. Use Case diagrams) oder Sequenzdiagramme genannt werden. Abbildung 6 gibt einen Überblick über die verschiedenen UML-Diagramme und deren Zuordnungshierarchie. Es ist darüber hinaus möglich, verschiedene Diagrammtypen miteinander zu kombinieren; dies gilt für alle Typen, unabhängig davon, ob sie den Struktur- oder den Verhaltensdiagrammen zugeordnet sind. Damit wird es möglich, auch Diagramme zu erstellen, die statische und dynamische Konzepte gleichzeitig in sich vereinen. Alle weiteren Details zu den einzelnen Diagrammtypen sollen hier nicht vertieft werden und sind der Spezifikation zu entnehmen (OMG 2012a).

Das Diagrammformat richtet sich am Metamodell aus, welches bestimmte Regeln zugrunde legt (OMG 2012b). Diese Regeln umfassen beispielsweise das Verständnis, dass es sich bei einer Assoziation mit einem Pfeil um eine gerichtete Verbindung handelt. Eine Assoziation ohne Markierung ist eine bidirektionale Verbindung, wobei der Klassifizierer immer zur entgegengesetzten Seite gehört. Weiterhin gilt, dass, wenn keine Multiplizität (engl. multiplicity) angegeben ist, diese implizit 1 ist. Diese und weitere Regeln dienen als sinnvolle Konventionen, die beim Verständnis von Diagrammen und bei der Einarbeitung in UML helfen.

Die Verwendung von UML bringt viele Vorteile mit sich. Zunächst bieten die Diagramme eine sehr gute Grundlage für die Kommunikation zwischen den verschiedenen Beteiligten (Agarwal und Sinha 2003). Weiterhin zählt Page-Jones (1999) folgende Ziele der UML auf: die Erweiterbarkeit der Kernkonzepte, die Unabhängigkeit von spezifischen Programmiersprachen, die Unterstützung von höheren Entwicklungskonzepten und die Integration bewährter Methoden (engl. best practices).

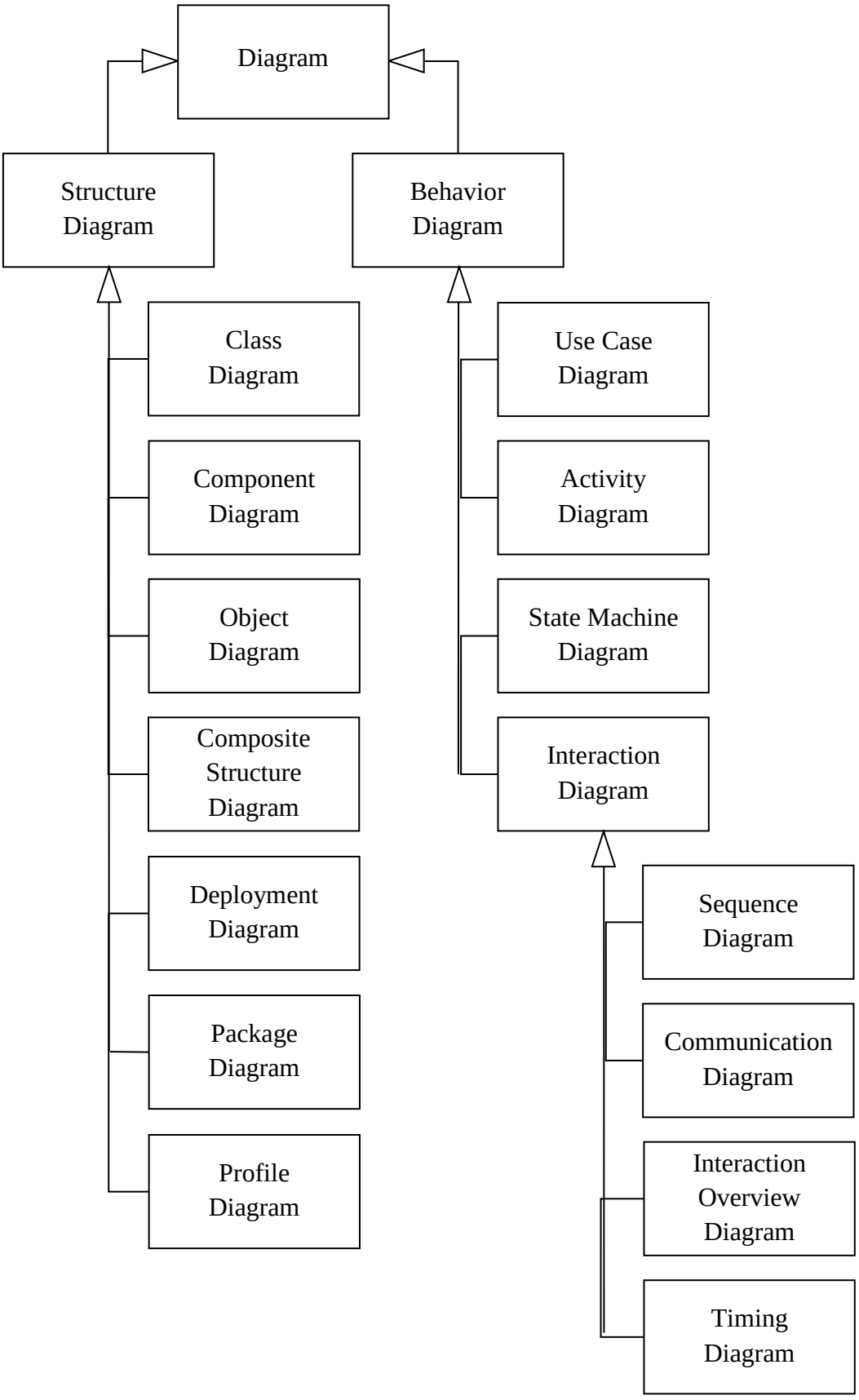


Abbildung 6: UML-Diagrammtypen (OMG 2012a)

Softwaremodellierung mithilfe von UML kann allerdings nicht alles erfassen. So ist es nicht möglich, Entscheidungen hinsichtlich des Designs oder Systemanforderungen in vollem Umfang abzubilden (Agarwal und Sinha 2003). Systemanforderungen beschreiben in detaillierter Form die Funktionen und Beschränkungen eines Systems, während Benutzeranforderungen die Dienste dokumentieren, welche das System dem Benutzer zur Verfügung stellen soll (Sommerville 2012). Benutzeranforderungen werden durch UML-Diagramme und schriftliche Anforderungen erfasst. Es ist dabei nicht Ziel der UML, schriftliche Beschreibungen zu ersetzen (Page-Jones 1999). Insgesamt ergänzen sich UML-Diagramme und schriftliche Anforderungen also und dienen gemeinsam der möglichst kompletten und korrekten Erfassung der Benutzeranforderungen.

2.2.1.3 Weitere Artefakte

In den letzten beiden Abschnitten wurden mit schriftlichen Anforderungsspezifikationen und UML-Diagrammen zur Softwaremodellierung bereits zwei Typen von Artefakten vorgestellt. Darüber hinaus existieren weitere Artefakttypen, die hier zusammengefasst behandelt werden.

Es ist naheliegend, die Dokumente der Anforderungsspezifikation mit dem Quellcode zu verbinden und diesen selbst als ein Artefakt zu betrachten (Egyed et al. 2005; Lindvall und Sandahl 1996; Pinheiro 2003). Mit Einschränkungen ist es dabei möglich, Quellcode aus vorhandenen Artefakten zu gewinnen, insbesondere aus UML-Diagrammen (beispielsweise Amdouni et al. 2011, Costa und da Silva 2007). Dabei wird der Quellcode in der Regel nur in einer groben, skelettartigen Struktur generiert, da eine komplette Erstellung der gesamten Funktionalität nicht möglich und auch nicht das Ziel ist, weil sich die Modelle auf einer anderen Abstraktionsstufe befinden. Informationen zur Nachverfolgbarkeit von Quellcode zu erstellen und zu pflegen, ist deshalb besonders wichtig, weil dies dem Programmierer erlaubt, die konkrete Implementierung schnell mit den Anforderungsspezifikationen abzugleichen, auf deren Basis sie entsteht.

Wenn der Quellcode als ein Artefakt eingebunden ist, so ist es ebenfalls sinnvoll, die mit ihm verbundenen Qualitätssicherungsmechanismen in die Nachverfolgbarkeit mit einzubeziehen. Dazu zählen zum einen Tests (Arkley und Riddle 2005) und zum anderen eine konsistente Verwaltung von Fehlermeldungen (Asuncion et al. 2010). Auch hier können Verknüpfungen mit anderen Artefakten im Rahmen der Nachverfolgbarkeit dazu dienen, Anforderungen nachzuvollziehen und bei Problemen die beteiligten Artefakte zu identifizieren.

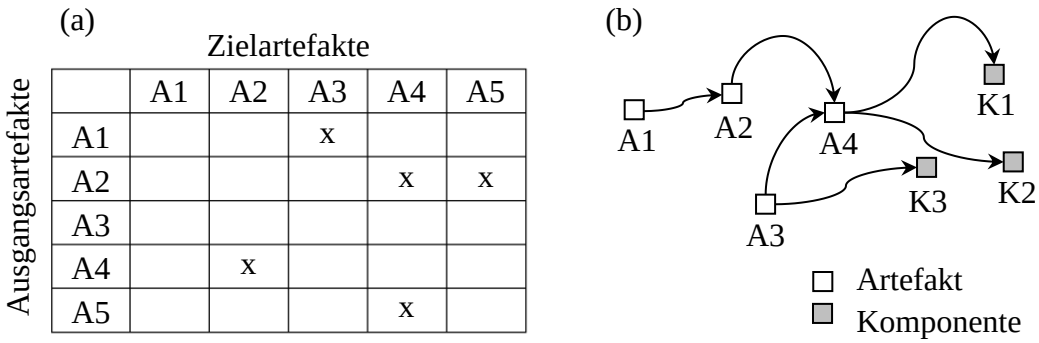
Neben den bisher genannten technischen Artefakten können nach Pinheiro (2003) auch die sozialen Aspekte der Softwareentwicklung in die Nachverfolgbarkeit einbezogen werden. Er nennt hier beispielsweise Personen, Strategien oder Richtlinien, Entscheidungen oder sogar Ziele und Konzepte. Dies unterstreicht die beschriebene Unvollständigkeit der bisher gegebenen Übersicht, da es durchaus möglich ist, weitere Aspekte als Artefakt zu konzeptualisieren.

2.2.2 Verbindungen zwischen Artefakten

Es stellt sich nun die Frage, auf welche Weise die Artefakte miteinander verbunden werden. Zunächst kann dabei der Bezugspunkt betrachtet werden. Aufbauend auf der Unterscheidung zwischen funktionalen und nichtfunktionalen Anforderungen (vgl. Abschnitt 2.2.1.1), differenziert Pinheiro (2003) in seiner Arbeit zwischen funktionalen und nichtfunktionalen Ver-

bindungen. Funktionale Verbindungen entstehen, wenn ein und dasselbe Objekt in verschiedenen Notationen verwendet wird, es also beispielsweise in einem UML-Klassendiagramm und in Quellcode auftritt. Nichtfunktionale Verbindungen erfassen hingegen die qualitätsbezogenen Aspekte von Konzeptbeziehungen, also zum Beispiel die Verbindungen von Anforderungsspezifikationen mit Zielen oder Entscheidungen (Pinheiro 2003). Liegt eine funktionale Verbindung vor, dann besteht gleichzeitig die Möglichkeit, eine Transformation von einem in ein anderes Artefakt unter Verwendung definierter Regeln zu vollziehen (Winkler und Pilgrim 2009).

Weiterhin können Nachverfolgbarkeitsinformationen auf unterschiedliche Art und Weise repräsentiert werden. Unter anderem können diese Informationen in Form schriftlicher Referenzen, Hyperlinks, Matrizen oder Graphen sowie Listen erfasst werden (Kotonya und Sommerville 2004; Pohl und Rupp 2011; Winkler und Pilgrim 2009). Am häufigsten anzutreffen sind Matrizen und Graphen (vgl. Abbildung 7). In einer Matrix bezeichnen die Zeilen und Spalten jeweils die Artefakte und die Zellen deren Verbindungen; in Nachverfolgbarkeitsgraphen stellen die Knoten die Artefakte dar, während die Kanten die Verbindungen repräsentieren.



	A1	A2	A3	A4	A5
A1			x		
A2				x	x
A3					
A4		x			
A5				x	

Abbildung 7: (a) Matrix und (b) Graph zur Erfassung der Nachverfolgbarkeit (in Anlehnung an Kotonya und Sommerville 2004; Pohl und Rupp 2011)

Die Repräsentation als Matrix ist auf der einen Seite weit verbreitet, bringt allerdings auf der anderen Seite einen entscheidenden Nachteil mit sich, da bei großen Softwareprojekten keine Skalierbarkeit gegeben ist (Pohl und Rupp 2011). Des Weiteren müssen für verschiedene Arten von Verbindungen auch verschiedene Matrizen erstellt werden, sodass hier ein erheblicher Aufwand entsteht. Günstiger ist daher die Verwendung von Nachverfolgbarkeitsgraphen, welche teilweise auch als Nachverfolgbarkeitsnetzwerke bezeichnet werden (Berkling et al. 2007). Entgegen der hier stark vereinfachten Darstellung, können auch bidirektionale Verbindungen und weitere Annexionen mithilfe eines Nachverfolgbarkeitsnetzwerks erfasst werden.

Darüber hinaus ist es möglich, verschiedene Typen von Verbindungen zu betrachten. Beispiele für Verbindungstypen sind Ähnlichkeiten, Konflikte, Abstraktionen, Vererbungen, Abhängigkeiten oder Schlussfolgerungen (Aizenbud-Reshef et al. 2005; Hildenbrand et al. 2009; Jirapanthong und Zisman 2007; Pohl und Rupp 2011, Ramesh und Jarke 2001). Da diese Typen konzeptuell wenig fundiert sind und teilweise auf empirischen Befunden beruhen, klassifiziert die Arbeit von Aizenbud-Reshef et al. (2005) Verbindungen als ein Set, bestehend aus drei Eigenschaften:

- Ereignisse: Ereignisse treten bei der Veränderung eines Modells auf, also wenn eine Verbindung erstellt, verändert oder gelöscht wird.
- Bedingungen: Bedingungen erlauben es, Ereignisse zu steuern beziehungsweise Auslöser (engl. trigger) für diese zu sein.
- Aktionen: Aktionen zielen auf die Validität des Gesamtsystems ab, indem sie beispielsweise die Ausführung eines Ereignisses verhindern oder eine Synchronisation zwischen verschiedenen Elementen herstellen.

Die Betrachtung auf dieser übergeordneten Ebene liefert einen Beitrag zur Theorie, lässt allerdings offen, wie genau die eben angesprochenen Verbindungstypen wie beispielsweise Vererbungen oder Abhängigkeiten anhand dieser Eigenschaften beschrieben werden sollen. Bevor diese Beschreibung für die bekannten Konzepte nicht vorliegt, sei es in dieser oder einer anderen Abstraktion, ist auch in Zukunft ein uneinheitliches Verständnis von Verbindungstypen zu erwarten.

2.2.3 Metamodelle der Nachverfolgbarkeit

Wie in den vorangestellten Abschnitten dargestellt, existiert weder für die Artefakte noch für die Verbindungen ein Standardansatz. Eine Vielzahl an Autoren (beispielsweise Piprani et al. 2008; Vanhooff et al. 2007; Walderhaug und Johansen 2006, für weitere siehe Winkler und Pilgrim 2009) haben sich mit der Erstellung eines Metamodells der Nachverfolgbarkeit beschäftigt. Häufig referenziert wird die Arbeit von Ramesh und Jarke (2001), in welcher das Metamodell aus Daten eines empirischen Forschungsprozesses abgeleitet wird. Das Modell beschreibt dabei, welche Informationen (hier genannt Objekt, engl. object) von wem (hier bezeichnet als Beteiligter, engl. stakeholder) und wo (hier genannt Quelle, engl. source) zur Nachverfolgbarkeit vorliegen. Abbildung 8 visualisiert dieses Metamodell.

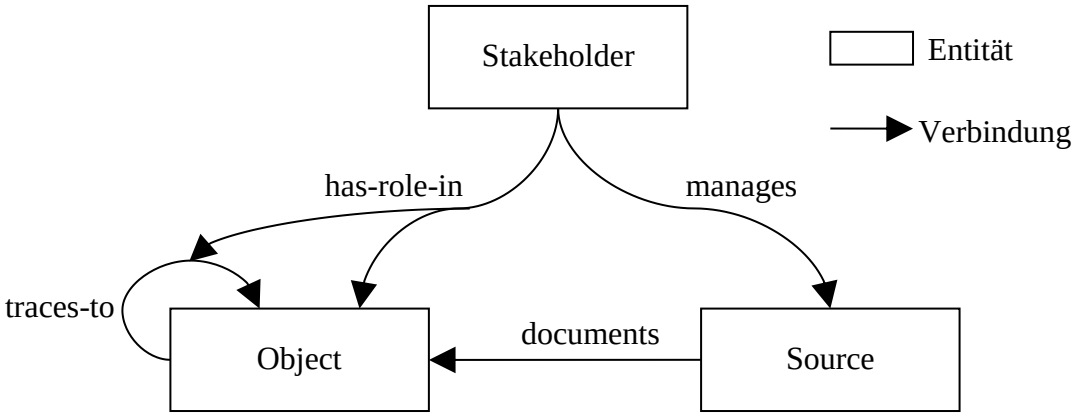


Abbildung 8: Konzeptuelles Modell der Nachverfolgbarkeit (Ramesh und Jarke 2001)

Auch wenn das Metamodell von Ramesh und Jarke gut verständlich und übersichtlich erscheint, so fehlt ihm doch die konzeptionelle Fundierung. Eine solche wird ausführlich in der Arbeit von Almeida et al. (2007) geliefert. Das Ziel dieses Metamodells ist die Unterstützung von Programmierwerkzeugen. Konzeptionell beschäftigt sich der Ansatz vor allem mit der Konformität zwischen Modellen. Das Konzept basiert auf einer Nachverfolgbarkeitsmatrix zur Visualisierung sowie auf dem Ecore Metamodell, welches wiederum durch das Eclipse

Modeling Framework (EMF) unterstützt wird. Für eine vertiefte Betrachtung des EMF wird auf Steinberg et al. (2008) verwiesen. Eine großer Nachteil dieses Modells ist die Beschränkung der Nachverfolgbarkeit auf eine Richtung: Es besteht ausschließlich die Möglichkeit, vorwärts gerichtete Verbindungen (vgl. Abschnitt 2.1.2) zu erfassen. Ebenfalls auf dem Ecore Metamodell des EMF beruht der Ansatz von Anquetil et al. (2010). Die Umsetzung ist flexibel und basiert auf einem Nachverfolgbarkeitsgraphen, der für alle Elemente zusätzliche Annotationen erlaubt. Die Autoren verfolgen dabei vor allem das Ziel, eine generische und anpassbare Lösung bereitzustellen. Auch wenn das Metamodell explizit für Softwareproduktlinien (engl. software product lines) entwickelt wurde, so ist es doch hinreichend flexibel für Erweiterungen. Auf der anderen Seite ist es sehr rudimentär und muss vom Anwender selbst angepasst werden.

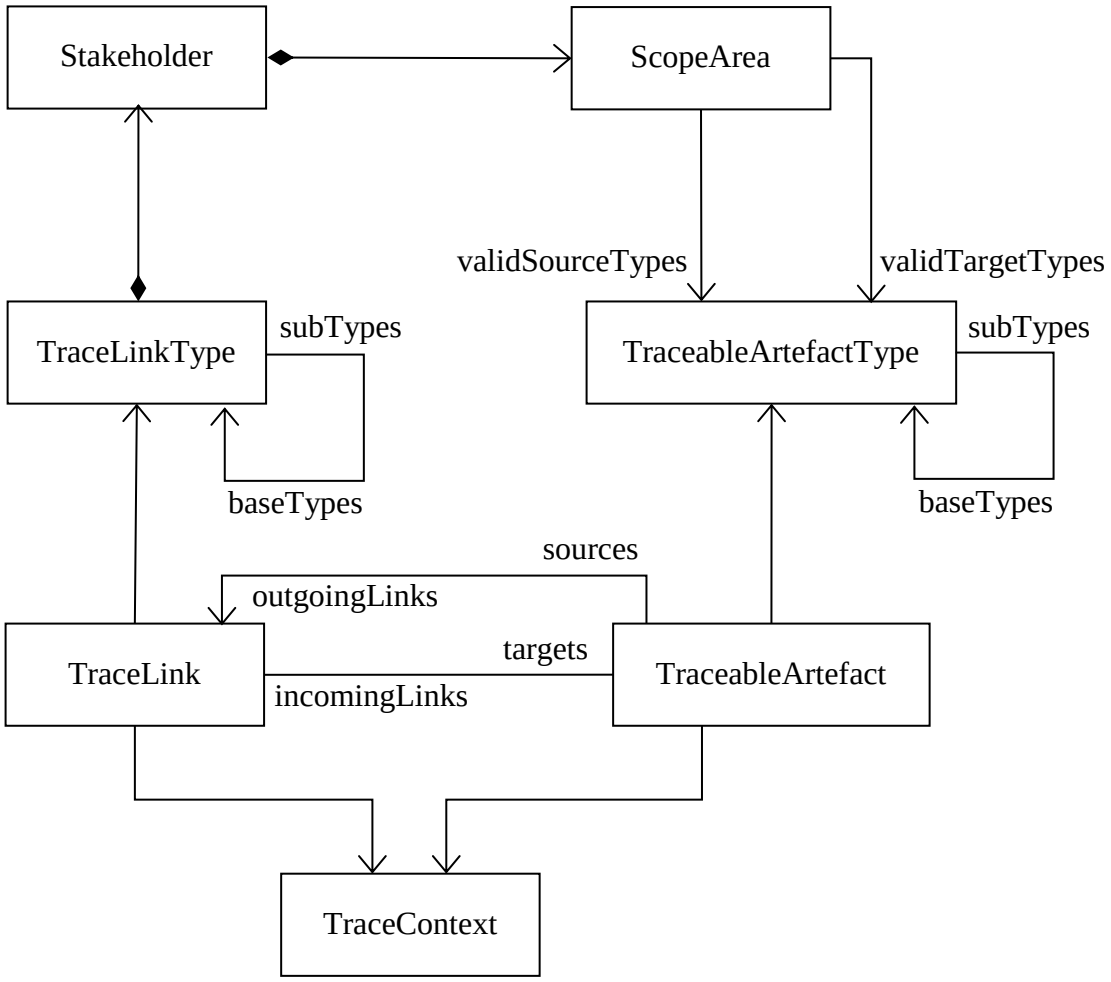


Abbildung 9: Metamodell von Anquetil et al. (2010)

Insgesamt betrachtet, spiegelt sich in den Metamodellen die bereits angesprochene Problematik der Pluralität der Ansätze wider. Die meisten Autoren verwenden daher ein ad hoc erstelltes und nur auf einzelne Szenarien zugeschnittenes Modell (Winkler und Pilgrim 2009). Es existiert damit kein Metamodell, welches für jede Problemstellung eingesetzt werden kann.

2.3 Umsetzung

Nachdem im vorherigen Unterkapitel die verschiedenen Artefakte und Verbindungen aufgezeigt wurden, wird nun darauf eingegangen, wie Nachverfolgbarkeitsinformationen erfasst werden können und dieser Prozess durch Werkzeuge unterstützt werden kann. Des Weiteren werden verschiedene Einsatzmöglichkeiten der Nachverfolgbarkeit aufgezeigt.

2.3.1 Erfassung von Nachverfolgbarkeitsinformationen

Es existieren mehrere Möglichkeiten, Nachverfolgbarkeitsinformationen zu erfassen. Diese lassen sich nach Spanoudakis und Zisman (2005) grob gliedern in (1) manuelle, (2) halbautomatische und (3) automatische Ansätze. Die manuelle Erfassung ist in vielerlei Hinsicht problematisch. Zum einen ist sie komplex und fehleranfällig (Spanoudakis und Zisman 2005); zum anderen entsteht für sie ein nicht unerheblicher Aufwand, der zu einem sensiblen Zeitpunkt wie der Endabnahme der Anforderungsspezifikationen durch den Kunden zu erheblichen Verzögerungen führen kann (Egyed et al. 2005). Dahingegen erfordern halbautomatische Ansätze zwar zum Teil einen Eingriff durch den Benutzer, erstellen die Nachverfolgbarkeitsinformationen aber teilweise auch ohne zusätzliche Eingaben und verringern so den Aufwand. Automatische Ansätze bedienen sich verschiedener Techniken, wie des Information Retrieval, oder bestimmter Regeln (Spanoudakis und Zisman 2005). Eine Übersicht über verschiedene Ansätze und deren Zuordnung zu dieser Gliederung bietet die Arbeit von Bashir und Qadir (2006).

Aizenbud-Reshef et al. (2006) unterscheiden darüber hinaus auch zwei Arten der berechneten Verbindungen: solche, basierend auf einer Ableitung (engl. derivation), und Analyse-basierte. Ableitungs-basierte Verbindungen erlauben es, valide Inhalte aus verwandten Artefakten zu berechnen. Typischerweise treten Ableitungs-basierte Verbindungen bei der Modelltransformation oder der Codegenerierung auf. Analyse-basierte Verbindungen werden hingegen bei der Code- oder Modellanalyse erstellt. Dabei werden sie beispielsweise anhand einer Abhängigkeitsanalyse des Quellcodes errechnet. Je nach verwendetem Ansatz kann dieses Verfahren sehr rechenaufwendig sein, da auch bei kleinen Änderungen jeweils eine Neuberechnung angestoßen werden kann (Aizenbud-Reshef et al. 2006).

Eine weiterentwickelte Form der Erfassung von Nachverfolgbarkeitsinformationen ist die Werte-basierte (engl. value based) Methode von Egyed et al. (2005). Im Unterschied zur klassischen Softwareentwicklung, bei der alle Artefakte als gleichwertig behandelt werden, bezieht der Werte-basierte Ansatz mit ein, dass Entscheidungen auf Systemebene einen Einfluss auf die Gesamtkosten, die Zeitplanung sowie den Wert einer Software als Ganzes haben (Boehm 2006). Hildenbrand et al. (2008) schlagen darüber hinaus vor, die Werte-basierte Vorgehensweise mit der einer Verwaltung von Entscheidungen zu kombinieren. Dieses integrierte Konzept nennen sie Traceability and Rationale Management. Es umfasst ein bidirektionales Verständnis von Nachverfolgbarkeitsbeziehungen und bezieht sich auf den gesamten Softwarelebenszyklus. Da dieser Ansatz auch Elemente semantischer Technologien beinhaltet, wird er in Kapitel 4 (vgl. 4.2.2.1) detaillierter erläutert.

2.3.2 Werkzeuge

Die Bandbreite bereits existierender Werkzeuge zur Erfassung von Nachverfolgbarkeitsinformationen ist groß; einen aktuellen Überblick gibt die Arbeit von Torkar et al. (2012). Eng verbunden mit der reinen Erfassung ist der Wunsch des Benutzers nach einer Visualisierung der Nachverfolgbarkeitsinformationen im Sinne einer grafischen Darstellung über eine Textdarstellung hinaus. Die meisten Werkzeuge bieten dazu eine Auswahl an Möglichkeiten wie beispielsweise Matrizen oder Graphen (vgl. Abschnitt 2.2.2) an. Die Visualisierung zielt in erster Linie auf die Maximierung des Verständnisses und der Kommunikation ab (Gotel et al. 2007), was wiederum eine Reduzierung des Risikos für den Erfolg des Softwareprojekts bewirken soll. Die Auswahl und Verwendung geeigneter Werkzeuge sind daher zentral für ein zielorientiertes Projektmanagement.

In der Arbeit von Geisser et al. (2007b) werden zwei Werkzeuge zur Erfassung von schriftlichen Anforderungsspezifikationen sowie deren Verbindungen untereinander vorgestellt. Eines davon ist RequisitePro[2]. Anforderungsspezifikationen können hier mit weiteren Attributen annotiert werden, was eine genauere Beschreibung erlaubt. RequisitePro verfügt über Schnittstellen mit anderen Produkten von IBM wie beispielsweise Rational Rose sowie eine Versionierungskontrolle. Als weiteres kommerzielles Werkzeug wird Dimensions[3] der Firma Serena beschrieben. Dimensions erlaubt es, Dokumente – insbesondere Microsoft Word und Excel Dokumente – miteinander zu verbinden. Es erlaubt eine Annotierung von Anforderungen mit zusätzlichen Attributen und stellt ebenfalls eine Versionierungskontrolle bereit. Nicht unterstützt wird von beiden Produkten die Überprüfung formaler Sprachkonstrukte (Geisser et al. 2007b).

Dass Anforderungsspezifikationen nicht zwei-, sondern auch dreidimensional dargestellt werden können, zeigt die Arbeit von Teyseyre und Campo (2009). Sie liefert einen Überblick über die Vielzahl an Werkzeugen, die beispielsweise die Darstellung von Bäumen, hierarchischen Netzen oder auch UML-Diagrammen in 3D erlauben. Koike (1993) sieht die Hinzunahme einer weiteren Dimension bei der Darstellung als großen Vorteil, da vormals separat abgebildete Beziehungen in einem Diagramm vereinigt werden können. Allerdings wird dadurch die Übersichtlichkeit beeinträchtigt, sodass insgesamt fraglich bleibt, inwieweit die dreidimensionale Darstellung einen Mehrwert gegenüber der traditionellen hat. Die meisten dieser Systeme befanden sich zum Zeitpunkt der Publikation von Teyseyre und Campo (2009) allerdings noch im Status eines Prototyps; auch bis heute hat diese Idee keine großflächige Verbreitung gefunden.

Wenn der gesamte Lebenszyklus eines Softwareprodukts betrachtet werden soll, ist die Erfassung von Anforderungsspezifikationen allein unzureichend. Systeme, die dies liefern wollen, müssen die Möglichkeit zur Erfassung mehrerer Artefakttypen bereitstellen. Beispiele für solche Werkzeuge sind CaliberRM[4] von Borland und DOORS[5], welches wie auch RequisitePro inzwischen zur IBM-Produktfamilie gehört. In der Arbeit von Anquetil et al. (2010) werden die Vor- und Nachteile dieser beiden Werkzeuge beschrieben. Beispielsweise erlaubt

[2] http://www-142.ibm.com/software/products/us/en/reqpro
[3] http://www.serena.com/index.php/en/products/dimensions-rm
[4] http://www.borland.com/products/caliber
[5] http://www-01.ibm.com/software/awdtools/doors/productline

DOORS die Spezifikation neuer Verbindungstypen, während CaliberRM die Identifikation bestimmter Inkonsistenzen bei der Verbindungs- und Artefaktdefinition bereitstellt. In CaliberRM werden Verbindungen ausschließlich manuell verwaltet, während DOORS darüber hinaus über die Möglichkeit des Imports verfügt. Beide Systeme bieten Visualisierungen, unter anderem als Nachverfolgbarkeitsmatrix, an und arbeiten darüber hinaus mit vielen Schnittstellen zusammen. Damit können zum Beispiel Verbindungen in externe Datenbanken exportiert werden. Auf der anderen Seite sind beide Werkzeuge proprietär beziehungsweise nicht Quellcode-offen und stehen damit nicht für die Forschung und eigene Weiterentwicklung zur Verfügung. Informationen zu weiteren kommerziellen Produkten sind der Arbeit von Anquetil et al. (2010) zu entnehmen.

Neben kommerziellen Lösungen wurden auch im akademischen Bereich mehrere Werkzeuge entwickelt. Geeignet für die Erstellung von Verbindungen zwischen Artefakten sind zum Beispiel Ariadne (Trainer et al. 2005) oder Augur (Froehlich und Dourish 2004). Ariadne, eine Erweiterung für Eclipse, zielt darauf ab, die technischen Abhängigkeiten zwischen Softwarekomponenten und den Autoren herzustellen. Dazu wird zunächst ein Graph erstellt, der anhand der Aufrufstruktur die Abhängigkeiten aus dem Quellcode extrahiert. Darauf aufbauend, wird ein sozialer Abhängigkeitsgraph erstellt, da davon ausgegangen wird, dass voneinander abhängige Komponenten auch von Autoren beziehungsweise Programmierern mit Aufgabenabhängigkeiten verantwortet werden. Auf der einen Seite lässt sich dieses Werkzeug gut in kollaborative Entwicklungsumgebungen integrieren. Auf der anderen Seite sind die Einsatzmöglichkeiten als begrenzt zu bezeichnen (Hildenbrand 2008). Das zweite Werkzeug, Augur, ermöglicht es, gleichzeitig Softwareartefakte und die Aktivitäten der Softwareentwicklung zu visualisieren. Dieses wird mithilfe von Annotationen erreicht, welche auch die Verbindung eines Quellcode-Abschnitts mit den Autoren ermöglichen. Auch eine Änderungshistorie steht zur Verfügung. Weitere Werkzeuge aus dem akademischen Bereich sind Palantir (Sarma et al. 2003) oder OPHELIA (Hapke et al. 2004). Palantir informiert die beteiligten Entwickler, wenn Änderungen an einem Artefakt durch andere Entwickler vorgenommen wurden. Eine vergleichbare Funktionalität bietet OPHELIA an, welches außerdem Funktionalitäten für Metriken und Wissensmanagement bereitstellt. Gemeinsam ist allen akademischen Werkzeugen, dass sie keine semantischen Techniken zur Analyse der Artefakte einsetzen.

2.3.3 Einsatzmöglichkeiten

Grundsätzlich kann eine systematische Erfassung von Nachverfolgbarkeitsinformationen wie bereits beschrieben (vgl. 2.1.3) die Akzeptanz durch den Endbenutzer sowie das Systemverständnis unterstützen. Wie auch UML-Diagramme, dient die Dokumentation von Nachverfolgbarkeitsinformationen der Kommunikation zwischen Entwicklern und weiteren Beteiligten (Asuncion et al. 2007).

Darüber hinaus beschreiben Spanoudakis und Zisman (2005) ausführlich, für welche Bereiche Nachverfolgbarkeit eingesetzt werden kann. Als erstes Einsatzgebiet nennen sie die Analyse von Auswirkungen (engl. impact analysis) und das Veränderungsmanagement (engl. change management). Die einfachste Form der Auswirkungsanalyse ist dabei die Identifikation aller von einer Änderung betroffenen Artefakte. Komplexere Ansätze zielen außerdem auf die Klassifikation der betroffenen Artefakte ab, die Identifikation von Nebeneffekten und die Ab-

schätzung entstehender Kosten. Im Rahmen des Veränderungsmanagements werden geplante Veränderungen priorisiert, beziehungsweise es wird eine Entscheidung getroffen, ob eine bestimme Änderung am System überhaupt durchgeführt werden sollte. Als zweites Einsatzgebiet bezeichnen Spanoudakis und Zisman Validierung, Verifizierung, Testen und die Einhaltung von Standards. Das dritte Einsatzgebiet ist die Wiederverwendung, wobei neben der Wiederverwendung von einzelnen Softwarekomponenten durchaus auch Anforderungsspezifikationen selbst im Rahmen des sogenannten „requirement recycling“ wiederverwendet werden können. Als viertes Einsatzgebiet benennen Spanoudakis und Zisman den Beitrag von Nachverfolgbarkeitsinformationen zum Verständnis von Artefakten; insbesondere, wenn es sich um Artefakte handelt, an denen der Betreffende nicht selbst mitgearbeitet hat. Dies ist ein häufig auftretendes Szenario in der Wartung von Software. Das fünfte und letzte Einsatzgebiet ist die Untersuchung durch empirische Studien, die die Verwendung von Nachverfolgbarkeit in Unternehmen analysieren (Spanoudakis und Zisman 2005).

Softwareentwicklung findet häufig im Team statt. Somit ist die Zusammenarbeit zwischen mehreren Beteiligten während des Softwareerstellungsprozesses ein weiterer Aspekt, für den Nachverfolgbarkeit von Bedeutung ist. Dabei spielen zum einen die Zurechenbarkeit und die Nachweisbarkeit eine Rolle (vgl. 2.1.3), zum anderen stellen Nachverfolgbarkeitsinformationen eine zentrale Wissensquelle dar, die unter anderem zur Koordination verwendet werden kann (Hildenbrand et al. 2009). Damit ist nicht nur die Nachverfolgbarkeit bezogen auf die Artefakte von Bedeutung, sondern es kann auch eine Erweiterung hinsichtlich der Informationen bezogen auf die Vernetzung der Beteiligten vorgenommen werden.

Schlussendlich kann sich Nachverfolgbarkeit auch für die Verwendung in anderen Themenfeldern als nützlich erweisen und auf diese übertragen werden, wie beispielsweise auf Service-orientierte Architekturen (SOA). SOA, wie beschrieben in der Arbeit von Seedorf et al. (2009), umfasst über Services hinaus unter anderem Geschäftsprozesse, Modelle und Testfälle. Es wird hier die Möglichkeit vorgestellt, Nachverfolgbarkeit zwischen Komponenten des SOA-Lebenszyklus und des Lebenszyklus von Geschäftsprozessen zu etablieren. Das vorgestellte Framework bezieht dabei sowohl die verschiedenen Artefakte als auch eine Ontologie und verschiedene Werkzeuge mit ein (Seedorf et al. 2009).

Insgesamt werden Nachverfolgbarkeitsinformationen also für eine ganze Reihe verschiedener Einsatzmöglichkeiten verwendet. Da die Variabilität bei der Auswahl der erfassten Artefakte und Verbindungen sehr hoch ist, bestimmt die Konzeption der Erfassung der Nachverfolgbarkeitsinformationen auch, für welche Zwecke sie eingesetzt werden kann. Da, wie beschrieben, die Nachverfolgbarkeit auch nach Ablauf eines erfolgreichen Entwicklungsprojekts bedeutsam sein kann, sollten Nachverfolgbarkeitsinformationen über den gesamten Softwareentwicklungszyklus hin sorgfältig erfasst werden.

2.4 Zusammenfassung

In diesem Kapitel wurden die Grundlagen der Nachverfolgbarkeit dargestellt. Ausgehend von der in dieser Arbeit verwendeten Definition des Begriffs der Nachverfolgbarkeit sowie den möglichen Herangehensweisen zur Klassifikation, wurde die Verbesserung der Qualität als die wichtigste Bedeutung der Nachverfolgbarkeit für die Softwareentwicklung herausgestellt. Des Weiteren wurden die Phasen der Nachverfolgbarkeit neben die des Softwarelebenszyklus

gestellt. Anschließend wurden Artefakte sowie deren Verbindungen als Grundbestandteile der Nachverfolgbarkeit beschrieben. Vertieft wurden die Artefakte der schriftlichen Anforderungsspezifikation und der Softwaremodellierung betrachtet. Verbindungen zwischen Artefakten unterscheiden sich hinsichtlich des Bezugs, der Repräsentationsform und des Typs. Da weder Artefakte noch deren Verbindungen ein einheitlich verstandenes Konstrukt darstellen, gibt es auch bei der Betrachtung auf der Ebene des Metamodells verschiedene Ansätze. Es wurden zwei Metamodelle vorgestellt. Abschließend wurde auf die Umsetzung der Nachverfolgbarkeit eingegangen, wobei neben den Konzepten der Erfassung auch bereits implementierte Werkzeuge vorgestellt wurden. Ebenfalls wurden die wichtigsten Einsatzmöglichkeiten aufgezeigt.

Insgesamt betrachtet, ist das Feld der Nachverfolgbarkeit in der Softwareentwicklung ein aktuelles Forschungsgebiet, für das bereits eine Vielzahl verschiedener Ansätze vorgeschlagen wurde. Bedingt durch die Heterogenität der unterschiedlichen Artefakte, konnten sich allerdings noch keine allgemein akzeptierten Standards etablieren. Eine Ausnahme bilden die Artefakttypen der Softwaremodellierung, für welche UML als Standard angesehen werden kann. Für die weiteren Artefakte und insbesondere auch für die Verbindungen zwischen den Artefakten ist dies nicht der Fall.

3 Semantische Analyse

Semantik wird als die Lehre von der Bedeutung definiert (Goddard 1998). Die semantische Analyse im Bereich der Informatik beschreibt eine Familie an Techniken, die nicht nur eine große Bandbreite an Einsatzmöglichkeiten bietet, sondern auch beständig verbessert und weiterentwickelt wird. Es kann daher nur ein Ausschnitt der Möglichkeiten präsentiert werden, wobei sich diese Arbeit ausschließlich auf die Analyse natürlicher Sprache fokussiert. Explizit nicht betrachtet wird die Analyse gesprochener Texte, da diese weitere Vorbereitungsschritte zur Transformation in die Schriftform erfordert.

Natürliche Sprache mithilfe von Algorithmen zu analysieren, ist bereits in der Vergangenheit eine der großen Herausforderungen auf dem Gebiet der künstlichen Intelligenz gewesen (Hofmann 2001). Basierend auf der Analyse natürlicher Sprache (engl. Natural Language Processing, kurz NLP), können Texte so aufbereitet werden, dass sie von Systemen für vielfache Zwecke weiterverarbeitet werden können. Das Ziel ist es, „human-like language processing“ zu erreichen (Liddy 2007, S. 2126).

Für die Analyse von Anforderungsspezifikationen in der Softwareentwicklung wurde bisher ausschließlich die Erkennung von Wortarten als basale Technologie der Bedeutungsanalyse eingesetzt. In dieser Arbeit werden auch fortgeschrittenere Technologien für die Analyse benötigt und deshalb in diesem Kapitel vorgestellt.

Im Aufbau folgt dieses Kapitel dem Konzept des ersten Grundlagenkapitels: Es wendet sich zunächst den linguistischen Grundlagen der Sprachverarbeitung zu und dann der Umsetzung. Die Umsetzung gliedert sich in die vorbereitenden Schritte sowie die verschiedenen Ansätze der Bedeutungsanalyse, wobei diese in ihrer Anordnung die basalen Technologien an den Anfang stellen. Anschließend werden die Werkzeuge und Einsatzmöglichkeiten semantischer Technologien präsentiert, bevor eine Zusammenfassung das Kapitel abschließt.

3.1 Sprachverarbeitung

3.1.1 Komplexität natürlicher Sprache

Die Entscheidung, ob ein Wort w zu einer Sprache gehört, die durch eine Grammatik gegeben ist, wird nach Pfister und Kaufmann (2008) als Wortproblem bezeichnet. Es gilt der Satz: „Das Wortproblem ist für jede Typ-1-Grammatik G lösbar, d. h. es existiert ein Algorithmus, der nach endlich vielen Schritten entscheidet, ob ein konkretes Wort w zur Sprache […] gehört oder nicht“ (Pfister und Kaufmann 2008, S. 150). Dabei bezieht sich die Typ-1-Grammatik auf die zweite Stufe der Sprachhierarchie von Chomsky, die insgesamt vier Stufen unterscheidet (vgl. Abbildung 10). Auf der ersten Stufe (Typ-0-Sprachen) werden allgemeine Sprachen angesiedelt, auf der zweiten kontextsensitive Sprachen, auf der dritten kontextfreie Sprachen und auf der vierten und letzten reguläre Sprachen. Für den Beweis des oben angegebenen Satzes sowie für die genaue Definition der Stufen der Sprachhierarchie von Chomsky wird an dieser Stelle auf die Arbeit von Pfister und Kaufmann (2008) verwiesen.

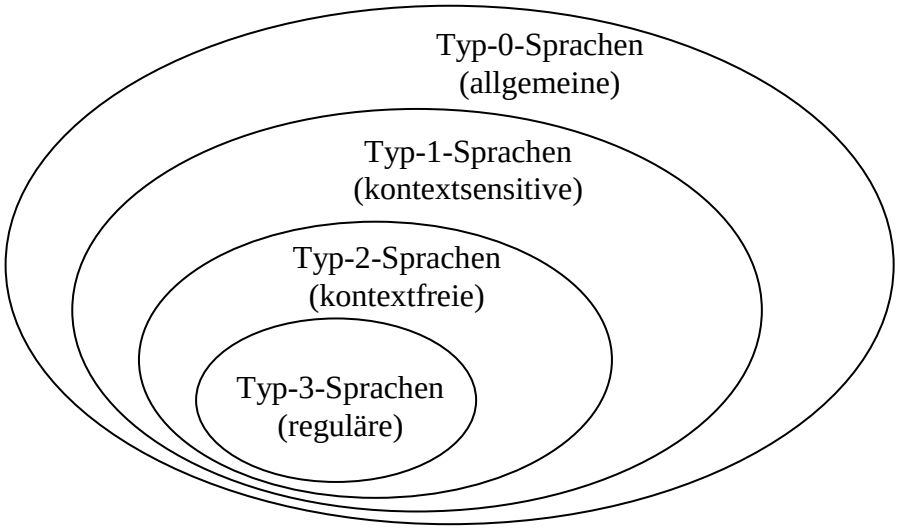


Abbildung 10: Sprachhierarchie nach Chomsky (Pfister und Kaufmann 2008)

Relevant für diese Arbeit sind die Schlussfolgerungen, die sich für die Analyse von geschriebenen Texten ergeben. Diese Schlussfolgerungen werden in der Arbeit von Winter (2010) beschrieben. Zunächst ist festzustellen, dass die Stufen der Hierarchie ungefähr die Ordnung der Komplexität reflektieren. Das heißt, je niedriger die Stufe in der Hierarchie ist, umso einfacher sind die möglichen Konstruktionen. Um natürliche Sprachen mithilfe formaler Grammatiken zu beschreiben, muss die Einordnung der Sprache in die jeweilige Hierarchiestufe bekannt sein. Nach dem Theorem von Chomsky ist Englisch keine reguläre Sprache. Ob Englisch eine kontextfreie Sprache ist, konnte noch nicht geklärt werden. Trotzdem wird im Allgemeinen davon ausgegangen, dass natürliche Sprachen in ihrer Aussagekraft über die von kontextfreien Grammatiken hinausgehen, was als trans-kontextfrei bezeichnet wird. Bisher wurde diese Eigenschaft lediglich für Niederländisch und einen Schweizer Dialekt bewiesen. Abschließend soll bemerkt werden, dass kontextfreie Grammatiken nicht ausreichen, Modelle natürlicher Sprachen zu erzeugen; hierzu sind Erweiterungen notwendig (Winter 2010). Zusammenfassend ist die Komplexität natürlicher Sprache sowie der Grammatiken, durch die sie erzeugt wird, hoch.

3.1.2 Linguistische Konzepte

Ist entschieden, ob ein Wort zu einer Sprache gehört, ist dessen Bedeutung von Interesse, was über die formale Betrachtung der Grammatik hinausgeht. Die Bedeutungsebene eines Wortes, auch eines Satzes oder Textes, wird als Semantik bezeichnet und unterliegt der Problematik des Auftretens von Mehrdeutigkeiten.

Um die verschiedenen Probleme der Eindeutigkeit begrifflich abzugrenzen, werden an dieser Stelle kurz relevante Konzepte der Linguistik (Ebert et al. 2010) vorgestellt. Das erste Konzept ist die sogenannte Synonymie, bei der zwei verschiedene Wörter auf das gleiche Konzept verweisen wie zu Beispiel „Semmel“ und „Brötchen“. Diese Wörter werden Synonyme genannt. Im Gegensatz dazu bezieht sich bei der Homonymie ein Wort auf mehrere Konzepte. Beispielsweise kann das Homonym „Bank“ zum einen auf ein Sitzmöbel und zum andern auf ein Geldinstitut referenzieren. Polysemie ist die Eigenschaft von Wörtern, verschiedene, aber

semantisch verwandte Konzepte zur Bedeutung zu haben. Ein Beispiel hierfür ist der Unterschied zwischen einer Zeitung als Institution („Die Zeitung wurde 1949 gegründet.“) und als Objekt („Die Zeitung liegt auf dem Tisch.“). Als Metonymie wird das Auftreten einer Verschiebung in der begrifflichen Interpretation bezeichnet. Im Unterscheid zur Polysemie liegt hier allerdings eine nicht-wörtliche Bedeutung vor, wie das folgende Beispiel verdeutlicht: „Das grüne Trikot hat die Sprintankunft gewonnen“. Hier steht „das grüne Trikot“ für den Träger und nicht das Kleidungsstück selbst. Die nicht-wörtliche Bedeutung ist auch eine Eigenschaft der Metaphorik, wobei bei einer Metapher eine Ähnlichkeit zwischen dem Ausdruck und der intendierten Bedeutung bestehen muss (Ebert et al. 2010).

Eine weiteres wichtiges Konzept ist die Hyponymie (Lüdeling 2009). Es beschreibt die Beziehung zwischen zwei Wörtern, wobei die Bedeutung des einen (Hyponym) in der Bedeutung des anderen (Hyperonym) enthalten ist. Beispielsweise besitzen die Hyponyme „Eiche“ und „Birke“ beide das Hyperonym „Baum“. Mengentheoretisch betrachtet, bilden Hyponyme eine Untermenge ihrer Hyperonyme. Ein ebenso bedeutsames Konzept ist die Meronymie (Kunze 2010), die sich auf die Teil-Ganzes-Beziehung zwischen zwei Substantiven bezieht. Als Beispiel bildet das Meronym „Dach“ den Teil eines Holonyms „Gebäude“. Weitere Konzepte, auf die im Folgenden zurückgegriffen werden wird, sind die Antonymie-Beziehung, bei der zwei Wörter das jeweilige Gegenteil voneinander sind, sowie die Troponymie-Beziehung, bei der ein Verb einem anderem – in der Regel spezifischeren – untergeordnet ist (Jurafsky und Martin 2009). Ein Beispiel für eine Troponymie-Beziehung ist die zwischen „sprechen“ und „flüstern“.

Ein menschlicher Leser kann die Bedeutung und Beziehung von oder zwischen Wörtern in den meisten Fällen anhand des Satzkontextes erkennen. Automatisierte Systeme stehen hingegen bei der Auflösung von Mehrdeutigkeiten vor großen Herausforderungen. Darüber hinaus ist das Erkennen morphologischer Variationen von Wörtern schwierig, was im nächsten Abschnitt erläutert wird.

3.1.3 Morphologie

Ein Wort besteht aus verschiedenen Bestandteilen. Die Bildung und Struktur von Wörtern wird als Morphologie bezeichnet (Amtrup 2010). Dabei wird zwischen einem Lemma und Wortformen unterschieden (Lüdeling 2009). Ein Lemma ist eine abstrakte Einheit und steht für eine oder mehrere Wortformen. Beispielsweise besitzen die Wortformen „laufe“, „läufst“ und „gelaufen“ das gleiche Lemma. Dieses kann beliebig benannt werden, solange die Zuordnung klar ist. In der Regel folgt die Benennung allerdings bestimmten Konventionen wie beispielsweise auf Basis der Infinitivform für ein Verb im Deutschen. Der Prozess, der ein Wort in seine Lemma-Form überführt, wird als Lemmatisierung bezeichnet.

Eine Herausforderung der linguistischen – und damit auch der automatischen – Analyse stellen zusammengesetzte Wörter wie „unhelpful“ dar. Diese können zum einen als ganzes Wort, zum anderen aber hinsichtlich des Präfixes „un-“, des Wortstamms „help“ und auch des Suffixes „-ful“ betrachtet werden (Goldsmith 2010). Ein weiteres Problem der Morphologie sind die Komparativ- und Superlativform eines Adjektivs (Lüdeling 2009). Beispielsweise sollten die Steigerungen „besser“ und „am besten“ genauso als zueinander gehörig erkannt werden wie die Reihe „schön“, schöner“ und „am schönsten“. Die Beispiele zeigen, dass Algorithmen

nicht ausschließlich über die reine Verkürzung eines Wortes auf seinen Wortstamm die passende Lösung finden können, sondern eine fortgeschrittenere Analyse benötigt wird.

Eine weitere Möglichkeit, mit der Morphologie umzugehen, stellt das sogenannte Stemming dar. Dabei ist es das Ziel, ein Wort in einen sinnvollen Wortstamm zu überführen. Im Unterschied zur Lemmatisierung, die eine zu definierende Basisform liefert, wird hier auf eine verkürzte Form eines Wortes abgezielt. Wie in der Arbeit von Willett (2006) beschrieben, werden dafür im Englischen häufig die rechtsseitigen Wortbestandteile abgetrennt. Dabei können zwei Arten von Fehlern auftreten. (1) Als Over-truncation wird bezeichnet, wenn zu viel abgetrennt wird und damit ein zu kurzer Wortstamm verbleibt, der nicht ausreichend zwischen zwei ansonsten unverwandten Wörtern differenziert. Ein Beispiel hierfür sind die englischen Wörter „medical“ und „media“, welche beide fälschlicherweise ein Wortstammelement „med*“ zugeordnet bekommen könnten. (2) Dem gegenüber steht die Under-truncation, bei der zu wenige Buchstaben abgetrennt werden. Zum Beispiel wäre die Zuordnung des Wortstammelements „bibliographic*“ für das Wort „bibliographically“ unzureichend, um auch das Wort „bibliography“ zu erfassen. Passender wäre hier das Wurzelelement „bibliograph*“ (Willett 2006).

Das automatisierte Erstellen der Lemma-Form bzw. des Wortstamms eines Wortes stellt somit neben der Bedeutungszuordnung einen weiteren Aufgabenbereich von Algorithmen dar. Wie diese Ziele konkret umgesetzt werden, wird im folgenden Unterkapitel dargestellt.

3.2 Umsetzung

Nachdem mit der Komplexität der Sprache, den linguistischen Konzepten und der Morphologie einige wichtige Grundlagen und Herausforderungen der Sprachverarbeitung vorgestellt wurden, stehen nun die Lösungsansätze im Vordergrund. Zunächst werden die vorbereitenden Schritte und anschließend wird die Bedeutungsanalyse selbst angesprochen. Wie auch schon im Themenbereich der Nachverfolgbarkeit, werden außerdem die vorhandenen Werkzeuge sowie deren verschiedene Einsatzmöglichkeiten aufgezeigt.

Zwei in der Informationsverarbeitung häufig verwendete Gütekriterien sind Genauigkeit (engl. precision) und Trefferquote (engl. recall). Diese Gütekriterien werden in einzelnen der folgenden Abschnitte aufgegriffen werden. Für die Definition (Olson und Delen 2008) werden die möglichen Ergebnisse einer Zuordnung in vier Ergebnisklassen eingeteilt: (1) korrekt als zugehörig klassifiziert (engl. true positive, TP), (2) fälschlicherweise als zugehörig klassifiziert (engl. false positive, FP), (3) fälschlicherweise nicht als zugehörig klassifiziert (engl. false negative, FN) und (4) korrekt nicht als zugehörig klassifiziert. Die Genauigkeit ist definiert als der Quotient TP / (TP+FP) und die Trefferquote als der Quotient TP / (TP+FN). Häufig werden diese Gütekriterien im Kontext der Suche nach Dokumenten eingesetzt. In Anlehnung an Manning et al. (2009) können die Quotienten auch allgemein wie folgt interpretiert werden: Die Genauigkeit gibt an, wie viele relevante Ergebnisse vom System zurückgeliefert werden, während die Trefferquote angibt, wie viele relevante Ergebnisse bezogen auf die Grundmenge – inklusive der nicht gefundenen – das System zurückliefert. Diese beiden Gütekriterien stehen in einem inversen Verhältnis zueinander, d. h., dass mit steigender Genauigkeit die Trefferquote sinkt und umgekehrt (Frakes und Baeza-Yates 1992).

3.2.1 Vorbereitung der Analyse

Bevor ein Text analysiert werden kann, muss dieser durch verschiedene Maßnahmen vorbereitet werden. Selbst die grundlegende Frage, was genau ein Wort auszeichnet, ist nicht trivial. In nicht-segmentierten Schriftsystemen, wie beispielsweise dem Japanischen, können die Piktogramme aufgrund fehlender Leerzeichen und Interpunktion nicht explizit voneinander abgegrenzt werden. Aber auch in segmentierten Schriftsystemen, zu denen auch das Deutsche und das Englische gehören, treten Probleme aufgrund von Ambiguitäten auf.

Das Problem der Ambiguität betrifft nicht nur Wörter, sondern auch Interpunktionszeichen selbst (Hagenbruch 2010). So kann ein Punkt als Satzzeichen oder nach einer Abkürzung auftreten. Auch Bindestriche oder Apostrophe können problematisch sein. Der Bindestrich kann zum einen Bedeutungseinheiten trennen, also beispielsweise „Blaue Reiter-Ausstellung“, sie zum anderen aber auch verbinden wie in „Hochschul-Strukturkommission“. Außerdem kann der Bindestrich am Zeilenende stehen, wo in bestimmten Fällen ebenfalls die Bedeutung verschieden interpretiert werden kann, wie in „Streik-Ende“ im Gegensatz zur Trennung von „der Streik-ende“.

Der erste Schritt der Textanalyse ist somit die Zerlegung in sinnvolle Einheiten unter Beachtung der bereits aufgeführten Problematiken. Ein weiterer – optionaler – Schritt ist die Lemmatisierung von Wörtern, die im Abschnitt 3.1.3 bereits hinsichtlich ihres linguistischen Hintergrundes vorgestellt wurde. Es stehen darüber hinaus weitere Schritte zur Verfügung, die im Folgenden kurz aufgezeigt werden.

3.2.1.1 Tokenisierung

Die Zerlegung des Textes bildet die Grundlage für alle weiteren möglichen Analyseschritte. Allgemein wird diese Zerlegung als Segmentierung bezeichnet, wobei der Begriff auch die Zerlegung von Texten in größere Abschnitte wie Sätze oder Absätze bezeichnen kann (Hagenbruch 2010). Die Tokenisierung zielt auf die Zerlegung in Token ab. Als Token werden in diesem Zusammenhang linguistisch zusammengehörige Einheiten bezeichnet, also z. B. ein Wort, aber auch zusammengesetzte Bezeichner wie der aus mehreren Wörtern bestehende Name einer Stadt wie „San Francisco“.

Es stehen mehrere Methoden für die Verarbeitung von Texten zur Verfügung, unter anderem Maximum-Entropy-Modelle, Entscheidungsbäume oder künstliche neuronale Netze. Maximum-Entropy-Modelle, auch als log-lineare Modelle bezeichnet, versuchen sowohl die Randbedingungen der realen Welt einzubeziehen als auch bekanntes Wissen durch die Wahl bestimmter Wahrscheinlichkeitsverteilungen abzubilden (Malouf 2010). Als Basis für die Maximum-Entropy-Modelle wird die Informationstheorie von Shannon (1951) herangezogen, nach der die Entropie ein statistischer Parameter ist. Dieser Parameter misst, wie viel Information im Durchschnitt von einem Buchstaben in einem Text einer gegebenen Sprache produziert wird. Wird die Sprache beispielsweise so effizient wie möglich in binärer Form kodiert, dann ist die Entropie die durchschnittliche Anzahl der binären Ziffern, die pro Buchstabe benötigt wird. Die Wahrscheinlichkeitsverteilung wird dabei in der Art gewählt, dass sowohl die Entropie maximiert wird als auch die Randbedingungen erfüllt werden (Malouf 2010).

Entscheidungsbäume können unter anderem für die Bearbeitung von Klassifikationsproblemen eingesetzt werden (Schmid 2010). Ein solches Klassifikationsproblem stellt beispielsweise die Auflösung von Mehrdeutigkeiten von Begriffen dar. Wie bereits einleitend erwähnt, können Punkte in englischen und auch in deutschen Texten zum einen das Ende eines Satzes markieren, zum anderen nach einer Abkürzung folgen; oder beides kann gleichzeitig zutreffen. Abbildung 11 zeigt einen möglichen Entscheidungsbaum zur Lösung dieses Problems.

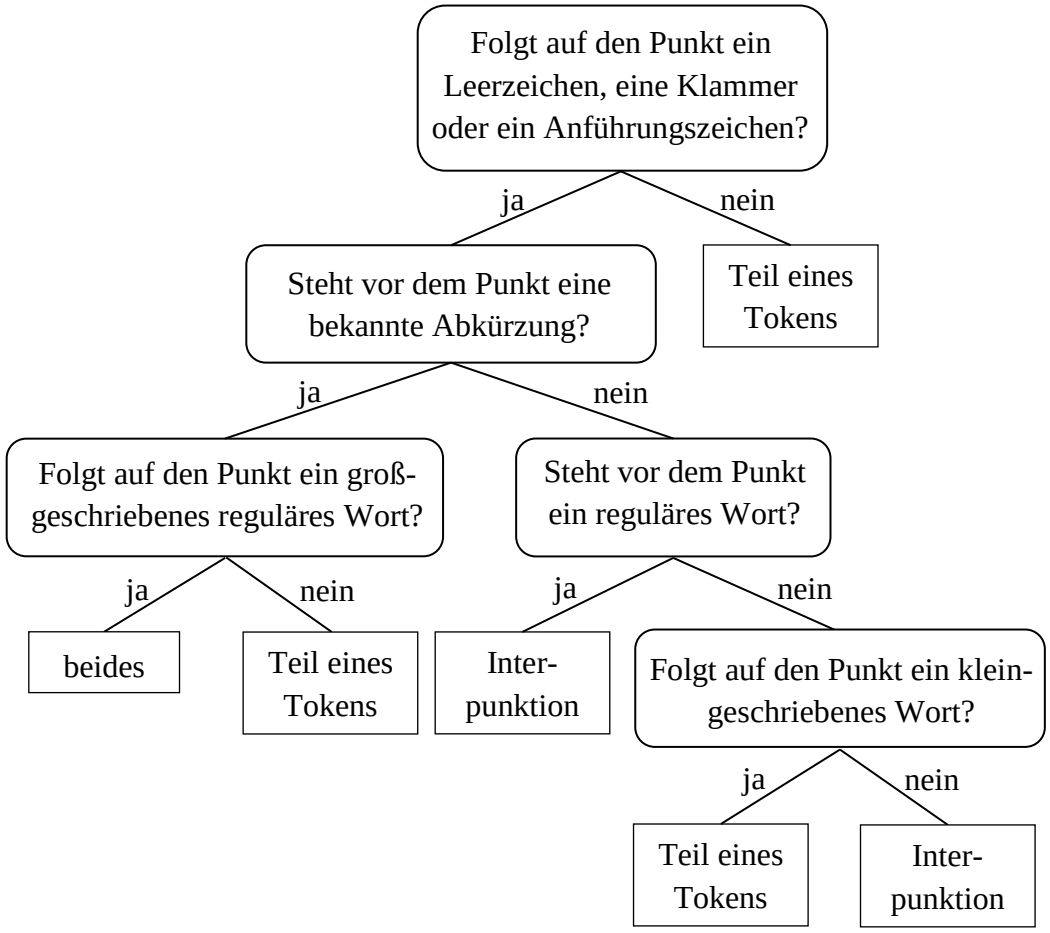


Abbildung 11: Entscheidungsbaum für einen Punkt (Malouf 2010)

Entscheidungsbäume sind eine schnelle Art der Klassifikation, sowohl hinsichtlich des Trainings als auch der für die Verarbeitung benötigten Zeit. Des Weiteren sind sie einfach zu erstellen, gut verständlich und liefern größtenteils akkurate Ergebnisse. Allerdings sind sie in ihrer Grundform nicht hinreichend leitungsstark, sodass mehrere Weiterentwicklungen vorgeschlagen wurden (Schmid 2010).

Eine weitere statistische Methode der Sprachverarbeitung sind künstliche neuronale Netze, insbesondere mehrschichtige Perzeptronennetze, die in der Arbeit von Henderson (2010) vorgestellt werden. Diese mehrschichtigen Perzeptronennetze bestehen aus mehreren Verarbeitungseinheiten, die in der Regel mit Trainingsdaten arbeiten. Bei mehrschichtigen Netzen, also solchen mit mindestens einer mittleren Schicht, ist es möglich, auch nicht-lineare Funktionen zwischen den Eingabe- und den Ausgabedaten abzubilden. Teilweise wird argumentiert, dass künstliche neuronale Netze auch zur Modellierung von Maximum-Entropy-Modellen eingesetzt werden können (Henderson 2010).

Alle hier vorgestellten Methoden werden der statistischen Sprachmodellierung zugeordnet. Diese Methoden beruhen nach Pfister und Kaufmann (2008) auf der Annahme, dass Sprache über einen Zufallsprozess erzeugt wird. Dahingegen werden Modelle, die grammatikalisches Expertenwissen wie die Verbkonjugation enthalten, als wissens- oder regelbasierte Sprachmodelle bezeichnet. Dieses Expertenwissen muss explizit in das Modell eingebracht werden, während statistische Verfahren in der Regel anhand von Datensätzen trainiert werden. Diese Trainingsdatensätze stellen einen Schwachpunkt der statistischen Sprachmodellierung dar, da für gute Resultate die Erstellung geeigneter Datensätze in der Praxis sehr aufwendig ist (Pfister und Kaufmann 2008). Ein entscheidender Nachteil der regelbasierten Verfahren ist es, dass die Erstellung eines zuverlässigen Systems aufwendig ist und nicht ohne Weiteres auf andere Dokumente übertragen werden kann (Hagenbruch 2010). Statistische Verfahren sind hier anpassungsfähiger; es existieren ebenfalls statistische Verfahren, die keine Trainingsdatensätze benötigen. Diese werden als unbeaufsichtigtes Lernen (engl. unsupervised learning) bezeichnet und beruhen auf der Annahme, dass beispielsweise Interpunktionszeichen nur in Ausnahmefällen uneindeutig sind und sich Regeln aus den eindeutigen Fällen ableiten lassen (Hagenbruch 2010).

Unabhängig vom eingesetzten Algorithmus bleiben diverse Problemfälle bestehen, wie zum Beispiel die Behandlung von gemischten Token wie „mp3“, zusammengesetzten Token wie „anglo-amerikanisch“ oder mehrteiligen Token wie der eingangs erwähnte Städtename „San Francisco“. Um solche Probleme zu beheben, empfehlen Perkuhn et al. (2012) den Einsatz mehrere Tokenisierer. Dabei sollte sowohl ein konservativer als auch ein radikaler Tokenisierer verwendet werden, also einer, der möglichst viele Zeichen als Wortbestandteil annimmt, beziehungsweise einer, der dies für möglichst wenige Zeichen durchführt. Beide Ergebnisse werden abschließend anhand von Segmentierungspunkten synchronisiert.

3.2.1.2 Lemmatisierung

Um dem bereits angesprochene Problem der Morphologie (vgl. 3.1.3) zu begegnen, können vollautomatische Algorithmen verwendet werden. Diese werden auch als Stemmer bezeichnet. Es können vier verschiedene Typen des Stemming (Baeza-Yates und Ribeiro-Neto 2011) unterschieden werden, von denen zwei kurz erläutert werden sollen. Zum Ersten können die Wörter nachgeschlagen und anschließend durch die zugehörige Form ersetzt werden. Auch wenn diese Methode wenig komplex in der Durchführung ist, so setzt sie doch eine entsprechende Datenbasis voraus, die wiederum zu hohem Speicheraufwand führt und damit für die praktische Umsetzung je nach Einsatzumgebung ungeeignet sein kann. Die zweite Methode beinhaltet das Abtrennen von Affixen, also vor allem von Suffixen in Sprachen wie dem Englischen, in welchen Wörter vornehmlich in ihren Endungen variieren. Für das Englische wird dabei häufig der Porter-Algorithmus eingesetzt, der trotz seiner Simplizität Resultate liefern kann, die vergleichbar mit anderen, komplexeren Algorithmen sind. Das Vorgehen ist dabei so aufgebaut, dass anhand bestimmter Regeln und Listen, Suffixe identifiziert und vom Wort abgetrennt oder ersetzt werden. Beispiele hierfür sind das Plural „s“, also die Verkürzung von „customers“ zu „customer“, oder die Entfernung der Endung „ing“ bei Verben (Baeza-Yates und Ribeiro-Neto 2011). Die Regeln werden über verschiedene Schritte hinweg angewandt. Für eine ausführliche Darstellung wird auf die Arbeit von Frakes und Baeza-Yates (1992, S. 139 ff.) verwiesen.

Anhand des bereits in Abschnitt 3.1.3 vorgestellten Beispiels „unhelpful“ ist ersichtlich, dass ein automatischer Stemmer allein auf Basis der Entfernung von Suffixen nicht immer zu einer zufriedenstellenden Lösung führen kann. Dies trifft insbesondere auf Sprachen zu, in denen häufig Zusammensetzungen auftreten, wie zum Beispiel zusammengesetzte Substantive im Deutschen, oder in denen viele Unregelmäßigkeiten vorkommen, wie zum Beispiel im Spanischen (Baeza-Yates und Ribeiro-Neto 2011). Ebenfalls angesprochen wurde bereits das Problem des Umgangs mit Steigerungsformen wie „besser“. Daher ist für ein gutes Ergebnis in diesen Fällen das Nachschlagen von Wörtern in einem Wörterbuch ebenfalls notwendig. Inzwischen existieren auch Versionen des Porter-Algorithmus, die Ausnahmen im Englischen wie die Zuordnung von „skies“ zu „sky“ beherrschen, sowie solche für weitere Sprachen wie Deutsch, Russisch oder Französisch. Diese stehen in der eigens erschaffenen Sprache Snowball im Internet zur Verfügung[6].

Sobald es möglich ist, Wörter nachzuschlagen, verschwinden die Grenzen zwischen Lemmatisierung und Stemming, da es anhand der definierten Ersetzungen möglich wird, jede gewünschte Form eines Wortes zu erreichen. Die jeweiligen Regeln sollten daher in Einklang mit dem Ziel dieses Vorbereitungsschritts entwickelt werden. Nicht immer ist Stemming notwendig, sodass dieser Schritt als optional zu betrachten ist.

3.2.1.3 Weitere Schritte und Überblick

Es können weitere Schritte der Vorbereitung durchlaufen werden, die die Textanalyse im Folgenden vereinfachen. Zum einen können sogenannte Stop Words aus dem Text entfernt werden. Es handelt sich dabei um Wörter, die eine grammatikalische Funktion haben, aber keinen Beitrag zum Inhalt leisten (Wilbur und Sirotkin 1992) und somit irrelevant für die weitere Analyse sind. Beispiele hierfür sind die Wörter „aber“, „falls“ oder Artikel wie „die“. Auch Interpunktionszeichen und Zahlen können – je nach Kontext – als irrelevant eingestuft werden. In Texten mit statistischen Inhalten können aber auch Zahlen von Bedeutung sein. Eine Möglichkeit, solche Listen mit Stop Words zu erstellen, liegt in der Durchführung einer Häufigkeitsanalyse basierend auf dem vorliegenden Set an Dokumenten. Es können ebenfalls Listen eingesetzt werden, die auf der generellen Häufigkeit von Wörtern beruhen. Eine solche Liste mit 421 Wörtern wird in der Arbeit von Fox (1989) für das Englische anhand eines großen Textkorpus (neutrum: das Korpus) erstellt. Die häufigsten englischen Wörter sind laut dieser Liste „the“, „and“ und „a“. Allerdings sind unter den häufigsten Wörtern auch solche, die durchaus wichtig sein können, wie beispielsweise „business“ oder „system“. Des Weiteren weist der Autor darauf hin, dass Wörter mit Prä- und Suffixen gesondert berücksichtigt werden müssen. Bei der Erstellung bzw. Auswahl einer solchen Liste muss daher sorgfältig vorgegangen werden und unter Umständen auch der Kontext berücksichtigt werden. Ein großer Vorteil der Eliminierung von Stop Words ist die Reduktion des untersuchten Textes, die bis zu 40 % betragen kann (Baeza-Yates und Ribeiro-Neto 2011). Ein Nachteil ist, dass zusammengehörige Phrasen unter Umständen im Anschluss an die Stop Word-Eliminierung nicht mehr erkannt werden können. Je nach Ziel der Analyse muss daher abgewogen werden, wie und ob dieser Schritt ausgeführt werden soll. Sollen zusammengehörige Phrasen wie „Sein oder nicht Sein“ erkannt werden, so sollte die Eliminierung von Stop Words nur mit Einschränkungen oder gar nicht erfolgen.

[6] http://snowball.tartarus.org/texts/stemmersoverview.html

Ein weiterer Vorbereitungsschritt ist bei Bedarf die Transformation aller Groß- in Kleinbuchstaben. Die Überführung eines Textes mithilfe einer Auswahl der hier insgesamt vorgestellten Schritte – oder unter Umständen auch weiterer individueller Maßnahmen – in eine bestimmte Form wird in verschiedenen Quellen als Normalisierung bezeichnet. Da dieser Begriff nicht einheitlich verwendet wird, werden in dieser Arbeit die verwendeten Schritte jeweils explizit genannt. Ein Beispiel dafür, wie die Vorbereitung der Analyse eines Textes ablaufen kann, wird in Abbildung 12 gezeigt. Dabei werden nacheinander die Schritte der Tokenisierung, der Eliminierung der Stop Words und als letztes der Lemmatisierung für einen kurzen Beispieltext durchlaufen.

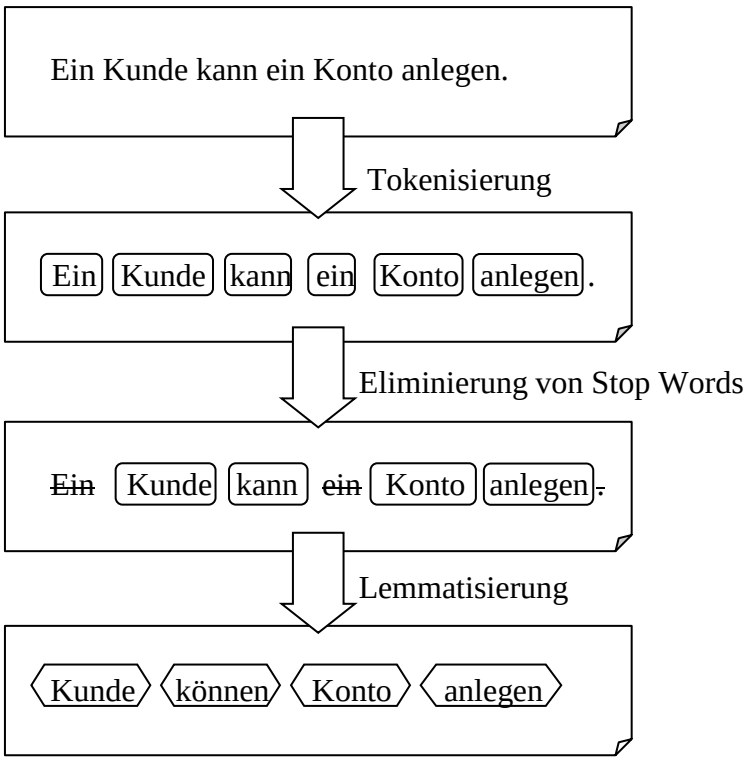


Abbildung 12: Beispiel für die Vorbereitung der Analyse

Nachdem ein Text auf diese Weise vorbereitet wurde, kann im Anschluss dessen Inhalt analysiert werden. Eine Auswahl für die dafür zur Verfügung stehenden Techniken wird im folgenden Unterkapitel vorgestellt.

3.2.2 Bedeutungsanalyse

Im Folgenden werden verschiedene algorithmische Ansätze vorgestellt, mit der die für diese Arbeit relevanten Analyseziele erreicht werden können. Die Reihenfolge der Unterabschnitte entspricht der aufsteigenden Größe des Analyseobjekts: Zunächst werden einzelne Wörter analysiert, dann zusammenhängende Wörter, darauf folgend ganze Sätze, und schließlich wird eine Analyse gezeigt, die über die Grenzen des Satzes hinweg arbeitet. Damit einhergeht eine Steigerung der Komplexität der Ansätze, wobei die ersten Unterabschnitte eher basale Techniken beschreiben, während die letzteren sich mit fortgeschritteneren Techniken auseinandersetzen.

3.2.2.1 Erkennung der Wortarten

Die mithilfe der Tokenisierung und gegebenenfalls weiterer Vorbereitungsschritte erhaltenen Wörter können mit grammatikalischen Informationen angereichert werden. Ein solches Verfahren zur Erkennung von Wortarten wird als Part-of-Speech (POS) Tagging bezeichnet. Da inzwischen Genauigkeiten von 97 %–98 % erreicht werden, kann POS Tagging als ein gelöstes Problem betrachtet werden (Giesbrecht und Evert 2009). Dies bedeutet, dass sich keine entscheidenden Verbesserungen mehr erreichen lassen (Perkuhn et al. 2012).

Wie schon bei der Tokenisierung (vgl. 3.2.1.1) können nach Hagenbruch (2010) auch hier regelbasierte oder stochastische Methoden eingesetzt werden. Stochastische Verfahren arbeiten dabei mit der Wahrscheinlichkeit, mit der ein Wort einer bestimmten Wortart angehört. Regelbasierte Verfahren ordnen in einem ersten Schritt jedem Wort das Tag zu, mit welchem es im Trainingskorpus am häufigsten annotiert war. Die Genauigkeit liegt für diesen ersten Schritt bei ungefähr 90 % und wird anschließend iterativ verbessert. Unabhängig vom gewählten Verfahren hat dabei die Auswahl des Tagsets, also der Menge aller zur Verfügung stehenden Tags, einen großen Einfluss auf die Genauigkeit des Ergebnisses. Je detailreicher das Tagset, desto mehr Unterscheidungen muss der POS Tagger treffen (Hagenbruch 2010). Besonders in Sprachen mit reichhaltiger Morphologie, wie beispielsweise dem Russischen, können Tagsets sehr umfangreich werden (Hajič und Hladká 1998). Auf der anderen Seite ergeben sich in diesem Fall auch weniger Ambiguitäten (Hagenbruch 2010).

Ein für das Englische einsetzbares Tagset ist das der Penn Treebank. Es enthält insgesamt 36 verschiedene Tags für Wörter sowie zusätzlich 12 Tags für Sonderzeichen wie „$“ oder „)“. Das Tagset wurde basierend auf einem 4,5 Mio. Wörter umfassenden Korpus erstellt (Marcus et al. 1993). Tabelle 1 zeigt eine Auswahl der Tags, die jeweilige Beschreibung und Beispiele.

Tag	Englische Beschreibung	Deutsche Beschreibung	Beispiel
CC	conjunction	Konjunktion	and, but, neither, therefore
CD	numeral, cardinal	Ziffer	forty-two, ten, 0.5, 1982
FW	foreign word	Fremdword	fiche, fille, jouer, oui
JJ	adjective	Adjektiv	first, separable, multilingual
JJR	adjective, comparative	Adjektiv in Komparativform	brighter, broader, clearer, closer
DT	determiner	Bestimmungswort	every, many, some, the
NN	noun	Substantiv	shed, slide, override, machinist
NNS	noun, plural	Substantiv in Pluralform	products, designs, facets, jobs
RB	adverb	Adverb	occasionally, prominently
VB	verb, base form	Verb in Grundform	ask, assess, begin, benefit
VBD	verb, past tense	Verb in Vergangenheitsform	registered, adopted, speculated
VBG	verb, present participle or gerund	Verb im Partizip Präsens oder Gerundium	focusing, capping, approaching, traveling

Tabelle 1: Bespiele für Tags der Penn Treebank (Atwell 2013)

Ein weitaus umfangreicheres Tagset für das Englische stellt NUPOS zur Verfügung (Mueller 2009). Mit über 200 Tags ist eine sehr viel differenzierte Analyse möglich, die auch für literarische Formen wie beispielsweise das inzwischen veraltete „thou art“ eingesetzt werden kann. Sollen hingegen nur übergeordnete Wortarten betrachtet werden, so kann ein universelleres Tagset verwendet werden wie z. B. das von Petrov et al. (2011). Dieses wurde auf Basis mehrerer Tagsets sowie 22 verschiedener Sprachen erstellt, darunter auch Arabisch, Chinesisch, Deutsch und Russisch. Das Ziel ist es, nur die notwendigsten und damit nützlichsten Kategorien bereitzustellen, die auch über mehrere Sprachen hinweg zu finden sind. Es wird daher zum Beispiel auf verschiedene Unterkategorien von Substantiven oder Verben verzichtet. Insgesamt verfügt dieses Tagset über 12 Tags für Substantive (NOUN), Verben (VERB), Adjektive (ADJ), Adverbien (ADV), Pronomen (PRON), Bestimmungswörter und Artikel (DET), Prä- und Postpositionen (ADP), Numerale (NUM), Konjunktionen (CONJ), Partikel (PRT), Satzzeichen (‘.’) sowie eine Restekategorie (X), die alle anderen Wörter wie Abkürzungen und Fremdwörter beinhaltet. Die unterschiedliche Verwendung des Penn Treebank Tagsets und des universalen Tagsets von Petrov et al. (2011) wird in Abbildung 13 anhand eines Beispielsatzes gegenübergestellt.

Satz:	The	oboist	Heinz	Hollinger	has	taken	a	hard	line	about	the	problems	.
Penn Treebank:	DT	NN	NNP	NNP	VBZ	VBN	DT	JJ	NN	IN	DT	NNS	.
Universal:	DET	NOUN	NOUN	NOUN	VERB	VERB	DET	ADJ	NOUN	ADP	DET	NOUN	.

Abbildung 13: Gegenüberstellung des Penn Treebank Tagsets und des universalen Tagsets (Petrov et al. 2011)

Für das universelle Tagset stehen Zuordnungen für alle einbezogenen Tagsets zur Verfügung, sodass es auch für deutsche Texte eingesetzt werden kann. Als das populärste deutsche Tagset bezeichnet Hagenbruch (2010) das Stuttgart-Tübingen-Tagset (TSST). Es enthält ein kleines Tagset mit 11 und ein großes Tagset mit 54 verschiedenen Tags (Schiller et al. 1999). Das kleine Tagset deckt sich bis auf kleine Abweichungen mit dem universellen Tagset von Petrov et al. (2011). Es wurde, mit kleinen Modifikationen, auf ein größeres Textkorpus angewandt, das als Referenzkorpus dienen kann (Brants et al. 2002). Dieses TIGER Treebank genannte annotierte Korpus basiert auf der Analyse von Artikeln großer deutscher Tageszeitungen.

Ein für englische Texte häufig verwendetes Textkorpus ist das Brown Korpus, das aus einer großen Bandbreite englischsprachiger Veröffentlichungen erstellt wurde (Fox 1989); inzwischen ist es im Korpus der Penn Treebank enthalten. Die Penn Treebank enthält darüber hinaus ebenfalls Texte aus Zeitungen, wie dem Wall Street Journal (Marcus et al. 1993). Tagsets werden in Verbindung mit einem bestimmten Textkorpus entwickelt und getestet. Da es sich häufig um Zeitungstexte handelt, besteht die Gefahr der Überanpassung an diese speziellen Daten, und es bleibt unklar, inwieweit andere Genres damit akkurat abgedeckt werden können (Giesbrecht und Evert 2009). Eine Verbesserungsmöglichkeit zur Erreichung einer möglichst breiten Literaturabdeckung ist der Ausbau des Referenzkorpus. So wurden für die Erstellung des NUPOS Tagsets unter anderem alle Werke von Shakespeare sowie einzelne Texte von Emily Brontë oder Herman Melville verwendet (Mueller 2009). Ein Hauptproblem, das allen POS Taggern gemeinsam ist, ist das Erkennen unbekannter Wörter (Hagenbruch 2010). In der Regel werden die umstehenden Wörter in die Analyse mit einbezogen, es existiert aber keine

global richtige Lösung. Häufig handelt es sich bei unbekannten Wörtern um Eigennamen, deren Erkennung weiter unten behandelt werden wird.

3.2.2.2 Disambiguierung

Als Disambiguierung wird die Auflösung von Mehrdeutigkeiten bezeichnet. Dabei wird der Begriff für zwei verschiedene Verfahren eingesetzt: zum Ersten für die Zuordnung des korrekten POS Tags und zum Zweiten zur inhaltlichen Unterscheidung beim Auftreten von Homonymen (vgl. 3.1.2).

Im Kontext der Zuordnung von POS Tags soll folgendes Beispiel von Jurafsky und Martin (2009) die Problematik verdeutlichen: Das Wort „book“ kann im Englischen sowohl als Verb („to book a flight“) als auch als Substantiv („to read a book“) auftreten. Im Englischen besteht zwar für die Mehrheit der Wörter keine Ambiguität, das heißt, ihnen ist genau ein Tag zuordenbar. Allerdings sind viele der am häufigsten auftretenden Wörter uneindeutig (Jurafsky und Martin 2009). So enthält das bereits erwähnte Brown Korpus 11,5 % ambigue Worttypen, aber insgesamt sind 40 % aller Token mehrdeutig (DeRose 1988). Das englische Wort mit dem höchsten Ambiguitätsgrad ist in dieser Analyse „still“ mit sieben verschiedenen möglichen Tags. DeRose legt gleichzeitig einen Algorithmus zur Disambiguierung vor, der über eine Genauigkeit von 96 % verfügt. Zur Ausführung dieses Algorithmus werden zunächst allen Wörtern alle jeweils möglichen Tags zugeordnet. Anschließend werden die Wörter in Form eines Stapelverfahrens analysiert. Dabei werden zunächst die ersten beiden Wörter betrachtet und mit einer Matrix abgeglichen, die die Häufigkeiten für die Kollokationen von Wortarten enthält. Diese Matrix basiert auf der Berechnung eines Referenzkorpus, sie ist in Tabelle 2 ausschnittsweise dargestellt. Es wird hier das Tagset des Brown Korpus[7] verwendet. Dieses ist im Vergleich zu der bereits vorgestellten Penn Treebank deutlich umfangreicher und erlaubt auch zusammengesetzte Tags. Bei vielen Tags bestehen Übereinstimmungen: NN für Substantive, RB für Adverbien, VB für die Basisform eines Verbs sowie VBD für die Vergangenheitsform eines Verbs. Zusätzlich für dieses Beispiel relevant sind die Tags AT für Artikel und PP$ für Possessivartikel (engl. possessive determiner). Aus Tabelle 2 ersichtlich ist beispielsweise, dass nach einem Artikel häufig ein Substantiv folgt, selten aber nach einem Verb direkt ein Substantiv folgt.

		Zweites Wort				
		NN	PP$	RB	VB	VBD
Erstes Wort	AT	186	0	8	1	8
	NN	40	1	40	9	66
	PP$	176	0	5	1	1
	RB	5	16	71	118	152
	VB	22	146	98	9	1
	VBD	11	143	160	2	1

Tabelle 2: Beispiele für Kollokationshäufigkeiten (nach DeRose 1988)

Für das Stapelverfahren werden die Werte für die ersten beiden Wörter in der Tabelle nachgeschlagen und die Tags mit dem höchsten Wert für die Kollokationshäufigkeit ausgewählt; anschließend wird das nächste Wort zum Stapel hinzugefügt. Nun werden die Werte für jede

[7] http://www.comp.leeds.ac.uk/ccalas/tagsets/brown.html

mögliche Kombination der Tags der beiden obersten auf dem Stapel befindlichen Elemente multipliziert und das Ergebnis mit dem höchsten Produkt als optimaler Pfad gespeichert. In jedem folgenden Prozessschritt werden ebenfalls die zwei vorhergehenden Elemente auf dem Stapel berücksichtigt, sodass der Algorithmus den Pfad sukzessive aus den jeweils besten Ergebnissen aufbaut, die bei der Kombination aus zwei aufeinander folgenden Tags entstehen. Bei bestimmen Wörtern kann sich dieses Vorgehen allerdings als Irrweg erweisen; so handelt es sich bei „time“ unabhängig vom Kontext so gut wie immer um ein Substantiv. Insgesamt ist der Algorithmus anderen Verfahren überlegen wie z. B. dem Verwenden großer Datenbanken, bestehend aus manuell erstellten Kontextregeln, und dem CLAWS-Ansatz von Marshall (1987). CLAWS berechnet die Werte nicht paarweise, sondern für alle möglichen Pfade und besitzt damit eine exponentielle Laufzeit, wohingegen der eben vorgestellte Algorithmus eine lineare Laufzeitabschätzung ermöglicht (DeRose 1988). Es existieren neben diesen stochastischen Verfahren ebenfalls regelbasierte Ansätze, z. B. von Voutilainen (1999), oder solche, die beide Verfahren kombinieren wie der Brill Tagger (Brill 1992) und dessen Weiterentwicklungen, z. B. von Acedański (2010) oder Schneider und Volk (1998).

Wie eingangs beschrieben, kann Disambiguierung auch das Vorgehen bei der inhaltlichen Unterscheidung von Homonymen bezeichnen. Die Problematik geht dabei über die des POS Taggings hinaus. So besitzt im Deutschen beispielsweise das Wort „Läufer“ die meisten Bedeutungen (Dengel 2012) – insgesamt sechs verschiedene, unter anderem die einer Schachfigur, eines Teppichformats oder eines Sportlers –, bei denen das zugeordnete POS Tag jeweils korrekterweise gleich wäre. Auch wenn die Problematiken der Disambiguierung im Rahmen von POS Tagging und inhaltlicher Natur (engl. word sense disambiguation, WSD) ähnlich scheinen, so stellen Jayanthi und Prema (2011) heraus, dass „algorithms used for one do not tend to work well for the other, mainly because the part of speech of a word is primarily determined by the immediately adjacent one to three words, whereas the sense of a word may be determined by words further away“ (S. 360). Die Autorinnen sehen einen weiteren Grund für die Unterschiede darin, dass für WSD nur wenige Trainingsdaten vorliegen. In Folge dieser Schwierigkeiten wurde für die WSD in dieser Studie nur eine Genauigkeit von 75 % erreicht. Jayanthi und Prema betonen allerdings, dass diese Zahl nur für das Englische gilt und für andere Sprachen stark abweichen kann. Insgesamt steht für die WSD eine große Bandbreite algorithmischer Ansätze zur Lösung bereit, von denen Entscheidungsbäume oder neuronale Netze bereits weiter oben angesprochen wurden (vgl. 3.2.1.1). Einen ausführlichen Überblick über die verschiedenen Verfahren gibt die Arbeit von Navigli (2009). Dengel (2012) meint, dass im Allgemeinen sprachliches Hintergrundwissen in Form eines Thesaurus benötigt wird. Ein Thesaurus ist dabei ein Set von Elementen, Wörtern oder Phrasen sowie ein Set von Beziehungen der Elemente untereinander (Jing und Croft 1994). Als Beispiel für einen Thesaurus-basierten Ansatz benennen Jurafsky und Martin (2009) den Lesk-Algorithmus. Dieser Algorithmus geht in seiner vereinfachten Form zunächst von der häufigsten Wortbedeutung aus und bezieht anschließend die umliegenden Wörter in die Analyse mit ein. Gibt es eine Bedeutung, für die sich eine größere Überlappung zwischen diesem Kontext und der Defintion im Thesaurus ergibt, so wird diese gewählt, bis alle Bedeutungen abgearbeitet wurden. In den erweiterten Versionen des Algorithmus werden auch Worte einbezogen, die im Thesaurus nicht in der Zieldefinition enthalten, aber mit dieser verwandt sind. Insgesamt betrachtet, lieferte zu diesem Zeipunkt ein solcher wissensbasierter Ansatz in Kombination

mit überwachten Lernstrategien die besten Ergebnisse für WSD-Probleme (Jurafsky und Martin 2009). Liegt kein Thesaurus vor, können mehrere Korpora parallel zueinander verwendet werden (Ide et al. 2002; Tufiş et al. 2004).

3.2.2.3 Erkennung benannter Entitäten

Die Erkennung benannter Entitäten oder auch NER (engl. named entity recognition) ist eine zentrale Aufgabe innerhalb der NLP; sie wird in der Arbeit von Neumann (2010) beschrieben. Eigennamen sind dabei sprachliche Ausdrücke, die sich auf Individuen von Klassen oder Typen beziehen, wie z. B. die Namen von Personen, Produkten oder Unternehmen, oder auch Ausdrücke für Datums-, Zeit-, Orts- oder Maßangaben. Ein Ansatz wäre es, eine Liste aller möglichen benannten Entitäten zu erstellen. Allerdings ist dies aufgrund der Variabilität und Produktivität von Eigennamen unzureichend. Ein Beispiel für die Variabilität sind die verschiedenen Schreibweisen, die allein ein Personenname haben kann. So kommt der Name „Albert Einstein“ auch in der Schreibweise „A. Einstein“, „Einstein, A.“ oder „Prof. Einstein“ vor. Für Produkt- oder Unternehmensnamen ist die Vielfalt bei der Namensfindung als Alleinstellungsmerkmal wichtig, sodass hier stetig neue Namen kreiert werden.

Aus diesen Gründen, so argumentiert Neumann, sollten überwachte und semi-überwachte Strategien angewandt werden, die anhand eines bereits annotierten Trainingskorpus sowie verschiedener Merkmale eine Klassifikation vornehmen. Diese Merkmale können sich auf den Eigennamen selbst, beispielsweise Groß- und Kleinschreibung oder Wortlänge, sowie auf den Kontext des Wortes beziehen. Die eingesetzten Verfahren sind häufig Anpassungen bekannter Verfahren wie beispielsweise Maximum-Entropie-Modellen (vgl. 3.2.1.1). Wie bereits erläutert, benötigen diese Verfahren in der Regel Trainingsdaten, die bei einer großen Anzahl verschiedener Klassen von benannten Entitäten auch ein sehr großes Trainingskorpus benötigen. Deshalb wurden semi-überwachte Verfahren entwickelt, die als zentrale Technologie das Bootstrapping beinhalten. Dabei wird anhand einer kleinen Menge an benannten Entitäten, sogenannten Seeds, versucht, entsprechende Muster zu erkennen, und Wörter, die diesen Mustern entsprechen, werden wiederum der Liste der Seeds hinzugefügt. Dieser Prozess wird im Anschluss immer wieder ausgeführt. Der Prozess kann entweder manuell beendet werden oder wenn das Abbruchkriterium erreicht wird, also keine weiteren Einträge mehr gefunden werden. Die Nachteile dieses Verfahrens liegen darin, dass ein größeres Korpus im Vergleich zum überwachten Lernen benötigt wird und außerdem die Seedelemente äußerst sorgfältig ausgewählt werden müssen (Neumann 2010).

Es existieren ebenfalls nicht-überwachte Verfahren, welche unter Einbezug externer lexikalischer Ressourcen arbeiten (Nadeau und Sekine 2007). Auf Basis dieser Ressourcen kann dann die automatische Annotierung stattfinden. Hierfür stehen untere anderem Ansätze zur Verfügung, die häufig gemeinsam auftretende Wörter in einem Korpus berücksichtigen oder solche, die das zeitgleiche Auftreten von Wörtern in mehreren Nachrichtenquellen einbeziehen (Nadeau und Sekine 2007). Eine häufig benutze externe Ressource stellt WordNet[8] dar. Diese Datenbank wird an der Universität Princeton entwickelt und bietet eine frei zugängliche Verlinkung semantischer Konzepte. Es können zum Ersten Wörter identifiziert werden, die synonym zueinander verwendet werden. Zum Zweiten können Meronymie- sowie Hyponymie-

[8] http://wordnet.princeton.edu

Beziehungen erfasst werden. Die Hyponymie-Beziehung ist die häufigste Art der Verlinkung und wird in der Informatik auch als „is-a"- oder Vererbungsbeziehung referenziert. Meronymie-Beziehungen werden in der Informatik häufig als „has-a"- oder „has-part"-Beziehungen bezeichnet. Drittens bietet WordNet die Möglichkeit des Ordnens von Verben in Hierarchien von unspezifisch zu spezifisch (Troponymie-Beziehung) und auch von Adjektiven in Gegensatzpaaren (Antonymie-Beziehung, beide vgl. 3.1.2). Die Mehrheit der Beziehungen in WordNet bestehen innerhalb bestimmter Wortarten, sodass vier Subnetzte für Substantive, Verben, Adjektive und Adverbien existieren. Zusätzlich werden auch Beziehungen zwischen Wörtern basierend auf ihrer Morphologie (vgl. 3.1.3) berücksichtigt. Zusammen spannen diese Beziehungen ein semantisches Netz auf, welches ausschnittsweise in Abbildung 14 visualisiert wird.

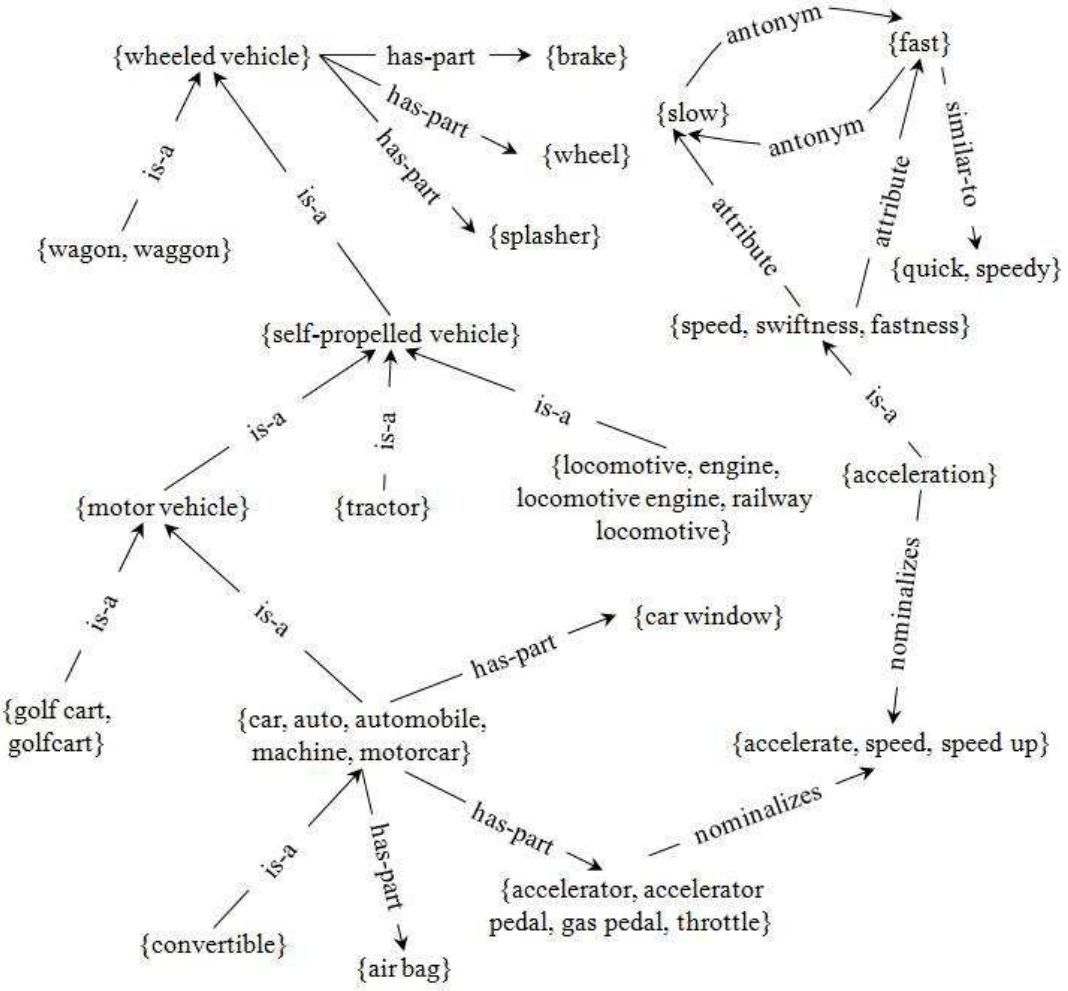


Abbildung 14: Ausschnitt eines semantischen Netzes aus WordNet (Navigli 2009)

Für die Analyse deutscher Texte kann GermaNet verwendet werden, welches in das EuroWordNet integriert ist, das insgesamt acht europäische Sprachen umfasst (Kunze 2010). Diesen Wortnetzen gemeinsam ist die Menge der Base Concepts. Diese 1000 Substantive und 300 Verben dienen der Kompatibilität der einzelnen Sprachnetze untereinander. EuroWordNet bzw. seine jeweiligen Unternetze müssen lizenziert werden; es ist damit im Gegensatz zu WordNet nicht frei verfügbar. Eine frei zugängliche Alternative für eine sprachübergreifende Erkennung benannter Entitäten wurde kurz nach der Jahrtausendwende von einer Initiative

der „Conference on Computational Natural Language Learning“ (CoNLL) als sogenannter Shared Task vorangetrieben9. Es werden verschiedene Sets für Kategorien angeboten, beispielsweise werden unterschieden: Personen (engl. persons, PER), Orte (engl. locations, LOC), Organisationen (engl. organizations, ORG) und weitere (engl. miscellaneous, MISC). Trainingsdatensätze stehen ebenfalls zur Verfügung. Im Rahmen des Shared Task entstand eine ganze Reihe von Veröffentlichungen (z. B. Florian et al. 2003; Klein et al. 2003; Zhang und Johnson 2003), die als Teil der Aufgabenstellung verschiedene Lernalgorithmen auf ein deutsches sowie ein englisches Trainingskorpus anwenden. Sie kommen – wie bereit für die Tokenisierung (vgl. 3.2.1.1) beschrieben – zum besten Ergebnis, wenn mehrere Verfahren parallel eingesetzt werden.

Ob der mit der Implementierung einer algorithmischen Lösung verbundene Aufwand in der Praxis allerdings von den Benutzern akzeptiert wird, ist fraglich. So wird laut Dengel (2012) für die Erkennung benannter Entitäten meistens auf Listen für den Abgleich von Wörtern zurückgegriffen, trotz guter Argumente gegen dieses Verfahren. Diese Listen mit bekannten Personen, Orten, Organisationen oder anderen Entitäten werden auch als Gazetteer bezeichnet (Mikheev et al. 1999).

3.2.2.4 Syntaktische Analyse

Nachdem die bisherigen Unterabschnitte sich auf die Analyse einzelner Token bezogen, werden im Folgenden auch die statischen Beziehungen von Wörtern untereinander betrachtet. Eine dafür eingesetzte Technologie ist das sogenannte Parsen (engl. parsing), welches zum Ziel hat, die syntaktischen Strukturen eines Satzes zu bestimmen (Nederhof und Satta 2010). Dabei wird in der Regel ein Baum aufgebaut (Manning und Schütze 1999), wie dies in Abbildung 15 gezeigt wird. S steht dabei für Satz (engl. simple clause, sentence), NP für Nominalphrase (engl. noun phrase), VP für Verbphrase (engl. verb phrase) und PP für Präpositionalphrase (engl. prepositional phrase). Die einzelnen Tags, mit Ausnahme von P für Partikel, sind Tabelle 1 zu entnehmen.

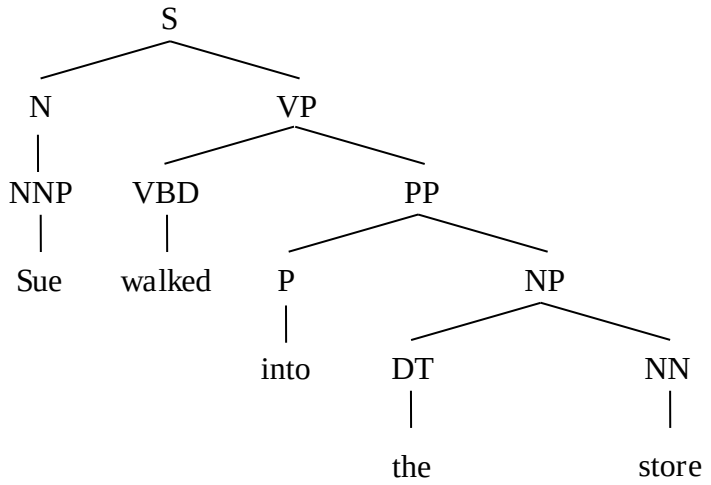


Abbildung 15: Parse-Baum (Manning und Schütze 1999)

[9] http://www.clips.ua.ac.be/conll2003/ner

Anhand der Beispielsätze „John broke the window.“ und „The window broke.“ wird deutlich, dass Parsen allein zwar die Zuordnung von Subjekt und Objekt des Satzes erlaubt, nicht aber die verschiedenen syntaktischen Rollen berücksichtigt, die diese spielen (Palmer et al. 2005). So ist „window“ in einem Satz das syntaktische Objekt, im andern das syntaktische Subjekt, obwohl die gleiche Entität referenziert wird. Es ist daher notwendig, eine syntaktische Annotation zu ergänzen. Hierzu werden zwei verschiedene Ansätze vorgestellt.

Bei der semantischen Annotierung von Rollen (engl. semantic role labeling) wird darauf abgezielt, den semantischen Zusammenhang zwischen einem Verb und seinen Argumenten herzustellen. Es existiert kein generell anerkanntes Set für diese Annotationen, aber es gibt prototypische Rollen (Jurafsky und Martin 2009). Beispielsweise wird zwischen einem Verursacher (engl. agent) und einem Beeinflussten (engl. patient, experiencer) unterschieden. Jurafsky und Martin benennen als weitere prototypische Rollen unter anderem Resultat, Instrument, Begünstiger, Quelle oder Ziel der ausgeführten Handlung. Als Basis für eine algorithmische Lösung stehen verschiedene bereits annotierte Korpora zur Verfügung, wie die auf der Penn Treebank aufgebaute Proposition Bank, kurz PropBank. Für die Erstellung der PropBank wurden zunächst regelbasierte Tagger verwendet, das Ergebnis wurde anschließend von Hand korrigiert (Palmer et al. 2005). In der PropBank bildet das Verb die Ausgangsbasis für die Annotierung. Die Argumente im Satz werden nummeriert; für jede Verbbedeutung werden diese Argumente jeweils spezifisch einer Rolle zugeordnet. Weiterhin werden verschiedene Tags für Beifügungen vergeben, welche als Nummer „ArgM“ sowie einen näher bestimmenden Zusatz beispielsweise für Ort, Zeit, Ausmaß oder Richtung erhalten. Als Beispiel soll folgender annotierter Satz dienen: „[$_{Arg0}$ Chuck] bought [$_{Arg1}$ a car] [$_{Arg2}$ from Jerry] [$_{Arg3}$ for $1000]”. Für die Wortbedeutung des Verbs „buy“ können nun die Rollen für die einzelnen Argumente in der PropBank nachgeschlagen werden: Arg0 wird „buyer“ zugeordnet, Arg1 erhält das Tag „thing bought“ und Arg3 das Tag „price paid“ (Palmer et al. 2005). Für diese Art der Annotierung befindet sich zurzeit ein Vorschlag für einen ISO-Standard[10] für die Annotierung semantischer Rollen in Bearbeitung (Bunt und Palmer 2013).

Neben dieser Rollen-bezogenen Analyse können auch Beziehungen zwischen zwei Wörtern innerhalb eines Satzes betrachtet werden. Wie für die semantische Rollenannotierung liegt auch hier keine Standardisierung vor, es wird daher die häufig zitierte Arbeit von Marneffe und Kollegen (Marneffe und Manning 2008; Marneffe et al. 2006) herangezogen. Auf Basis eines Parse-Baums werden dazu unter Verwendung eines regelbasierten Algorithmus sogenannte typisierte Beziehungen (engl. typed dependencies) erstellt. Eine typisierte Beziehung ist dabei folgendermaßen aufgebaut: Beziehungsbezeichner(Governor, Dependent). Für den in Abbildung 15 gezeigten Beispielsatz „Sue walked into the store.“ sieht das Ergebnis die Analyse wie folgt aus:

nsubj(walks, Sue) = nominal subject
root(ROOT, walks)
prep(walks, into) = preposition
det(store, the) = determiner
pobj(into, store) = object of preposition

[10] International Organization for Standardization

Die Beziehungen sind hierarchisch aufgebaut, wobei „root“ die generischste der Beziehungen ist. Für die weiteren Beziehungen wird jeweils ein möglichst spezifischer Beziehungsbezeichner vergeben, wobei beispielsweise Objekte in direkte (dobj), indirekte (iobj) und präpositionale (pobj) Objekte unterschieden werden. Darüber hinaus können die Beziehungen auch weiter miteinander verbunden werden, sodass zusammengezogene Beziehungen (engl. collapsed dependencies) entstehen. Damit können beispielsweise die Präposition und das präpositionale Objekt in die zusammengezogene Beziehung „prep_into(walks, store)“ überführt werden. Innerhalb dieser zusammengezogenen Beziehungen existiert auch der Beziehungsbezeichner „agent“, welcher die Identifikation eines Handelnden in einer passiven Satzkonstruktion ermöglicht. Dies ist vergleichbar mit der semantischen Rollenannotation, wobei diese einen auf das Verb ausgerichteten Fokus besitzen. Typisierte Beziehungen hingegen unterstützen darüber hinaus eine Vielzahl an weiteren Beziehungen, wie solche zwischen Substantiven und ihren Adjektiven (engl. adjectival modifier). Für den Satz „He has a red car.“ wird beispielsweise die typisierte Beziehung „amod(car, red)“ als eines der Ergebnisse ermittelt. Es liegt kein standardisiertes Set für die Beziehungsbezeichner vor; ein Überblick über die Beziehungsbezeichner des hier vorgestellten Ansatzes sowie deren genaue Beschreibung ist Marneffe und Manning (2012) zu entnehmen.

3.2.2.5 Koreferenzanalyse

Zur Erfassung semantischer Zusammenhänge genügt es unter Umständen nicht, einen einzelnen Satz zu analysieren. Beispielsweise kann es wünschenswert sein, auch den Zusammenhang zu erkennen, wenn in einem Satz von „Albert“ gesprochen wird und im nächsten von „er“. Bereits an diesem einleitenden Beispiel ist die Schwierigkeit ersichtlich, die eine solche Koreferenzanalyse beinhaltet. Noch deutlicher wird die Problematik bei Betrachtung des Beispiels von Dengel (2012), wobei die beiden Sätze „Albert lehnt sich aus dem Fenster.“ und „Er hatte es gestern offen gelassen.“ gleich zwei Korefenzketten beinhalten. Koreferenz bezeichnet im Kontext dieser Arbeit das Auftreten von zwei sprachlichen Segmenten, die auf die gleiche Entität referenzieren (Dengel 2012). Dabei können diese sprachlichen Segmente verschiedenen Wortarten angehören. Ein Beispiel für Koreferenzketten wird in Abbildung 16 gezeigt.

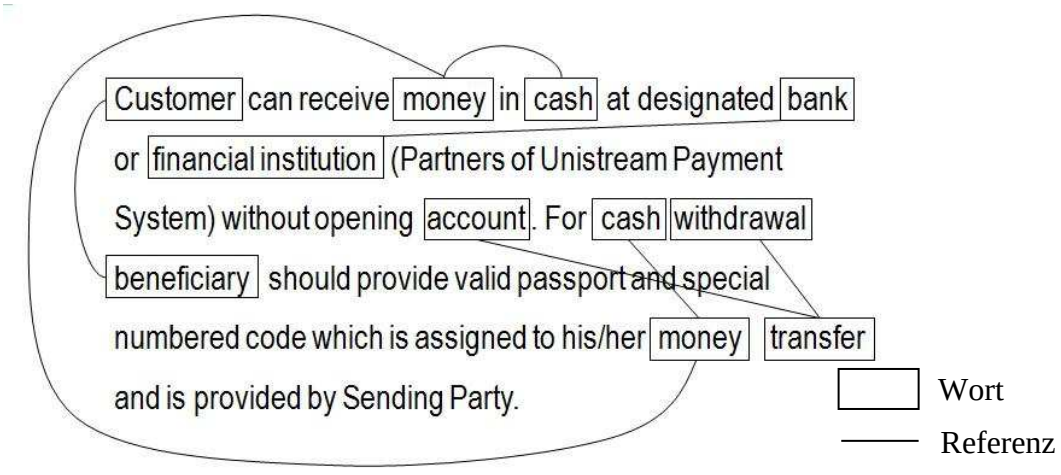


Abbildung 16: Beispiele für Koreferenzketten (Navigli 2009)

Zur Identifikation von Koreferenzen unterscheiden Jurafsky und Martin (2009) zunächst verschiedene Typen referenzierender Ausdrücke: Nominalphrasen, Pronomen, Demonstrativpronomen sowie Namen. Während bestimmte Typen – wie beispielsweise Pronomen – überwiegend auf bekannte Entitäten zurückreferenzieren, stellen andere – wie insbesondere bestimmte Nominalphrasen – dem Leser neue Entitäten vor, wie „Fenster“ im obigen Beispiel.

Um zu untersuchen, ob es sich um eine bereits bekannte oder eine neue Entität handelt, können auch hier verschiedene Verfahren eingesetzt werden. Jurafsky und Martin bieten dazu einen historischen Überblick, der die Grundlagenforschung zeitlich bereits in den späten 1970er-Jahren verortet. Dass die Lösung der Koreferenz-Problematik ebenfalls eine hochaktuelle Herausforderung der Sprachanalyse ist, zeigt sich in den CoNLL Shared Tasks (siehe 3.2.2.3) der Jahre 2011 und 2012, die sich mit dieser Thematik befassen. Alle eingereichten Beiträge wurden anhand von Trainingsdatensätzen getestet; als bester Beitrag 2011 erwies sich der eines Teams der Universität Stanford (Lee et al. 2011). Dieser beruht auf der Arbeit von Raghunathan et al. (2010), die ein sogenanntes Sieb über mehrere Durchgänge hinweg einsetzt. Der Sieb-Algorithmus unterscheidet sich erheblich von anderen Lernalgorithmen, da nicht versucht wird, alle Referenzen im Text aufzulösen. Stattdessen wird mit Heuristiken gearbeitet. So wird beispielsweise davon ausgegangen, dass neue Entitäten eher am Anfang eines Textes auftreten und damit gleichzeitig weniger vorhergehende Kandidaten für eine Referenz existieren. Des Weiteren wenden die meisten herkömmlichen Ansätze für die Identifikation einer Koreferenz jeweils eine einzige Funktion auf ein Paar aus zwei Ausdrücken an. Dabei wird eine kleine Zahl wichtiger Eigenschaften oft von einer größeren Zahl weniger wichtiger Eigenschaften verdrängt. Dem gegenüber werden im Sieb-Algorithmus mehrere Durchgänge mit absteigender Präzision durchgeführt, wobei jeder Durchgang auf dem Ergebnis des vorhergehenden aufbaut. Im ersten Durchgang werden zunächst nur Entitäten mit exakter Übereinstimmung gesucht. Anschließend werden im zweiten Durchgang Entitäten gesucht, die einer Reihe von sehr spezifischen Regeln entsprechen, welche sich unter anderem auf Relativpronomen, Abkürzungen oder auch Demonyme[11] beziehen. In den folgenden Durchgängen werden mit abnehmender Strenge zusammengehörige Substantive identifiziert. Dabei wird davon ausgegangen, dass Entitäten bei späteren Erwähnungen kürzer und mit weniger Informationen versehen sind, wie beispielsweise „… intervene in the [Florida Supreme Court]’s move … does look like very dramatic change made by [the Florida court] …“ (S. 495). Im letzten Durchgang werden die bis dahin gebildeten Cluster auf Pronomenbezogene Koreferenzen hin untersucht. Dabei werden bestimmte Randbedingungen – beispielsweise hinsichtlich Geschlecht oder Anzahl – aufgestellt, denen die Koreferenzen genügen müssen. Das Ausführen dieses letzten Durchgangs führte zu einer Erhöhung der paarweisen Trefferquote um 22 %.

Insgesamt ist der Sieb-Algorithmus im Vergleich zu überwachten und nicht-überwachten Ansätzen einfacher aufgebaut. Trotzdem erzielte er bessere Ergebnisse als diese im Vergleich aller Einreichungen und ist überdies einfach zu erweitern. Erweiterungen können beispielsweise über das Hinzufügen weiterer Durchgänge erfolgen, wie dies auch in dem bereits ange-

[11] Verwandtschaft von Wörtern basierend auf Ländern, deren Einwohnern oder ethischen Gruppen, wie z. B. „Deutschland“ und „die Deutschen“ (Grouin et al. 2011)

sprochenen Beitrag zum Shared Task von Lee et al. (2011) umgesetzt wurde. Inwieweit der Sieb-Algorithmus für alle Sprachen gleichermaßen geeignet ist, bleibt allerdings offen: Bereits im folgenden Shared Task 2012 – diesmal mit mehrsprachigem Fokus – gewann ein Beitrag von Fernandes et al. (2012), dessen Ansatz wieder auf einer Kombination nichtheuristischer Algorithmen basiert. Auch für das Deutsche liegen aktuelle Ansätze zur Koreferenzanalyse vor (z. B. Cahill und Riester 2012).

3.2.3 Werkzeuge

Die Menge der verfügbaren Werkzeuge zur semantischen Analyse ist sehr unübersichtlich. Allein die Auswahl[12] von Christopher Manning, Professor für Linguistik und Informatik an der Universität Stanford, umfasst mehr als 80 verschiedene Softwarepakete für verschiedene NLP-Bereiche. Laut Varjú et al. (2012) ist das Natural Language Toolkit (NLTK) das populärste Werkzeug zur Sprachanalyse. Es basiert auf der Programmiersprache Python und wurde ursprünglich für pädagogische Zwecke im Bereich der Linguistik entwickelt (Bird und Loper 2004). NLTK besteht aus mehreren Modulen, darunter die häufig als Standard betrachteten Module für Tokenisierung, Stemming und POS Tagging (Bird 2006). Darüber hinaus bringt es Module für bestimmte fortgeschrittenere Analyseverfahren mit, wie die Erkennung benannter Entitäten und die Korefernzanalyse. In der Dokumentation von NLTK[13] wird ebenfalls die Funktionalität beschrieben, externe Datenbanken wie WordNet einzulesen und darauf basierend eine Funktionalität für WSD anzubieten. Auch für die PropBank existiert eine Schnittstelle, was die Annotierung semantischer Rollen ermöglicht. Als die verlässlichsten und am weitesten entwickelten Java-basierten Werkzeuge bezeichnen Varjú et al. (2012) Stanford CoreNLP[14] und Apache OpenNLP[15]. Beide integrieren die Standardfunktionalitäten Tokenisierung, Stemming und POS Tagging. Die Erkennung benannter Entitäten, die Disambiguierung von POS Tags, Parsen sowie die Koreferenzanalyse stehen ebenfalls zur Verfügung, wobei Letztere in Apache OpenNL erst kürzlich hinzugefügt wurde und daher noch eingeschränkt arbeitet sowie nur wenig dokumentiert ist.

Alle drei Werkzeuge unterstützen die Verarbeitung mehrerer Sprachen, wobei dafür in der Regel Erweiterungen installiert oder zusätzliche Daten über Schnittstellen eingelesen werden müssen. Außerdem stehen die drei Werkzeuge unter Lizenzen, die die Nutzung zur Forschung ohne Einschränkungen erlauben. Sie sind darüber hinaus auch quelloffen, sodass eigene Erweiterungen möglich sind. Wie auch für NLTK, stehen in Stanford CoreNLP Methoden für die Einbindung von WordNet und PropBank zur Verfügung. Für Apache OpenNLP enthält die Dokumentation keine der genannten Schnittstellen. Allerdings gibt es ein als experimentell deklariertes Paket zur Konvertierung von Korpora. Der entscheidende Vorteil der Verwendung des Stanford CoreNLP ist dessen Implementierung für die Analyse von typisierten Beziehungen. Für die Implementierungen komplexerer Funktionalitäten der Bedeutungsanalyse ist es damit eine gute Option unter den Java-basierten Werkzeugen. Alterativ zu der Verwendung von Java stellt als freie Softwareumgebung die Programmiersprache R[16] eine weitere Möglichkeit dar, über verschiedene Pakete bestimmte semantische Analysen zu im-

[12] http://nlp.stanford.edu/links/statnlp.html
[13] http://nltk.org/index.html
[14] http://nlp.stanford.edu/software/corenlp.shtml
[15] http://opennlp.apache.org
[16] http://www.r-project.org

plementieren. Beispielsweise stellt das Paket Isa Funktionalitäten wie Stemming oder die Eliminierung von Stop Words zur Verfügung.

Neben den klassischen, Desktop-basierten Werkzeugen werden inzwischen auch Web-Services angeboten, die NLP-Funktionalitäten bereitstellen. Ein sehr umfangreiches Angebot an verschiedenen Funktionen bietet AlchemyAPI[17], welches unter anderem die Erkennung benannter Entitäten, die Extraktion von Konzepten und Themen sowie die Identifikation der Sprache beinhaltet, in der ein Text verfasst wurde. Allerdings ist die freie Verwendung nur bis zu einer begrenzten Menge an Anfragen pro Tag möglich. Ein weiterer Web-Service ist OpenCalais[18]. Im Unterschied zu AlchemyAPI wird Deutsch nicht unterstützt, und auch der Funktionsumfang ist etwas geringer. Dafür wird für jede gefundene Entität ein Prozentwert für deren Relevanz angegeben. Auch hier gibt es eine Grenze für die kostenfrei pro Tag möglichen Anfragen. Die beiden Werkzeuge verwenden jeweils eine eigene Liste an Tags und zielen auch auf unterschiedliche Entitäten ab. Es gibt Überschneidungen, wobei die Eingabe des gleichen Textes nicht zu identischen Ergebnissen führt. Beispielsweise wird der Textausschnitt „Northeastern University in Boston“ von OpenCalais als Organisation identifiziert, während von AlchemyAPI „Boston“ getrennt als Stadt getaggt wird. Insgesamt bieten beide Services einen großen Umfang an Möglichkeiten zur Anreicherung von Text mit Metadaten an, wobei hier vor allem Konzepte oder Themen im Vordergrund stehen; die Annotierung von Wortarten ist nicht möglich. Diese ist über den Web-Service des CLAWS-Projekts[19] (vgl. Abschnitt 3.2.2.2) umsetzbar.

3.2.4 Einsatzmöglichkeiten

Die Menge der elektronisch zur Verfügung stehenden Texte wächst sehr schnell, sodass die Notwendigkeit einer semantischen Analyse beständig größer wird. Dabei sind deren Einsatzzwecke vielfältig. Beispielsweise erwartet ein Nutzer, dass Texte in Nachrichtenarchiven thematisch geordnet sind. Hier wird in den meisten Fällen ein Redakteur die Einordnung in Kategorien wie „Sport“, „Wirtschaft“ etc. vornehmen. Mit dem zunehmenden Aufkommen von Benutzer-generierten Inhalten wird es in Zukunft unumgänglich sein, automatisierte Verfahren für diese Einordnung zu verwenden. Weitere fortgeschrittene Verfahren werden für die Analyse von Meinungen, Gefühlen und Fakten eingesetzt. Die Erkennung von Meinungen oder Gefühlen (engl. sentiment analysis) ermöglicht unter anderem die Klassifizierung einer schriftlichen Produktbewertung als positiv oder negativ (Pang und Lee 2008). Die Erkennung von Fakten (Dengel 2012) befasst sich mit der Identifizierung von Umschreibungen. So können die Sätze „Albert Einstein erblickte in Ulm das Licht der Welt.“ und „Der Geburtsort von Albert Einstein ist Ulm.“ als identisches Faktum erkannt werden. Die oben beschriebene Koreferenzanalyse wird unter anderem in der Medizin beispielweise für die Verbesserung der Zugänglichkeit und der Qualität von Patientenakten eingesetzt (z. B. Uzuner et al. 2013; Zheng et al. 2013).

Neben der Analyse von Texten können diese auch transformiert werden. Ein häufiger Einsatzzweck ist die automatische Übersetzung in andere Sprachen, die inzwischen gute Ergeb-

[17] http://www.alchemyapi.com
[18] http://www.opencalais.com
[19] http://ucrel.lancs.ac.uk/claws

nisse erzielt, aber auch beständig weiterentwickelt wird (Way 2010). Weiterhin kann der Umfang einzelner Texte reduziert werden, indem automatisch Zusammenfassungen erstellt werden. Diese Idee ist nicht neu und wurde bereits vor der Jahrtausendwende erforscht (eine Übersicht bieten Mani und Maybury 1999). Allerdings gelangt sie durch die Verbreitung mobiler Endgeräte zu neuer Bedeutung, da diese vermehrt zum Abrufen von Informationen verwendet werden, aber im Vergleich zur Nutzung eines stationären Rechners über einen kleineren Bildschirm verfügen und für kürzere Zeiträume (z. B. Überbrückung von Wartezeiten unterwegs) genutzt werden. Aktuell wurde die für das iPhone angebotene Applikation Summly von Yahoo für einen Betrag von 23 Mio. Euro gekauft; diese kürzt Nachrichtentexte auf eine Länge von maximal 400 Zeichen (Heise 2013b).

Eine weitere zukunftsträchtige Technologie ist die Beantwortung von Fragen (engl. question answering), die weit über die Eingabe von Stichwörtern in Suchmaschinen hinausgeht (Webber und Web 2010). Stattdessen erfolgt die Eingabe als ganzer Satz, wobei auch die Antwort in diesem Format erwartet wird und nicht eine Liste von Links zu eventuell relevanten Webseiten. Bekannte Beispiele sind die Applikationen Siri für iPhone oder Google Now für Android-basierte Geräte. Wie auch Textzusammenfassungen, ist die Beantwortung von Fragen als Idee nicht neu. Sie ist im Gegenteil eine der ältesten Aufgaben von NLP-Systemen und wurde bereits in den 1960er- und 70er-Jahren thematisiert (Jurafsky und Martin 2009). Erst kürzlich wurden die Fortschritte auf diesem Gebiet sichtbar, als das von IBM entwickelte System Watson 2011 die Spielshow Jeopardy gegen menschliche Mitspieler gewann (Markoff 2011). Watson führt dabei eine statistische Analyse für mehrere Alternativen parallel durch und berechnet anhand eines Trainingsdatensatzes das beste Ergebnis (Murdock et al. 2012). Im Hintergrund wird für die Berechnung ein eigens entwickeltes Cluster benötigt, damit ausreichend Rechenleistung bereitgestellt werden kann (Brown et al. 2013). In Zukunft soll diese Technologie unter anderem im Kundenservice eingesetzt werden (Heise 2013a).

3.3 Zusammenfassung

Dieses Kapitel befasste sich mit ausgewählten semantischen Technologien, wobei zunächst die linguistischen Grundlagen vorgestellt wurden. Neben der inhärenten Komplexität natürlicher Sprache wurde verdeutlicht, vor welchen Herausforderungen die Sprachanalyse steht.

Von der Bedeutungsanalyse abgegrenzt wurden als vorbereitende Schritte die Tokenisierung und das Stemming vertieft betrachtet. Während mit der Tokenisierung die Zerlegung des Textes in analysierbare Einheiten ein grundlegender und unumgänglicher Schritt ist, handelt es sich beim Stemming um einen optionalen Schritt, der Problemen der Morphologie entgegenwirken kann. Ein weiterer – ebenfalls optionaler – Schritt der Vorbereitung ist die Entfernung irrelevanter Stop Words. Bei der anschließenden Vorstellung verschiedener Techniken der Bedeutungsanalyse konnte jeweils nur ein Ausschnitt der vorhandenen Algorithmen vorgestellt werden, da für die einzelnen Teilgebiete eine große Bandbreite überwachter, semiüberwachter und unüberwachter Verfahren vorliegt. Die Auswahl eines Algorithmus muss anhand des vorliegenden Problems getroffen werden, teilweise werden mehrere Algorithmen miteinander kombiniert. Wo zutreffend, stellt die Auswahl eines bestimmten Textkorpus zum Training sowie eines bestimmten Sets zur Annotierung weitere Entscheidungsaspekte dar.

Unter Umständen werden auch externe Ressourcen wie WordNet oder PropBank für die Analyse benötigt.

Insgesamt handelt es sich bei der semantischen Analyse um ein forschungsintensives Feld, in dem einige Probleme als bereits gelöst betrachtet werden, während andere Fragestellungen Gegenstand aktueller Forschungstätigkeit sind, wie die Erkennung von Meinungen oder Gefühlen sowie die Erkennung von Fakten. Dies zeigt sich auch in der Menge der vorhandenen Werkzeuge sowie der Breite der Einsatzmöglichkeiten, für die aktuelle Beispiele aufgeführt wurden.

4 Semantische Technologien in der Softwareentwicklung

Dieses Kapitel stellt den aktuellen Forschungsstand für den Einsatz semantischer Technologien in der Softwareentwicklung dar und bildet damit den Bezugsrahmen dieser Arbeit. Zunächst werden die wichtigsten Einsatzziele vorgestellt. Anschließend werden Möglichkeiten für die Umsetzung sowie eine Auswahl der vorhandenen Werkzeuge beschrieben.

4.1 Einsatzziele

Der Prozess der Softwareentwicklung ist eng mit der Erfassung und Bearbeitung von Wissen verbunden. Robillard beschreibt die Softwareentwicklung als „the progressive crystallization of knowledge into a language that can be read and executed by a computer" (1999, S. 92). Wissen wird beispielsweise während der Erfassung von Anforderungsspezifikationen von den Gedanken der Beteiligten in eine schriftliche Form überführt. Des Weiteren entsteht während des Softwareentwicklungsprozesses selbst Wissen über das Softwareprojekt. Dieses Wissen muss den Beteiligten zugänglich gemacht werden, sodass ein Bedarf nach Wissensrepräsentation entsteht. Die verschiedenen Formen der Wissensrepräsentation, insbesondere in Form von Ontologien, werden im ersten Abschnitt betrachtet. Liegt Wissen bereits in der formalisierten Gestalt eines Modells vor, so sind in der Softwareentwicklung drei Einsatzziele von Bedeutung. Zum einen soll Wissen repräsentiert werden, was in Abschnitt 1 beschrieben wird. Außerdem sollen verschiedene Modelle ineinander transformiert werden, womit sich der zweite Abschnitt befasst. Zudem sollen Modelle validiert werden, was im dritten Abschnitt beschrieben wird.

4.1.1 Wissensrepräsentation

Der Begriff der Wissensrepräsentation wird in verschiedenen Disziplinen verwendet und soll deshalb kurz abgegrenzt werden. In der Psychologie ist Wissensrepräsentation die Manifestierung von Wissen im Gedächtnis (Robillard 1999), in der Logik die Verwendung mathematischer Formeln und logischer Operatoren zum automatischen Schließen (Brachman und Levesque 2004). In der vorliegenden Arbeit wird das Verständnis von Stock und Stock (2008) aufgegriffen, die Wissensrepräsentation als den Einsatz von Techniken, Methoden und Werkzeugen beschreiben, mit dem Ziel, Wissen in digitalen Datenbanken optimal auffindbar zu machen. Als grundlegende Wissensordnungen werden Nomenklaturen, Klassifikationen, Thesauri und Ontologien genannt. Des Weiteren werden facettierte Wissensordnungen beschrieben, die jedoch den Rahmen der vorliegenden Arbeit übersteigen. Nomenklaturen bezeichnen Schlagwortsysteme, in denen einzelne Terme zur inhaltlichen Erschließung des Wissens vorliegen. Dabei werden Schlagwörter grundsätzlich kontrolliert. D. h. es werden nur Schlagwörter zugelassen, die ausdrücklich erlaubt sind. In Klassifikationen, wie beispielsweise dem System der Internationalen Patentklassifikation, werden Begriffe durch Notationen bezeichnet sowie nach bestimmten Merkmalen zu Klassen zusammengefasst. Klassen können wiederum hierarchisch zueinander angeordnet sein. Im Gegensatz dazu verknüpft ein Thesaurus (vgl. auch Abschnitt 3.2.2.2) Terme zu kleinen begrifflichen Einheiten und setzt diese in Relation zueinander.

Thesauri werden in ihrer Funktion und ihren Merkmalen durch bestimmte Regeln determiniert (Stock und Stock 2008). In der Softwareentwicklung werden Schlagwortsysteme vor allem in Form von Glossaren verwendet, während Thesauri unter anderem in der semantischen Analyse eingesetzt werden. Wissen in digitalen Datenbanken optimal auffindbar zu machen, bildet damit die Basis für Prozesse innerhalb der Softwareentwicklung.

Als übergeordnetes Ziel kann die erfolgreiche Kommunikation zwischen den Beteiligten und schlussendlich auch mit und zwischen Systemen angesehen werden, wozu eine gemeinsame Konzeptualisierung als Grundlage benötigt wird (Dengel et al. 2012; Sure et al. 2002). Hierfür werden unter anderem Ontologien eingesetzt. Der Begriff Ontologie stammt ursprünglich aus der Philosophie und bezeichnet dort die Lehre vom Seienden (Carstensen 2010). Eine häufig zitierte Definition im Bereich der Informatik ist die von Gruber (1993): „An ontology is an explicit specification of a conceptualization“ (S. 199). Ergänzt mit der Auffassung von Uschold und Gruninger (1996), die eine Ontologie als „shared understanding of some domain of interest“ (S. 97) sehen, kommen Dengel et al. (2012) zu einer umfassenden Definition:

> „Eine Ontologie ist eine formale, explizite Spezifikation einer gemeinsamen Konzeptualisierung.“ (S. 65)

Als Komponenten von Ontologien beschreiben Dengel et al. Klassen, Relationen, Regeln und Instanzen. Klassen sind Begriffskategorien, die meistens hierarchisch organisiert sind und damit Vererbungsmechanismen beinhalten. Relationen beschreiben, wie Klassen untereinander zusammenhängen, was wiederum eine Eigenschaft einer Klasse beschreiben kann. Regeln wie beispielsweise für die Suche oder die Überprüfung von Gültigkeiten bestimmen, was in der Domäne als wahr angesehen wird, während Instanzen die realen Elemente repräsentieren (Dengel et al. 2012). Anhand dieser Komponenten wird klar, dass Ontologien die Einsatzgebiete der eben erwähnten Wissensrepräsentationen, wie das Klassifizieren von Inhalten und das inhaltliche Verknüpfen, mit einschließen und erweitern. Daher stellen Ontologien die Repräsentationsform mit der höchsten semantischen Reichhaltigkeit dar und bilden damit die oberste Stufe der „Semantischen Treppe“ von Blumauer und Pellegrini (2006), die in Abbildung 17 dargestellt ist.

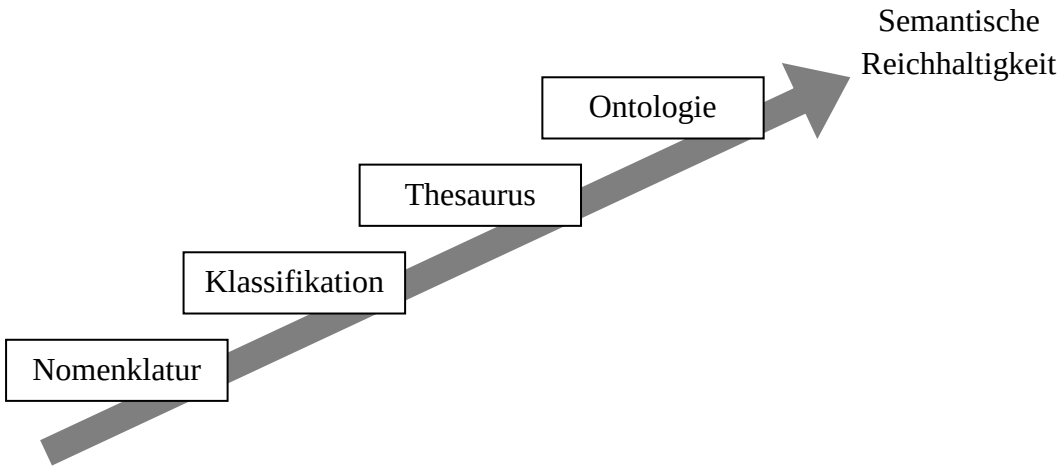


Abbildung 17: Semantische Treppe (nach Blumauer und Pellegrini 2006; Stock und Stock 2008)

Ein Beispiel für eine Ontologie ist WordNet (vgl. 3.2.2.3), welches über die Verknüpfung von Wörtern selbst zusätzlich die Unterscheidung verschiedener Wortbedeutungen erlaubt sowie die Beziehungen semantisch anreichert. Ontologien werden teilweise auch als semantische Modelle bezeichnet (Baclawski et al. 2001).

Blumauer und Pellegrini (2006) gehen auf den Nutzen von Ontologien ein. So können diese nicht nur, wie eben gefordert, dem Austausch zwischen Programmen und der Unterstützung der Kommunikation zwischen den Beteiligten dienen, sondern auch dafür eingesetzt werden, verschiedene Wissensrepräsentationsformen ineinander zu übersetzen oder Theorien abzubilden. Dabei werden alle vorgestellten Arten der Wissensrepräsentation zur formalen Beschreibung der zugrunde liegenden Modelle verwendet, aber nicht für deren Visualisierung (Blumauer und Pellegrini 2006). Für die Visualisierung werden häufig Graphen oder Hierarchien sowohl in 2D als auch in 3D verwendet. Allerdings gibt es keine Form, die für alle Anwendungen angemessen ist, sodass dem Benutzer verschiedene Visualisierungen angeboten werden sollten (Katifori et al. 2007).

Insgesamt stellen Ontologien in der Softwareentwicklung eine der wichtigsten semantischen Technologien dar, da sie sowohl den Austausch zwischen Systemen als auch die Kommunikation der Beteiligten untereinander vereinfachen. Neben allgemeinen Ontologien werden für spezifische Anwendungsbereiche in der Regel Domänen-spezifische Ontologien erstellt und eingesetzt (Dengel et al. 2012). Einen Überblick über die vielfältigen Einsatzmöglichkeiten von Ontologien in der Softwareentwicklung geben Happel und Seedorf (2006). Speziell auf den Bereich der kollaborativen Softwareentwicklung wird in der Arbeit von Happel et al. (2010) eingegangen.

4.1.2 Modelltransformation

Mens und Van Gorp (2006) schlagen eine Klassifikation für Modelltransformationen vor. Dazu müssen Modelle zunächst in einer Modellierungssprache vorliegen, also beispielsweise in Form eines UML-Diagramms (vgl. 2.2.1.2) für Entwurfsmodelle oder in Form einer Programmiersprache für Quellcodemodelle. Die erste Dimension der Klassifikation von Mens und Van Gorp wird durch die Sprache bestimmt, in der das Ausgangs- bzw. das Zielmodell der Transformation sich befinden. Befinden sich beide Modelle in der gleichen Sprache, wird die Transformation als endogen bezeichnet, anderenfalls als exogen. Die zweite Dimension bezieht sich auf den Abstraktionslevel des Ausgangs- und des Zielmodells. Befinden sich beide Modelle auf dem gleichen Abstraktionslevel, so wird von einer horizontalen Transformation gesprochen, bei unterschiedlichem Abstraktionslevel von einer vertikalen Transformation. Die beiden Dimensionen spannen eine Klassifikationsmatrix auf, die in Abbildung 18 aufgezeigt wird.

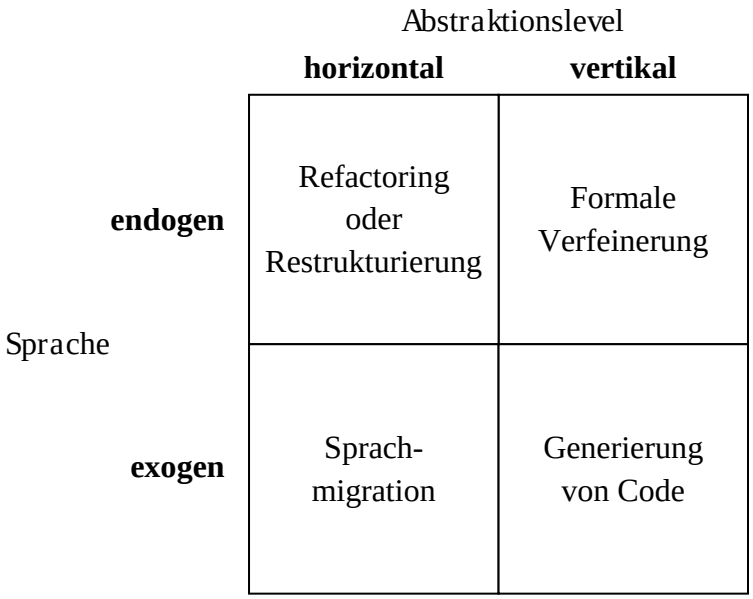


Abbildung 18: Beispiele für die Klassifikation von Transformationen (Mens und Van Gorp 2006)

In den Zellen der Klassifikationsmatrix stehen jeweils Beispiele für die einzelnen Transformationen. Einige Transformationen, wie Refactoring, sind eher simpel, während andere, wie insbesondere die Generierung von Quellcode, komplex und aufwendig sind. Eine weitere Charakteristik der Modelltransformation ist der Grad der Automatisierung, wobei Mens und Van Gorp explizit darauf hinweisen, dass die Transformation von einer natürlichsprachlichen Anforderungssprezifikation in ein Analysemodell eine manuelle Intervention erfordert und nicht vollständig automatisierbar ist (Mens und Van Gorp 2006). Auch Ibrahim und Ahmad (2010) bezeichnen die Automatisierung der Generation eines UML-Klassendiagramms aus natürlichsprachlichen Anforderungssprezifikation als große Herausforderung. In ihrem Lösungsansatz verwenden die Autoren mehrere NLP-Funktionalitäten und eine Domänenspezifische Ontologie. Der essenzielle Schritt ist hierbei die Identifikation von relevanten Konzepten im Text, die für ein bestimmtes Modell benötigt werden. Die zur Verfügung stehenden Verfahren werden im nächsten Unterkapitel (4.2) vertieft betrachtet.

Für Modelltransformationen zwischen verschiedenen UML-Diagrammen oder von UML-Diagrammen und Quellcode liegen dagegen sehr viele automatisierte Ansätze vor. Auch hier ist der grundlegende Schritt die Identifikation aller Elemente des Ausgangsmodells, die auf das Zielmodell abgebildet werden sollen (Atkinson und Kühne 2007). Die Autoren beschreiben verschiedene Ansätze, wobei auf der OMG-Architektur basierende Transformationen den großen Vorteil mit sich bringen, dass die Transformationen selbst wiederverwendet werden können. Dies gelingt über die Festlegung eines standardisierten Metamodells. Ein solches vereinfachtes Metamodell wird in Abbildung 19 gezeigt. Ein Problem des Ansatzes ist, dass UML viele Anpassungen erlaubt. Eine benötigte Information kann auf verschiedene Weise ausgedrückt werden, sodass Transformationen komplexer werden und nur schwerer wiederzuverwenden sind (Atkinson und Kühne 2007).

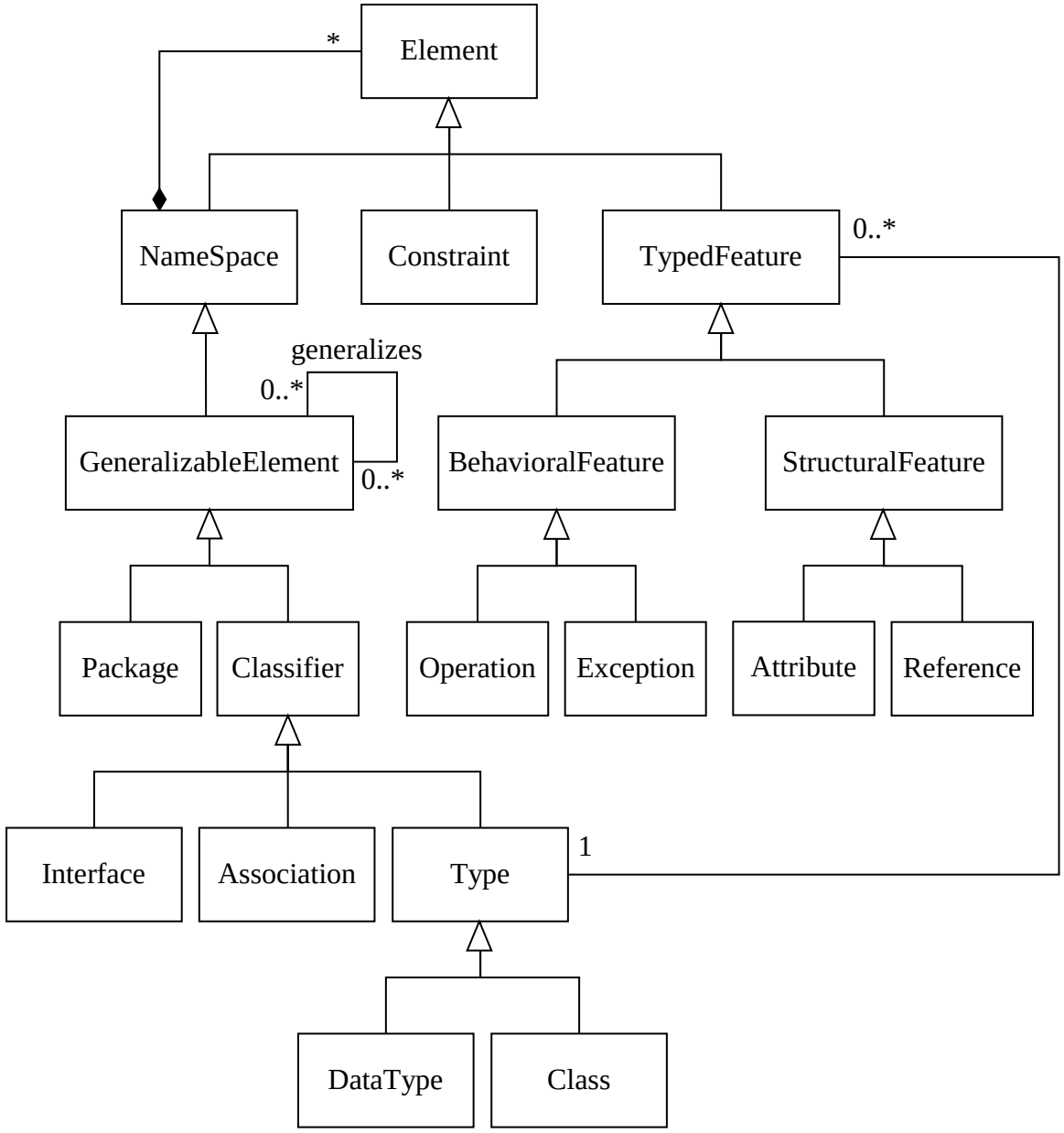


Abbildung 19: Vereinfachtes UML-Metamodell (Atkinson und Kühne 2007)

Als technische Grundlage für Modelltransformationen wird häufig XML (Extensible Markup Language) beziehungsweise das darauf aufbauende Format XMI (XML Metadata Interchange) verwendet (Sendall und Kozaczynski 2003). Für die Transformation stehen zwei Wege zur Verfügung (Mens und Van Gorp 2006): Erstens kann XSLT (Extensible Stylesheet Language Transformation) oder XQuery verwendet werden. Zweitens kann ein Metamodell einbezogen werden, welches konform zu MOF (Meta Object Facility) ist. Der beste Weg hängt vom Ziel der Modelltransforamtion ab. Um zu überprüfen, ob schlussendlich das gewünschte Verhalten erreicht wird, können Validierung und Verifikation eingesetzt werden, welche im folgenden Abschnitt beschrieben werden.

4.1.3 Validierung und Verifikation

In der Softwareentwicklung bezeichnet der Begriff der Validierung im Allgemeinen den Beleg dafür, dass ein Programm den Bedürfnissen der Beteiligten entspricht; Verifikation be-

zeichnet den Beleg dafür, dass ein Programm den Spezifikationen entspricht (Sommerville 2012). Häufig werden die beiden Terme nicht in dieser Differenzierungstiefe betrachtet, sodass verschiedene Autoren den einen oder den anderen Begriff verwenden, ohne ihn vorher abgegrenzt zu haben, wobei häufig die Prüfung des Programms mittels Testens referenziert wird. Die Begriffe können auch automatisierte Prozesse umfassen und, bezogen auf einzelne Modelle, wie UML-Diagramme oder die im ersten Abschnitt dieses Unterkapitels beschriebenen Ontologien verwendet werden. Auch Abläufe auf Programmebene können Gegenstand der Validierung sein (Cook und Wolf 1999; Holzmann 1997). Die automatisierte Verifikation wird häufig mit dem englischen Term Model Checkers bezeichnet. Die dafür implementierten Werkzeuge erlauben es, alle möglichen Kombinationen aus Ein- und Ausgaben eines Programms zu bestimmen und auf diese Weise mit höherer Wahrscheinlichkeit Fehler zu finden als mit manuellem Testen (Miller et al. 2010).

Validierung und Verifikation werden unter anderem im Zusammenhang mit den im vorangegangenen Abschnitt beschriebenen Modelltransformationen benötigt. Zum einen kann vor der Transformation überprüft werden, ob das Ausgangsmodell valide ist, da anderenfalls das Ergebnis der Transformation undefiniert wäre (Volter 2006). Ebenfalls ist es sinnvoll, das Zielmodell zu verifizieren. Umgekehrt schlagen Van Amstel et al. (2011) vor, Modelltransformationen als Hilfsmittel für die Verifikation zu verwenden. Generell werden für die Überprüfung der Korrektheit eines Modells häufig endliche Automaten verwendet (Holzmann 1997; Saxena und Kumar 2012; Visser und Havelund 2000). Die Anzahl der zu überprüfenden Zustände kann allerdings sehr groß werden, sodass andere Techniken wie Vereinfachungen und Abstraktionen verwendet werden (Van Amstel et al. 2011).

Für die Anwendung von Verifikationsmethoden liegen verschiedene Standards vor, insbesondere für sicherheitskritische Systeme wie beispielsweise in der Luftfahrt oder in der Medizintechnik (Almeida et al. 2011). Almeida et al. heben hervor, dass in der Softwareentwicklung generell Standards zunehmend an Bedeutung gewinnen. Dazu wurde von mehreren Ländern, darunter auch Deutschland und die USA, eine Intitiative zur Regulierung vorangetrieben, die im ISO-Standard 15408 festgeschrieben wurde. Dieser als Common Criteria bezeichnete Standard regelt die Einhaltung der Anforderungensspezifikationen und dient als Grundlage für die Evaluation eines Softwareprodukts oder -services. Allein das Vorhandensein eines solchen Rahmenwerkes ist ein Vorteil für Softwareentwickler und Kunden. Allerdings ist der Einsatz formaler Methoden auch immer mit Kosten verbunden, wobei Projekte gleichzeitig Budgetrestriktionen unterliegen. Almeida et al. (2011) kommen zu dem Schluss, dass es unrealistisch ist, die absolute Korrektheit eines Programms zu erwarten.

4.2 Umsetzung

Es werden nun Methoden zur Umsetzung spezifischer Einsatzziele besprochen. Für die Wissensrepräsentation werden getrennt die Unterstützung für die Erstellung von Anforderungsspezifikationen und die Erfassung von Nachverfolgbarkeitsinformationen betrachtet. Für die Modelltransformation wird der Einsatz semantischer Technologien für die Analyse von Anforderungsspezifikationen vorgestellt, die eine automatische Extraktion von UML-Diagrammen zum Ziel hat. Des Weiteren wird kurz auf Ansätze zur Validierung und Verifikation sowie abschließend auf bereits vorhandene Werkzeuge eingegangen.

4.2.1 Semantische Technologien zur Unterstützung von Anforderungsspezifikationen

Die Erfassung von schriftlichen Anforderungsspezifikationen kann auf vielfältige Weise durch semantische Technologien unterstützt werden. Zunächst werden Wikis und semantische Wikis als semantische Technologien vorgestellt; es wird jeweils beschrieben, wie diese während der Erstellung von Anforderungsspezifikationen eingesetzt werden können. Darüber hinaus wird auf die Unterstützung durch Ontologien für diese Phase der Softwareentwicklung eingegangen.

4.2.1.1 Unterstützung durch Wikis

Der Begriff Wiki beschreibt sowohl das Konzept als auch die Implementierung. Wikis wurden erstmals 1995 von Ward Cunningham beschrieben (Grace 2009) und können definiert werden als eine „freely expandable collection of interlinked webpages, a hypertext system for modifying and storing information – a database where each page is easily editable by users" (Leuf und Cunningham 2001, S. 4). Dabei wird Wiki als ein essenzielles Konzept, nicht als konkrete Implementierung verstanden.

Da jeder Nutzer – oder zumindest jeder Nutzer, der über die entsprechenden Rechte verfügt – die Seiten verändern und somit sein eigenes Wissen teilen kann, bieten sich Wikis als Grundlage für die verteilte Zusammenarbeit an. Durch das Bereitstellen einer einfachen Syntax sowie nachvollziehbarer Konventionen für die Navigation besteht auch für Nutzer mit geringen Vorkenntnissen nur eine niedrige Einstiegshürde. Das Auffinden von Wissen wird durch eine Suchfunktion erleichtert. Die Einbindung anderer Kontexte kann durch Verlinkung mit anderen Webseiten realisiert werden (Grace 2009). Generell verfügen Wikis über folgende charakteristische Funktionen (Ebersbach et al. 2007; Leuf und Cunningham 2001):

- einfache Editierbarkeit der Wiki-Seiten
- Browser-basierter Zugriff
- Verlinkung der Seiten untereinander
- Versionshistorie & Übersicht über kürzlich vorgenommene Änderungen
- Unterstützung der Kooperation

All diese Funktionalitäten sind grundsätzlich auch für Softwareentwicklung wünschenswert. Da an der Entwicklung von Softwareprojekten eine hohe Anzahl an Beteiligten mitwirkt, ist besonders die Unterstützung der Kooperation ein wichtiges Argument für den Einsatz von Wikis (Happel et al. 2008). Speziell für den Einsatz von Wikis im Requirements Engineering (RE) legen Lai et al. (2012) einen Übersichtsartikel vor, der sinnvolle Erweiterungen für RE-spezifische Wikis zusammenfassend darstellt. Unter anderem werden folgende Mechanismen genannt: die Möglichkeit, Benachrichtigungen zu verschicken, der Export beispielsweise in Dokumente im XML-Format oder der Microsoft Office Suite, das Bereitstellen von fixen oder anpassbaren Vorlagen (engl. templates) sowie die Einbindung von Multimedia-Materialien. Dabei sind nach Meinung der Autoren weitere Entwicklungen in diesem Bereich wünschenswert, wie beispielsweise die Verbesserung und Erweiterung der Vorlagen, die Ergänzung von Mechanismen zum Vorschlag und zur Aushandlung von Anforderungen sowie die Entwicklung von Standards für die Erhebung, Analyse und Validierung. Obwohl also Forschungsbe-

darf besteht, können Wikis bereits jetzt einen sinnvollen Beitrag für das Requirements Engineering liefern (Lai et al. 2012).

Auf der anderen Seite können Wikis generell und damit ebenfalls im Zusammenhang mit dem Einsatz in der Softwareentwicklung mit Nachteilen verbunden sein. Grace (2009) hebt vier Bereiche hervor: Sicherheit, Datenmigration, Training und die Kategorisierung von Informationen. Wie eingangs erwähnt, können Benutzer von Wikis die Webseiten verändern. Auch wenn diese Änderungen in der Regel über einen Mechanismus zurückverfolgt und ältere Versionen wiederhergestellt werden können, so kann dennoch eine Überwachung der Inhalte notwendig sein. Besonders in öffentlich zugänglichen Wikis kann es sonst zu sogenannten Edit Wars kommen, in denen Benutzer ihre Konflikte in Form von sehr häufigen Änderungen austragen (Decker et al. 2007). Auch bei einem eingeschränkten Nutzerkreis ist es häufig sinnvoll – gerade in Hinblick auf die Sicherheit von Daten in Organisationen –, entsprechende Vorkehrungen zu treffen. Der zweite Kritikpunkt von Grace betrifft die Datenmigration. Die Autorin gibt zu bedenken, dass durch Updates, Fehlerkorrekturen und neue Versionen die Stabilität und die Integrität der Plattform gefährdet sein können. Darüber hinaus ist die Bedienung eines Wikis nicht für jeden Benutzer intuitiv, sodass eventuell doch ein Bedarf an Training entsteht (Raman 2006). Der vierte Kritikpunkt betrifft die Kategorisierung von Informationen, die Grace als subjektiv kritisiert; die Flexibilität und die Freiheiten bei der Gestaltung von Wikis können zu Schwierigkeiten beim Auffingen von Informationen für andere Nutzer führen.

4.2.1.2 Unterstützung durch semantische Wikis

Semantische Wikis erweitern traditionelle Wikis derartig, dass sie semantische Annotationen der Webseiten sowie deren Links untereinander ermöglichen, wobei diese Annotationen in der Regel einer Ontologie entsprechen (Schaffert et al. 2008). Nordheimer et al. (2012) präsentieren ein passendes Beispiel: Der Link zwischen „Stakeholder“ und „Requirement“ kann mit der Beziehung „responsible for“ annotiert werden, was das Stellen semantischer Suchanfragen wie „who is responsible for requirement X“ oder „which requirements are managed by person Y“ ermöglicht. Der Hauptunterschied zwischen traditionellen und semantischen Wikis liegt damit in der Möglichkeit, in semantischen Wikis neben der natürlichen Sprache auch eine formalisierte Sprache benutzen zu können (Maalej et al. 2008). Davis (2006) führt als Vorteile semantischer Wikis an:

- **Konzeptbasierte Suche**. Diese ermöglicht, wie soeben im Beispiel gezeigt, eine neue Art von Anfragen, die in ihrer Mächtigkeit deutlich über die bisherigen Möglichkeiten der Sprach-basierten Suche hinausgehen.
- **Beantwortung von Fragen**. Semantische Wikis können als Wissensbasis für die Beantwortung der Fragen nach dem Was, Warum oder Was wenn und sind überdies mit Suchmaschinen integrierbar.
- **Reichhaltig strukturierte Navigation**. Informationen können unter anderem aus mehreren Perspektiven, auf verschiedenen Abstraktionsleveln und auch anhand unterschiedlicher Beziehungen betrachtet werden.
- **Einfache und flexible Visualisierung**. Die Inhaltstruktur ist nicht nur auf verschiedene Weisen darstellbar, sondern auch direkt editierbar.

Diese Funktionalitäten sind mit traditionellen Wikis nicht zu erreichen. Darüber hinaus stellen Nordheimer et al. (2012) heraus, dass semantische Wikis als leichtgewichtiger Ansatz betrachtet werden können, da sie einfach zu verwenden sind und dem Benutzer die Verwendung der semantischen Annotationen nicht aufzwingen, sondern benutzerfreundliche Elemente für die Bedienung bereitstellen. Solche Elemente sind beispielsweise Buttons, Checkboxen oder auch Dropdown-Menüs für die Verlinkung der Wiki-Seiten. Des Weiteren können semantische Suchanfragen vordefiniert, gespeichert und wiederverwendet werden, was für den Benutzer eine Aufwands- und Zeitersparnis bedeutet.

Diese Stärken können während der Anforderungserhebung eingesetzt werden, beispielsweise um Dokumentation von Funktionen oder den Prozess der Fehlerbehebung zu strukturieren (Panagiotou und Mentzas 2009). Darüber hinaus kann die formale Semantik auch von Maschinen verstanden und verarbeitet werden, sodass Benutzer besser durch das System unterstützt werden können. Beispielsweise kann auf diese Weise Wissen, wie in Form von Anforderungsspezifikationen, bearbeitet und sogar neu erstellt werden (Maalej et al. 2008).

Geisser et al. (2007a) beschreiben, wie die Anforderungserhebung abläuft und an welchen Stellen Wikis sinnvoll eingesetzt werden können. Zunächst wird idealerweise ein initiales Treffen durchgeführt, bei dem alle beziehungsweise möglichst viele der Beteiligten anwesend sind. Das Ziel ist es, ein einheitliches Bild des geplanten Systems zu erlangen. Anschließend findet eine asynchrone Anforderungserhebung und -analyse statt. Dabei kommen dem Interessenvertreter auf Kundenseite Aufgaben zu wie das Übermitteln und Kommentieren von Anforderungsspezifikationen, die Identifikation und Definition von erklärungsbedürftigen Fachtermini sowie die Erweiterung und Präzisierung der Vision. Die Implementierungsverantwortlichen kommentieren die Anforderungsspezifikationen und bewerten deren Machbarkeit. Ein Moderator überwacht den Aushandlungsprozess, indem er oder sie die zu präzisierenden Elemente identifiziert und die Anforderungsspezifikationen konsolidiert und kategorisiert. Wikis können in der Art eingesetzt werden, dass für jede Anforderungsspezifikation eine eigene Wiki-Seite erstellt wird. Hier ist insbesondere die integrierte Versionsverwaltung von Bedeutung, da sie das Nachverfolgen von Änderungen erlaubt. Für die Fachtermini können innerhalb des Wikis Glossare erstellt und gepflegt werden (Geisser et al. 2007a).

Statt Glossare zu verwenden, bietet es sich an, eine höhere Form der semantischen Reichhaltigkeit zu wählen, gewissermaßen die Stufen der semantischen Treppe (vgl. Abbildung 17) heraufzusteigen und eine Ontologie zur Wissensrepräsentation einzusetzen. Einige der semantischen Wikis bieten dazu einen Editor zur Bearbeitung von Ontologien. Teilweise werden auch Import- und Exportfunktionalitäten für Ontologien zur Verfügung gestellt (Maalej et al. 2008). Für die Implementierung der semantischen Strukturen werden häufig die Web Ontology Language (OWL) oder das Resource Description Framework (RDF) eingesetzt (Lohmann et al. 2007). Diese werden auch als Grundlagen für das semantische Web gesehen, welches als Weiterentwicklung des Internets beschrieben wird und durch Standardisierung eine verbesserte Bewältigung komplexer Aufgaben erreichen möchte (Blumauer und Pellegrini 2006). Damit vereinen semantische Wikis die Charakteristiken der Technologien des semantischen Webs mit denen des Wissensmanagements und Social Software, wie z. B. Blogs (Hagen et al. 2007). Abbildung 20 zeigt, wie Wiki-basiertes Requirements Engineering sich in diesem Kontext einordnen lässt.

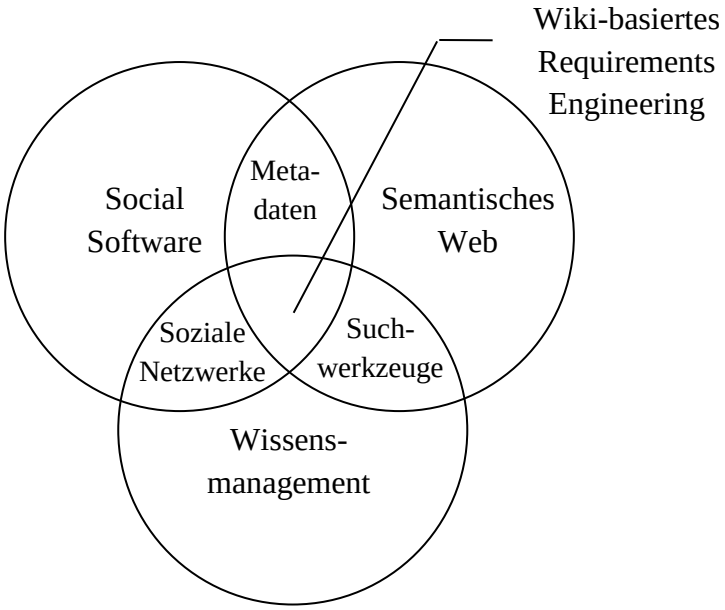


Abbildung 20: Wiki-basiertes Requirements Engineering (in Anlehnung an Hagen et al. 2007)

Zusammengefasst bieten semantische Wikis eine Vielzahl an Funktionalitäten, die während der Erstellung von Anforderungsspezifikationen hilfreich sind. Ihre Stärken heben sie von konventionellen Content Management Systemen (CMS) ab, insbesondere in Hinblick auf die Unterstützung der Kollaboration der Benutzer untereinander. Kritisch zu bemerken ist allerdings, dass der Einsatz semantischer Wikis gewisse Voraussetzungen an das Vorwissen der Benutzer stellt, beziehungsweise Zeit für die Einarbeitung benötigt wird, um die Funktionalitäten – insbesondere die semantischen – in vollem Umfang verwenden zu können.

4.2.1.3 Unterstützung durch Ontologien

In den frühen Phasen der Softwareentwicklung spielt laut Kaiya und Saeki (2006) das Domänen-spezifische Wissen eine weitaus größere Rolle als das Wissen darüber, wie die Anforderungsspezifikationen im Allgemeinen erstellt werden. Zur Unterstützung des Prozesses der Anforderungserhebung schlagen die Autoren daher den Einsatz von Ontologien vor. Genauer gesagt, soll eine Domänen-spezifische Ontologie zusammen mit bestimmten Inferenzregeln dazu verwendet werden, die Qualität der Anforderungsspezifikationen sicherzustellen. Es werden vier Qualitätsmaße betrachtet: die Korrektheit, die Vollständigkeit, die Konsistenz sowie die Eindeutigkeit. Zuerst werden dazu die Anforderungsspezifikationen satzweise in semantische Elemente zerlegt und den Bestandteilen der Ontologie zugeordnet. Dabei werden sowohl Konzepte als auch Beziehungen berücksichtigt. Auf diese Weise können später Abhängigkeiten der Anforderungsspezifikationen und damit deren Vollständigkeit überprüft werden (Kaiya und Saeki 2005, 2006). Es ist zu beachten, dass für diesen Ansatz bereits eine Domänen-spezifische Ontologie vorliegen muss.

Da eine dem Problem angemessene Ontologie nicht immer vorliegt, kann es sinnvoll sein, eine Ontologie aus den Anforderungsspezifikationen selbst zu extrahieren. Ein solcher Ansatz wird in der Arbeit von Kof (2004) beschrieben. Er geht dazu in mehreren Schritten vor: Zuerst werden Terme aus den Anforderungsspezifikationen extrahiert, dann werden diese Terme gruppiert (engl. clustering), und abschließend werden die Beziehungen zwischen den Termen ergänzt. Dazu werden NLP-Techniken wie POS Tagging (vgl. 3.2.2.1) verwendet, wobei in

mehreren Schritten manuelle Eingriffe notwendig sind. Dieses Vorgehen erlaubt es, Inkonsistenzen innerhalb der Anforderungsspezifikationen aufzudecken und zu korrigieren. Der Autor führt ebenfalls eine Fallstudie durch, die belegt, dass das Verfahren auch für größere Anforderungsdokumente skaliert. Aus der Fallstudie werden mehrere Regeln abgeleitet, die unter anderem die naheliegende Forderung beinhalten, eine konsistente Terminologie zu verwenden.

Eine weitere auf Ontologien basierende Vorgehensweise legen Assawamekin et al. (2008, 2009a) vor. Die Autoren betonen, dass die verschiedenen Beteiligten eine unterschiedliche Perspektive hinsichtlich der Anforderungen haben und deshalb bei der kollektiven Erstellung von Anforderungsspezifikationen Diskrepanzen in der Terminologie resultieren. Das Ziel des Ansatzes ist es deshalb, durch den Abgleich von Ontologien (engl. ontology matching) ein gemeinsames Verständnis sowie eine gemeinsame Darstellungsform für die unterschiedlichen Anforderungsspezifikationen herzustellen. Der Abgleich findet zwischen zwei Ontologien statt und resultiert in einem Ergebnis für die Übereinstimmung, welches in eine der folgenden fünf Kategorien klassifiziert wird: Äquivalenz (=, engl. equivalence), höhere Generalität ($\supseteq$, engl. more general), geringere Generalität ($\subseteq$, engl. less general), keine Übereinstimmung ($\perp$, engl. mismatch) und Überlappung ($\cap$, engl. overlapping). Falls keine dieser Kategorien zutrifft, wird die Beziehung idk (engl. I don't know) zurückgegeben. Wenn eine Übereinstimmung gefunden wird, d.$\perp$ oder idk, wird automatisch ein Nachverfolgbarkeits-Link generiert. Die Beziehungen $\supseteq$ und $\subseteq$ werden dabei zu einer Nachverfolgbarkeitsbeziehung zusammengefasst. Einen Überblick gibt Tabelle 3.

Lexikalische Beziehung	Semantische Beziehung	Nachverfolgbarkeitsbeziehung
Synonym	a = b	overlapTotally
Hypernym oder Holonym	a $\subseteq$ b	overlapInclusively
Hyponym oder Meronym	a $\supseteq$ b	
Antonym	a $\perp$ b	noOverlap
/	a $\cap$ b	overlapPartially

Tabelle 3: Zuordnung lexikalischer und semantischer Beziehungen zu Nachverfolgbarkeitsbeziehungen (nach Assawamekin et al. 2008)

Zur Bestimmung der lexikalischen Beziehungen werden sowohl WordNet als externe Ressource als auch verschiedene Techniken zum Abgleich zwischen Zeichenketten (engl. string matching) eingesetzt. In einem nächsten Schritt werden die Konzepte mithilfe von Regeln aussagenlogisch untersucht, damit verschiedene Typen von Nachverfolgbarkeitsbeziehungen zugeordnet werden können. Die Autoren beschreiben ihren Ansatz als voll-automatisch, wobei sie offen lassen, wie verfahren wird, wenn Beziehungen inkorrekt zugeordnet werden. Eine große Stärke des Ansatzes liegt in seinem multiperspektivischen Ausgangspunkt, der explizit von einer Kollaboration verschiedener Benutzer während der Erstellung der Anforderungsspezifikationen ausgeht. Dies erscheint realistischer als die weiter oben erwähnte Forderung nach einer konsistenten Terminologie. Die Autoren erweitern ihren Ansatz, sodass er über Anforderungsspezifikationen hinaus auch weitere Artefakte einschließt. Damit ist die Brücke zur Unterstützung der Nachverfolgbarkeit durch semantische Technologien geschlagen. Der erweiterte Ansatz wird, neben anderen, im folgenden Abschnitt vertieft betrachtet.

4.2.2 Semantische Technologien zur Unterstützung der Nachverfolgbarkeit

Die Erweiterung des Einsatzes semantischer Technologien von der Verwendung im Zusammenhang mit Anforderungsspezifikationen auf die Verwendung über mehrere Artefakte und über den gesamten Softwarelebenszyklus hinweg mag naheliegend erscheinen. Trotzdem liegen bisher nur wenige Ansätze vor. Teilweise wird nur die Idee beschrieben, wie beispielsweise von Noll und Ribeiro (2007a, 2007b). Allerdings wird dabei nicht auf die Umsetzung eingegangen. Daher werden in diesem Unterabschnitt nur Ansätze vorgestellt, die einen für die Umsetzung relevanten Beitrag leisten.

4.2.2.1 Unterstützung durch Wikis

Hildenbrand et al. (2008) stellen einen sehr weit entwickelten Ansatz vor, der unter anderem für die Annotierung von Artefakten auf einer Wiki-Engine aufsetzt. Neben der Unterstützung der Nachverfolgbarkeitsbeziehungen wird ebenfalls die Verwaltung von Entscheidungen und deren Gründen unterstützt. Entscheidungen, die während des Softwareentwicklungsprozesses getroffen werden, sind Werte-bezogen und reflektieren somit die unterschiedliche Bedeutung, die verschiedene Softwareartefakte für den Kunden haben. Gleichzeitig ist es möglich, Änderungswünsche im System zu erfassen und mit den anderen Konzepten zu verknüpfen. Darüber hinaus legt der Ansatz großen Wert auf die Unterstützung der Zusammenarbeit zwischen den Beteiligten und integriert deshalb eine Plattform zur Kollaboration.

Die Umsetzung wird ausführlich in der Arbeit von Hildenbrand (2008) dargestellt. Einer der Grundbausteine des Ansatzes ist das dediziert entwickelte Informationsmodell. Es basiert auf dem bereits vorgestellten Metamodell von Ramesh und Jarke, welches Objects, Stakeholders und Sources beinhaltet (vgl. 2.2.3). Diese Metaklassen werden im konzeptuellen Informationsmodell verwendet, welches als Objects unter anderem Change und Rationale beinhaltet. Change dient der Abbildung der eben erwähnten Änderungswünsche, während Rationale Entscheidungen repräsentiert. Alle Aktivitäten (Activity) und Änderungswünsche sind jeweils mit der ihnen zugrunde liegenden Entscheidung verknüpft. Artefakte – abgebildet durch die Klasse Artifact – beziehen sich auf Anforderungsspezifikationen, Entwurfsmodelle, Quellcode und Testfälle. Für Artefakte und Aktivitäten können sowohl der jeweilige Status als auch der zugehörige User erfasst werden. Abbildung 21 stellt das konzeptuelle Informationsmodell von Hildenbrand (2008) grafisch dar.

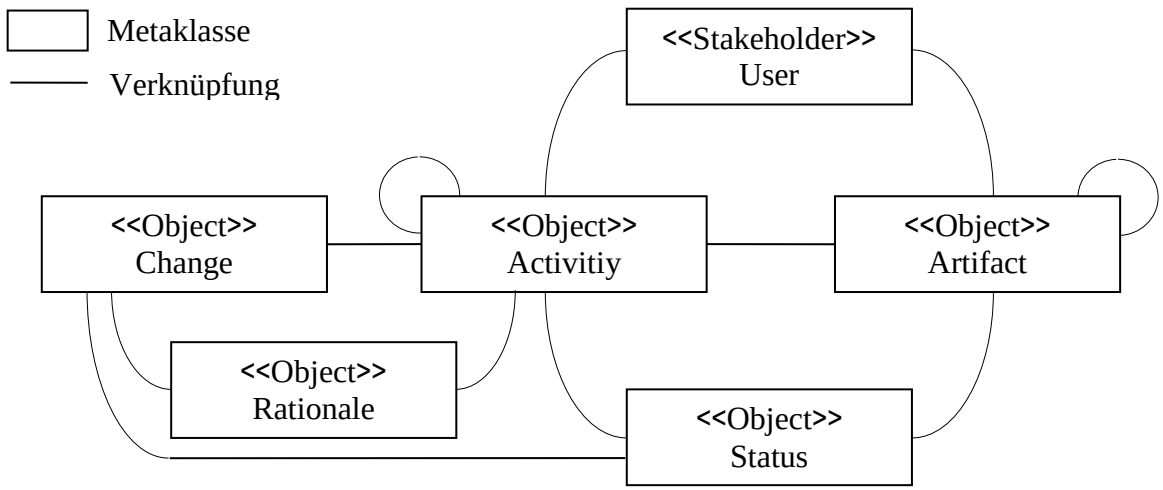


Abbildung 21: Konzeptuelles Informationsmodell (nach Hildenbrand 2008)

Neben der Kollaborationsunterstützung als einem zentralen Vorteil des Ansatzes sind ebenfalls die Konzepte zur Visualisierung hervorzuheben. Es handelt sich dabei um Graph-basierte Visualisierungen, die die Nachverfolgbarkeitsbeziehungen in Form eines Netzwerkes abbilden. Diese Graphen sind vielfältig in ihrer Komplexität anpassbar, so können vorgefertigte Sichten bereitgestellt sowie Filter definiert werden. Falls gewünscht, können beispielsweise auch die Bedeutungen von bestimmten Artefakten visualisiert werden, indem eine spezielle Sicht die Knoten im Graphen in verschiedenen Größen darstellt. Da, wie beschrieben, auch die Benutzer und deren Aktivitäten im System erfasst werden, können unter anderem ebenfalls die Strukturen der Zusammenarbeit visualisiert werden. Das Ziel dieser umfassenden Visualisierungen ist, dem Benutzer eine Grundlage zur Analyse der relevanten Objekte zu geben. Beispielsweise können anhand einer Visualisierung fehlende oder überflüssige Nachverfolgbarkeitsbeziehungen identifiziert werden. Da auch Funktionalitäten zur Modifikation der Graphen bereitstehen, können die Informationen direkt bearbeitet und korrigiert werden. Ein Beispiel für einen Graphen zur Visualisierung findet sich in Abbildung 22.

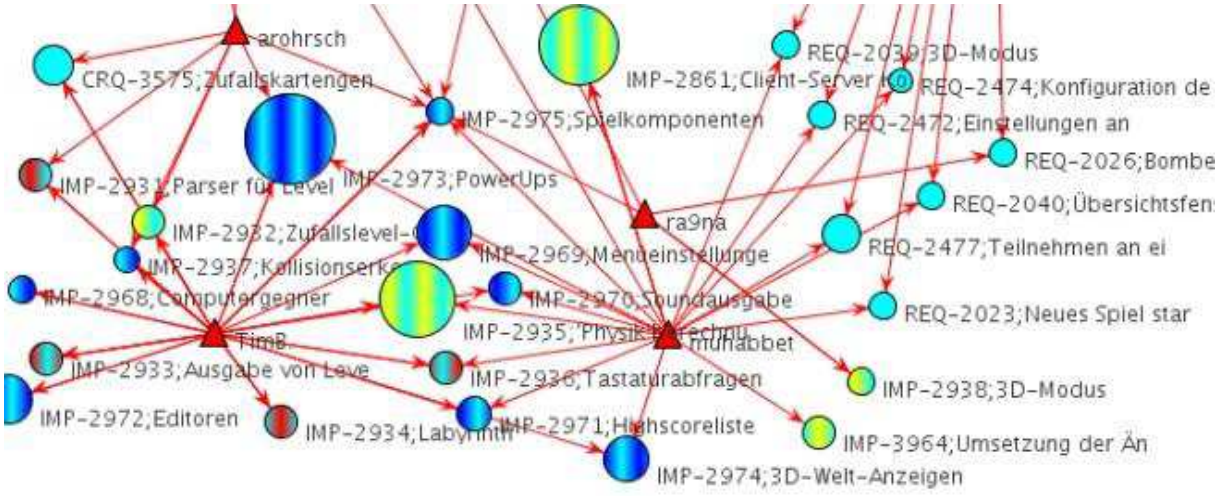


Abbildung 22: Visualisierung von Nachverfolgbarkeitsinformationen (Hildenbrand et al. 2009)

Insgesamt betrachtet, kann der Ansatz nicht nur während der Erstellung von Nachverfolgbarkeitsbeziehungen unterstützend eingesetzt werden, sondern ist auch für die weiteren Phasen der Softwareentwicklung bis hin zur Wartung hilfreich. Allerdings werden neben den Wiki-bezogenen Mechanismen keine weiteren semantischen Technologien eingesetzt. Dies wäre – wie der Autor selbst anspricht – beispielsweise für automatisierte Vorschläge denkbar. Auch eine semantische Analyse der Artefakte oder der Nachverfolgbarkeitsbeziehungen wäre eine sinnvolle Erweiterung.

4.2.2.2 Unterstützung durch semantische Wikis

Nordheimer et al. (2012) setzen in ihrer aktuellen Forschungsarbeit ein semantisches Wiki für die Erstellung von Nachverfolgbarkeitsbeziehungen ein, wobei sowohl Anforderungen als auch Software- und Architekturkomponenten, Geschäftsprozesse und Prozessschritte einbezogen werden. Der Collaborative Lightweight Extension for Software Engineering (CLEoS) genannte Ansatz unterstützt die Verwendung durch mehrere Benutzer gleichzeitig wie auch der im letzten Abschnitt beschriebene Ansatz von Hildenbrand (2008). Im Gegensatz zu diesem, der sich auf den Einsatz in groß angelegten Entwicklungsprojekten bezieht, ist es das

Ziel von CLEoS, auch kleine und mittlere Unternehmen in der Softwareentwicklung zu unterstützen. Damit wird der Möglichkeit eines leichtgewichtigen Einsatzes eine hohe Bedeutung eingeräumt. Das heißt, dass auch weniger erfahrene Anwender sich schnell in die Benutzerschnittstelle einarbeiten können und wählen können, welche Annotationen definiert und implementiert werden sollen, ohne dass dies zwangsweise immer geschehen muss. Weiterhin können vordefinierte semantische Suchanfragen gespeichert und wiederverwendet werden. Für die Umsetzung werden Vorlagen verwendet, sodass auch ohne Erlernen der Wiki-Syntax Geschäftsprozesse oder Anforderungen beschrieben werden können. Auch für die Beschreibung der Nachverfolgbarkeitsbeziehungen stehen Formulare bereit. Für jeden Geschäftsprozess, jede Softwarekomponente etc. wird jeweils eine Wiki-Seite angelegt, wobei auch die Beziehungen der einzelnen Konzepte untereinander auf Eigenschaften im Wiki abgebildet werden. Das semantische Datenmodell von CLEoS ist in Abbildung 23 dargestellt.

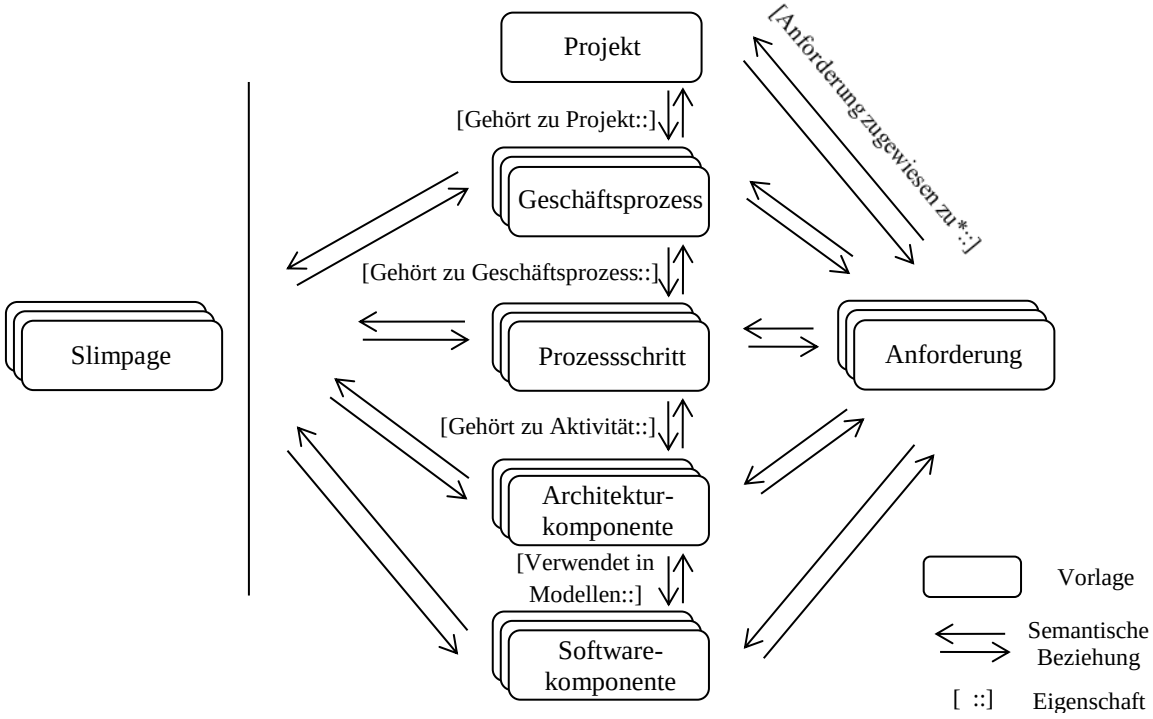


Abbildung 23: Semantisches Datenmodell von CLEoS (nach Nordheimer et al. 2012)

Eine wichtige Grundlage, auf der die Umsetzung beruht, ist dabei der Ansatz von Thum et al. (2009), genannt SLIM (Synchronous Lightweight Modeling). Dieser erlaubt die synchrone Zusammenarbeit verschiedener Benutzer an einem UML-Modell direkt im Web-Browser. Dies ist ohne die Installation weiterer Software möglich und kann so ebenfalls als leichtgewichtig bezeichnet werden. Ein zusätzlicher Vorteil ist die zentrale Überwachung der Änderungen, die die Kontrolle der Konsistenz entscheidend vereinfacht. Der effiziente Umgang mit gleichzeitigen Änderungsversuchen durch mehrere Benutzer wird durch einen Sperr-Mechanismus auf Element-Ebene gelöst. Die Verwendung von SLIM ist ein Alleinstellungsmerkmal des CLEoS-Ansatzes und damit auch, neben der Berücksichtigung von Geschäftsprozessen, ein entscheidender Vorteil. Insgesamt hat der Einsatz semantischer Wikis für die Unterstützung von Nachverfolgbarkeitsbeziehungen großes Potenzial, den Softwareentwicklungsprozess entscheidend zu verbessern.

4.2.2.3 Unterstützung durch Ontologien

Bereits in Abschnitt 4.2.1.3 wurde der Ansatz von Assawamekin et al. (2008, 2009a) für die Unterstützung der Erstellung von Anforderungsspezifikationen durch Ontologien eingeführt. Dieser Ansatz wurde durch die Forschergruppe für weitere Artefakte wie Klassen- und Anwendungsfalldiagramme erweitert (Assawamekin et al. 2009a; Assawamekin 2010). Erneut wird hier die automatische Generierung der Nachverfolgbarkeitsbeziehungen als ein Hauptziel angesehen. Dazu werden mehrere Module verwendet, die, unter anderem mithilfe von NLP-Technologien, Elemente aus den Anforderungsspezifikationen und den Diagrammen extrahieren. Des Weiteren befasst sich ein Modul mit der Klassifikation der Anforderungen aufbauend auf einer Basisontologie, welche wiederum dazu dient, eine Anforderungsontologie für jeden Beteiligten zu erstellen. Die Basisontologie ist vorgegeben und wird jeweils für die einzelnen Diagrammtypen erweitert. Unter anderem wird auch die Erweiterung um Entity Relationship-Diagramme beschrieben. Wie auch schon beim ursprünglichen Ansatz, wird anschließend ein Abgleich zwischen den verschiedenen Ontologien durchgeführt, der in den oben gezeigten Nachverfolgbarkeitsbeziehungen resultiert (siehe Tabelle 3). Abbildung 24 zeigt ein Beispiel für die eben beschriebene Vorgehensweise der Generierung von Nachverfolgbarkeitsbeziehungen aus einer schriftlichen Anforderungsspezifikation.

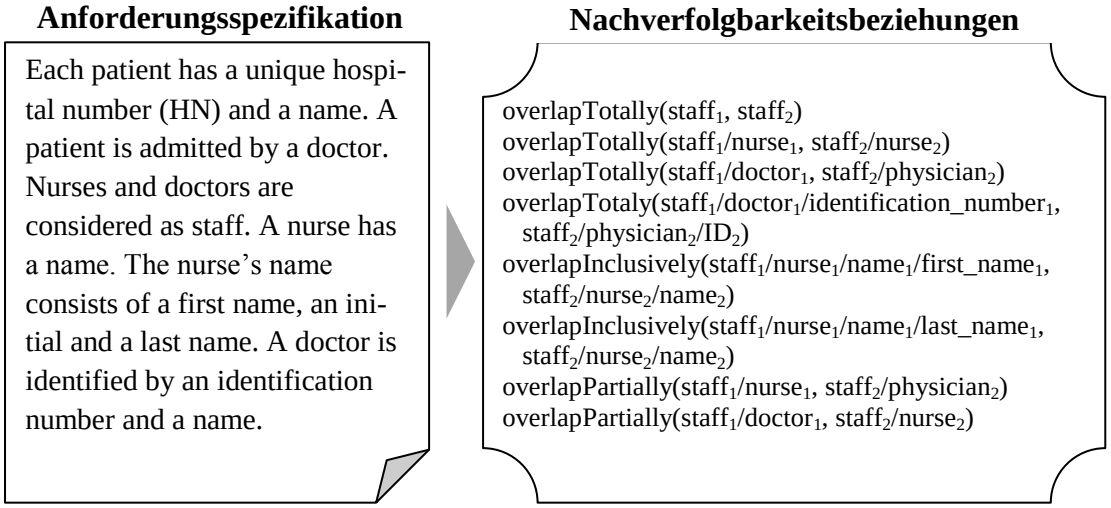


Abbildung 24: Beispiel für die Generierung von Nachverfolgbarkeitsbeziehungen aus einer textuellen Anforderungsspezifikation (nach Assawamekin et al. 2009a)

Der Ansatz der Forschergruppe um Assawamekin erscheint vielversprechend, wurde aber vermutlich technisch nicht komplett umgesetzt; zumindest stimmen die gezeigten Beispiele für die unterschiedlichen Perspektiven nicht miteinander überein. Des Weiteren wurde die Entwicklung nach 2010 nicht weitergeführt.

Eine andere Herangehensweise, wie Ontologien die Nachverfolgbarkeit unterstützen können, zeigt die Arbeit von Seedorf (2010), der eine Ontologie für die Entwicklung komponentenbasierter Anwendungssysteme konzipiert. Sie umfasst sowohl das Domänenkonzept als auch Software- und Architekturkomponenten sowie Geschäftsprozesse. Damit gelingt die Integration von fachlichen und technischen Aspekten, die bis dahin nur unzureichend erforscht wurde. Mithilfe dieses Ansatzes können sich die Beteiligten somit vorab auf eine Terminologie festlegen und diese über den gesamten Entwicklungsprozess verwenden. Bei Bedarf können auch weitere Aspekte ergänzt werden. Des Weiteren geht auch dieser Ansatz von verschiede-

nen Beteiligten aus. Die Ontologie enthält drei dementsprechende Sichten (engl. views): die Architektursicht, die Geschäftssicht und die Anwendungsfallsicht. Dank der Architektursicht können mit den Softwarekomponenten die kleineren Einheiten eines Softwaresystems eingeordnet werden. In der Geschäftssicht werden die Unternehmensmodelle mit den Softwarekomponenten verbunden Die Anwendungsfallsicht erlaubt eine Verbindung zwischen den funktionalen Anforderungen und den Softwarekomponenten der Lösungsarchitektur. Die drei Sichten werden über die Spezifikationselemente der Softwarekomponenten verbunden, diese werden in Abbildung 25 gezeigt.

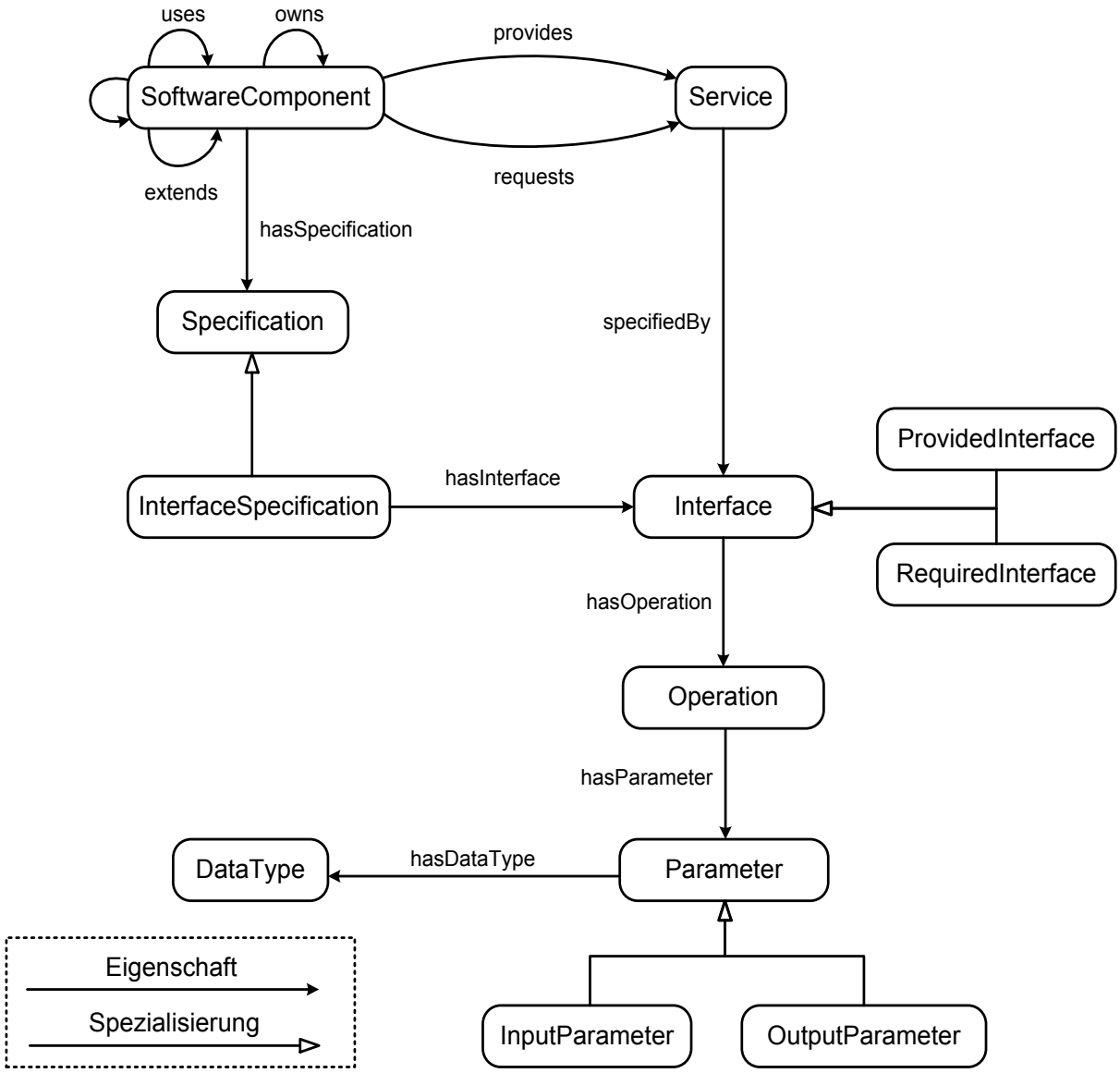


Abbildung 25: Spezifikationselemente von Softwarekomponenten (Seedorf 2010)

Auf Basis der Ontologie werden die Komponenten in einer vereinheitlichten Form zu einer gemeinsamen Wissensbasis zusammengeführt. Als Benutzerschnittstelle wird ein semantisches Wiki eingesetzt. Dieses erlaubt unter anderem eine semantische Suche und eine Visualisierung der Komponentenbeziehungen in Form eines Graphen. Der Ansatz verwendet damit gleich mehrere semantische Technologien und integriert darüber hinaus auch Geschäftsprozesse. Aus diesem Grund befindet er sich auf einem eher hohen Abstraktionsniveau, sodass er

für die Verwendung für die Nachverfolgbarkeit erweitert werden müsste, da beispielsweise Artefakte wie Klassendiagramme von einer spezifischeren Betrachtung profitieren würden.

Zusammengefasst kann eine Ontologie einen bedeutenden Beitrag zum phasenübergreifenden und technologieunabhängigen Verständnis der Artefakte eines Softwareprojekts liefern. Dieses Verständnis hat insbesondere Vorteile für die gemeinsame Auffassung der Terminologie durch die Beteiligten. Es sollte aber beachtet werden, dass die Einrichtung und Pflege einer solchen Wissensbasis mit Kosten verbunden sind, die gegenüber dem zu erwartenden Nutzen abzuwägen sind.

4.2.2.4 Analyse bestehender Systeme

Eine separat zu betrachtende Einsatzmöglichkeit semantischer Technologien ist die der Analyse bestehender Systeme. Häufig wird in diesem Zusammenhang auch der englische Term „Legacy Systems“ verwendet. Es ist unter Umständen wünschenswert zur Unterstützung insbesondere von Wartungsprozessen, post hoc die Beziehungen zwischen Artefakten zu analysieren. Hierzu legt die Forschergruppe um Yoshikawa und Saeki zwei Publikationen vor (Hayashi et al. 2010; Yoshikawa et al. 2009), in denen mithilfe einer Domänen-spezifischen Ontologie die Nachverfolgbarkeitsbeziehungen zwischen Quellcode und natürlichsprachlichen Funktionsbeschreibungen (wieder)entdeckt werden können. Die Erstellung einer Domänen-spezifischen Ontologie ist allerdings mit zusätzlichem Aufwand verbunden. Dieser entfällt bei der Herangehensweise, die von der Forschergruppe um Witte und Zhang präsentiert wird (Witte et al. 2007; Zhang et al. 2006, 2008). Wie eben, wird auch hier vom Vorliegen des Quellcodes ausgegangen, für den Nachverfolgbarkeitsbeziehungen zur ebenfalls vorliegenden Dokumentation in natürlicher Sprache hergestellt werden sollen. Die Grundidee ist hierbei, Ontologien für die einzelnen Artefakte zu erstellen und diese anschließend abzugleichen. Dies erinnert stark an das Ontology Matching von Assawamekin et al. (2008, 2009a), welches im letzten Abschnitt beschrieben wurde. Allerdings wird nicht genauer auf den Abgleichprozess eingegangen, sodass hier keine Aussagen über die Ähnlichkeit getroffen werden können. Ein großer Vorteil Ontologie-basierter Methoden ist, dass im Vergleich zu klassischen Information Retrieval-Techniken mehr als reine Stichwortabgleiche durchgeführt werden können. Dabei kann bereits mit Information Retrieval-Techniken eine beachtliche Verbesserung erzielt werden: In ihrer Studie fanden Hayes et al. (2007) signifikant mehr korrekte Nachverfolgbarkeitsbeziehungen, und dies in einem Drittel der Zeit, verglichen mit manueller Bearbeitung. Alles in allem ist die Analyse bestehender Systeme ein insbesondere für Wartungszwecke wichtiger Bereich, der weiter erforscht werden sollte.

4.2.3 Semantische Technologien für die Transformation von Anforderungsspezifikationen in UML-Diagramme

Formal betrachtet, handelt es sich bei der Erstellung von UML-Diagrammen basierend auf schriftlichen Anforderungsspezifikationen um eine Art der Modelltransformation (vgl. 4.1.2). Eingeordnet in die Klassifikation von Mens und Van Gorp (2006), liegt eine endogene Transformation vor, da verschiedene Sprachen – einmal die natürliche Sprache, zum anderen die formale Sprache der UML – verwendet werden. Ob es sich um eine vertikale oder eine horizontale Transformation handelt, ist abhängig davon, ob ein Unterschied hinsichtlich der Abstraktionslevel vorliegt. Der Abstraktionslevel des Ausgangsmodells – der natürlichen Sprache

– und des Zielmodells – eines oder mehrerer UML-Diagramme – ist wiederum abhängig von Präferenzen und Bedürfnissen der Beteiligten. Zentral für die Erstellung von UML-Diagrammen ist die Identifikation von Kandidaten für die verschiedenen Elemente der Diagramme aus dem vorliegenden Text. Meyer (1998) beschreibt das damit verbundene Vorgehen folgendermaßen: „Class elicitation is a dual process: class suggestion [and] class rejection“ (S. 724). Diese Idee lässt sich von Klassen auf alle andere Elemente übertragen; zunächst müssen Kandidaten für die Elemente gefunden und muss diese Kandidatenliste anschließend eingeschränkt werden. Neben den Grundkonzepten der UML – Klassen, Operationen und Attribute – werden auch Beziehungen zwischen verschiedenen Klassen für diese Arbeit als relevant erachtet, und deren Extraktion wird in den folgenden Unterkapiteln beschrieben. Dabei werden jeweils zunächst konzeptuelle Ansätze aufgezeigt und diese auf eine Automatisierbarkeit hin untersucht.

4.2.3.1 Identifikation von Klassen

Eine grundlegende Idee zur Identifikation von Klassen ist die Überlegung, von Substantiven auszugehen. Allerdings stellen nicht alle Substantive auch automatisch Klassen dar. Eine detailreiche Methode zur Elimination von Kandidaten für Klassen in natürlichsprachlichen Dokumenten schlagen Song et al. (2004) vor. Im ersten Schritt wird dabei für jedes Substantiv in den Anforderungsspezifikationen entschieden, ob es als Kandidat eliminiert werden kann. Die ersten sieben Regeln zur Eliminierung werden aus der Arbeit von Rumbaugh et al. (1991) adaptiert:

- **Regel K1: Überflüssige Klassen** (engl. redundant classes). Wenn zwei Klassen die gleiche Information beinhalten, sollte die behalten werden, die den Sachverhalt besser beschreibt. Ein Beispiel wären zwei Klassen „Kunde“ und „Passagier“, wobei hier die erstere eliminiert werden sollte, da die zweite passender ist.
- **Regel K2: Irrelevante Klassen** (engl. irrelevant classes). Wenn ein Kandidat für eine Klasse nur wenig oder gar nichts mit dem Problem zu tun hat, sollte er eliminiert werden. Beispielsweise ist der Beschäftigungsstatus eines Flugticketkäufers vernachlässigbar.
- **Regel K3: Ungenaue Klassen** (engl. vague classes). Eine Klasse sollte spezifisch und gut abgegrenzt sein. Zum Beispiel bleibt bei einer Klasse „Dokumentation“ im Unklaren, wie sie zu verschiedenen anderen Klassen zugeordnet werden soll.
- **Regel K4: Attribute**. Wenn ein Namenskandidat vor allem ein individuelles Objekt beschreibt, sollte es als Attribut umgesetzt werden. Als Beispiele sind hier die Attribute „Alter“ oder „Adresse“ zu nennen.
- **Regel K5: Operationen**. Analog zu den eben genannten Attributen sollten auch Operationen als Methoden reformuliert werden. Als Beispiel wird von Song et al. die Kalkulation eines Bonusses benannt, es besteht kein Bedarf an einer eigenen Klasse „Bonus“.
- **Regel K6: Rollen**. Der Kandidat für einen Klassennamen sollte nicht die Rolle in einer Assoziation beschreiben, sondern vielmehr die inneren Merkmale. Ein Autoverkäufer sollte demnach nicht die Klasse „Käufer“ verwenden, weil auch andere Personen das Fahrzeug als Fahrer oder Leasingkunde verwenden könnten.

- **Regel K7: Konstrukte der Implementierung**. Falls Konstrukte ausschließlich Implementierungs-bezogen sind und nicht in der realen Welt vorkommen, sollten sie nicht als Klassen im Analysemodell aufgenommen werden. Beispiele sind hier „CPU“ oder „Algorithmus“. In späteren Phasen der Softwareentwicklung können diese Klassen aber durchaus ergänzt werden.

Song et al. (2004) verzichten auf die Anwendung von Regel 6, ergänzen aber darüber hinaus drei weitere Regeln, die teilweise auf den Arbeiten anderer Autoren basieren:

- **Regel K8: Meta-Sprache**. Wenn ein Klassenname die Systemanforderungen nur auf einem sehr hohen Niveau beschreibt, sollte er entfernt werden. Zum Beispiel bezieht sich diese Regel auf Namen wie „System“ oder „Information“; sie wurde aus der Arbeit von Pooley und Stevens (1999) abgeleitet.
- **Regel K9: Werte**. Wenn ein Kandidat für einen Klassennamen einen Wert an sich ausdrückt, sollte er eliminiert werden. Beispielsweise ist „Wartestatus“ die Ausprägung eines Attributwerts und sollte nicht als eigene Klasse abgebildet werden.
- **Regel K10: Abgeleitete Klassen**. Wenn Konzepte von anderen abgeleitet werden können, sollte die Entscheidung über die Inklusion der abgeleiteten Klasse auf die spätere Design-Phase verschoben werden. Diese Regel basiert auf der Arbeit von Blaha und Premerlani (1998).

Im nächsten Schritt führen Song et al. (2004) eine Kategorisierung aller nicht eliminierten Klassen durch. Als Kategorien werden unter anderem Rollen von Teilnehmern, Orte, Organisationen und Interaktionen verwendet. Die Autoren untersuchen weiterhin, ob auch bestimmte Verbphrasen als Klassennamen in Frage kommen – was in der vorgelegten Fallstudie allerdings nicht der Fall war – sowie ob versteckte Klassen vorliegen. Versteckte Klassen sind solche, die nicht explizit den Anforderungsspezifikationen zu entnehmen sind und nur durch die Analyse eines Domänen-Experten identifiziert werden können. Verbphrasen bezeichnen zusammengehörige Segmente eines Satzes, also beispielsweise „ist angekommen“. Sie werden über das Verfahren des sogenannten Chunking ermittelt. Das Vorgehen wird in Jurafsky und Martin (2009) erläutert. Auch wenn diese Methode sehr detailreich und gut nachvollziehbar ist, so ist sie dennoch nicht direkt auf eine automatisierte Lösung übertragbar, da beispielsweise irrelevante oder ungenaue Klassen (Regel K2 und K3) nur schwer durch einen Algorithmus identifiziert werden können. Den Regeln K7 bis K9 folgend, wäre es möglich, bestimmte Wörter anhand einer vordefinieren Liste auszuschließen. Jedoch existiert in der Literatur kein geteiltes Verständnis über eine solche Ausschlussliste.

Ein sehr einfaches, automatisiertes Vorgehen wird von Overmyer et al. (2001) vorgeschlagen, die davon ausgehen, dass die Häufigkeit des Auftretens von Substantiven dazu herangezogen werden kann, ihre Eignung als Klassen zu beschreiben: Je häufiger ein Substantiv auftritt, desto wahrscheinlicher handelt es sich um einen guten Kandidaten für eine Klasse. Daher werden dem Benutzer die Substantive in absteigender Häufigkeit vorgelegt, damit dieser entscheidet, ob es sich um eine Klasse handelt oder nicht. Ebenfalls auf die Häufigkeit als Eliminationskriterium gehen Harmain und Gaizauskas (2003) in ihrem Ansatz ein. Alle Kandidaten für Klassen werden eliminiert, wenn sie nur sehr selten auftreten oder nicht an einer Beziehung beteiligt sind.

Ein fortgeschritteneres System aus Regeln zur Identifikation von Klassen legen Shinde et al. (2012) vor, die einige der bereits genannten Regeln und Ansätze erneut aufgreifen. Zum einen sollen auch hier alle Konzepte eliminiert werden, die sich auf Attribute beziehen (siehe Regel K4), zum anderen Konzepte, das sich auf Elemente des Designs beziehen, wie beispielsweise: „Application“, „System“, oder „Data“ (siehe Regel K8). Des Weiteren werden alle Konzepte ausgeschlossen, die nur einmal auftreten und eine Häufigkeit von unter 2 % besitzen. Die Autoren definieren darüber hinaus eigene Regeln zur Eliminierung:

- **Regel K11: Benannte Entitäten**. Konzepte, die Orte, Personennamen o. Ä. referenzieren, sollten eliminiert werden.
- **Regel K12**: **Substantivphrasen**. In Substantivphrasen (Substantiv+Substantiv), bei denen das zweite Substantiv ein Attribut ist, beschreibt das erste Substantiv die Klasse, wie beispielsweise bei „customer name“.
- **Regel K13: Übergeordnete Klassen**. Generelle Konzepte, d. h., solche, die auf einer hohen Ebene des Hyponymiebaums angesiedelt sind und von einem spezifischen Konzept ersetzt werden können, sollten eliminiert werden.

Die Regel K13 steht im Widerspruch zu Regel K10 von Rumbaugh et al. (1991), wobei die Entscheidung für die eine oder die andere Variante letztendlich abhängig vom gewünschten Abstraktionsniveau der jeweiligen Betrachtung ist. Hyponyme werden mithilfe von WordNet identifiziert. Für die anderen Regeln gehen Shinde et al. (2012) nicht im Einzelnen auf die technische Umsetzung ein. Da an anderen Stellen vordefinierte Listen – beispielsweise für die Identifikation von Attributen – verwendet werden, ist davon auszugehen, dass diese für die Identifikation von Designelementen, Orten und Personennamen eingesetzt werden. Der Ansatz ist praktikabel, limitiert aber die Mächtigkeit der Identifikationsfunktionalität durch den Einsatz der vordefinierten Listen. Nur dort definierte Wörter können gefunden werden. Die Aufnahme eines Wortes in solch eine Liste ist aber in hohem Maße abhängig vom Kontext, sodass für optimale Resultate für jede Applikation neue Listen erstellt werden sollten. Dies verursacht wiederum einen nicht zu unterschätzenden Aufwand.

Auch unter Hinzunahme weiterer Regeln werden aufgrund der Komplexität natürlicher Sprache mit hoher Sicherheit nicht alle Klassen korrekt identifiziert werden können. Grundsätzlich beinhalten die Regeln von Rumbaugh et al. verschiedene Herausforderungen, die konzeptuelle Entscheidungen widerspiegeln und nicht von einem Algorithmus allein getroffen werden können. Beispielsweise ist die Deklaration eines Kandidaten für eine Klasse als „irrelevant“ (Regel K2) durchaus möglich – wie anhand der 2 %-Regel ersichtlich –, aber es ist auch hier vom Benutzer eine Parametrisierung notwendig. Ob ein Kandidat für eine Klasse ungenau ist (Regel K3), kann nur von einem menschlichen Entscheider abgeschätzt werden. Daher obliegt die endgültige Entscheidung, ob ein Substantiv als Klasse verwendet werden soll oder nicht, letztendlich immer dem Benutzer. Eine vollständige Automatisierung ist nicht erreichbar und aus Sicht des Benutzers auch nicht wünschenswert, da die Gestaltungsspielräume sonst zu sehr eingeschränkt wären.

4.2.3.2 Identifikation von Attributen

Attribute im linguistischen Sinn dürfen nicht mit Attributen innerhalb der UML-Notation verwechselt werden, in der sie die Eigenschaften von Klassen beschreiben. Wie auch für

Klassen, schlagen Rumbaugh et al. (1991) eine Vorgehensweise auf konzeptueller Ebene vor. Diese geht von der Annahme aus, dass Attribute normalerweise durch bestimmte Substantive oder Adjektive im Text repräsentiert werden. Damit ein Substantiv als Kandidat für ein Attribut infrage kommt, muss auf es eine Possessiv-Phrase folgen, wie das Beispiel „the color of the car“ verdeutlicht. Als Adjektiv würde die Formulierung „the red car“ auf ein Attribut hindeuten. Des Weiteren stellt bereits Regel K4 in den Eliminierungsregeln für Klassen eine weitere Möglichkeit für die Identifikation von Kandidaten für Attribute dar. Als Eliminationsregeln für Attribute stellen Rumbaugh et al. (1991) unter anderem folgende auf:

- **Regel A1: Objekte**. Attribute sollten sich nicht auf andere Objekte beziehen; in diesem Fall sollte eine Beziehung zwischen den beiden zur Darstellung verwendet werden. Ein anderes Objekt wird immer dann benötigt, wenn es in seiner Existenz unabhängig vom ersten ist. Beispielsweise sollte „Boss“ nicht als Attribut realisiert werden.
- **Regel A2: Qualifizierer**. Falls der Attributwert von einem bestimmten Kontext abhängt, sollte er die Beziehung zwischen zwei Klassen qualifizieren. Als Beispiel sollte die Identifikationsnummer eines Mitarbeiters nicht als Attribut realisiert werden, wenn ein Mitarbeiter auch für mehrere Organisationen gleichzeitig arbeiten kann. In diesem Fall würde die Identifikationsnummer die Beziehung zwischen „Mitarbeiter“ und „Organisation“ qualifizieren.
- **Regel A3: Interne Werte**. Falls der Attributwert einen inneren Zustand beschreibt, der außerhalb des Objekts unsichtbar ist, so sollte er in der Analysephase nicht in das Modell aufgenommen werden.
- **Regel A4: Details**. Attribute, die mit hoher Wahrscheinlichkeit keinen Einfluss auf die meisten Operationen haben, sollten ebenfalls nicht in das Modell aufgenommen werden.
- **Regel A5: Unvereinbare Attribute** (engl. discordant attributes). Attribute, die sich stark von allen anderen Attributen unterscheiden, können darauf hindeuten, dass hier zwei separate Klassen erstellt werden sollten.

Diese Aufstellung der Regeln ist nicht komplett; Rumbaugh et al. (1991) führen zusätzlich drei weitere, sehr detaillierte Regeln auf, die an dieser Stelle aber aufgrund ihrer Detailtiefe zu weit führen würden.

Analog zu den Eliminierungsregeln für Klassen sollte auch hier hinterfragt werden, inwieweit sich diese Regeln für ein automatisiertes Verfahren eignen. Bereits oben wurde der Ansatz von Shinde et al. (2012) vorgestellt, wobei Regel K12 sich ebenfalls auf Attribute bezieht: Folgen zwei Substantive aufeinander, so gehen die Autoren generell davon aus, dass es sich bei dem zweiten Substantiv um das Attribut handelt. Im bereits angeführten Beispiel „customer name“ stellt „name“ dann das Attribut dar. Für die weitere Identifikation von Attributen werden Listen vordefinierter Wörter verwendet. Die damit einhergehenden Problematiken wurden bereits im letzten Abschnitt verdeutlicht.

Einen grundsätzlich anderen Ansatz wählen Harmain und Gaizauskas (2003), die für die automatische Identifikation von Attributen auf die eingangs dieses Abschnitts vorgestellte Idee zurückgreifen, dass Adjektive für die Identifikation von Attribut-Kandidaten herangezogen werden können. Da, wie im Beispiel „the red car“ deutlich wird, „red“ lediglich die Ausprä-

gung beschreibt und nicht als Name für das Attribut verwendet werden kann, wird weiteres lexikalisches Wissen benötigt, welches in diesem Fall über die Einbindung von WordNet umgesetzt wird. Als weitere Regel benutzen Harmain und Gaizauskas die Heuristik, dass die Verben „have“, „denote“ und „identify“ ein Attribut indizieren. Die Elimination von Attributen wird in diesem Fall dem Benutzer überlassen, der aus einer Liste von Attributen jeweils diejenigen auswählt, die für eine bestimmte Klasse passen.

Aus den Unterschieden der gezeigten Ansätze ergibt sich die Problematik einer vollständigen Automatisierung: Attribute können in Verbindung mit bestimmten Substantiven oder Verben sowie in Form von Adjektiven auftreten. Die hier vorgestellten konzeptuellen Eliminationsregeln sind dabei nur schwer oder gar nicht automatisierbar. Rumbaugh et al. (1991) heben als weitere generelle Problematik hervor, dass Attribute im Gegensatz zu Klassen oder Beziehungen häufig nicht sehr ausführlich beschrieben werden. Beispielsweise ist aus der Formulierung „the red car“ nicht ableitbar, welche anderen Farben ein Auto haben kann. Es ist daher notwendig, dass zusätzliche Rückschlüsse bezogen auf den Anwendungsbereich getroffen werden. Allerdings beeinflussen Attribute selten die Grundstruktur des Problems, sodass fehlende Attribute auch in späteren Phasen der Softwareentwicklung noch ergänzt werden können, ohne dass tiefgreifende Änderungen am Gesamtsystem vorgenommen werden müssen.

4.2.3.3 Identifikation von Operationen

Die Ausführung einer Operation kann zu Änderungen an den Attributwerten oder zur Rückgabe eines Werts führen, wobei Klassen in der Regel Methoden oder anderes Verhalten für die Bearbeitung eines Operationsaufrufs bereitzustellen (OMG 2012a). Eine Operation kann durch eine, aber auch durch mehrere Methoden implementiert sein. Wie auch für die Identifikation von Klassen, legt Abbott (1983) eine Regel für die Identifikation von Operationen vor: „A verb, attribute, predicate, or descriptive expression suggests an operator“ (S. 884). Diese Regel ist sehr allgemein und macht eine weitere Präzisierung notwendig. Eine detaillierte Verfahrensweise zur Identifikation von Operationen beschreiben Bajwa et al. (2009), die die Annahme treffen, dass Kandidaten für Methoden durch bestimmte Verben repräsentiert werden. Diese Verben müssen für die Identifikation als Operationen bestimmte Anforderungen erfüllen:

- **Regel O1**: Das Verb tritt nach einem Substantiv oder einem Pronomen auf, und es ist kein Hilfsverb vorhanden. Beispielsweise impliziert für den Satz „The customer buys a pen“ das Verb „buy“ eine entsprechende Methode für die Klasse „Customer“.
- **Regel O2**: Das Verb tritt nach einem Hilfsverb auf. Wiederum auf das soeben genannte Beispiel bezogen, könnte der Satz auch folgendermaßen formuliert werden: „The customer is buying a pen“, wobei sich erneut „buy“ zur Extraktion anbietet.
- **Regel O3**: Das Verb tritt nach einer Präposition wie „to“ oder „of“ auf. Als Beispielsatz soll hier dienen: „The customer is demanding to cancel his order”, wobei „cancel“ als Methode extrahiert werden sollte.

Der Vorteil dieser Regeln ist, dass sie sich gut automatisieren lassen. Allerdings ergibt sich eine bisher nicht aufgeführte Problematik, da die Regeln nicht ohne Weiteres auf andere Sprachen wie das Deutsche übertragbar sind. Die Satzstellung ist im Deutschen teilweise anders aufgebaut, so werden Hilfsverb und Verb häufig durch andere Satzteile getrennt. Als

Beispiel soll hier der Satz „Die Bestellung ist zur Stornierung freigegeben.“ dienen. Auch für die bereits besprochene Identifikation von Klassen und Attributen können sprachspezifische Hürden auftreten, wie für die Substantivphrase „customer name“, die in beiden Unterkapiteln bereits als Beispiel verwendet wurde. Da (teil-)automatisierte Ansätze sich in der Regel auf das Englische beziehen, setzt die Übertragung der hier gezeigten Regeln auf deutsche Texte voraus, dass überprüft werden muss, ob für die Anwendung der Regeln äquivalente linguistische Konstrukte existieren. Dies dürfte zumindest für einen Teil der Regeln der Fall sein, sofern geeignete Werkzeuge beispielsweise für die Zerlegung zusammengesetzter Substantive vorliegen, wie für „Kundenname“ im eben genannten Beispiel.

Auf eine sprachübergreifende Problematik weisen Booch et al. (2007) hin: Bei der Identifikation von Klassen beziehungsweise Objekten werden häufig Substantive herangezogen, während für die Identifikation von Operationen vor allem Verben im Fokus stehen. Allerdings können Substantive auch als Verben ausgedrückt werden und umgekehrt Verben substantiviert werden. Dies dürfte ein weiterer Grund dafür sein, dass eine vollständige Automatisierung bisher nicht umgesetzt wurde.

4.2.3.4 Identifikation von Beziehungen

Als grundlegende Regel für Beziehungen (engl. associations), in diesem Fall im Kontext von Entity Relationship-Diagrammen, stellt Chen (1983) fest: „A transitive verb in English corresponds to a relationship type“ (S. 130). Dies lässt sich auf UML-Diagramme übertragen; Rumbaugh et al. (1991) erstellen analog zu den Regeln für die Elimination von Klassen auch eine Vorgehensweise für Beziehungen zwischen Klassen. Als Ausgangsbasis für Beziehungen werden Zustandsverben und Verbphrasen verwendet, Beispiele hierfür sind Angaben wie „ist enthalten in“ oder „arbeitet für“. Es werden analog zu den oben (siehe 4.2.3.1) angeführten Regeln alle Kandidaten eliminiert, die irrelevant (Regel K2) oder rein Implementierungsbezogen sind (Regel K7). Weiterhin schlagen Rumbaugh et al. (1991) folgende Regeln zur Elimination von Beziehungen vor:

- **Regel B1: Beziehungen zwischen eliminierten Klassen**. In dem Fall, dass eine der Klassen einer Beziehung entfernt wird, muss entweder auch die Beziehung entfernt werden oder so umgestaltet werden, dass eine andere Klasse deren Platz einnimmt.
- **Regel B2**: **Aktionen** (engl. actions). Beziehungen sollten beständige Konzepte repräsentieren, also strukturelle Eigenschaften statt transienter Ereignisse wie „Geldautomat akzeptiert Karte“.
- **Regel B3: Ternäre Beziehungen**. Die meisten – aber nicht alle – Beziehungen zwischen drei oder mehr Klassen können in binäre Beziehungen zerlegt werden.
- **Regel B4: Abgeleitete Beziehungen**. Alle Beziehungen, die durch andere Beziehungen ausgedrückt werden können, sollten entfernt werden. So kann die Beziehung „Großvater von“ auch ausgedrückt werden durch zwei „Elternteil von“-Beziehungen.

Wie in den vorangegangenen Abschnitten, stellt sich auch hier die Frage, inwieweit diese Regeln automatisierbar sind. Erneut liefert die Arbeit von Shinde et al. (2012) einen pragmatischen Ansatz. Wenn ein Satz mit dem Aufbau Konzept1-Verb-Konzept2 gefunden wird, bei dem Konzept1 und Konzept2 jeweils eine Klasse sind, dann wird das Verb als eine Beziehung identifiziert. Bei einem Satzaufbau von Konzept1-Beziehung1-Konzept2-UND-Konzept3

wird die Beziehung zwischen den Klassen Konzept1 und Konzept2 sowie zwischen den Klassen Konzept1 und Konzept3 erstellt. Die gleiche Satzformation, die anstelle des UND ein UND_NICHT enthält, stellt die Beziehung lediglich zwischen den Klassen von Konzept1 und Konzept2 her. Um Meronymie- und Hyponymie-Beziehungen zu erfassen, wird überprüft, ob das gefundene Verb in der folgenden Liste enthalten ist: „consists of“, „contain“, „hold“, „include“, „divided to“, „has part“, „comprise“, „carry“, „involve“, „imply“, „embrace“.

Ebenfalls auf der Analyse des Satzbaus beruht der Ansatz von Harmain und Gaizauskas (2003). Dabei werden für alle Verben das logische Subjekt, das logische Objekt und gegebenenfalls weitere Präpositional-Phrasen bestimmt. Verben, die nicht über solche gebundenen Ergänzungen verfügen, werden von der Kandidatenliste für Beziehungen entfernt. Dies basiert auf der Annahme, dass intransitive Verben vor allem eine Statusänderung indizieren, nicht aber eine Beziehung.

Abschließend ist zu bemerken, dass Verben als grundlegende Kandidaten sowohl für die Identifikation von Beziehungen als auch die von Operationen herangezogen werden. Dies ist nicht weiter problematisch, da beide Elemente sich überschneiden. Für den Fall, dass eine Klasse die Operation beziehungsweise Methode einer anderen Klasse aufruft, kann zwischen den beiden Klassen auch eine Beziehung bestehen. So ist es sinnvoll, dass die Klasse „Organisation“ sowohl eine Beziehung zur Klasse „Mitarbeiter“ aufweist als auch eine Methode in mindestens einer der beiden Klassen implementiert wird, die die Referenz auf die andere Klasse speichert oder darüber hinaus weitere damit verbundene Funktionalitäten bereitstellt.

4.2.3.5 Identifikation weiterer Elemente

Die bisher beschriebenen Identifikationen beziehen sich ausschließlich auf grundlegende Elemente eines Klassendiagramms. Weitere Notationselemente wie Stereotyp, Schnittstelle oder Template werden in den Ansätzen nicht betrachtet. Für Notationselemente anderer UML-Diagrammtypen liegen bisher nur sehr wenige Ansätze vor. Für die Identifikation von Anwendungsfalldiagrammen stellt Börstler (1996) einen Ansatz vor, welcher allerdings die Eingabe nur über vorgefertigte Formulare erlaubt. Diese Formulare haben beispielsweise ein separates Feld für den Akteur und enthalten als semantisches Konzept lediglich eine Nomenklatur. Kamalrudin et al. (2010) analysieren schriftliche Anforderungsspezifikationen mit dem Ziel, diese in ein vereinfachtes Anwendungsfalldiagramm umzuwandeln. Dabei gleichen sie Textbausteine gegen eine selbsterstellte Bibliothek mit vordefinierten Phrasen ab. Semantische Funktionalitäten wie das POS Tagging werden nicht verwendet.

Vinay et al. (2009) verfolgen ebenfalls das Ziel, Klassen- und Anwendungsfalldiagramme automatisch aus Text zu extrahieren. Für Klassendiagramme werden Regeln aufgestellt, die eher basaler Natur sind und nicht über die bereits vorgestellten hinausgehen. Als Kandidaten für Akteure im Anwendungsfalldiagramm wird von Substantiven ausgegangen. Dabei werden nur die häufigsten verwendet. Der Wert für die Häufigkeit wird allerdings nicht angegeben. Als Anwendungsfälle (engl. Use Cases) werden jeweils die mit dem gewählten Substantiv assoziierten Verbteile extrahiert. Was ein Verbteil genau ist und wie das Vorgehen algorithmisch gestaltet ist, wird nicht weiter beschrieben.

Die Arbeit der Forschungsgruppe um Pérez-González (Pérez-González et al. 2005; Pérez-González und Kalita 2002a, 2002b) unterstützt neben der Erstellung von Klassendiagrammen auch die von Objekt-, Sequenz- und Aktivitätsdiagrammen, beruht aber auf einer semi-natürlichen Sprache. Das heißt, dass Eingaben nur sehr strukturiert vorgenommen werden können und somit argumentiert werden kann, dass hier überhaupt keine semantische Analyse vorliegt. Einzig der Ansatz von Shinde et al. (2012) verwendet NLP-Technologien zur Erstellung von Klassen- und Sequenzdiagrammen. Dabei werden für Letztere Akteure und Nachrichten identifiziert. Die Bezeichnung Akteur wird hier fälschlicherweise für Lebenslinie (OMG 2012a) verwendet und durch Substantive identifiziert. Die Identifikation von Nachrichten wird nicht erläutert, auch Beispiele für erzeugte Sequenzdiagramme werden nicht gezeigt. Daher gestaltet sich die Bewertung des konzeptuellen Beitrags schwierig.

4.2.3.6 Weitere Transformationen

Neben der Transformation von Anforderungsspezifikationen in UML-Diagramme können auch andere Transformationen interessant sein. Zunächst können andere Diagrammtypen wie Entity Relationship-Diagramme im Fokus stehen (Chen 1983). Weiterhin können Transformationen zwischen verschiedenen Modellen das Ziel sein, wie beispielsweise zwischen UML-Diagrammen und Ontologie-Sprachen (Baclawski et al. 2001). Nicht immer müssen Diagramme das Resultat einer Transformation sein. So zeigen Kuhlmann und Gogolla (2012), wie UML-Diagramme mithilfe der Object Constraint Language (OCL) in ein relationales Modell überführt werden.

Da in dieser Arbeit natürlichsprachliche Anforderungsspezifikationen im Vordergrund stehen, soll an dieser Stelle die Arbeit von Meth und Kollegen angesprochen werden (Meth et al. 2012; Meth 2013). Die Autoren entwerfen ein Konzept zur Analyse von Anforderungsspezifikationen mit dem Ziel, die Produktivität der Beteiligten zu erhöhen, indem entweder die Qualität erhöht oder der Aufwand verringert wird. In der Umsetzung werden mithilfe von NLP-Techniken Tags für Wörter oder Abschnitte in einer Anforderungsspezifikation vergeben. Mögliche Tags sind Actor, Activity, Data und Non-functional. Diese werden dem Benutzer vorgeschlagen, wobei er das Ergebnis anschließend modifizieren kann. Anhand des implementierten Prototyps konnte nachgewiesen werden, dass dieser semi-automatische Ansatz die Produktivität bei der Identifikation und Klassifikation von Anforderungsspezifikationen verbessert. Dieses Konzept liefert einen wichtigen Beitrag zur Überführung von schriftlichen Anforderungsspezifikationen in eine strukturierte Form, bietet aber keinen Verbindungspunkt zur Etablierung von Nachverfolgbarkeitsbeziehungen mit anderen Artefakten.

Transformationen müssen nicht immer von einem Text ausgehen. Es besteht ebenfalls die Möglichkeit, Text aus formalen Modellen zu generieren (Dalianis 1992). Diese sollen als Diskursgrundlage für die Validierung dienen. Da in dieser Arbeit davon ausgegangen wird, dass Anforderungsspezifikationen in Textform vorliegen, wird im Folgenden deren Validierung erläutert.

4.2.4 Semantische Technologien für die Validierung von Anforderungsspezifikationen

Wie bereits in der konzeptuellen Besprechung im ersten Unterkapitel (vgl. 4.1.3) angesprochen, bezieht sich die Validierung häufig auf das gesamte System. Es existieren auch Ansätze, die sich explizit der Überprüfung natürlichsprachlicher Anforderungsspezifikationen widmen.

Wie eben bereits diskutiert, verwenden Autoren den Begriff Validierung in inkonsistenter Weise. In der Regel wird weniger die Übereinstimmung mit den Bedürfnissen des Kunden überprüft, sondern eher die Konsistenz innerhalb der Anforderungsspezifikationen.

Gervasi und Nuseibeh (2002) schlagen vor, Anforderungsspezifikationen mithilfe eines Parsers in ein formales Modell zu überführen. Dieses Modell beschreibt, welche Aktionen vom System bei einem bestimmten Ereignis ausgeführt werden und welche Bedingungen dafür erfüllt sein müssen. Über logische Operationen können dann Anforderungsspezifikationen identifiziert werden, die im Konflikt miteinander stehen. Die Autoren verwenden die Anforderungsspezifikationen eines der Kontrollsysteme der Internationalen Raumstation ISS als Fallstudie. Anhand dieser zeigen sie, dass der Ansatz zum Aufdecken von Fehlern verwendet werden kann, die sonst durch eine manuelle Überprüfung hätten gefunden werden müssen. Es wird also vielmehr auf die Fehlerfreiheit abgezielt, was sicher auch eines der Bedürfnisse des Kunden ist, aber diese nicht abdeckt. Als Beispiel hierfür ist die nicht überprüfte Vollständigkeit der Anforderungsspezifikationen genannt. Ein Modell, welches neben der Konsistenz auch andere Qualitätskriterien berücksichtigt, ist das von Fantechi et al. (2003). Das Qualitätsmodell baut auf drei Grundeigenschaften auf: Testbarkeit, Konsistenz und Verständlichkeit. Für jede Grundeigenschaft werden mehrere Indikatoren gebildet, die wiederum über Metriken messbar sind. Für deren Anwendung werden NLP-Techniken verwendet. Die Funktionalität wird nur anhand eines einzelnen Satzes getestet, sodass die Ergebnisse der Metriken nur wenig Aussagekraft haben. Ein weiterer Ansatz, der in mehr als 500 Projekten getestet wurde, ist der von Verma et al. (2011). Hier werden ausführliche Glossare zur Aufdeckung von Inkonsistenzen erstellt. Allerdings können Anforderungen nur in einer restriktierten Sprache formuliert werden, was die Anwendbarkeit des Ansatzes einschränkt.

Insgesamt betrachtet, bilden automatisierte Ansätze zur Validierung natürlichsprachlicher Anforderungsspezifikationen ein bisher wenig erforschtes Feld. Dies kann darin begründet sein, dass die verschiedenen Ziele teilweise in Konflikt zueinander stehen. So kann die Verbesserung der Konsistenz zu einer Verringerung der Vollständigkeit und damit auch einer verminderten Korrektheit führen (Zowghi und Gervasi 2003). Die Validierung von Anforderungsspezifikationen oder auch von ganzen Systemen ist äußert komplex und wird hier nicht weiter behandelt.

4.2.5 Werkzeuge

Für den Einsatz im Requirements Engineering liegen der bereits angesprochene Übersichtsartikel für Wikis von Lai et al. (2012) sowie für semantische Wikis von Hoenderboom und Liang (2009) vor. Insbesondere da der Einsatz eines speziellen (semantischen) Wikis jeweils von den Anforderungen an die Funktionalitäten abgeleitet werden sollte, wird auf die dahingehende Auswertung in den Übersichtsartikeln verwiesen. Als Fazit sei erwähnt, dass unter den 19 von Hoenderboom und Liang untersuchten semantischen Wikis zwei herausstanden, da sie die meisten betrachteten Funktionalitäten erfüllen. Eines davon ist das Semantic Media Wiki, welches auch die Grundlage für die Entwicklung des von Nordheimer et al. (2012) vorstellten Prototyps ist. Das CLEoS-Wiki implementiert das oben beschriebene semantische Datenmodell (vgl. 4.2.2.2) und unterstützt auf diese Weise Nachverfolgbarkeitsbeziehungen

zwischen Anforderungsspezifikationen, Architektur- und Softwarekomponentendiagrammen, Geschäftsprozessen sowie deren Aktivitäten.

Speziell für die Unterstützung von Nachverfolgbarkeitsbeziehungen können die Werkzeuge Trace/Analyzer (Egyed 2003), XTraQue (Jirapanthong und Zisman 2007), ADAMS (Lucia et al. 2007) und TraVis (Hildenbrand et al. 2009) verwendet werden. Trace/Analyzer und XTraQue verwenden jeweils ein regelbasiertes Vorgehen zur automatischen Generierung von Nachverfolgbarkeitsbeziehungen, die verschiedenen Typen zugeordnet werden können. ADAMS unterstützt darüber hinaus die Kooperation zwischen den Beteiligten. Das am weitesten entwickelte Werkzeug ist TraVis, welches als zusätzliche Funktionalität auch eine sehr fortgeschrittene Visualisierung bereitstellt (vgl. 4.2.1.1). Ein Nachverfolgbarkeits-Werkzeug, das auch Ontologien einbezieht, ist MUPRET (Assawamekin et al. 2009a, 2009b). Es setzt die oben (vgl. 4.2.1.3 und 4.2.2.3) theoretisch beschriebenen Konzepte praktisch um. Ebenfalls auf Ontologien basiert das Werkzeug von Zhang et al. (2006, 2008), welches außerdem versucht, verschiedene Entwurfsmuster (engl. Design Pattern) in der Dokumentation zu identifizieren.

Für die Transformation von Anforderungsspezifikationen mithilfe von NLP-Technologien wurden Werkzeuge wie NL-OOPS von Mich (1996) oder SMART von Dori et al. (2004) entwickelt, die zwar Objekt-orientierte Diagramme erstellen, aber nicht dem heutigen UML-Standard entsprechen. Mit der Verbreitung von UML als Standard wurden auch entsprechende Werkzeuge implementiert. Die meisten von diesen konzentrieren sich auf bestimmte Diagramme, wie beispielsweise Klassendiagramme (Amdouni et al. 2011; Hudaib et al. 2007; Ibrahim und Ahmad 2010; Vinay et al. 2009). Ein Werkzeug, das mehrere UML-Diagrammtypen unterstützt, ist CIRCE von Ambriola und Gervasi (2006), welches allerdings eine Restriktion der verwendeten Sprache voraussetzt. Als Basis für die Analyse kommt dabei auch das NLP-Werkzeug GATE[20] (General Architecture for Text Engineering) zum Einsatz. Diese Open Source Software enthält die wichtigsten Funktionalitäten für die Tokenisierung, POS Tagging etc., ist aber im Funktionsumfang kleiner als die beiden bereits vorgestellten Stanford CoreNLP und Apache OpenNLP (vgl. 3.2.3).

4.3 Zusammenfassung

Die in diesem Kapitel vorgestellten Ansätze bilden den aktuellen Stand der Forschung bezüglich des Einsatzes semantischer Technologien für die Anforderungsanalyse und die Nachverfolgbarkeit ab. Eine der wichtigsten semantischen Technologien sind Wikis und deren Erweiterung, die semantischen Wikis. Letztere haben sich in den letzten Jahren sehr schnell entwickelt und stehen häufig unter einer Open Source-Lizenz zur Verfügung (Maalej et al. 2008). Darüber hinaus sinken die Einstiegshürden weiter, da sich immer mehr Editoren im „What you see is what you get“-Stil durchsetzten. So wurde erst kürzlich ein solcher für Wikipedia fertiggestellt (Kleinz 2013), während bisher eher vereinzelt spezielle (semantische) Wikis diese Funktionalität anboten.

Für die Identifikation von UML-Diagrammen aus schriftlichen Anforderungsspezifikationen kann das Fazit gezogen werden, dass bisher vor allem Klassendiagramme im Vordergrund

[20] http://gate.ac.uk

stehen. Die Verknüpfungen zwischen Anforderungsspezifikationen in unrestriktierter natürlicher Sprache und anderen Diagrammtypen bilden eine Forschungslücke. Dies mag zum einen der Komplexität der natürlichen Sprache und den Limitationen der vorhandenen Methoden geschuldet sein. Zum anderen spiegelt es das nicht umgesetzte Konzept der Nachvollziehbarkeit zwischen allen Artefakten wider. Es fehlt daher ein Ansatz, der semantische Technologien und Nachvollziehbarkeit zusammenbringt. Ein solcher Lösungsansatz wird im folgenden Kapitel beschrieben.

5 Lösungsansatz

Die Artefakte eines Softwareprojekts zu verwalten, ist eine wichtige Aufgabe in der Softwareentwicklung. Wie bereits im Grundlagenkapitel 2 beschrieben, dienen die Artefakte selbst und die mit ihnen verbundenen Nachverfolgbarkeitsinformationen als Basis für die Kommunikation zwischen den Beteiligten. Sowohl auf dem Gebiet der Nachverfolgbarkeit als auch im Bereich der semantischen Technologien wurden in den letzten Jahren zahlreiche Forschungsbeiträge publiziert. Dabei haben sich die Funktionalitäten, die durch semantische Technologien bereitgestellt werden können, deutlich vergrößert, wie im Grundlagenkapitel 3 dargestellt.

Die Unterstützung des Softwareentwicklungsprozesses durch semantische Technologien wurde im vorigen Kapitel vorgestellt und dabei die Forschungslücke identifiziert. Bisher beschränken sich die Ansätze auf den Einsatz von NLP-Technologien für die Erstellung von Klassendiagrammen, wobei keine Verbindung zu anderen Artefakten vorgesehen ist und damit keine Nachverfolgbarkeit möglich ist. Ansätze, die sich mit der Nachverfolgbarkeit beschäftigen, verwenden zwar semantische Technologien wie (semantische) Wikis, bieten aber keine semantische Analyse der Anforderungsspezifikationen. Diese Arbeit bringt semantische Analyse und Nachverfolgbarkeit zusammen und zeigt auf, wie diese Verbindung für die Softwareentwicklung eingesetzt werden kann.

Dieses Kapitel gibt zunächst einen Überblick über den Lösungsansatz. Dabei werden die einzelnen Anforderungen herausgearbeitet, die im zweiten Unterkapitel vertieft behandelt werden. Das dritte Unterkapitel widmet sich der Architektur des Lösungsansatzes und stellt neben den Begriffen und dem Anwendungskonzept auch die Module dar. Anschließend werden das Ziel, das Konzept sowie der Ablauf der Analyse vorgestellt, die zur Generierung der Diagramme führt. Die Beschreibung der Limitationen des Lösungsansatzes und eine Zusammenfassung schließen das Kapitel ab.

5.1 Überblick

Bereits eingangs dieser Arbeit wurden die übergeordnete Forschungsfrage sowie die konkreten Fragestellungen beschrieben (vgl. 1.2). Die Fragestellungen beziehen sich auf die Artefakte selbst sowie die Nachverfolgbarkeitsbeziehungen, die Automatisierbarkeit der semantischen Analyse sowie die Präsentation der Ergebnisse. Für diese Fragestellungen werden nun Anforderungen abgeleitet und zunächst in Form eines Überblicks vorgestellt. Die einzelnen Anforderungen werden im nächsten Unterkapitel vertieft und direkt den Erkenntnissen gegenübergestellt, die in Kapitel 2 bis 4 erarbeitet wurden.

Die erste Fragestellung betrifft die Artefaktauswahl. Laut der gezeigten Definition (vgl. 2.1.1) umfassen Artefakte alles, was im Laufe des Softwareentwicklungsprozesses erstellt wird. Aus dieser fast unbegrenzten Anzahl an möglichen Artefakten muss eine Auswahl getroffen werden. Die schriftlichen Anforderungsspezifikationen stehen dabei im Fokus der Betrachtung. Sie sollen in natürlicher Sprache formuliert werden (Anforderung 1). Daneben sollen Diagramme der Softwaremodellierung als Artefakte einbezogen werden. Klassendiagramme stellen das zentrale Konzept der UML dar und können über den gesamten Softwareentwicklungs-

prozess hinweg eingesetzt werden (Kecher 2011). Aus diesem Grund sollen sie auch in diesem Lösungsansatz berücksichtigt werden. Klassendiagramme repräsentieren als Strukturdiagramme (vgl. 2.2.1.2) die statischen Elemente einer Software, weshalb der Lösungsansatz als Erweiterung auch dynamische Elemente in Form eines Verhaltensdiagramms berücksichtigen soll. Das grundlegendste Verhaltensdiagramm ist das Anwendungsfalldiagramm. Dieses liefert eine grobe Sicht auf die Funktionalität des Systems und beschreibt die Erwartungen, die ein Benutzer an das Verhalten eines Systems hat (Kecher 2011). Daher werden Anwendungsfalldiagramme als Vertreter der Verhaltensdiagramme ausgewählt und als Artefakt einbezogen. Die Verbindung zu Geschäftsprozessen wurde in der Forschung bisher vernachlässigt. Es wird daher der Vorschlag von Nordheimer et al. (2012) aufgegriffen, Geschäftsprozesse in die Betrachtung der Nachverfolgbarkeit mit einzubeziehen und so zur Schließung dieser Forschungslücke weiter beizutragen (Anforderung 2). Insgesamt werden damit die Artefakte der schriftlichen Anforderungsspezifikationen in natürlicher Sprache, Anwendungsfall- und Klassendiagramme sowie Geschäftsprozesse einbezogen.

Die zweite Fragestellung bezieht sich auf die Gestaltung der Nachverfolgbarkeitsbeziehungen. Der Lösungsansatz soll Nachverfolgbarkeitsbeziehungen in der Art umsetzten, dass die Bidirektionalität sichergestellt wird. Das bedeutet, dass die Möglichkeit der Nachverfolgbarkeit zwischen zwei Artefakten sowohl vorwärts als auch rückwärts (vgl. 2.1.2) gewährleistet ist. Erweitert wird die Betrachtung der Nachverfolgbarkeitsbeziehungen um den Aspekt der zeitlichen Trennung der Artefakte. Die Überwindung der zeitlichen Zäsur, die zwischen der Fertigstellung der Anforderungsspezifikationen und dem Beginn der Erstellung der weiteren Artefakte entsteht, bildet eine Forschungslücke. Der hier vorgestellte Lösungsansatz schließt diese Forschungslücke, indem er die Erstellung der Anforderungsspezifikationen zeitgleich mit der Erstellung und Weiterentwicklung der weiteren Artefakte ermöglicht. Dazu soll der eingegebene Text der Anforderungsspezifikation direkt während der Eingabe analysiert werden (Anforderung 3). Diese Analyse während der Eingabe steht in direktem Zusammenhang mit der dritten Fragestellung, der nach der Automatisierbarkeit der semantischen Analyse. Der Lösungsansatz soll dazu beitragen, die Erstellung der weiteren Artefakte zu beschleunigen, da die in den schriftlichen Anforderungsspezifikationen enthaltenen Informationen durch die algorithmische Unterstützung weiterverwendet werden können. Dazu soll durch den Einsatz aktueller semantischer Technologien die Analyse für die Generierung von Vorschlägen für Diagramme verwendet werden.

Die vierte Fragestellung umfasst die Ergebnispräsentation der Analyse. Der Lösungsansatz soll dem Paradigmenwechsel Rechnung tragen, der sich zurzeit vollzieht. Statt Anwendungen auf einem einzelnen Rechner oder innerhalb eines Netzwerkes als Desktop-Anwendung anzubieten, findet mehr und mehr eine Verlagerung hin zu Web-basierten Lösungen statt (Miller 2009). Dabei werden die Daten auf einem Server gespeichert, und der Zugriff findet über das Internet statt. Der zu entwickelnde Lösungsansatz soll eine Browser-native Oberfläche bieten (Anforderung 4). Weiterhin soll der Lösungsansatz einen Fokus auf die Benutzerfreundlichkeit (Anforderung 5) legen, damit die Einstiegsbarrieren für den Benutzer gering gehalten werden können.

Insgesamt bietet die vorliegende Arbeit damit einen neuen Lösungsansatz für die Integration der Nachverfolgbarkeit und des Softwarelebenszyklus (vgl. 2.1.4). Bisher wurden schriftliche

Anforderungsspezifikationen häufig getrennt von den weiteren Artefakten betrachtet. Dies führte zum Verlust der Kontinuität, da während der Entwicklung auftretende Änderungen nicht mehr eingepflegt werden (Pohl 2008). Letztendlich führt diese Trennung auch zum Auftreten von Diskrepanzen zwischen Anforderungsspezifikationen und anderen Artefakten, womit auch die Nachverfolgbarkeitsbeziehungen verloren gehen. Ein Grund für die getrennte Betrachtung könnte darin liegen, dass bestehende Ansätze sich zu sehr auf die Rollen der Beteiligten konzentrieren. Allerdings kann eine Person auch mehrere Rollen einnehmen, d. h. gleichzeitig für die Erstellung der Anforderungsspezifikationen und die Softwaremodellierung zuständig sein, oder es können mehrere Beteiligte bei diesen Aufgaben zusammenarbeiten.

5.2 Anforderungen

Die eben identifizierten Anforderungen werden nun mit den Grundlagen aus den Kapiteln 2 bis 4 verknüpft und dabei jeweils die besonderen Herausforderungen dargestellt. Die Nummerierung der Abschnitte entspricht dabei der Nummerierung der Anforderungen.

5.2.1 Unterstützung natürlicher Sprache

Es gibt grundsätzlich zwei Möglichkeiten für die Formulierung schriftlicher Anforderungsspezifikationen: zum einen die Verwendung unrestriktierter natürlicher Sprache und zum anderen den Einsatz einer strukturierten semi-natürlichen Sprache. Beide Optionen haben Vor- und Nachteile; so wurde die Komplexität natürlicher Sprache bereits in Abschnitt 3.1.1 dargestellt. Ein weiterer Nachteil, der speziell auf natürlichsprachliche Anforderungsspezifikationen zutrifft, ist die Vermischung der verschiedenen Perspektiven, d. h. der Struktur, der Funktion und von Verhalten (Pohl 2008). Zum anderen unterstützt die natürliche Sprache den Benutzer in einer ganz besonderen Weise: Dalianis (1992) beschreibt mehrere Vorteile. So verfügen nicht alle Beteiligten über die notwendige Ausbildung, formale Beschreibungen nachvollziehen zu können. Außerdem ist die Einstiegshürde für neue Benutzer niedriger, da keine Formalia für die Verwendung des Systems zu erlernen sind. Damit liefert die natürliche Sprache auch einen wichtigen Beitrag zur Benutzerfreundlichkeit (Dalianis 1992).

Strukturierte Sprache baut im Gegensatz dazu auf einer Einschränkung der verwendeten Wortformen, Satzstrukturen oder einem ausgewählten Vokabular auf (Yue et al. 2011). Restriktierungen der Wortformen können sich beispielsweise in der Einschränkung auf bestimmte Zeitformen oder die Singular-/Pluralform eines Substantives äußern. Satzstrukturen können zum Beispiel dadurch eingeschränkt werden, dass Bedingungen immer in der Form „if-then“ ausgedrückt werden müssen. Bei der Einschränkung des Vokabulars muss zum Beispiel zwangsweise ein bestimmtes Schlüsselwort für die Vererbung verwendet werden. Ein Beispiel für eine sehr restriktierte Sprache ist Simplified English. Dieser Industriestandard stammt aus der Luft- und Raumfahrtbranche und beschränkt das Vokabular auf ungefähr 1500 Wörter, von denen weniger als 300 Verben sind (Hoard et al. 1992). Jedes dieser Wörter darf nur in genau einer Bedeutung verwendet werden, sodass Ambiguitäten vermieden werden. Auch hier ist das Ziel, die Verständlichkeit zu erhöhen, wobei dabei insbesondere Unterstützung für eine automatische Übersetzung sowie für Benutzer bereitgestellt werden soll, deren Muttersprache nicht Englisch ist (Mencl 2004). In der Softwareentwicklung werden häufig eigene Restriktionen definiert, die auf die jeweiligen Bedürfnisse der Ansätze zugeschnitten sind, wie

beispielsweise in den bereits aufgeführten von Pérez-González und Kalita (2002a) oder Verma et al. (2011).

Neben diesen Argumenten wird davon ausgegangen, dass schriftliche Anforderungsspezifikationen mehrheitlich in natürlicher Sprache vorliegen. Aktuelle Zahlen stehen hierfür nicht zur Verfügung. Luisa et al. (2004) finden in einer Marktstudie zu diesem Thema, dass 79 % der Anforderungsspezifikationen in natürlicher Sprache erstellt werden. Da die Datengrundlage der Studie selbst aus dem Jahr 1999 stammt, muss der Wert als veraltet betrachtet werden. Allerdings liegen auch keine Hinweise darauf vor, dass sich seitdem eine Verlagerung hin zu strukturierten Anforderungsspezifikationen ergeben hat. Die daraus resultierende Annahme, dass Anforderungsspezifikationen häufiger in natürlicher Sprache vorliegen, führt zu der Entscheidung, mit diesem Lösungsansatz unrestriktierte, natürliche Sprache zu unterstützen.

5.2.2 Einbezug von Geschäftsprozessen

Generell ist es eine große Herausforderung in der Softwareentwicklung, Nachverfolgbarkeitsbeziehungen aufrechtzuerhalten, da jedes Artefakt seine eigene Repräsentation besitzt und jedes neue Artefakt mit einem speziellen Integrationsaufwand verbunden ist (Aizenbud-Reshef et al. 2006). Aus diesem Grund wäre eine umfangreiche Ausweitung der angebotenen Schnittstellen für die Zusammenarbeit der vorhandenen Ansätze wünschenswert, was aber aufgrund mangelnder Standards im Moment nur schwer zu erreichen ist. Gleichzeitig ist dies eine mögliche Erklärung dafür, warum im Bereich der Nachverfolgbarkeit zwar häufig betont wird, dass alle Arten von Artefakten einbezogen werden, diese Aussage dann allerdings in der Regel strikt auf Software-bezogene Artefakte wie UML-Diagramme und Code beschränkt bleibt. Seedorf et al. (2009) bemängeln diese fehlende Verknüpfung mit Geschäftsprozessen und liefern mit ihrem Ansatz (vgl. 4.2.2.2) den Grundgedanken, dem diese Arbeit folgt.

Geschäftsprozesse werden daher im Lösungsansatz der vorliegenden Arbeit explizit als ein Artefakt berücksichtigt. Für die Repräsentation von Geschäftsprozessen liegt eine Reihe an Notationen vor; einen Überblick gibt die Studie von Recker et al. (2009). Von den Autoren werden die Vor- und Nachteile der einzelnen Notationen untersucht, wobei BPMN[21] (Business Process Model and Notation) als die mit dem höchsten Grad an Vollständigkeit identifiziert wird. Allerdings wird damit auch ein Nachteil erkauft. So ist BPMN gleichzeitig die Notation mit der höchsten Redundanz, d. h., es steht, gemessen an einem Referenzmodell, im Vergleich zu den anderen untersuchten Notationen der höchste Grad an überlappenden Konstrukten zur Verfügung. Redundanz kann zu Konfusion beim Benutzer führen (Recker et al. 2009). Insgesamt überwiegen die Vorteile: BPMN ist ausdrucksstark, in der Praxis weit verbreitet und darüber hinaus auch ein Standard der Object Management Group. Damit ist BPMN als Notation für den Lösungsansatz in dieser Arbeit geeignet. Einen Überblick über den BMPN 2.0 Standard gibt Allweyer (2009).

In Anbetracht des großen Umfangs der Elemente von BPMN-Diagrammen kann hier nur ein kleiner Ausschnitt abgebildet werden. Es ist außerdem nicht Kernaufgabe der BPMN, die IT-Landschaft abzubilden (Freund und Rücker 2012). Vielmehr werden Prozesse aufgezeigt, und zwar unabhängig davon, ob diese von der IT gestützt werden oder nicht. Deshalb ist es der

[21] http://bpmn.org

Fokus des Lösungsansatzes, die Verbindung herzustellen, in dem Bewusstsein, dass die Vorschläge für BPMN-Diagramme sich nur auf die IT-gestützten Aktivitäten beziehen können. Alle weiteren Aktivitäten und Elemente müssen vom Benutzer modelliert werden.

5.2.3 Semantische Analyse während der Eingabe

Alle bisher existierenden Ansätze (vgl. 4.2.3) lesen als Grundlage der Analyse bereits vorhandene Anforderungsdokumente ein. Der Ablauf der Softwareentwicklung ist aber in der Regel dynamisch, d. h., Rückkopplungen mit vorherigen Entwicklungsschritten sind explizit vorgesehen (vgl. Abbildung 4). Während der Entwicklung eines Projekts kann immer wieder die Notwendigkeit auftreten, bereits erstellte Anforderungsspezifikationen anzupassen, z. B., wenn sich neue Anforderungen ergeben oder Unklarheiten beseitigt werden sollen. In den existierenden Ansätzen tritt damit die bereits angesprochene Trennung zwischen der Erstellung der schriftlichen Anforderungsspezifikationen und allen weiteren Artefakten auf. Diese Trennung wird durch den hier vorgestellten Lösungsansatz in der Art überwunden, dass Anforderungsspezifikationen direkt in einen Editor eingegeben und nicht aus Dokumenten ausgelesen werden. Während der Eingabe wird der Text fortlaufend analysiert, d. h., die Ergebnisse der Analyse werden direkt visualisiert. Damit entfällt der sonst notwendige Schritt, eine Analyse zu bestimmten Zeitpunkten durchzuführen, beispielsweise jeweils nach einer Bewertungsrunde oder einer neuen Version der Anforderungsdokumente. Die Ergebnisse der Analyse sind sofort erkennbar und – wenn der Benutzer dies wünscht – auch sofort anpassbar.

Aus den identifizierten Elementen sollen Vorschläge für Diagramme generiert werden. Es wird hier explizit von Vorschlägen gesprochen, da aufgrund der Komplexität natürlicher Sprache nicht zu erwarten ist, dass Diagramme umfassend korrekt und vollständig aus dem Text extrahierbar sind. Die extrahierten Informationen können nur so konsistent und so vollständig wie die Anforderungsspezifikation selbst sein. Außerdem sollen auch die Diagramme selbst modifizierbar sein, damit der Benutzer sie bei Bedarf anpassen kann. Er entscheidet selbst, ob er die Hervorhebungen im Editor ändert und damit die Grundlage für die Generierung anpasst oder das generierte Diagramm bearbeiten möchte. Dies sichert dem Benutzer die maximale Flexibilität zu.

Bei der Analyse selbst beschränken sich existierende Ansätze für die Identifikation von Elementen für Diagramme auf eher grundlegende semantische Technologien. Auf Basis des POS Taggings werden Elementkandidaten identifiziert, und diese Kandidatenliste wird anschließend anhand bestimmter Regeln eingeschränkt. Die Disambiguierung wird dabei nicht explizit erwähnt, wird aber als implizit enthalten betrachtet, da Werkzeuge wie Stanford CoreNLP oder NLTK sie bereits in die Vergabe von POS Tags integrieren. Alle darüber hinausgehenden Möglichkeiten der semantischen Technologien werden bisher nicht ausgeschöpft. Der Lösungsansatz soll daher auch semantische Analysetechniken einsetzen, die über das POS Tagging hinausgehen. Konkret sollen die Erkennung benannter Wortarten (vgl. 3.2.2.3), typisierte Beziehungen (vgl. 3.2.2.4) und die Koreferenzanalyse (vgl. 3.2.2.5) einbezogen werden. Wie diese Technologien genau eingesetzt werden, wird aufgrund der Notwendigkeit einer ausführlichen Beschreibung weiter unten konkretisiert.

Hier sei noch eine Implikation der Analyse während er Eingabe erwähnt. Dadurch, dass der Text nicht komplett, sondern nur bis zum aktuellen Stand der Eingabe vorliegt, sind Doku-

ment-bezogene Berechnungen nicht sinnvoll. Solche Berechnungen wurden bereits in Verbindung mit der Identifikation von Klassen (vgl. 4.2.3.1) vorgestellt. Dabei wird über den Ausschluss eines Klassenkandidaten anhand der Häufigkeit seines Auftretens in Verbindung mit einem bestimmten Schwellenwert entschieden. Da während der Eingabe immer auch neu hinzukommende Elemente beschrieben werden können, ist eine solche Vorgehensweise hier nicht vorgesehen.

5.2.4 Browser-native Oberfläche

Die Oberfläche einer Anwendung über einen Web-Browser anzubieten, bringt viele Vorteile mit sich. Thum (2011) beschreibt dazu die Eigenschaften verschiedener Plattformen. In Abgrenzung zu Internet-basierten Applikationen, die auch andere Protokolle verwenden, werden Plattformen, die ausschließlich über HTTP (Hyper Text Transfer Protocol) kommunizieren, als Web-basiert bezeichnet. Browser-basierte Plattformen bieten darüber hinaus eine Oberfläche an, die ausschließlich in einem Web-Browser residiert. Als Browser-nativ werden alle Plattformen bezeichnet, die Browser-basiert sind und keinerlei Erweiterungen wie Java oder Flash benötigen. Eine Browser-basierte Plattform benötigt damit weder eine Konfiguration des Netzwerks oder der Firewall noch die Installation weiterer Programme neben dem Browser. Browser-native Plattformen bieten noch mehr Vorteile, da keine Erweiterungen installiert oder gewartet werden müssen. Außerdem hinterlassen sie keinen Footprint, was in diesem Zusammenhang bedeutet, dass das System des Benutzers nicht modifiziert wird (Thum 2011). Aufgrund dieser Vorteile soll der Lösungsansatz eine Browser-native Oberfläche anbieten.

Diese Überlegung legt nahe, dass der Lösungsansatz dem klassischen Client-Server-Prinzip folgt. Alle Darstellungs-bezogenen Aufgaben wie die Bearbeitung von Anforderungsspezifikationen und Diagrammen werden dabei vom Client, also dem Web-Browser übernommen, während der Server die Analyse der Anforderungsspezifikationen sowie das Generieren der Diagramm-Vorschläge übernimmt. Zum Speichern der Daten wird eine Datenbank verwendet, deren Zugriff durch die Logik auf dem Server der Anwendung verwaltet wird. Diese Aufgabenverteilung des Lösungsansatzes wird in Abbildung 26 dargestellt.

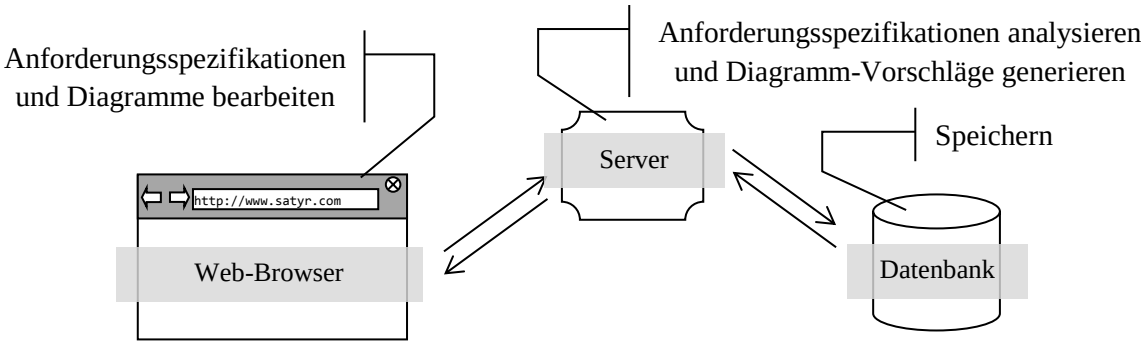


Abbildung 26: Aufgabenverteilung des Lösungsansatzes

Eine alternative Lösungsmöglichkeit wäre es, den Desktop als Plattform zu wählen. Dabei könnte die Software als eigenständige Anwendung oder als Plugin, beispielsweise für die bekannte Entwicklungsumgebung Eclipse, realisiert werden. Dies hätte den Vorteil, dass Aspekte hinsichtlich der Datensicherheit einfacher zu handhaben sind. Zumindest solange keine Server-Anbindung oder Kommunikation mit anderen Systemen über ein Netzwerk stattfindet,

wird keine Verschlüsselung der Übertragung benötigt. Wenn außerdem von einer Mehrbenutzer-Bedienung abgesehen wird, braucht keine Verwaltung von Zugriffsrechten implementiert zu werden. Dies erscheint allerdings nur selten praktikabel, da, wie bereits festgestellt, die Bedeutung von Kollaboration während der Softwareentwicklung durch viele existierende Arbeiten belegt wird (z. B. 4.2.2.1). Die Browser-native Plattform bietet dazu die ideale Grundlage: Jeder kann ohne besondere softwaretechnische Vorrausetzungen das System benutzen. Sind ein aktueller Web-Browser und ein Internetzugang vorhanden, werden keine weiteren Installationen oder Konfigurationen benötigt und damit die Eintrittsbarrieren beim Benutzer so niedrig wie möglich gehalten. Gleichzeitig sind der Aufwand und die Kosten für die Einrichtung des gesamten Systems überschaubar, da die Software allein über den Server bereitgestellt wird. Browser-basierte Plattformen sind haben allerdings auch einen Nachteil. Es gibt eine Vielzahl an verschiedenen Browsern, die trotz aller Bemühungen um die Standardisierung nicht immer die gleiche Darstellung garantieren. Der Benutzer beziehungsweise das Unternehmen hat die Kontrolle darüber, welcher Browser und welche Version verwendet werden; damit liegt diese Komponente außerhalb des Einflussbereichs des Systems. Eine möglichst hohe Kompatibilität mit vielen Browsern und Versionen zu erzielen, erfordert zusätzlichen Aufwand bei der Implementierung. Dieser Nachteil wird allerdings dadurch aufgewogen, dass sich der Aufwand bei der Wartung nur auf den Server als zentralen Punkt konzentriert.

5.2.5 Benutzerfreundlichkeit

Bereits in den letzten Abschnitten wurden Aspekte der Benutzerfreundlichkeit angesprochen. So trägt die Unterstützung natürlicher Sprache zur Verständlichkeit der Inhalte bei, während die Verwendung von Standardnotationen wie UML und BPMN ein schnelles Einarbeiten bei den verwendeten Diagrammen ermöglicht. Die Analyse während der Eingabe ist nach den Prinzipien der Übersichtlichkeit und Flexibilität gestaltet; gleichzeitig unterstützt die Browser-native Oberfläche die Kollaboration. Hier ergeben sich viele Überschneidungen mit dem Wiki-Konzept (vgl. 4.2.2.1), welches ebenfalls auf Flexibilität, einen einfachen Zugriff und einer Unterstützung der Kooperation abzielt. Anders als in einem Wiki soll aber keine Seitenbasierte Aufteilung der Inhalte stattfinden, sondern der Benutzer wird über Buttons geleitet.

Es stellt sich ebenfalls die Frage, inwieweit andere semantische Technologien wie die Eigenschaften semantischer Wikis und/oder von Ontologien in den Lösungsansatz einfließen können. Diese bieten, wie gezeigt (vgl. 4.2.1.2 & 4.2.1.3), viele Vorteile. Allerdings erfordern sie ein hohes Vorwissen beziehungsweise den Willen und die Kapazitäten des Benutzers, sich in diese Konzepte einzuarbeiten. Insbesondere sind alle Ansätze, die Ontologien involvieren, abhängig von der Qualität der eingesetzten Ontologie. So wird eine Ontologie, die relevante Konzepte nicht beinhaltet, kaum benutzt werden (Missikoff et al. 2002). Außerdem ist zwischen der Sparsamkeit und der Vollständigkeit abzuwägen, wozu Mädche und Staab (2001) anmerken: „Targeting completeness for the domain model appears to be practically unmanageable and computationally intractable, but targeting the scarcest model overly limits expressiveness“ (S. 77). Es muss also die Balance zwischen diesen beiden Zielen angestrebt werden. Für die meisten Nutzer würde dies eine Überforderung darstellen. Weiterhin wurde der Einsatz von Ontologien für die Nachverfolgbarkeit bereits in der Arbeit von Assawamekin beschrieben (vgl. 4.2.2.3).

Eine wichtige Erwartung an den Lösungsansatz ist, dass die Analyse der Anforderungsspezifikationen und die Generierung der Diagramme hinreichend schnell umgesetzt werden. Da beides, wie beschrieben, auf dem Server stattfindet, müssen die Funktionalitäten sowie die Kommunikation zwischen Client und Server diese Kriterien erfüllen. Des Weiteren sollten die Wartezeiten für den Benutzer unsichtbar bleiben. Das heißt, dass während der Eingabe – beispielsweise im Editor – nicht auf Antwort des Servers gewartet werden muss, sondern der Benutzer weiterarbeiten kann.

5.3 Architektur

Basierend auf den Anforderungen, wird nun die Architektur des Lösungsansatzes beschrieben. Dazu werden zunächst die verwendeten Begriffe voneinander abgegrenzt und anschließend der Aufbau der verschiedenen Module aufgezeigt. Dann wird der Ablauf der Arbeit mit dem System aus Benutzersicht veranschaulicht, bevor im letzten Unterabschnitt eine Abgrenzung des Lösungsansatzes vorgenommen wird.

5.3.1 Begriffe

Wie eingangs dieser Arbeit (vgl. 2.1.1) hergeleitet, müssen für die Umsetzung der Nachverfolgbarkeit alle Arten von Artefakten bidirektional und über alle Phasen des Softwarelebenszyklus hinweg miteinander verbunden werden können. Dazu werden gemäß den Anforderungen auch Geschäftsprozesse als eine Art von Artefakt einbezogen und damit die bisher auf die Produkte der Softwareentwicklung beschränkte Betrachtung erweitert. Um den Lösungsansatz konsistent beschreiben zu können, werden nun die weiteren Begriffe vorgestellt. Ein Benutzer ist eine Person, die mit der Anwendung interagiert. Für jedes neue Entwicklungsvorhaben wird durch den Benutzer ein neues Projekt angelegt, welches alle dazugehörigen Artefakte beinhaltet. Da der Lösungsansatz auf Kooperation angelegt ist, kann ein Benutzer ein Projekt für die gemeinsame Nutzung freigeben. Ein Projekt umfasst alle damit in Verbindung stehenden Artefakte. Eine Visualisierung der Begriffe findet sich in Abbildung 27.

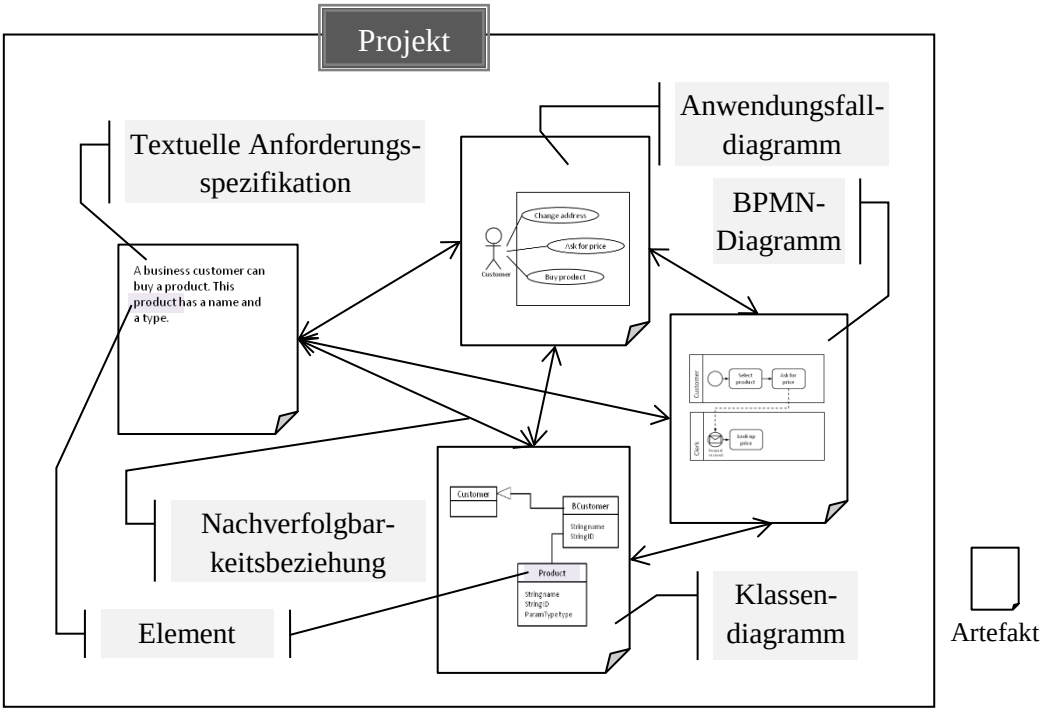


Abbildung 27: Überblick über die Begriffe

Wie oben bereits aufgeführt, werden als Artefakte schriftliche Anforderungsspezifikationen, UML-Anwendungsfalldiagramme, UML-Klassendiagramme und BPMN-Diagramme einbezogen. Die Artefakte sind über Nachverfolgbarkeitsbeziehungen miteinander verbunden. Die im Text durch die Analyse als relevant identifizierten Informationen werden als Elemente bezeichnet. Wenn der Benutzer dem Vorschlag für ein Element zustimmt, wird dieses für die Generierung der Diagramme verwendet, d. h., es findet sich in diesen wieder.

Für ein Projekt können beliebig viele Anforderungsspezifikationen erstellt werden. Bei der Generierung eines Diagramms aus einer Anforderungsspezifikation sollen bereits erstellte und damit eventuell bereits angepasste Diagramme des gleichen Typs nicht überschrieben werden. Deshalb können für jede schriftliche Anforderungsspezifikation wiederum beliebig viele Diagramme für jeden der verschiedenen Typen generiert werden. Dabei wird automatisch eine Nachverfolgbarkeitsbeziehung zwischen der Anforderungsspezifikation und dem generierten Diagramm angelegt. Da diese Beziehung bidirektional sein soll, erhält jedes Diagramm eine Referenz auf die Anforderungsspezifikation, aus der es generiert wurde. Auch zwischen den verschiedenen Diagrammen kann der Benutzer direkt zu den weiteren dieser Anforderungsspezifikation zugehörigen Diagrammen gelangen. Insgesamt ist es damit möglich, zu jeder Anforderungsspezifikation und jedem Diagramm jeweils eine Liste der zugehörigen Diagramme eines Typs zu erhalten. Von einem Diagramm aus kann direkt die zugehörige Anforderungsspezifikation erreicht werden.

5.3.2 Anwendungskonzept

Ein Benutzer, der mit dem System arbeiten möchte, benötigt zunächst ein Benutzerkonto. Ist dieses angelegt, kann ein neues Projekt erstellt werden. Für dieses Projekt sind auch Zusatzinformationen erfassbar, wie eine Projektbeschreibung und die Zugriffsrechte, die an andere Benutzer vergeben wurden. Ein Benutzer mit Zugriffsrechten kann Ergänzungen und Änderungen vornehmen, aber keine Löschungen. Zu jedem Projekt können mehrere schriftliche Anforderungsspezifikationen angelegt und einzeln bearbeitet werden. Während der Eingabe werden vom System Vorschläge für Elemente in der Form gemacht, dass Elemente für die verschiedenen Diagramme empfohlen werden. Diese Vorschläge können direkt im Editor angepasst werden. Falls die schriftliche Anforderungsspezifikation ausreichend beschrieben ist und keine weiteren Anpassungen mehr vorgenommen werden sollen, kann ein Diagramm generiert werden. Stellt der Benutzer fest, dass die Anforderungsspezifikation ergänzt oder verändert werden muss, dann kann ein direkter Zugriff auf diese erfolgen. Ist der Diagramm-Vorschlag ausreichend für die direkte Weiterbearbeitung des Diagramms, kann dies im Diagrammeditor erfolgen. Abbildung 28 veranschaulicht die wichtigsten Schritte bei der Arbeit mit dem System.

Nicht gezeigt werden die Schritte der Benutzerverwaltung, wie das Anlegen eines Benutzerkontos oder das Einloggen. Diese sollen erwartungsgemäß umgesetzt werden und liefen deshalb an dieser Stelle keinen Beitrag zum Verständnis des Anwendungskonzeptes.

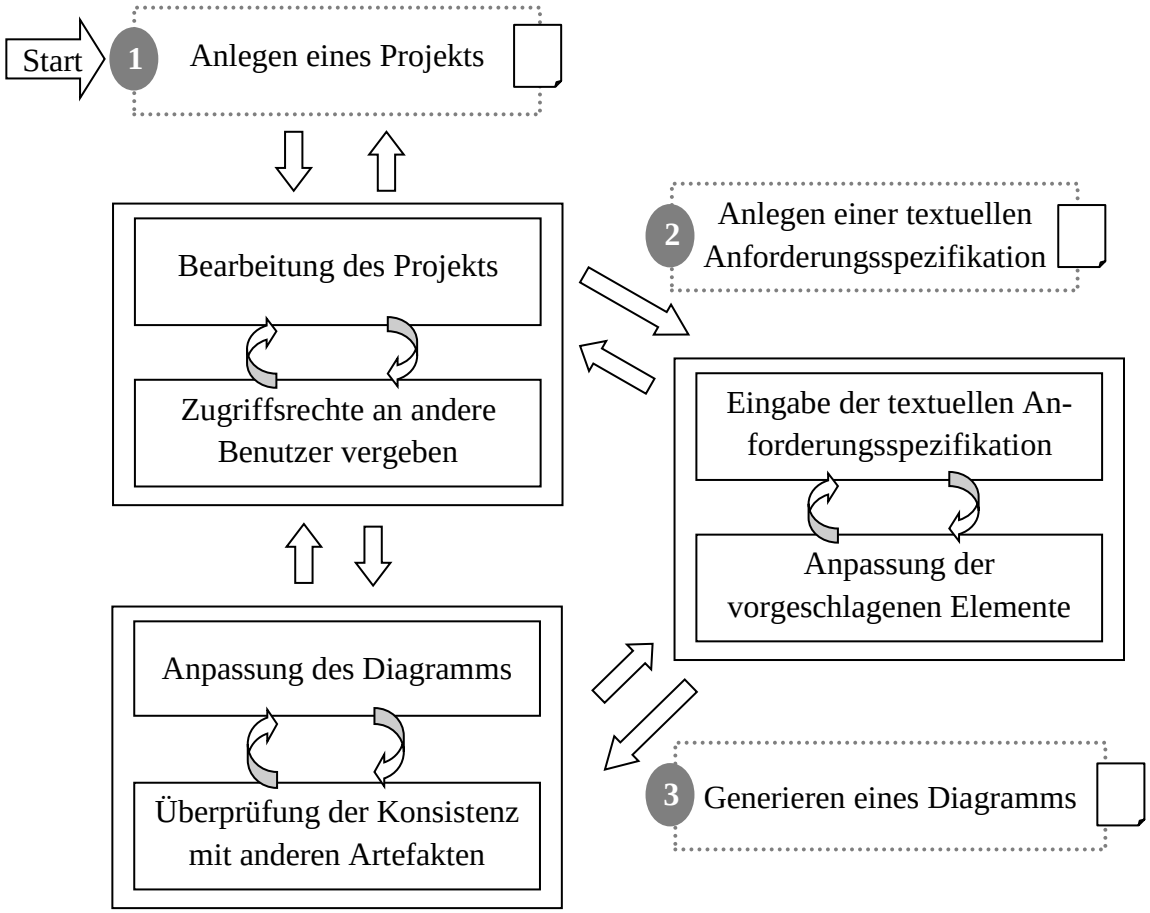


Abbildung 28: Ablauf der Bearbeitung aus Sicht des Benutzers

Der Benutzer kann jederzeit die Konsistenz der Artefakte untereinander prüfen. Alle Artefakte sind an einem zentralen Ort abrufbar und, wie eben beschrieben, durch Nachverfolgbarkeitsbeziehungen miteinander verbunden. Dadurch kann zwischen den Artefakten schnell navigiert werden und der Benutzer auf diese Weise bei der Suche nach Unstimmigkeiten unterstützt werden. Es gibt damit keine Medienbrüche mehr beispielsweise zwischen schriftlichen Anforderungsspezifikationen, die in Word-Dokumenten enthalten sind und mit Diagrammen abgeglichen werden sollen, die mit speziellen Werkzeug erstellt wurden. Auch die anderen Benutzer, die mit einem Projekt assoziiert sind, können bei Nachfragen direkt angesprochen werden.

Aus diesem auf den Benutzer ausgerichteten Ablauf ergeben sich die für die Seite der Analyse benötigten Funktionalitäten. Zum Ersten müssen dem Benutzer Vorschläge für Diagrammelemente unterbreitet werden. Diese Vorschläge sollen direkt im Eingabefeld der Anforderungsspezifikationen durch farbige Markierungen des Textes hervorgehoben werden. Zum Zweiten soll der Benutzer dann die Möglichkeit haben, diese farblichen Markierungen anzupassen. Auf Basis der farblichen Markierungen soll dann zum Dritten die Diagrammgenerierung stattfinden.

5.3.3 Module

Die Architektur des Lösungsansatzes ist modular aufgebaut. Dabei werden die verschiedenen Aufgaben durch die einzelnen Module bearbeitet. Das Visualisierungsmodul stellt alle Funktionalitäten bereit, die für die Darstellung im Web-Browser benötigt werden, während das Persistierungsmodul alle die Speicherung in der Datenbank betreffenden Aufgaben übernimmt. Alle Logik-bezogenen Aufgaben werden durch drei Module umgesetzt: das Analysemodul für die semantische Analyse der schriftlichen Anforderungsspezifikationen, das Generatormodul für die Generierung und Modifikation der Diagramme und das Nachverfolgbarkeitsmodul für die Verwaltung der Artefakte, Projekte und Benutzer.

Das Analysemodul unterteilt sich in zwei Unterbereiche, einen für den Basistagger und einen für die Spezialtagger der einzelnen Diagrammtypen. Der Basistagger extrahiert die Informationen aus dem Text und greift dabei auf die Technologien der semantischen Analyse zurück, die bereits in den Abschnitten 3.2.1.1 bis 3.2.2.5 vorgestellt wurden. Die Spezialtagger verarbeiten die Ergebnisse des Basistaggers in der Art weiter, dass sie die relevanten Elemente für den zugehörigen Diagrammtyp extrahieren. Der Benutzertagger basiert allein auf den Eingaben des Benutzers, er besitzt eine Sonderstellung. Die Analyse wird im folgenden Abschnitt konkret dargestellt werden.

Das Generatormodul beinhaltet als zweites Logik-Modul alle benötigten Funktionalitäten zur Generierung und Modifikation von Diagrammen. Bei der Generierung werden alle im Text markierten Elemente in ihr entsprechendes Gegenstück im Diagramm überführt, und das Ergebnis wird grafisch dargestellt. Markierte Elemente sind solche, die entweder vom Analysemodul vorgeschlagen werden oder, wie eben beschrieben, vom Benutzer selbst als entsprechende Elemente markiert werden. Ein generiertes Diagramm kann weiter modifiziert werden. Die Generierung der Diagramme wird ebenfalls im folgenden Abschnitt konkretisiert.

Das dritte Logik-bezogene Modul stellt die Nachverfolgbarkeit zwischen den Artefakten sicher. Das Nachverfolgbarkeitsmodul bietet alle Grundfunktionalitäten zum Anlegen und Löschen von Artefakten, Projekten und Benutzern. Dieses Modul stellt die bidirektionale Nachverfolgbarkeit sicher. Wie oben beschrieben, sind einem Projekt mehrere Anforderungsspezifikationen und diesen wiederum mehrere Diagramme zuordenbar, wobei die Nachverfolgbarkeitsbeziehung automatisch beim Anlegen beziehungsweise Generieren eines Artefakts erfolgt. Bei der Änderung innerhalb eines Artefakts bleiben die Nachverfolgbarkeitsbeziehungen unberührt. Eine Neuzuordnung der Artefakte untereinander ist nicht vorgesehen. Für den Fall, dass ein Artefakt gelöscht wird, bleiben die damit verbundenen Artefakte erhalten. Allerdings ist nun für den Fall, dass eine Anforderungsspezifikation gelöscht wurde, die Nachverfolgbarkeitsbeziehung erloschen, und der Benutzer wird auf das gesamte Projekt zurückverwiesen. Wird ein Projekt gelöscht, so werden alle damit verbundenen Artefakte ebenfalls gelöscht.

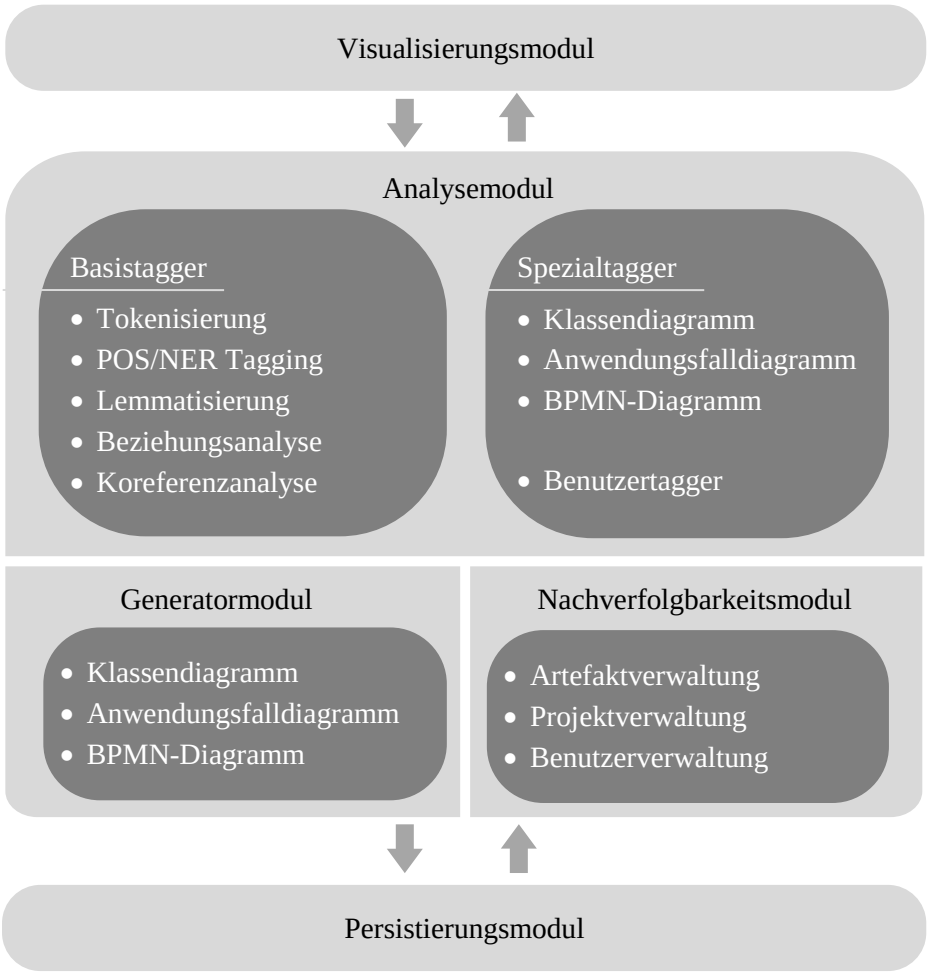


Abbildung 29: Module des Lösungsansatzes

Einen Überblick über alle Module bietet Abbildung 29. Falls die Notwendigkeit entsteht, die Visualisierung oder die Persistierung auszutauschen, so ist dies einfach durchführbar, da diese Module so weit wie möglich unabhängig von den anderen Modulen sind. Die weiteren Module, die die Logik des Lösungsansatzes implementieren, greifen dagegen ineinander und ziehen bei Änderungen notwendigerweise Anpassungen in den anderen Modulen nach sich.

5.4 Analyse

Die wesentliche Analyseeinheit des Lösungsansatzes ist der Text, der für die Anforderungsspezifikation eingegeben wird. Dieser Text kann mehrere Sätze umfassen, die wiederum aus mehreren Wörtern bestehen. Das Satzende wird durch einen Punkt markiert. Das Wort ist die kleinste Analyseeinheit und Hauptinformationsträger dieses Lösungskonzepts. Ein Wort speichert alle Informationen, mit denen es getaggt wird, wobei als Tags in diesem Fall der Einfachheit halber auch das Lemma sowie Referenzen auf andere Wörter oder Listen anderer Wörter bezeichnet werden. Abbildung 30 veranschaulicht diese Zusammenhänge.

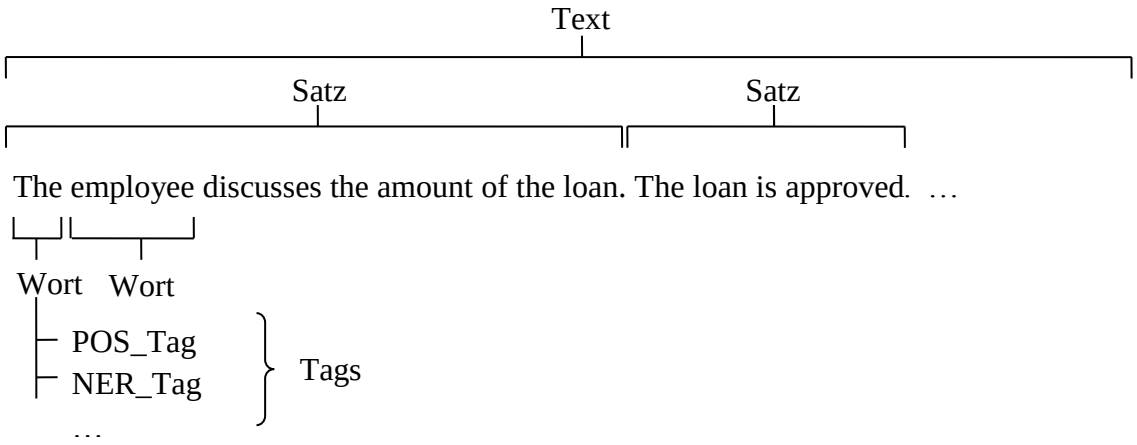


Abbildung 30: Analyseeinheiten des Lösungsansatzes

Es stellt sich nun die Frage, welche Tags ein Wort benötigt und wie aus diesen Grundinformationen im Anschluss die Generierung der Diagramme stattfindet. Dazu wird zunächst das Ziel der Analyse dargestellt.

5.4.1 Ziel

Aus dem jeweils vorliegenden Text sollen die Informationen extrahiert werden, die für die farblichen Markierungen verwendet werden können und damit als Basis für die Generierung der Diagramme dienen. Bereits in Abschnitt 4.2.3 wurde dazu vorgestellt, welche Elemente relevant sind und wie diese durch die existierenden Arbeiten ermittelt werden. Dabei werden Kandidaten für Elemente über Wortarten (POS Tags) – beispielsweise Substantive für eine Klasse – gefunden, und diese Kandidatenliste anschließend über Eliminationsregeln eingeschränkt. Wie herausgearbeitet wurde, sind diese Eliminationsregeln teilweise nur sehr schwer zu automatisieren. Daher erweitert der hier vorgestellte Lösungsansatz die bekannten Strategien in folgender Weise: Die aus der Analyse der Anforderungsspezifikation resultierenden Informationen zu einem Wort sollen um die Beziehungen der Wörter untereinander sowie deren syntaktische Bedeutung im Satz ergänzt werden. Dies kann über die Verwendung typisierter Beziehungen erreicht werden (vgl. 3.2.2.4).

Elemente, die für Klassendiagramme identifiziert werden sollen, sind die **Klassen**, deren **Attribute**, **Operationen** und **Beziehungen**. Als Kandidaten für Klassen werden die syntaktischen Subjekte und die syntaktischen Objekte des Satzes herangezogen. Weiterhin wird festgelegt, dass benannte Entitäten nicht als Kandidaten für Klassen infrage kommen (vgl. Regel K11), da davon ausgegangen wird, dass sie Instanziierungen von Klassen und nicht die dahinter stehenden Konzepte beschreiben. Dies kann über die Erkennung benannter Entitäten (vgl. 3.2.2.2) gelöst werden. Kandidaten für Attribute sind Adverbien oder andere Substantive (vgl. Regel K12), die das jeweilige Substantiv näher beschreiben. Kandidaten für Beziehungen sind transitive Verben, also solche, die mit mindestens zwei Klassenkandidaten in Beziehung stehen. Die Überschneidung zu Operationen wurde bereits erläutert. Eine mögliche Strategie ist, ein Wort gleichzeitig als Beziehung und als Operation zu markieren. Der Lösungsansatz soll aus Gründen der Übersichtlichkeit nur eine farbliche Markierung pro Wort vornehmen, sodass Kandidaten für Operationen die Verben sind, die nur mit weniger als zwei Klassenkandidaten in Beziehung stehen.

Die für Anwendungsfalldiagramme zu identifizierenden Elemente sind **Akteure** und **Anwendungsfälle**. Die existierenden Ansätze gehen für den Akteur zunächst von allen Substantiven aus. Akteure sind aber aktiv Handelnde, sodass in diesem Lösungsansatz nur Substantive ausgewählt werden, die auch syntaktische Subjekte eines Satzes beziehungsweise Teilsatzes sind. Benannte Entitäten werden beibehalten, wenn sie vom Typ Organisation sind. Beispielsweise ist die IRS (Internal Revenue Service) als Steuerbehörde der USA durchaus ein für das Anwendungsfalldiagramm interessanter Akteur. Kandidaten für Anwendungsfälle sind Verben, die mit den als Akteuren identifizierten Substantiven in Verbindung stehen.

Für BPMN-Diagramme sollen Elemente für **Aktivitäten** und sogenannte **Swimlanes** (dt. Schwimmbahnen) identifiziert werden. Eine Swimlane beantwortet die Frage danach, wer für die jeweiligen Aktivitäten zuständig ist (Freund und Rücker 2012). BPMN-Diagramme basieren wie Anwendungsfalldiagramme auch auf Aktivitäten und Handelnden. Der Unterschied zwischen den beiden Diagramm-Typen besteht darin, dass BPMN-Diagramme die Abbildung der zeitlich-logischen Abfolge erlauben. Aktivitäten können damit analog zu Anwendungsfällen, Swimlanes analog zu Akteuren extrahiert werden. Zusätzlich werden die Verbindungen des Sequenzflusses im Diagramm-Vorschlag aufgenommen. Dazu wird die Abfolge im Text übernommen, und die Aktivitäten werden entsprechend miteinander verbunden. Es lassen sich nun die benötigten Beziehungen von Wörtern untereinander sowie die relevanten Informationen ableiten, die aus dem Text extrahiert werden müssen. Dazu wird im Folgenden das Konzept dargestellt.

5.4.2 Konzept

Für die Identifikation von syntaktischen Subjekten und Objekten liegen bereits Algorithmen vor (vgl. 3.2.2.4). Für diesen Lösungsansatz wird darüber hinaus das Teilkonzept des Hauptverbs benötigt. Hauptverben sind Verben, die keine Hilfsverben sind. In der Konstruktion „is approved“ ist „is“ das Hilfsverb. Des Weiteren werden als Teilkonzept des Adjektivs alle Wörter verstanden, die mit einem Substantiv in Beziehung stehen. Das sind zum einen Adjektive in der Originalbedeutung. Zwar sind diese nicht immer direkt übertragbar, wie im bereits vorgestellten Beispiel „the red car“; sie dienen dem Benutzer aber als Hinweis, ein entsprechendes Attribut „color“ zu definieren. Zum anderen können Substantive auch mit anderen Substantiven in Beziehung stehen (vgl. 4.2.3.2). So kann in „the customer name“, „the customer’s name“ oder „the name of the customer“ das Substantiv „name“ jeweils dem Substantiv „customer“ zugeordnet werden. Ein Substantiv, das einem anderen Substantiv zugeordnet werden kann, wird damit wie ein Adjektiv behandelt.

Zusammengefasst werden damit die folgenden Teilkonzepte benötigt: syntaktische Subjekte, syntaktische Objekte, Adjektive (in angepasster Form) und Hauptverben. Diese Zusammenhänge können durch die Analyse für typisierte Beziehungen erkannt werden (vgl. 3.2.2.4). Das übergeordnete Konzept setzt diese Teilkonzepte miteinander in Beziehung. Jeder korrekt formulierte Satz besitzt ein Hauptverb (VERB), weshalb es zentral für das Konzept ist. Falls keine Passivkonstruktion vorliegt, besitzt das Hauptverb ein syntaktisches Subjekt (SUBJ). Weiterhin kann ein Hauptverb mehrere syntaktische Objekte (OBJ) besitzen. Jedes syntaktische Subjekt und jedes syntaktische Objekt kann mehrere Adjektive (ADJ) besitzen. Das sich dadurch ergebende Beziehungskonzept wird in Abbildung 31 dargestellt.

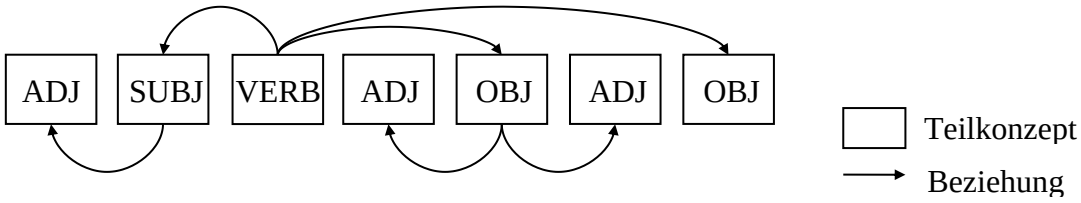


Abbildung 31: Beziehungskonzept

Aus Vereinfachungsgründen wird in Abbildung 31 nicht die Möglichkeit gezeigt, dass mehrere Hauptverben und/oder syntaktische Subjekte auftreten. Dies kann durch Konjunktionen entstehen. So treten im Beispielsatz „The loan is approved or rejected.“ mehrere Hauptverben auf. In diesem Fall werden die Beziehungen für die weiteren Hauptverben analog angelegt. Für den Fall einer Konjunktion zwischen syntaktischen Subjekten, wie in „The customer and the employee discuss the loan.“, werden alle zusätzlichen syntaktischen Subjekte ebenfalls dem Hauptverb zugeordnet. Eine weitere Möglichkeit für das Auftreten mehrerer Hauptverben und/oder syntaktischer Subjekte in einem Satz ist, dass der Satz aus mehreren Teilsätzen besteht. Für den Satz „The office informs the employee, he calls the customer.“ werden mehrere Hauptverben gespeichert. Die Erstellung der weiteren Beziehungen ist problemlos möglich, da Beziehungen immer zwischen zwei Wörtern untersucht werden.

Die auf Basis dieses Konzepts für ein Wort erstellten Tags können zunächst in farbliche Markierungen für die Diagrammelemente umgewandelt werden. Wie bereits in den Zielen angesprochen, werden syntaktische Subjekte für Klassen, Akteure und Swimlanes vorgeschlagen. Syntaktische Objekte werden ebenfalls als Klassen vorgeschlagen. Hauptverben werden als Anwendungsfall und Aktivität vorgeschlagen. Für den Fall, dass ein Hauptverb mindestens ein syntaktisches Subjekt und mindestens ein syntaktisches Objekt besitzt, wird es als transitiv angesehen und für das Klassendiagramm als Beziehung vorgeschlagen, anderenfalls als Operation. Dieses Vorgehen wird in Abbildung 32 anhand eines Beispiels gezeigt.

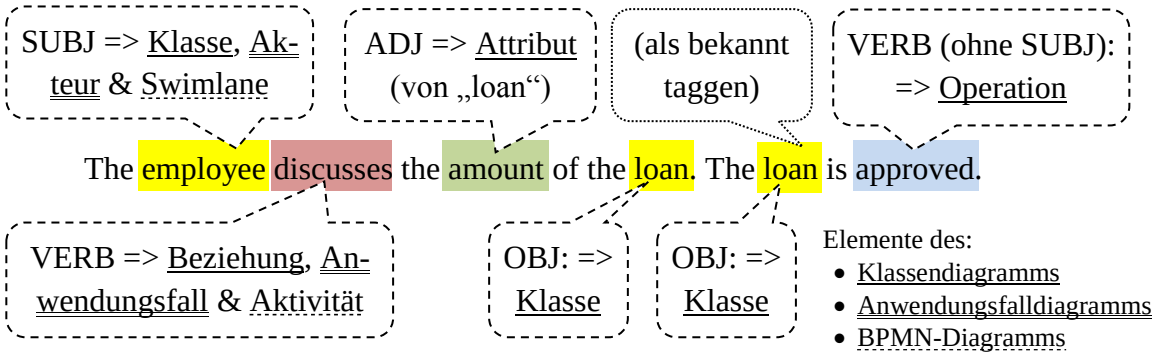


Abbildung 32: Beispiel für aus einem Satz extrahierbare Informationen

Das Beispiel in Abbildung 32 beinhaltet außerdem den Fall, dass über Satzgrenzen hinweg Beziehungen zwischen zwei Wörtern bestehen, wie für das Wort „loan“. Um diese Beziehung aufzudecken, wird die Koreferenzanalyse eingesetzt (vgl. 3.2.2.5). Diese identifiziert zusammengehörige Substantive und erlaubt es auf diese Weise, die entsprechenden Diagrammelemente zusammenzuführen.

Nachdem damit das Konzept für die Identifikation von Elementen für die Diagramme beschrieben ist, wird im Folgenden ein Überblick über den Ablauf gegeben, der von der Eingabe der Anforderungsspezifikation bis hin zur Generierung der Diagramme führt.

5.4.3 Ablauf

Das eben vorgestellte Konzept wird nun in den gesamten Ablauf der Analyse eingeordnet, und die weiteren Teilschritte bis zur Generierung der Diagramme werden vorgestellt. Gemeinsam mit den grundlegenden semantischen Analysen und der Koreferenzanalyse liefert das Konzept die Ausgangsbasis für die farblichen Markierungen. Deshalb werden diese Teilschritte vom Basistagger implementiert. Die Umsetzung der Teilkonzepte in die farblichen Markierungen ist vom jeweiligen Diagrammtyp abhängig und wird von den jeweiligen Spezialtaggern bereitgestellt. Es gibt einen dedizierten Spezialtagger für jeden Diagrammtyp. Darüber hinaus behandelt der Benutzertagger die Anpassungen, die der Benutzer an den Markierungen vornimmt. Mit jedem Schritt der Analyse werden einem Wort weitere Tags hinzugefügt; auf Basis aller Tags werden dann die Diagramme generiert. Der Ablauf gestaltet sich insgesamt in folgender Weise:

1. Basistagger: Vorbereitung des Textes
    a. Tokenisierung
    b. Vergabe von (Standard-)Tags für POS, NER und Lemma
    c. Vergabe von Tags auf Basis der Koreferenzanalyse
    d. Vergabe selbstdefinierter Tags auf Basis typisierter Beziehungen
2. Spezialtagger:
    a. Vergabe Diagramm-spezifischer Tags für die Diagrammelemente
    b. Anzeigen der farblichen Markierungen für die Diagrammelemente
    c. (optional:) Vergabe von Tags durch den Benutzer (falls ja: zurück zu 2.a)
3. Generieren: Erstellen eines Diagramms auf Basis aller vorhandenen Tags

Die Tokenisierung ist die Voraussetzung für alle weiteren Teilschritte und steht deshalb am Anfang der Analyse. Es schließt sich die Vergabe der Tags für POS, NER und Lemma an. Die Teilschritte 1.a und 1.b sollen durch das verwendete Werkzeug übernommen werden. Auch die Koreferenzanalyse (1.c) wird – obwohl eine vergleichbar neue Funktionalität – bereits von mehreren Werkzeugen implementiert und soll durch dieses vorgenommen werden. Funktionalitäten für die Analyse typisierter Beziehungen (vgl. 3.2.2.4) werden unter den betrachteten Werkzeugen bisher allein durch das Stanford CoreNLP implementiert. An dieser Stelle verliert der Lösungsansatz die Unabhängigkeit vom eingesetzten Werkzeug. Die Implementierung des Stanford CoreNLP stützt sich auf die Arbeit von Marneffe und Manning (2008) sowie Marneffe et al. (2006) und ist damit theoretisch fundiert.

Noch nicht erläutert wurden das Vorgehen bei der Vergabe von Tags durch den Benutzer sowie die Generierung der Diagramme. Bevor ein Diagramm generiert wird, kann der Benutzer die Vorschläge für die Elemente anpassen. Diese Anpassung hat nicht nur Auswirkungen auf die farbliche Markierung des angezeigten Textes, sondern es wird auch ein erneuter Analysedurchlauf für den entsprechenden Spezialtagger gestartet. Beispielsweise möchte der Benutzer für den Satz „The customer calls the bank.“ das Wort „bank“ nicht als Klasse übernehmen. Damit kann „calls“ nicht mehr in eine Beziehung im Klassendiagramm übersetzt werden und

wird deshalb als Operation von „customer“ angepasst. Dies wird dadurch ermöglicht, dass die Beziehung zwischen „customer“ und „calls“ weiterhin verwendet werden kann. Das generelle Vorgehen ist hier, dass in der Beziehungskette zurückgegangen und ein Ersatzelement gesucht wird. Dazu werden die Listen syntaktischer Subjekte und Objekte der Hauptverben verwendet. Wird ein Subjekt benötigt, so wird in der Liste der Subjekte nach einem weiteren gesucht; ist sie leer, so wird das erste Objekt verwendet, falls es eines gibt. Analog wird für ein Objekt zunächst in der Liste der Objekte nachgesehen und im Misserfolgsfall ersatzweise ein Subjekt verwendet, falls vorhanden. Adverbien gehen außerdem einen zusätzlichen Schritt zu ihrem Hauptverb zurück. Falls keine Zuordnung mehr stattfinden kann, wird das letzte im Text gefundene Element verwendet. Damit würde ein Attribut dem letzten gefundenen Objekt oder Subjekt zugeordnet, das als Klasse markiert ist. Dieses Vorgehen verhindert, dass Informationen verloren gehen. Der Benutzer kann dann in der Diagrammanpassung entscheiden, wie er mit diesen Elementen weiter verfahren möchte. Damit der Benutzer die Möglichkeit hat, seine Entscheidung rückgängig zu machen, werden die Tags des Basistaggers nicht überschrieben. Stattdessen werden zusätzliche Tags erstellt, die die Entscheidungen des Benutzers speichern.

Die Generatoren verwenden alle vorhandenen Tags, wobei, wie eben beschrieben, die Tags des Benutzers Vorrang haben. Um dem Benutzer ein ansprechendes Ergebnis zu liefern, wird für jedes Wort das Lemma verwendet. Damit wird aus dem eben vorgestellten Textausschnitt die Operation „call()“. Den Konventionen entsprechend, wird der erste Buchstabe von Klassen, Akteuren und Swimlanes großgeschrieben, jedes weitere Element klein. Beziehungen in Klassendiagrammen mit dem Lemma „be“ werden in „is-a“ umgewandelt und als Vererbungsbeziehung markiert. Analog werden Beziehungen mit dem Lemma „have“ in „has-a“ umgewandelt und als Aggregation markiert. Um Anwendungsfälle ausführlicher als nur mit dem Hauptverb zu beschreiben, werden direkte Objekte zusätzlich markiert. Für das bekannte Beispiel „The customer calls the bank.“ wird damit der Anwendungsfall „call bank“ erstellt. Die genaue algorithmische Umsetzung wird im folgenden Kapitel 6 beschrieben werden.

Der Lösungsansatz ermöglicht es somit, fortgeschrittene semantische Technologien für die Analyse von Anforderungsspezifikationen anzuwenden. Das zugrunde liegende Beziehungskonzept wurde bisher nicht für die Analyse von Anforderungsspezifikationen eingesetzt und stellt damit einen wesentlichen Beitrag der vorliegenden Arbeit dar. Der Benutzer wird auf intelligente Weise bei der Informationsextraktion unterstützt, indem Elemente für die Diagrammerstellung vorgeschlagen werden. Die Limitationen des Lösungsansatzes werden im Folgenden aufgezeigt.

5.5 Limitationen des Lösungsansatzes

Für die ausgewählten Artefakte steht nur eine Untermenge des tatsächlichen Umfangs der von den unterschiedlichen Diagrammen angebotenen Elemente bereit. Der Lösungsansatz beschränkt sich auf die wichtigsten Elemente. Es ist möglich, weitere Elemente zu ergänzen; der bereitgestellte Umfang ist aber ausreichend für die Veranschaulichung des Konzeptes. Weiterhin wird auch für die einbezogenen Artefakte selbst eine Auswahl getroffen. Diese beschränkt sich auf die schriftlichen Anforderungsspezifikationen, Klassen-, Anwendungsfall- und BPMN-Diagramme. Eine Ergänzung um weitere Diagrammtypen ist durch das Hinzufügen eines entsprechenden Spezialtaggers problemlos möglich.

Wünschenswert wäre darüber hinaus die Verbindung mit Quellcode. Zum einen könnte die Anwendung dazu mit bekannten Desktop-Werkzeugen wie Eclipse verknüpft werden. Zum anderen wird seit Kurzem die Idee vorangetrieben, den Web-Browser als Entwicklungsumgebung zu verwenden. Für die Programmiersprache Python existiert eine Web-Seite[22], welche einen entsprechenden Service anbietet. Für den produktiven Einsatz ist die Verwendung kostenpflichtig. Da in der Zukunft durchaus zu erwarten ist, dass immer mehr Funktionalitäten vom Desktop weg und hin zu Browser-basierten Oberflächen verlagert oder zumindest zusätzlich angeboten werden, ist eine entsprechende Erweiterung des Lösungsansatzes um diese Funktionalität denkbar.

Änderungen in den Anforderungsspezifikationen können durch ein erneutes Generieren der Diagramme problemlos in die verschiedenen Diagramm-Typen übertragen werden. Im Sinne der bidirektionalen Nachverfolgbarkeit ist einem Diagrammartefakt die entsprechende Anforderungsspezifikation zuordenbar. Allerdings ist es nicht möglich, im Diagramm vorgenommene Änderungen automatisch in die Anforderungsspezifikation zurückzuübertragen. In der Softwareentwicklung wird die Möglichkeit, Änderungen automatisch in beide Richtungen zu propagieren, häufig als Round-Trip-Engineering bezeichnet. Existierende Ansätze befassen sich mit der Verarbeitung von Änderungen für Klassendiagramme und Code (beispielsweise Kleuker 2009). Für den vorliegenden Lösungsansatz ist es nicht sinnvoll, Änderungen in Diagrammen zurück auf die Anforderungsspezifikation zu übertragen, da beim Hinzufügen von Elementen neuer Text erstellt werden müsste. Bei der Löschung oder Umbenennung von Elementen müsste Text abgeändert werden. Dafür müssten entsprechende Strategien und Mechanismen vorhanden sein, damit die Konsistenz der Artefakte untereinander gewahrt bleibt. Dies liegt außerhalb des betrachteten Bereichs des Lösungsansatzes, sodass keine Propagation von Änderungen in den Diagrammen zurück zu der Anforderungsspezifikation erfolgt.

Der Lösungsansatz betrachtet als kleinste Analyseeinheit genau ein Wort. Aus diesem Grund werden zusammengesetzte Substantive im Englischen nicht als solche erkannt. Beispielsweise wird für „the interest rate of the loan“ das Wort „interest rate“ für diesen Zusammenhang als zwei Wörter erkannt. Wie am Beispiel „the customer name“ gezeigt, geht der Lösungsansatz hier davon aus, dass das zweite Substantiv das erste genauer beschreibt. Weiterhin wird von der Koreferenzanalyse für solche Satzkonstruktionen die Zusammengehörigkeit nicht mehr erkannt. Für „interest“ beziehungsweise „customer“ wird keine Beziehung zu weiterem Auftreten in Text hergestellt, da die Koreferenzanalyse das zweite Substantiv als ausschlaggebend betrachtet. Es handelt sich hierbei um eine Problematik, die sich aus der Komplexität natürlicher Sprache ergibt.

Für die Bearbeitung ist es wünschenswert, dass mehrere Benutzer ein und dasselbe Artefakt zur gleichen Zeit bearbeiten können. Die Umsetzung dieser Funktionalität liegt allerdings nicht in der Zielsetzung der vorliegenden Arbeit. Ein Lösungsansatz für die synchrone Bearbeitung von Web-Seiten ist konzeptuell bereits beschrieben und technisch umgesetzt worden (Thum et al. 2009; Thum 2011). Eine Integration des hier entworfenen Lösungsansatzes mit dieser Technologie ist durch die Eigenschaft der Browser-Nativität eine mögliche Ergänzung.

[22] www.pythonanywhere.com

5.6 Zusammenfassung

Der hier vorgestellte Lösungsansatz überwindet die Lücke zwischen Anforderungsspezifikationen und den weiteren Artefakten der Softwareentwicklung, indem er die Nachverfolgbarkeit nicht nur bereitstellt, sondern durch fortgeschrittene semantische Technologien unterstützt. Auf Basis des vorgestellten Beziehungskonzepts werden direkt in der Anforderungsspezifikation mögliche Diagrammelemente identifiziert und die anschließende Generierung von Vorschlägen von Diagrammen unterstützt. Das Beziehungskonzept ist ein wesentlicher Beitrag dieser Arbeit; bisher existieren keine vergleichbaren Ansätze.

Eine der besonderen Herausforderungen des Lösungsansatzes ist die Bearbeitung natürlicher Sprache. Gleichzeitig ist dies auch ein wichtiges Unterscheidungsmerkmal gegenüber anderen Ansätzen. Da die Analyse direkt während der Eingabe und nicht erst nach Vollendung aller Dokumente erfolgt, bekommt der Benutzer ein direktes Feedback zu seinen Eingaben und kann die Anforderungsspezifikationen sofort anpassen. Dank der implementierten Nachverfolgbarkeitsfunktionalitäten bietet der Lösungsansatz dem Benutzer die Möglichkeit, schriftlichen Anforderungsspezifikationen und die zugehörigen Diagramme im direkten Zusammenhang zu betrachten. Damit kann unproblematisch und ohne Verzögerungen die Konsistenz der einzelnen Artefakte untereinander überprüft werden.

Insgesamt sind die Voraussetzungen zur Arbeit mit dem Lösungsansatz so niedrig wie möglich gehalten. Die Benutzer müssen lediglich über Grundkenntnisse im Bereich der Softwaremodellierung sowie der verwendeten Diagramm-Notationen verfügen.

6 Implementierung

Die Anforderungen an die Implementierung wurden im vorangegangenen Kapitel dargelegt; in diesem Kapitel wird die technische Umsetzung des Lösungsansatzes beschrieben. Es wird dabei auf die verwendeten Technologien sowie die Besonderheiten der Umsetzung eingegangen.

Die Implementierung des Lösungsansatzes wurde im Rahmen eines Teamprojekts an der Universität Mannheim durchgeführt. Diese Veranstaltung wird für Studierende im Studiengang Master Wirtschaftsinformatik angeboten und erstreckt sich über zwei Semester. Das vorliegende Teamprojekt begann im Oktober 2012 und wurde von der Autorin dieser Arbeit fachlich betreut und angeleitet. Die Ziele der Implementierung sowie die Auswahl hinsichtlich der verwendeten Technologien oblagen den Entscheidungen der Betreuerin. Die Implementierung wurde durch die Studierenden Artur Gareis, Etienne Dilocker, Rebecca Diedrich, Torben Möller und Vasil Indzhov vorgenommen.

Die Software liegt in Form eines Prototyps vor, der von den Studierenden als Urheber der Software unter die GNU General Public License (GPL) 3.0[23] gestellt wurde. Diese Lizenz erlaubt eine Weiterentwicklung der Software. Nicht Teil dieser Abgabe war die mxGraph Bibliothek für die Visualisierung der Diagramme, für die separat eine akademische Lizenz vom Anbieter[24] bereitgestellt wurde. Im Rahmen der vorliegenden Arbeit wurde der Algorithmus für die Analyse implementiert. In der Originalversion war dieser ohne das Beziehungskonzept umgesetzt. Weiterhin wurde die Oberfläche an wenigen Stellen zur Verbesserung der Benutzerfreundlichkeit angepasst.

Dieses Kapitel stellt zunächst die verwendeten Softwarepakete als technische Grundlage der Implementierung vor. Anschließend wird die Umsetzung erläutert, bevor auf die Verwendung und die Limitationen der Implementierung eingegangen wird.

6.1 Softwarepakete

Wie in den Anforderungen (vgl. 5.2) festgelegt, soll der Lösungsansatz in Form einer Client-Server-Anwendung umgesetzt werden. Die Wahl des Frameworks und der weiteren relevanten Komponenten wird in den folgenden Abschnitten kurz vorgestellt.

6.1.1 Framework

Das Google Web Toolkit (GWT) ermöglicht es, die Implementierung von Server und Client in einer Programmiersprache – Java – vorzunehmen (Sowa et al. 2008). Der Client-seitige Code wird dann vom Java-zu-JavaScript-Compiler, dem Herzstück von GWT, für die Anzeige im Web-Browser in JavaScript übersetzt. Die Kommunikation zwischen Client und Server erfolgt über einen Remote Procedure Call (RPC), der dem Client erlaubt, eine bestimmte Methode des Servers aufzurufen (Gupta 2008). Dies erinnert an das bekannte Remote Method Invocation (RMI)-Prinzip, wobei RPC-Aufrufe einen entscheidenden Vorteil besitzen: Sie

[23] http://www.gnu.org/copyleft/gpl.html
[24] http://www.jgraph.com/mxgraph.html

können asynchron erfolgen, d. h., dass der Client nicht auf die Antwort des Servers warten muss und der Benutzer weiterarbeiten kann. Ermöglicht wird dies durch den die AJAX (Asynchronous JavaScript and XML)-Technologie. Diese erlaubt das dynamische Laden einzelner Objekte innerhalb einer Webseite, wobei der Client – in diesem Fall ein Browser – einen Teil der Verarbeitung übernimmt (Woychowsky 2006).

Eine weitere Besonderheit von GWT ist das sogenannte Deferred Binding. Dieses ermöglicht die Verwendung der erstellten Applikation in allen Web-Browsern. Da jeder Web-Browser JavaScript anders umsetzt und JavaScript selbst nicht über ausreichende Algorithmen zur Optimierung verfügt, wird in GWT die Optimierung während der Kompilierung durchgeführt (Gupta 2008). Dies bedeutet, dass der Compiler eine eigene Version der Applikation für den jeweiligen Browser erstellt. Es wird für die Implementierung die GWT-Version 2.5.1 eingesetzt, die laut Dokumentation die Browser Firefox, Internet Explorer, Safari, Chromium und Google Chrome sowie Opera unterstützt. In der Umsetzung hat sich allerdings gezeigt, dass die angestrebte einheitliche Umsetzung für alle Browser nicht vollständig funktioniert. Aufgrund der Eigenheiten einzelner Web-Browser – insbesondere des Internet Explorers – treten immer wieder Abweichungen zwischen den einzelnen Versionen auf. Teilweise sind diese Abweichungen sehr klein und betreffen nur die Darstellungen; bei komplexeren Applikationen kann unter Umständen die Funktionalität beeinträchtigt sein. Daher wurde Firefox als Referenz-Browser festgelegt. Ein weiterer Vorteil von GWT ist die Unterstützung durch die bewährte Entwicklungsumgebung Eclipse[25]. Für Eclipse und für Firefox wurden die aktuellen GWT-Plugins verwendet, die die Entwicklung entscheidend vereinfachen. Der Code kann auf diese Weise direkt im Browser getestet werden

6.1.2 Datenbank und Server

Für die Speicherung der Daten wird MySQL[26] verwendet. MySQL ist ein relationales Datenbankverwaltungssystem und zeichnet sich durch Stabilität und Skalierbarkeit aus. Unter anderem wird MySQL für YouTube verwendet, was diese Eigenschaften eindrucksvoll belegt. Der Zugriff auf die Datenbank in Java erfolgt über die JDBC (Java Database Connectivity)[27]-Schnittstelle. JDBC gilt als Standard für den Zugriff auf relationale Datenbanken mittels SQL (Structured Query Language). Dank dieser Abstraktionsschicht kann die Datenbank ausgetauscht werden, ohne dass der Code angepasst werden muss. In GWT wird dazu lediglich eine Anpassung in der Konfigurationsdatei benötigt, sodass ein Austausch der Datenbank unkompliziert möglich ist.

Zur Bereitstellung der Software wird Tomcat[28] eingesetzt. Tomcat ist einer der weitverbreitetsten Web-Server; im Jahr 2010 erreichte er einen Marktanteil von rund 57 %, gemessen an der Anzahl der Domains (Rodewig 2011). Auch sehr große und kommerzielle Webseiten werden mit Tomcat betrieben, sodass die Technologie als etabliert betrachtet werden kann. Ein weiterer Vorteil von Tomcat ist, dass jede Instanz mit genau einer Java Virtual Machine (JVM) assoziiert ist, sodass im Falle eines Ausfalls des Tomcats und/oder der JVM alle weiteren Applikationen geschützt sind, die auf demselben physikalischen Server laufen. Um die

[25] http://www.eclipse.org
[26] http://dev.mysql.com
[27] http://www.oracle.com/technetwork/java/javase/jdbc/index.html
[28] http://tomcat.apache.org/

Software auf dem Server bereitzustellen, werden der Code sowie alle benötigten Ressourcen in eine WAR (Web Application Archive)-Datei gepackt. Diese Datei kann auch auf anderen Web-Servern genutzt werden, sodass wie bei der Datenbank auch hier die Austauschbarkeit gewährleistet ist.

6.1.3 Externe Ressourcen

Die semantischen Funktionalitäten des Prototyps werden durch Stanford CoreNLP unterstützt. Bereits in Abschnitt 3.2.3 wurden dessen Vorteile als Werkzeug mit dem größten Funktionsumfang dargelegt, welches auch die benötige Funktionalität (vgl. 5.4.3) für typisierte Beziehungen beinhaltet. Es wird das Tagset der ebenfalls vorgestellten Penn Treebank (vgl. 3.2.2.1) verwendet. Die Einbindung findet über eine JAR (Java Archive)-Datei statt. Für die Diagrammerstellung wird mxGraph[29] eingebunden. Diese JavaScript-Bibliothek unterstützt die Erstellung von Diagrammen, indem sie gebräuchliche Funktionalitäten für das Zeichnen und die Interaktion bereitstellt. Konzeptuell basiert die Bibliothek auf der mathematischen Graphen- und Netzwerktheorie, also den Verbindungen zwischen Knoten und Kanten. Die Einbindung erfolgt als einzelne JavaScript-Datei in der HTML (Hypertext Markup Language)-Seite. Es werden keine weiteren Plugins o. Ä. benötigt, sodass die Anforderung der Browsernativen Oberfläche erfüllt werden kann.

6.1.4 Übersicht

Nachdem die einzelnen Softwarepakete vorgestellt wurden, soll an dieser Stelle eine kurze Übersicht bereitgestellt werden. Diese zeigt die getesteten Versionen sowie die benötigen Voraussetzungen (siehe Tabelle 4). Die verschiedenen Komponenten benötigen unterschiedliche Java-Versionen. Da GWT auf der aktuellen Version 1.7 aufbaut, wurde diese ebenfalls als Basis für alle weiteren Komponenten verwendet. Zu beachten ist, dass auch für den Server eine entsprechende JRE (Java Runtime Environment) bereitgestellt wird. Neben dem Referenz-Browser Firefox wurde der Prototyp auch mit Chrome (Version 29.0.1547.76) und Opera (Version 9.0) erfolgreich getestet. Mit Safari (Version 6) und Internet Explorer (Version 10) treten Probleme mit den farblichen Markierungen auf, die nicht korrekt gesetzt werden.

Einsatz für	Komponente	Softwarepaket	Version
Client	Browser	Firefox	23.0.1
Server	Web-Server	Tomcat	7.0
	Datenbank	MySQL	5.6
Entwicklung	Framework	GWT	2.5.1
	Entwicklungsumgebung	Eclipse	Juno 2
		GWT Eclipse-Plugin	4.2
		GWT Firefox-Plugin	1.23
	Bibliotheken	Stanford CoreNLP	3.2.0
		mxGraph	1.11.0.0

Tabelle 4: Für den Prototyp getestete Versionen

[29] http://www.jgraph.com/mxgraph.html

Auch wenn es sich bei GWT um eine vergleichsweise junge Technologie handelt, so hat sich die Abwärts-Kompatibilität sowohl für den Code selbst als auch bezogen auf den Web-Browser in den letzten Jahren deutlich verbessert. Im Laufe der Entwicklung des Prototyps wurden sukzessive neu erscheinende GWT-Versionen verwendet, ohne dass Probleme bei der Kompatibilität entstanden.

6.2 Umsetzung

Der Semantic Requirement Editor, kurz SemanticRE, setzt den im Kapitel 5 entworfenen Lösungsansatz in Form eines Prototyps um. Es werden zunächst die implementierten Algorithmen vorgestellt und anschließend die wichtigsten Funktionalitäten der Oberfläche beschrieben. Abschließend werden die Besonderheiten der Implementierung aufgezeigt.

6.2.1 Algorithmen

Die Analyse läuft in drei grundlegenden Schritten ab (vgl. 5.4.3). Zu Beginn vergibt der Basistagger die grundlegenden Tags für POS, NER, Lemma, Beziehungen und Koreferenzen. Anschließend werden auf dieser Basis Vorschläge für Diagrammelemente durch die einzelnen Spezialtagger im Text farblich markiert. Im dritten und letzten Schritt erstellen die Diagramm-spezifischen Generatoren das jeweilige Diagramm. Im Folgenden werden ausgewählte Stellen der einzelnen Algorithmen vorgestellt.

6.2.1.1 Basistagger

Eine Besonderheit des Stanford CoreNLP ist, dass die einzelnen Komponenten größtenteils unabhängig voneinander arbeiten. So enthält beispielsweise der Parser selbst die Funktionalität für das POS Tagging. Eine besondere Reihenfolge für die einzelnen vorbereitenden Schritte ist damit nicht notwendig.

Für die Analyse der Beziehungen der Wörter untereinander werden typisierte Beziehungen (vgl. 3.2.2.4) verwendet, wie im Rahmen des Konzepts bereits begründet wurde. In einem ersten Durchlauf durch einen Satz werden alle syntaktischen Subjekte und Hauptverben getaggt; hier wird der Algorithmus ausschnittsweise für die Identifikation innerhalb einer Passivkonstruktion gezeigt:

```
01  LexicalizedParser lp = LexicalizedParser.loadModel(grammar, options);…
02  List<CoreMap> nlpSentences = document.get(SentencesAnnotation.class);…
03  for (CoreMap sentence : nlpSentences) {
04    Sentence s = new Sentence();
05    List<Word> wordList = s.getWords();…
06    Tree t = lp.parse(s.getSentenceAsString());
07    Collection<TypedDependency> typedDependencies =
                structure.allTypedDependencies();
08    for (TypedDependency typedDependency : typedDependenciesCollapsed) {
09      TreeGraphNode gov = typedDependencyCollapsed. gov();
10      TreeGraphNode dep = typedDependencyCollapsed. dep();
11      if (typedDependency.reln().equals(EnglishGrammaticalRelations.AGENT)){
12        wordList.get(gov.index()-1).markAsMainVerb(true);
13        wordList.get(dep.index()-1).markAsSubject(true);
14        wordList.get(dep.index()-1).addVerbOfNoun(wordList.get(gov.index()-1));
15        wordList.get(gov.index()-1)
                              .addSubjectOfVerb(wordList.get(dep.index()-1));
16  }…
```

Als Ausgangsbasis dient eine Liste aller gefundenen Beziehungen, die Stanford CoreNLP für jeden Satz (Zeile 4) auf Basis eines Parse-Baums (Zeile 6) bereitstellt. Die Beziehung `reln()` wird in Zeile 11 abgefragt. Für die Passivkonstruktion „The status is saved by the employee.“ lautet diese `agent(saved, employee)`, wobei „saved“ der Governor ist und entsprechend als Hauptverb getaggt wird (Zeile 12) und „employee“, der Dependent, entsprechend als syntaktisches Subjekt (Zeile 13). Beziehungen werden jeweils von beiden Seiten verwaltet. Als Grundstruktur wird dazu eine Liste aller Wörter in einem Satz (Zeile 5) verwendet, und den `Word`-Objekten in den Zeilen 14 und 15 werden die entsprechenden Referenzen hinzugefügt. Dabei fällt auf, dass der Index um 1 verschoben ist, da in der Wortliste das erste Element den Index 0 hat, in der Struktur der typisierten Beziehungen aber den Index 1. Analog verläuft die Identifikation eines nominales Subjektes, wobei auch hier der Governor in der Regel das Hauptverb und der Dependent das Subjekt ist. Die Ausnahme bildet eine sogenannte Copular-Beziehung (dt. Bindeglied). Ein Beispiel hierfür ist der Satz „The identifier is unique.“, in dem `nsubj(unique, identifier)` als Ergebnis ausgegeben wird. In diesem Fall wird das Subjekt gespeichert und das Hauptverb später ergänzt.

Nachdem alle syntaktischen Subjekte erfasst worden sind, können in einem zweiten Durchlauf durch den Satz alle Adjektive erfasst werden. Diese Reihenfolge ist notwendig, da ein Substantiv – wie beschrieben – teilweise wie ein Adjektiv behandelt wird und dabei wichtig ist, ob es bereits als Subjekt getaggt wurde. Im Beispiel „The loan rate is flexible.“ wird im ersten Durchlauf „rate“ als Subjekt markiert. Im zweiten Durchlauf wird das Wort als Adjektiv von „loan“ getaggt, und die anderen Beziehungen werden entsprechend umgesetzt. Das heißt, dass dann „loan“ als Subjekt getaggt und die Beziehungen mit dem Hauptverb „is“ angepasst wird. Weiterhin als Adjektive werden Possessivbeziehungen, Präpositionen mit „of“ und Adjektive selbst identifiziert. Einen Überblick sowie Beispiele gibt Tabelle 5.

Im dritten und vierten Durchlauf werden alle syntaktischen Objekte gesucht, wobei jeweils geprüft wird, ob es sich um ein bereits als Adjektiv getaggtes Wort handelt. In diesem Fall werden die Beziehungen analog zum eben gezeigten Vorgehen für syntaktische Subjekte angepasst. Es werden alle vom Stanford CoreNLP implementierten Objekte erfasst: direkte, indirekte und präpositionale Objekte. Weiterhin werden nominale Subjekte in Passivkonstruktionen als syntaktische Objekte erfasst sowie alle ausstehenden Copular-Beziehungen korrekt gesetzt. Dies ist eine der wenigen Stellen, an der POS Tags benötigt werden, da Copular-Beziehungen mit Substantiven und auch mit Verben auftreten können. Präpositionale Objekte werden in einem eigenen Durchlauf erfasst, da auf Basis zusammengezogener Beziehungen eine direkte Verbindung zwischen Hauptverb und Objekt möglich ist. Tabelle 5 gibt auch hierzu einen Überblick.

Im fünften und letzten Durchlauf werden Konjunktionen behandelt. Konjunktionen können sich auf alle Teilkonzepte beziehen. Daher steht diese Behandlung am Ende; es werden jeweils die Beziehungen des Governors für den Dependent übernommen. Beispielsweise werden für den Satz „The office accepts or rejects the loan.“ in den ersten Durchläufen „office“ als syntaktisches Subjekt, „accepts“ als Hauptverb und „loan“ als syntaktisches Objekt getaggt. Dies wird nun auf „rejects“ übertragen. Tabelle 5 zeigt einen Überblick über alle Durchläufe.

Durch-lauf	Identifikation von:	EnglishGrammaticalRelation	Beispiel (Governor, Dependent)
1	syntaktischen Subjekten und Hauptverben	NOMINAL_SUBJECT	The employee saves the status.
		AGENT	The status is saved by the employee.
2	Adjektiven	NOUN_COMPOUND_MODIFIER	customer name
		POSSESSION_MODIFIER	customer's name
		equals("prep_of")	name of the customer
		ADJECTIVAL_MODIFIER	He buys a red car.
3	syntaktischen Objekten	DIRECT_OBJECT	The customer buys a car.
		INDIRECT_OBJECT	The customer gives the employee a call.
		NOMINAL_PASSIVE_SUBJECT	The status is saved by the employee.
		COPULA	The identifier is unique.
4	präpositionalen Objekten	startsWith("prep") && !equals("prep_of")	The employee talks with the customer.
5	Konjunktionen	startsWith("CON")	The office accepts or rejects the loan. The office calls the customer and the employee.

Tabelle 5: Durchläufe und identifizierte Beziehungen

Am Ende des Basistaggers wird das Koreferenz-Modul des Stanford CoreNLP eingesetzt. Dieses vergibt für Substantive aufsteigende ganzzahlige Werte, wobei zusammengehörige Wörter den gleichen Wert erhalten und die folgende Ganzzahl übersprungen wird. Im Beispielsatz „The office[1] informs the employee[2]. He[2] calls the customer[4].“ werden die Substantive beziehungsweise die Referenz-anzeigenden mit den entsprechenden Zahlen gekennzeichnet. Die Information der Koreferenz-Tags wird für die Diagrammgenerierung benötigt. Zunächst werden im Folgenden die Algorithmen der Spezialtagger beschrieben.

6.2.1.2 Spezialtagger

Die Spezialtagger operieren unabhängig voneinander auf den Tags, die durch den Basistagger vergeben wurden. Der umfangreichste Spezialtagger ist der für das Klassendiagramm, da dieser insgesamt vier verschiedene Elemente taggen muss: Klassen, Beziehungen, Operationen und Attribute. Der Ausschnitt des Quellcodes zeigt die Vergabe der entsprechenden Tags:

```
01  if ((word.isMarkedAsObject() || word.isMarkedAsSubject()) &&
                word.getNe().equals("O")){
02    word.setClassTag(ClassDiagramComponentEnum.Class);
03  } else if (word.isMarkedAsMainVerb() ){
04    if (!word.getObjectsOfVerb().isEmpty() &&
                !word.getSubjectsOfVerb().isEmpty()) {
05      word.setClassTag(ClassDiagramComponentEnum.Relation);
06    } else {
07      word.setClassTag(ClassDiagramComponentEnum.Method);
08    }
09  } else if (word.isMarkedAsAdjective()){
10    word.setClassTag(ClassDiagramComponentEnum.Attribute);
11  }…
```

Es wird in Zeile 1 überprüft, ob eine benannte Entität vorliegt. Ist dies der Fall, so wird dieses Wort nicht als Klasse markiert. In Zeile 4 wird überprüft, ob ein Hauptverb mindestens ein syntaktisches Subjekt und mindestens ein syntaktisches Objekt hat. In diesem Fall wird das Verb als transitiv verstanden und als Beziehung im Klassendiagramm getaggt. Hauptverben, die diese Bedingung nicht erfüllen, werden als Operationen getaggt (Zeile 7) und Adjektive als Attribute (Zeile 10).

Die beiden Spezialtagger für Anwendungsfall- und BPMN-Diagramme arbeiten sehr ähnlich, sodass es ausreicht, den relevanten Quellcode-Ausschnitt des Ersteren zu zeigen:

```
01  if (word.isMarkedAsSubject()
            && (!word.getNe().equals("NUMBER"))
            && (!word.getNe().equals("DURATION"))
            && (!word.getNe().equals("LOCATION"))) {
02    word.setUseCaseTag(UseCaseDiagramComponentEnum.Actor);
03  } else if (word.isMarkedAsMainVerb() && !word.getLemma().equals("be")){
04    word.setUseCaseTag(UseCaseDiagramComponentEnum.UseCase);
05  }…
```

Für Akteure werden – anders als für Klassen – nur syntaktische Subjekte und nicht auch syntaktische Objekte herangezogen (Zeile 1). Dies basiert auf der Überlegung, dass Akteure aktiv Handelnde sind. Ausgenommen davon sind benannte Entitäten der Art Zahl, Dauer und Ort, für die eine Übernahme als Akteur nicht sinnvoll ist. Als Anwendungsfall werden alle Hauptverben herangezogen (Zeile 3). Eine Ausnahme bildet der Fall, in dem „be“ das Hauptverb beziehungsweise das Lemma des Hauptverbs ist, da es sich nicht um eine Handlung handelt und damit die Übernahme als Anwendungsfall nicht sinnvoll ist. Wörter mit dem Lemma „be“ werden nur für das Klassendiagramm in eine Vererbungsbeziehung umgewandelt, und „is-a“ wird als Text eingesetzt. Der Spezialtagger für das BPMN-Diagramm verwendet das gleiche Vorgehen wie der für Anwendungsfalldiagramme, wobei Swimlanes analog zu Akteuren und Aktivitäten analog zu Anwendungsfällen getaggt werden.

6.2.1.3 Diagrammgeneratoren

Die Generatoren erstellen auf Basis der vorhandenen Tags die jeweiligen Diagramme. Für das Klassendiagramm werden dazu zunächst alle (UML-)Beziehungs-Elemente identifiziert. Die syntaktischen Subjekte des Hauptverbs werden als Ausgangsklasse für die Beziehung angelegt und diese jeweils mit allen syntaktischen Objekten verbunden. In einem zweiten Durchlauf durch den Satz werden Klassen gesucht, die bisher nicht angelegt wurden, also keiner Beziehung zugehörig sind. Jeweils nach dem Anlegen einer Klasse und auch nach dem Auffinden einer Referenz auf eine Klasse durch die Koreferenzanalyse werden alle zugehörigen Operationen und Attribute ergänzt. Dies ist möglich, da jedes syntaktische Subjekt und Objekt eine Liste mit den entsprechenden Bezugsobjekten verwaltet. Im Falle des Auftretens von nicht zuordenbaren Operationen und Attributen wird, wie oben (vgl. 5.4.3) beschrieben, in der Beziehungskette nach weiteren Möglichkeiten der Zuordnung gesucht.

Als Beispiel für das Anlegen eines Diagrammelements wird hier der Akteur für das Anwendungsfalldiagramm gezeigt:

```
01  for (Sentence sentence : taggedText.getSentences()){
02    for (Word word : sentence.getWords()){
03      if (word.getUseCaseTag() == UseCaseDiagramComponentEnum.Actor){
04        if (!createdActorsMap.containsKey(word.getCoRefID())){
05          String w = WordHelper.lemma(word);
06          w  = WordHelper.firstLettUpperCase(w);
07          Actor actor = new Actor(w);
08          diagram.actors.add(actor);
09          createdActorsMap.put(word.getCoRefID(), actor);
11        }
12        addUseCase(createdActorsMap.get(word), word);
13      }…
```

In Zeile 4 wird dabei überprüft, ob für den Akteur bereits durch die Koreferenzanalyse ein entsprechendes Element identifiziert wurde. Falls dem so ist, wird die Erstellung übersprungen und direkt in Zeile 12 die Methode für das Hinzufügen der Anwendungsfälle zum Akteur aufgerufen. Wurde der Akteur noch nicht angelegt, so wird für den Namen des Akteurs zunächst das Lemma verwendet und der erste Buchstabe großgeschrieben (Zeilen 5 und 6). In Zeile 8 wird der Akteur dem Diagramm hinzugefügt und in Zeile 9 als angelegt registriert. Für das Anlegen der Anwendungsfälle wird das als Anwendungsfall markierte Hauptverb mit allen seinen direkten Objekten kombiniert. Aus dem Satz „The customer gives the employee a call.“ wird so der Anwendungsfall „call employee“ extrahiert. Für Hauptverben ohne direktes Objekt wie „The customer waits.“ wird nur das Verb verwendet, also hier „wait“.

Erneut ist das Vorgehen für das BPMN-Diagramm weitgehend analog zu dem des Anwendungsfalldiagramms. Der Unterschied besteht darin, dass die Aktivitäten noch durch Pfeile durch den Sequenzfluss verbunden werden müssen. Dazu wird die Reihenfolge der Hauptverben im Text übernommen.

6.2.2 Oberfläche

Die Verwendung der Oberfläche des Semantic Requirement Editor wird anhand von ausgewählten Szenarien dargestellt. Die folgende Beschreibung orientiert sich am Ablauf des Arbeitens mit der Software (vgl. 5.3.2). Zunächst wird auf die Projekt-, Benutzer- und Artefaktverwaltung eingegangen. Dann werden der semantische Editor sowie die generierten Diagramme vorgestellt.

6.2.2.1 Projekt-, Benutzer- und Artefaktverwaltung

Wenn der Benutzer sich zum ersten Mal nach dem Anlegen des Kontos einloggt, wird zunächst die Projektübersicht angezeigt (siehe Abbildung 33). Falls noch kein anderer Benutzer ein Projekt freigeben hat, ist die Projektliste leer. Hier können neue Projekte angelegt oder importiert werden. Ein gewähltes Projekt kann bearbeitet, exportiert oder gelöscht werden. Ebenfalls können für jedes Projekt die Benutzerrechte angepasst werden. Nach dem Anlegen eines Projekts können die schriftlichen Anforderungsspezifikationen erfasst werden. Aus Gründen der Vereinfachung werden diese in der Software kurz als „Requirement“ bezeichnet.

Um dem Benutzer eine intuitive und verständliche Benutzerführung zu bieten, wird eine Navigationsleiste verwendet, die zu jedem Zeitpunkt sichtbar ist. Die Menüpunkte sind Project Manager, Project Details, Requirements, Use Case Diagrams, Class Diagrams, BPMN Diagrams, User Administration und Logout.

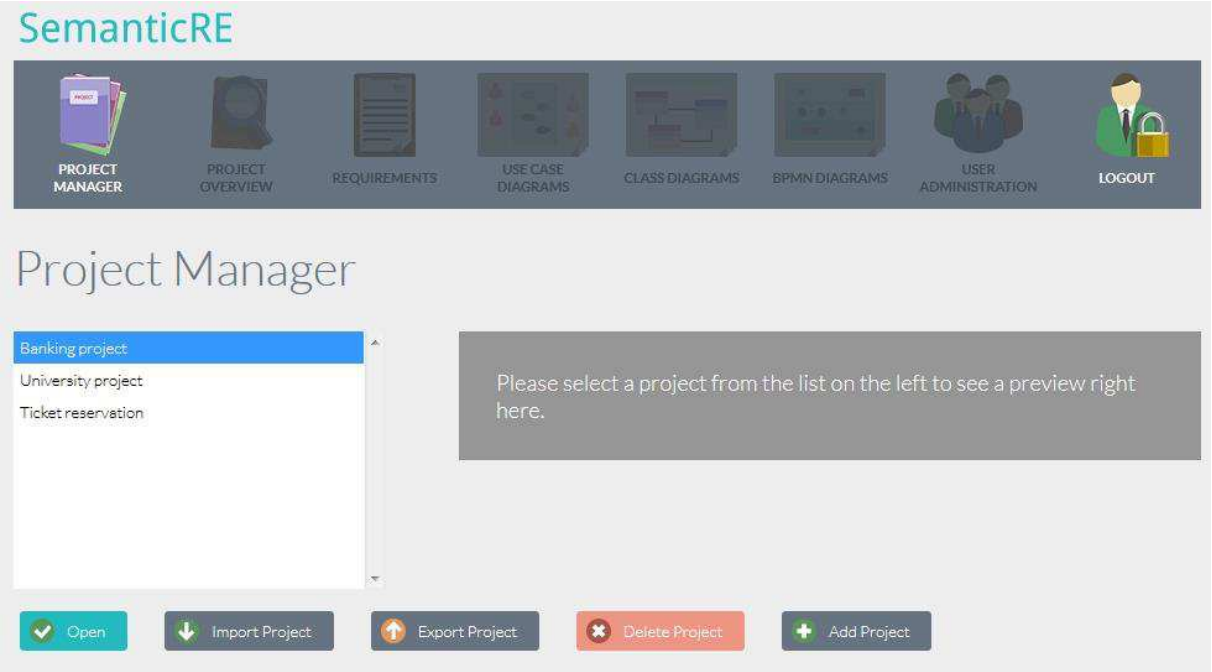


Abbildung 33: Übersicht über alle Projekte im Project Manager

Unter dem Menüpunkt Requirements können einzelne Anforderungsspezifikationen zu einem Projekt hinzugefügt, bearbeitet oder gelöscht werden, Gleiches gilt jeweils für die anderen Artefakte. Wird eine Anforderungsspezifikation für die Bearbeitung ausgewählt, so gelangt der Benutzer auf die Ansicht für den semantischen Editor, welche separat im nächsten Abschnitt behandelt wird. Die einzelnen Diagramme werden im darauf folgenden Abschnitt aufgezeigt. Die Benutzerverwaltung (engl. User Administration) erlaubt es einem Administrator, die Daten der einzelnen Benutzer anzupassen sowie das Passwort zurückzusetzen und Benutzerkonten zu aktivieren und deaktivieren. Es ist vorgesehen, dass Benutzer erst nach der Aktivierung durch einen Administrator die Software verwenden können. Weiterhin können hier Administratoren anderen Benutzern ebenfalls Administratorenrechte einräumen.

Einzelne Menüpunkte sind unter Umständen ausgegraut und damit nicht anwählbar. Beispielsweise ist in der Abbildung 33 die Benutzerverwaltung erkennbar inaktiv, da der im Moment eingeloggte Benutzer nicht über Administrationsrechte verfügt.

6.2.2.2 Semantischer Editor

Der semantische Editor ist die für den Benutzer sichtbare Oberfläche des Analysemoduls. Er erlaubt die Eingabe der schriftlichen Anforderungsspezifikationen. Da die Analyse eine Fülle an Informationen als Ergebnis bereitstellt, war es hier die Herausforderung, die verschiedenen visuellen Elemente übersichtlich anzuordnen. Im Mittelpunkt steht die Anforderungsspezifikation. Auf der linken Seite sind deren Name, das Datum des Anlegens sowie das Datum der letzten Bearbeitung zu sehen. Der Benutzer kann zwischen den jeweils einem Diagrammtyp zugehörigen Markierungen wählen, damit die Übersichtlichkeit gewahrt bleibt. Weiterhin soll der Benutzer die Möglichkeit haben, die Markierungen zu filtern, also beispielsweise nur bestimmte Elemente wie die Klassen des Klassendiagramms anzuzeigen. Dies wird durch ein baumartiges Element auf der linken unteren Seite verwirklicht. Abbildung 34 zeigt den semantischen Editor mit den Markierungen für ein Klassendiagramm.

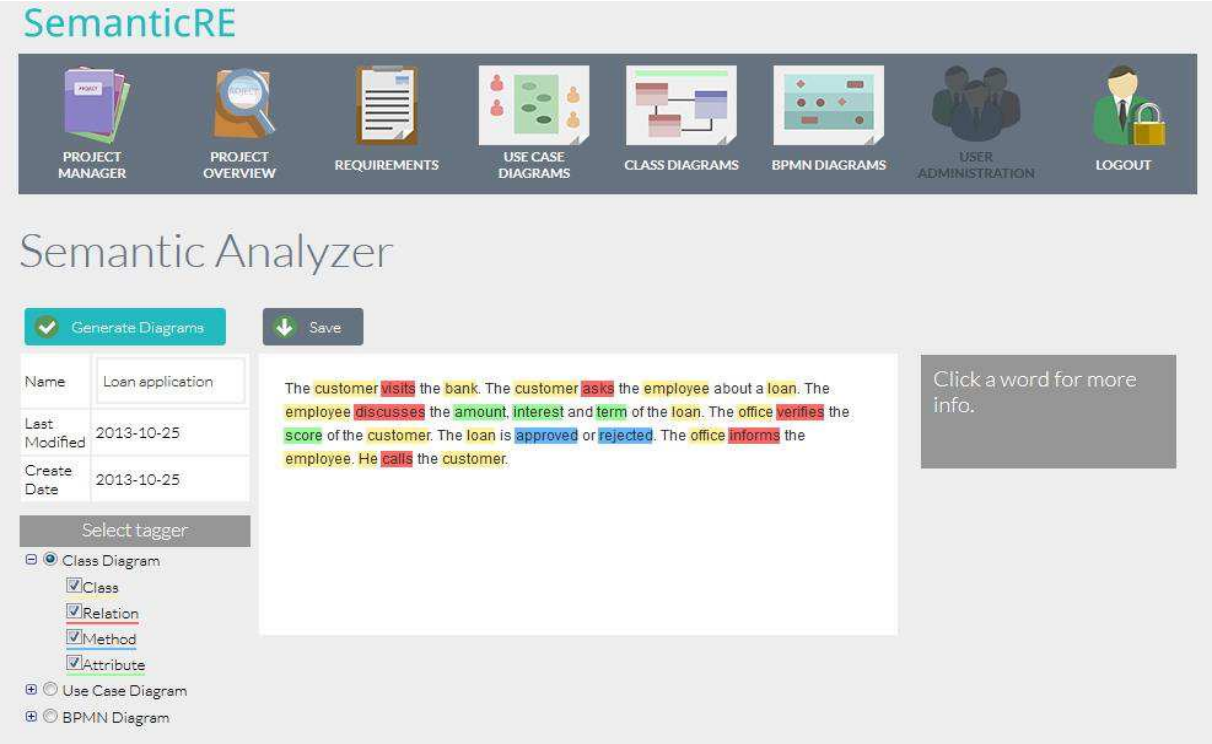


Abbildung 34: Semantischer Editor mit Hervorhebungen für das Klassendiagramm

Wenn ein bestimmtes Wort angeklickt wird, können zusätzliche Informationen abgerufen und die vergebenen Tags angepasst werden. Abbildung 35 zeigt, wie diese zusätzlichen Informationen aussehen können, wobei diesmal auf der linken Seite die Markierungen für ein Anwendungsfalldiagramm ausgewählt wurden.

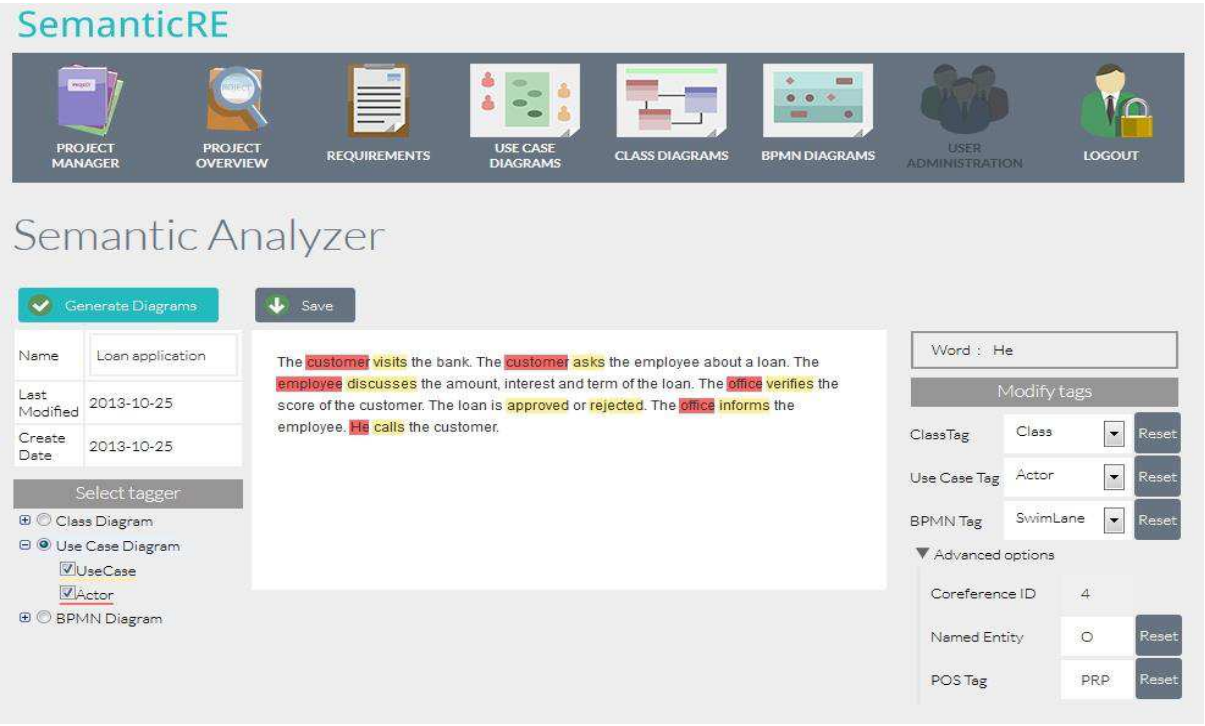


Abbildung 35: Semantischer Editor mit Hervorhebungen für das Anwendungsfalldiagramm

Zu sehen sind auf der rechen Seite die Zusatzinformationen für das Wort „He“. Durch die Koreferenzanalyse wurde dem Wort „employee“, auf das es sich bezieht, ebenfalls der Wert 4 zugeordnet; hier wird diese Referenz in Form des Felds Coreference ID sichtbar. Das Feld Named Entity enthält das vom entsprechenden Paket des Stanford CoreNLP vergebenen Tag. Das verwendete Set an Kategorien ist das größte der drei verfügbaren und unterscheidet die sieben verschiedenen Kategorien „Time“, „Location“, „Organization“, „Person“, „Money“, „Percent“ und „Date“. Das Feld POS Tag enthält das Part-of-Speech-Tag. Diese Felder dienen dem Benutzer vor allem als Zusatzinformation, bei Bedarf können sie mit Ausnahme des Felds für die Coreference ID auch angepasst werden. Die Werte der Koreferenzanalyse sind wie beschrieben aufsteigende Ganzzahlen, die bei jedem Analysedurchlauf neu vergeben werden. Sie verschieben sich bei Eingabe eines zusätzlichen Substantives weiter vorn im Text, sodass ein manuelles Setzen nicht sinnvoll ist.

Die Analyse wird jeweils angestoßen, wenn der Benutzer die Eingabe eines Wortes beendet hat, also ein Leerzeichen, ein Komma oder einen Punkt setzt. Es wäre alternativ möglich, einen Timer zu verwenden, der nach einer bestimmten Zeit die Analyse auslöst. Da die Analyse mit Wörtern als kleinste Analyseeinheit arbeitet, ist die erste Lösung die sinnvollere. Weil die Analyse Beziehungen innerhalb eines ganzes Satzes einbezieht, können sich Tags im Laufe der Eingabe eines Satzes ändern: Beispielsweise wird ein Hauptverb zunächst als Operation und erst mit Hinzufügen eines Objekts als Beziehung für das Klassendiagramm identifiziert.

6.2.2.3 Generierte Diagramme

Wenn der Benutzer alle Tags wie gewünscht angepasst hat, kann er eines der Diagramme generieren, zu dem er direkt weitergeleitet wird. Als Beispiel wird in Abbildung 36 das Klassendiagramm gezeigt, welches auf Basis der Eingabe aus Abbildung 34 erstellt wurde.

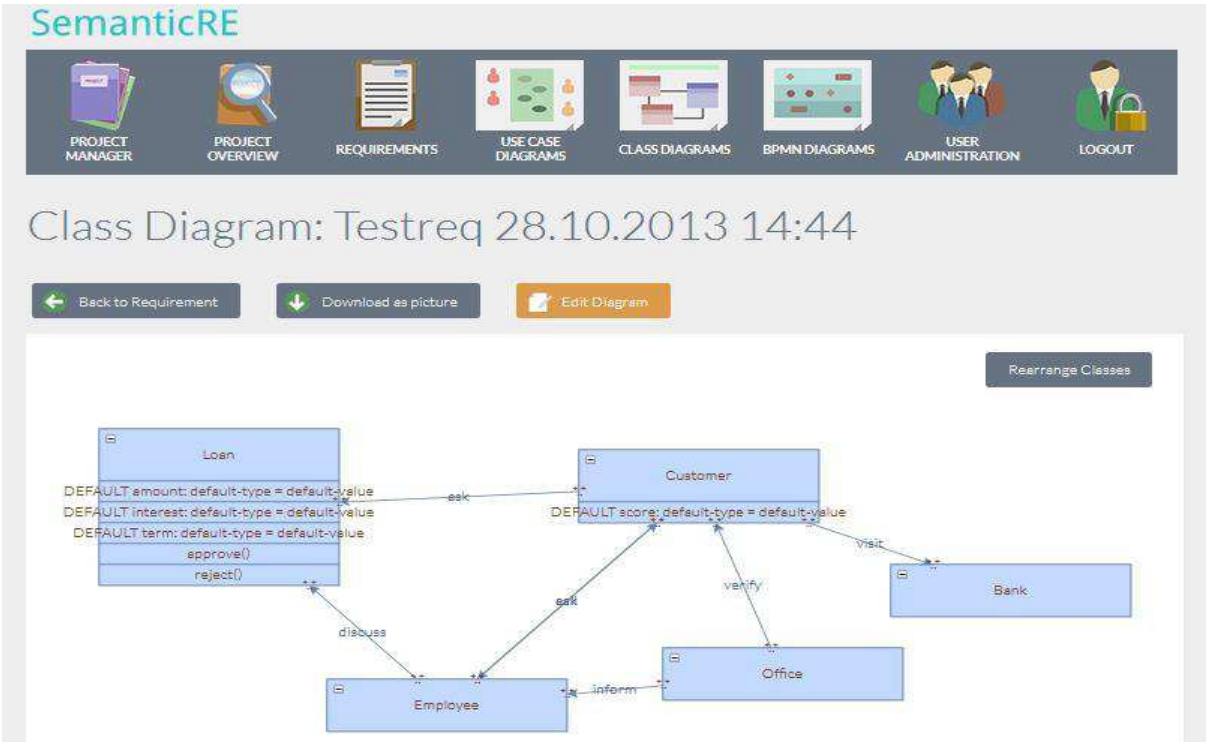


Abbildung 36: Generiertes Klassendiagramm

In Abbildung 37 wird das Anwendungsfalldiagramm gezeigt, welches für dieselbe Eingabe wie eben erstellt wurde. Die zugehörigen Markierungen wurden in Abbildung 35 gezeigt.

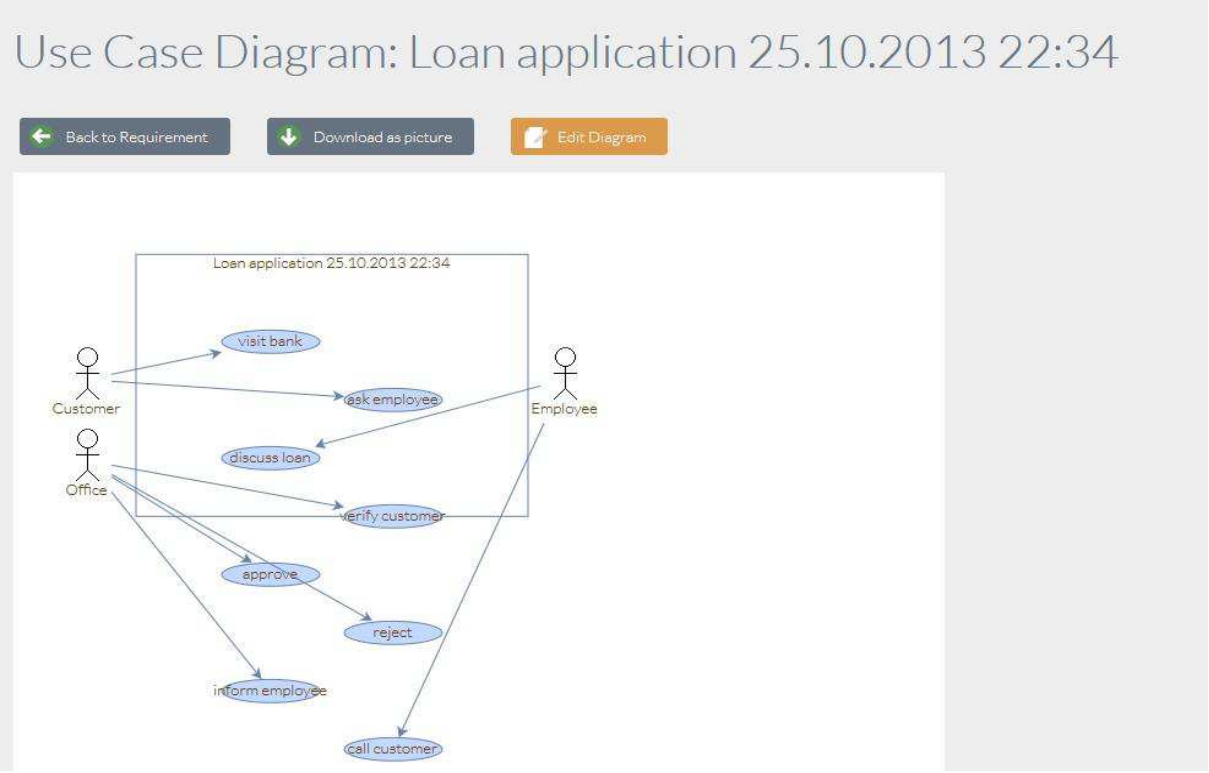


Abbildung 37: Generiertes Anwendungsfalldiagramm

Das BPMN-Diagramm für dieses Beispiel wird in Abbildung 38 gezeigt. Die Markierungen entsprechen aufgrund der Analogie der Algorithmen denen des Anwendungsfalldiagramms.

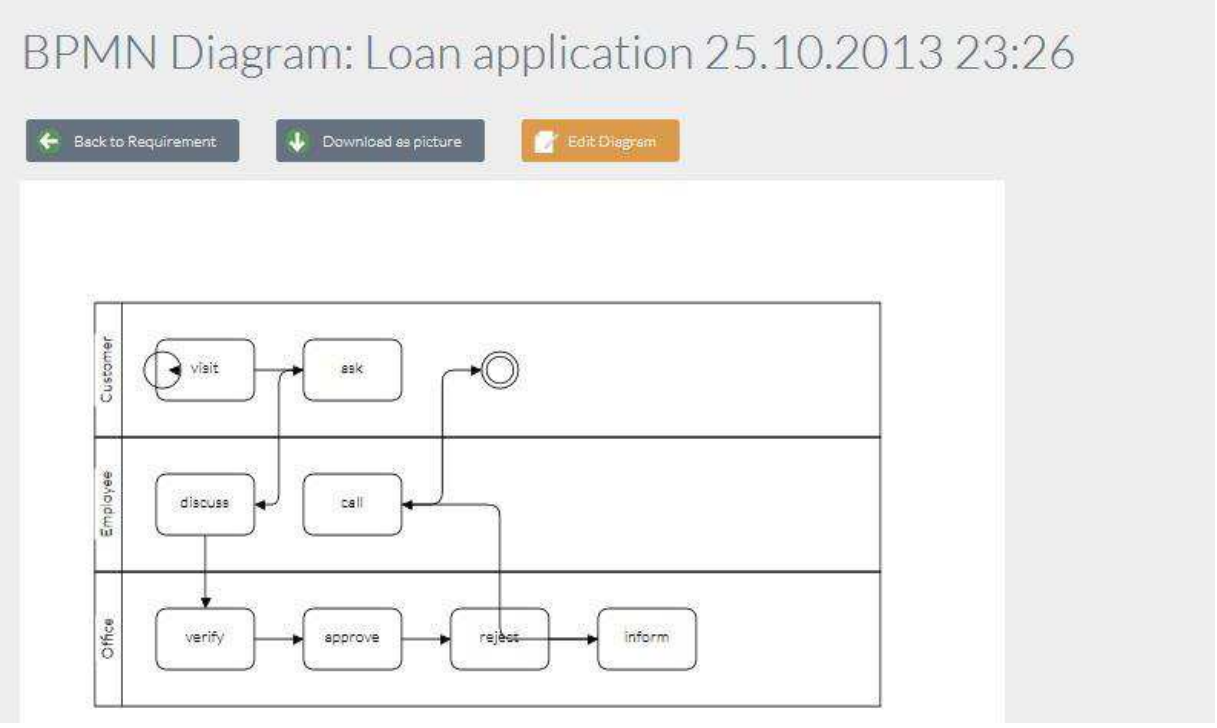


Abbildung 38: Generiertes BPMN-Diagramm

Für weitere Anpassungen kann jeweils der Button Edit Digram rechts oberhalb des Diagramms verwendet werden. Es besteht ebenfalls die Möglichkeit, das Diagramm als Bild zu speichern. Um Diagramme effizient unterscheiden zu können, wird jedes mit dem Zeitstempel seiner Generierung versehen. Alle Diagramme können dann in der Übersicht betrachtet werden und alte Versionen bei Bedarf erneut aufgerufen oder auch gelöscht werden. Wie diese Übersicht zur Verwaltung aussieht, wird in Abbildung 39 gezeigt.

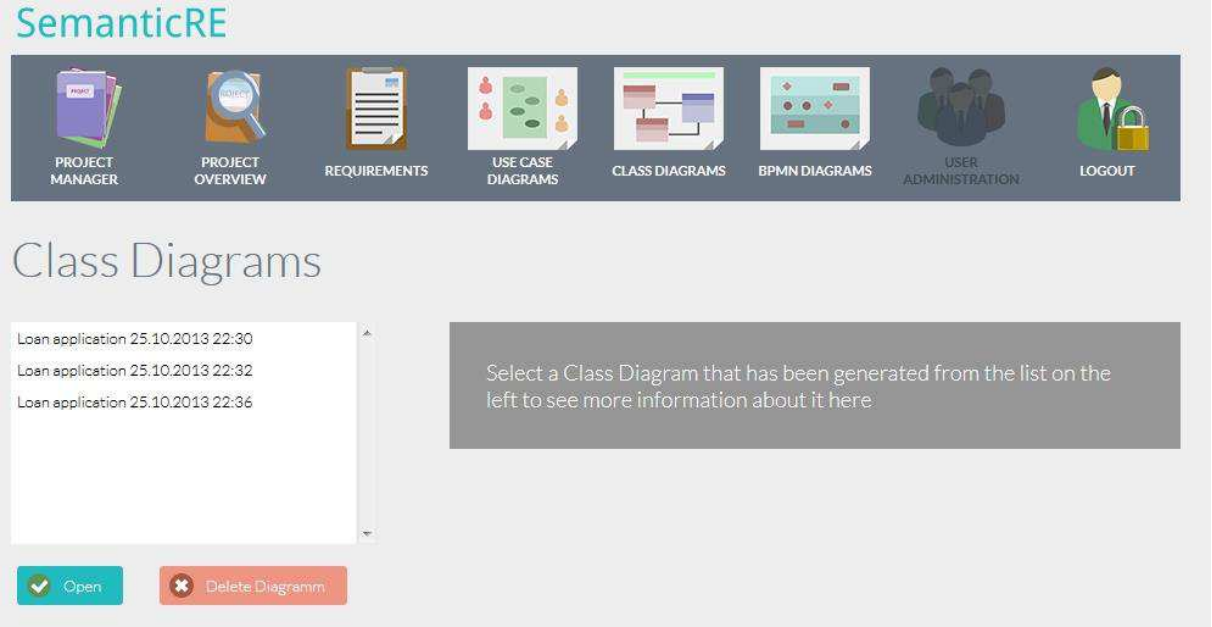


Abbildung 39: Verwaltung der Klassendiagramme

Der bisher nicht erklärte Menüpunkt Logout führt zur erwarteten Funktionalität, d. h., der Benutzer beendet seine Sitzung und kann sich bei Bedarf erneut einloggen.

6.2.3 Besonderheiten und Optimierungen

Bei der Implementierung mit GWT traten unerwartete Besonderheiten auf. Es gibt verschiedene Möglichkeiten für die Eingabe von Text, jedoch verfügt keine der vorhandenen GUI (Graphical User Interface)-Komponenten über alle benötigten Eigenschaften. In der `TextArea` können Wörter nicht farblich markiert werden; von der `RichTextArea` werden `ClickEvents` nicht unterstützt. Diese werden aber benötigt, damit die Position des Cursors ermittelt und anschließend zusätzliche Informationen zu einem Wort angezeigt werden können. Von den Mitgliedern des Teamprojekts wurde dieses Problem in der Art gelöst, dass eine transparente `TextArea` zum Anzeigen der farblichen Markierungen über die `RichTextArea` gelegt wurde, um so die gewünschte Funktionalität umzusetzen. Da GWT beständig weiterentwickelt wird, ist für die Zukunft eine direktere Lösung basierend auf nur einer GUI-Komponente zu erwarten.

Beim Einsatz des Stanford CoreNLP muss berücksichtigt werden, dass zur Ausführung eine hohe Arbeitsspeicherkapazität benötigt wird. Während der Entwicklung muss deshalb der entsprechende Parameter auf der Kommandozeile mit übergeben werden, es empfiehlt sich eine Größenordnung von mindestens 2 GB. Für die produktive Bereitstellung empfiehlt sich auf dem Server ein RAM von mindestens 5 GB.

6.3 Verwendung

Unabhängig von der Implementierung sollte für Anforderungsspezifikationen in natürlicher Sprache eine Reihe von Konventionen beachtet werden. Es handelt sich hier nicht um eine Einschränkung der zu verwendenden Sprache, sondern um Hilfestellungen, die es anderen Beteiligten erleichtern, die Anforderungsspezifikationen korrekt zu verstehen. Pohl (2008) beschreibt mehrere Regeln für Sprache und Grammatik; die wichtigsten sind die Verwendung der Aktivform und das Verfassen im Präsens. Beispielsweise sollte statt „Die Benutzerkennung wurde eingegeben." besser formuliert werden „Der Benutzer gibt seine Benutzerkennung in das System ein.", was die Aussagen lebendiger macht und die Verständlichkeit erleichtert. Die Aktivform erfordert außerdem die explizite Nennung eines Handelnden, was als wichtige Information dazu beiträgt, Missverständnisse zu vermeiden. Darüber hinaus sind Modalverben zu eliminieren, also statt „Das System sollte die Benutzerdaten überprüfen." besser als „Das System überprüft die Benutzerdaten." formuliert werden. Modalverben suggerieren eine Möglichkeit, was in diesem Kontext zu Missverständnissen führen kann (Pohl 2008). Bezüglich des zu verwendenden Vokabulars schlägt Berztiss (1997) vor, möglichst präzise im Sprachgebrauch zu sein. Als Beispiel führt er Folgendes an: „last year this publisher put out thirty-seven books; the publisher has 200000 books in its warehouse"(S. 48). Für die erste Bedeutung von „book" ist „title" passender, für die zweite „copy". Diese sprachlichen Feinheiten können von der NLP-Analyse nicht erfasst werden und benötigen den Beitrag konzeptueller Überlegungen durch den Benutzer.

Diese Regeln haben Vorschlagscharakter und werden vom System nicht überprüft, da in der vorliegenden Arbeit keine Einschränkung der natürlichen Sprache vorgenommen werden soll. Hinsichtlich der Analyse bestehen aufgrund der durch das Stanford CoreNLP bereitgestellten Funktionalitäten keine grammatikalischen Restriktionen, da Modalverben und auch die Aktiv- und Passivform automatisch erkannt werden. Nicht implementiert ist eine Erkennung von Synonymen. Über die Einbindung von WordNet wäre dies prinzipiell möglich, bleibt jedoch dem Benutzer überlassen. Er wird bereits bei der Eingabe der Anforderungsspezifikationen im Editor dabei unterstützt, die wichtigsten Elemente zu identifizieren, sodass Synonyme schneller auffallen als in einem Text ohne Hervorhebungen. Auch in den Diagrammen führen synonyme Begriffe zu einzelnstehenden Elementen, die dem Benutzer einen Hinweis darauf geben können, dass hier eine Anpassung notwendig ist.

6.4 Limitationen der Implementierung

Eine der wichtigsten Einschränkungen des Prototyps ist, dass keine grafischen Editoren für die Modifikation der Diagramme bereitgestellt werden. Stattdessen werden Modifikationen über einen Wizard vorgenommen, wobei in mehreren Schritten die einzelnen Elemente angepasst werden. Dies ist notwendig, da die verwendete JavaScript-Bibliothek keine grafischen Bearbeitungsoptionen zur Verfügung stellt. Innerhalb der schrittweisen Anpassung können alle Elemente modifiziert werden, wobei beispielsweise für ein Klassendiagramm zunächst die Klassen und erst im Anschluss die Verbindungen zwischen den Klassen festgelegt werden. Dies erleichtert über sinnvolle Speicherpunkte die Wahrung der Konsistenz des Modells in der Datenhaltungsschicht. Außerdem verbessert die Trennung der einzelnen Schritte die Übersichtlichkeit des Bearbeitungsprozesses und der Oberfläche selbst.

Im Moment beinhalten das Konzept und der Prototyp eine Sprachunterstützung nur für englische Anforderungsspezifikationen. Das Konzept könnte in großen Teilen auf andere, hinreichend ähnliche Sprachen wie das Deutsche übertragen werden. Auch das Stanford CoreNLP unterstützt ein deutsches Tagset sowie die Analyse typisierter Beziehungen für die deutsche Sprache. Eine Erweiterung der Sprachunterstützung ist damit technisch problemlos möglich.

Im Sinne der Wiederverwendung von Artefakten wäre es wünschenswert, diese zwischen verschiedenen Projekten auszutauschen oder sogar gemeinsam zu verwenden. Denkbar wäre beispielsweise eine Verknüpfung mit Kernkomponenten, die im System hinterlegt sind und für die ein neues Projekt eine Schnittstelle implementieren soll. Der Benutzer kann Projekte im vorliegenden Prototyp nur als Ganzes exportieren und auch wieder importieren. Sollen nur einzelne Artefakte ausgetauscht werden, so könnte die Export/Import-Funktionalität angepasst werden. Dadurch, dass die Speicherung der Projektdaten ausschließlich im XML-Format stattfindet, sind auch Schnittstellen mit anderen Programmen möglich.

6.5 Zusammenfassung

Der hier vorgestellt Prototyp erfüllt die im letzten Kapitel beschriebenen Anforderungen. Er demonstriert damit die Umsetzbarkeit des Lösungsansatzes. In diesem Kapitel wurden die technischen Voraussetzungen sowie die Implementierung der Algorithmen vorgestellt. Alle hier aufgeführten Beispiele zeigen ausschließlich von den Algorithmen erstellte Ergebnisse, es wurden keinerlei manuelle Veränderungen oder Ergänzungen vorgenommen. Die erzeugten farblichen Markierungen und die generierten Diagramme belegen, dass die Analyse sinnvolle Ergebnisse liefert.

Der Semantic Requirement Editor richtet sich an alle, die am Prozess der Softwareentwicklung beteiligt sind. Auch Benutzer ohne Vorwissen auf dem Gebiet der Softwaremodellierung können Anforderungsspezifikationen eingeben, welche dann von Experten im Bereich der Modellierung aufgerufen und weiterbearbeitet werden können. Eine weitere Möglichkeit ist die, dass die Anforderungen in Interviews oder Fokusgruppen besprochen und anschließend von den Softwareingenieuren in Textform umgesetzt werden. Die Auftraggeber der zu entwickelnden Software können dann im System die Anforderungsspezifikationen einsehen. Bei Änderungen in den schriftlichen Anforderungsspezifikationen können diese direkt in die Diagrammform überführt werden.

7 Evaluation

Um den vorgestellten Lösungsansatz sowie den implementierten Prototyp bewerten zu können, werden beide Arbeitsergebnisse evaluiert. Dafür wird zunächst das Konzept für die Evaluation herausgearbeitet, dann werden die Planung sowie die Umsetzung beschrieben. Anschließend werden die Ergebnisse der Evaluation vorgestellt und eine Zusammenfassung gegeben.

7.1 Evaluationskonzept

Für die Erarbeitung des Evaluationskonzepts wird die Methode von Kitchenham (1996) verwendet. Diese erlaubt es, den Lösungsansatz und das implementierte Werkzeug gleichzeitig zu evaluieren. Die Methode umfasst sieben Evaluationskriterien; wobei sich das erste Kriterium vor allem auf den Organisationskontext bezieht. Es beschreibt unter anderem die Wahl einer Methode oder eines Werkzeugs für ein bestimmtes Projekt oder für die Einbindung in eine eigene Produktlinie in der Organisation. Da diese Überlegungen auf den Forschungskontext nicht zutreffen, wird das Kriterium an dieser Stelle nicht einbezogen und es werden im Folgenden die sechs weiteren Kriterien erläutert.

Das zweite Kriterium betrifft die Art des Einflusses der Methode beziehungsweise des Werkzeugs. Diese wird in qualitativ und quantitativ unterschieden. Ein qualitativer Einfluss liegt vor, wenn beispielsweise die Sichtbarkeit eines Prozesses oder die Benutzbarkeit erhöht werden. Mit quantitativem Einfluss werden Auswirkungen auf die Produktivität, Wartbarkeit oder Qualität bezeichnet. Für die hier vorliegende Arbeit sollen beide Arten des Einflusses evaluiert werden. Die Art des Evaluationsobjekts, das dritte Kriterium, wurde bereits eingangs festgelegt: Es sollen sowohl Methode als auch Werkzeug in Kombination beurteilt werden. Als viertes Kriterium wird der Gestaltungsbereich des Einflusses betrachtet, der aus der Granularität und dem Umfang besteht. Die Granularität bezieht sich auf die Frage, ob ein Softwareprodukt als Ganzes oder jeweils die einzelnen Teile untersucht werden sollen. Für die vorliegende Evaluation soll beides erfolgen. Der Umfang betrifft den Einfluss der Methode beziehungsweise des Werkzeugs, er beschreibt wie viele Phasen des Softwarelebenszyklus abgedeckt werden, wobei der Entwicklungsphase und der Wiederverwendung besondere Aufmerksamkeit zukommen, da diese als einzelne Unterkriterien aufgeführt werden. In der vorliegenden Arbeit wird der gesamte Lebenszyklus betrachtet, allerdings muss dies aus Gründen des Umfangs für die Evaluation eingeschränkt werden, sodass nur die ersten Phasen der Anforderungsanalyse und des Entwurfs betrachtet werden. Das fünfte Kriterium unterscheidet die Reife des Evaluationsobjekts. Drei Reifegrade werden unterschieden: (1) noch nicht im kommerziellen Einsatz und damit noch in der Entwicklung, (2) der Einsatz für wenige durch die eigene Organisation erstellte Produkte und (3) die breite Verwendung. Hier zeigt sich deutlich, dass die Evaluationsmethode für kommerzielle Produkte entwickelt wurde. Im Forschungskontext wird in der Regel von Prototypen gesprochen, wenn eine funktionsfähige Implementierung zu Demonstrationszwecken vorliegt, was hier der Fall ist. Die Lernzeit ist das sechste Kriterium, wobei unterschieden wird zwischen der Zeit, die benötigt wird, grundlegende Prinzipien zu verstehen, und der, die Methode beziehungsweise das Werkzeugs vollständig zu beherrschen. Da die vollständige Beherrschung den Umfang der hier geplanten

Evaluation übersteigt, wird nur die Lernzeit für die grundlegenden Prinzipien betrachtet. Das siebte und letzte Kriterium bezieht sich auf die Reife der evaluierenden Organisation. Da diese Einstufung sich auf die Organisation und nicht den Evaluationsgegenstand bezieht, wird sie an dieser Stelle nicht weiter betrachtet. Einen Überblick über alle relevanten Evaluationskriterien gibt Tabelle 6, wobei die für die vorliegende Evaluation gewählten grün markiert sind.

Art des Einflusses	Qualitativ			Quantitativ	
Art des Evaluations-objekts	Methode		Werkzeug	Kombination aus Methode/Werkzeug	
Granularität	Softwareprodukt als Ganzes			Einzelne Teile des Softwareprodukts	
Einfluss auf den Soft-warelebenszyklus	Eine Phase	Zwei oder mehr Phasen	Entwick-lungsphase	Alle Phasen	Wiederver-wendung
Reife des Evaluations-objekts	In der Entwicklung		Prototyp		Breite Verwendung
Lernzeit	Zeit zum Erlernen der grundlegenden Prinzipien			Zeit zum Erlernendes Werkzeugs / der Methode	

Tabelle 6: Evaluationskriterien (in Anlehnung an Kitchenham 1996)

Neben den Kriterien beschreibt Kitchenham die einzelnen Evaluationsmethoden, die zum Einsatz kommen können: Experimente, Fallstudien und Umfragen. Für die Evaluation in der vorliegenden Arbeit wird eine Umfrage durchgeführt. Die Planung wird im Folgenden erläutert.

7.2 Planung

Die für die Evaluation erstellte Umfrage wurde online bereitgestellt, damit die Anonymität der Testpersonen sichergestellt werden konnte. Mit dem Ziel, die Abbruchquote möglichst gering zu halten, wurde besonderer Wert darauf gelegt, den Umfang der Umfrage und den sich daraus ergebenden Aufwand für die Testpersonen auf ein Minimum zu beschränken. Deshalb wurde auf das Validitäts-stiftende Vorgehen, jedes Konzept mit mehreren Frageitems zu erfassen, bewusst verzichtet. Alle Fragen befinden sich in Anhang A.

Im ersten Fragenblock werden zunächst die fünf grundlegenden Alleinstellungsmerkmale des Lösungsansatzes als einzelne Teile des Softwareprodukts evaluiert: die Unterstützung natürlicher Sprache, der Einbezug von Geschäftsprozessen, das Generieren von Diagramm-Vorschlägen, die Analyse während der Eingabe sowie die Browser-native Oberfläche. Kitchenham schlägt eine vergleichende Methode vor, was heißt, dass eine Methode/ein Werkzeug jeweils mit einer oder mehreren alternativen Methode(n)/Werkzeug(en) verglichen wird. Da der hier entworfene Lösungsansatz Ideen umsetzt, die bisher nicht beziehungsweise nicht für weitverbreitete Methoden oder Werkzeuge eingesetzt wurden, konnte dies nur für die Konzepte erfolgen, bei denen davon ausgegangen werden kann, dass die Testpersonen mit der Alternative vertraut sind. Dazu wurde das Generieren von Diagramm-Vorschlägen der manuellen Erstellung von Diagrammen gegenübergestellt. Außerdem wurde die Browser-native Oberfläche mit einer Desktop-basierten Lösung verglichen. Bei den anderen Funktionen wur-

de jeweils der Vergleich zu einer Software erfragt, die einzelne Funktionalität nicht besitzt. Als letzte Frage in diesem ersten Fragenblock wurde auf das gesamte Produkt bezogen erfragt, ob das Arbeiten mit der Software die Produktivität bei der Softwareentwicklung im Vergleich mit der manuellen Anforderungsanalyse erhöht. Alle Antworten in diesem Fragenblock wurden auf einer 7-stufigen Likert-Skala erfasst; es wurde für die niedrigste Ausprägung der Anker „Stimme überhaupt nicht zu“ und für die höchste Zustimmung der Anker „Stimme voll zu“ vorgegeben.

Der zweiten Fragenblock bezieht sich auf die Software als Ganzes. Nachdem bereits im ersten Fragenblock auf die Produktivitätssteigerung als quantitative Art des Einflusses eingegangen wurde, wurde im zweiten Fragenblock die Benutzerfreundlichkeit der Oberfläche als qualitatives Merkmal erhoben. Dazu werden folgende Aspekte erfragt:

- Zufriedenheit mit dem Umfang der Funktionalität der Software insgesamt.
- Zufriedenheit mit der Benutzerfreundlichkeit der Oberfläche.
- Zufriedenheit mit dem Aufwand, sich in die Bedienung einzuarbeiten.
- Zufriedenheit mit der Schnelligkeit der Software.
- Zufriedenheit mit der Fehlerfreiheit der Software.

Der letzte Aspekt diente ebenfalls dem Zweck, eventuell aufgetretene Fehler zu identifizieren und diese anschließend zu beheben. Deshalb wurden die Testpersonen direkt nach dieser letzten Frage gebeten, eine Fehlerbeschreibung zu hinterlassen, falls Fehler aufgetreten waren. Als Skala wurde erneut eine 7-stufige Likert-Skala eingesetzt, wobei die Ankerpunkte „Gar nicht zufrieden“ und „Sehr zufrieden“ waren.

Die Zielgruppe des Lösungsansatzes und damit auch der Evaluation sind Benutzer, die über Vorkenntnisse im Bereich Softwareentwicklung verfügen. Als Mindestvoraussetzungen ist Grundwissen im Bereich der verschiedenen Diagramme und deren Elementen erforderlich. Deshalb wurden gezielt potenzielle Testpersonen angesprochen, die bereits mindestens den Bachelor (Wirtschafs-)Informatik oder einem anderen Informatik-nahen Studiengang abgeschlossen haben. Weiterhin wurden Testpersonen angesprochen, die seit Längerem in der IT-Branche tätig sind. Auf die Mindestvoraussetzung bezüglich des Vorwissens wurde sowohl in der Einladung als auch in der Einleitung der Umfrage hingewiesen. Um eine möglichst hohe Rücklaufquote zu erreichen, wurde als weiterer Anreiz für die Teilnahme unter allen Testpersonen ein Gutschein im Wert von 20 € verlost.

Da der Link zur Teilnahme frei verfügbar war, wurden bereits während der Planung Maßnahmen zur Qualitätssicherung der Ergebnisse berücksichtigt. Dafür wurde eine Selbsteinschätzung hinsichtlich des Vorwissens erhoben. Die Testpersonen wurden gebeten, ihr Vorwissen im Bereich der Softwareentwicklung auf einer 5-stufigen Skala einzuschätzen; die Antwortoptionen waren „keine Kenntnisse“, geringe Kenntnisse“, „Grundkenntnisse“, „gute Kenntnisse“ und „sehr gute Kenntnisse“. Zusätzlich bestand die Möglichkeit, keine Angaben zu dieser Frage zu machen. Als weitere Maßnahme zur Sicherung der Qualität wurden auf technischer Ebene die Zeiten registriert, die die Testpersonen für die einzelnen Abschnitte der Umfrage benötigten. Weiterhin wurde der Browser-Typ erfasst, da die Implementierung für Firefox optimiert wurde. Die Testpersonen wurden vor der Umfrage auf diese Einschränkun-

gen hingewiesen, die Erfassung diente damit lediglich der Möglichkeit, Ursachen für Probleme bei der Durchführung auszuschließen. Am Ende der Umfrage konnten die Testpersonen in Form eines offenen Antwortformats optional weiteres Feedback hinterlassen.

7.3 Durchführung

Damit die Evaluierenden auf den Prototyp zugreifen konnten, wurde die Software auf einem Web-Server zur Verfügung gestellt. In die Umfrage wurde ein Link eingebettet, der beim Aufruf auf der Seite des Prototyps dazu führte, dass jeweils ein neues Benutzerkonto mit einem Evaluationsprojekt angelegt wird. Damit für jede Testperson ein eigenes Benutzerkonto angelegt wird, wurde als Übergabeparameter die Sitzungsnummer der Umfrageplattform verwendet, welche nur einmalig auftreten kann.

Um sich mit dem Prototyp vertraut zu machen, wurde die Testperson aufgefordert, entweder eigenständig mit der Software zu arbeiten oder einem Beispielszenario zu folgen. Dieses beinhaltete zunächst die Navigation zu einer Anforderungsspezifikation, die bereits für das Evaluationsprojekt angelegt wurde. Der Text dieser Anforderungsspezifikation entspricht dem, der auch im Rahmen der Implementierung gezeigt wurde (vgl. Abbildung 34), allerdings zunächst ohne den letzten Satz. Für diese Ansicht wurde die Testperson gebeten, folgende Aktionen durchzuführen:

- Wählen der Optionen für das Tagging
- Anklicken eines bestimmten Wortes zum Anzeigen der Zusatzinformationen
- Testen der Analyse während der Eingabe
- Generieren eines Diagramms
- Testen der Editiermöglichkeiten

Als Text für das Testen der Analyse während der Eingabe wurde der letzte Satz „He calls the customer.“ verwendet. Dieser ist kurz genug, damit die Testperson beim Wechsel zwischen den Instruktionen innerhalb der Umfrage und der Eingabe in den Prototyp sich ihn gut merken kann. Darüber hinaus demonstriert der Satz die Koreferenzanalyse anhand des ersten Wortes „He“. Wenn die Option für das Tagging eines Klassendiagramms gewählt ist, wird zudem für das Wort „calls“ zunächst die farbliche Markierung als Operation, dann als Beziehung des Klassendiagramms angezeigt.

Die Testperson konnte den Prototyp unbegrenzt, also über das Beispielszenario hinaus, weiterverwenden, wobei der volle Funktionsumfang zur Verfügung stand. Die Fragen waren auf derselben Seite wie das Beispielszenario platziert, sodass die Testperson zwischen der Beschreibung, dem Prototyp und den Fragen nach Bedarf wechseln konnte.

7.4 Ergebnisse

Insgesamt lag die Zahl der Testpersonen bei 63, wobei sechs Datensätze ausgeschlossen wurden; drei aufgrund von Unvollständigkeit und drei aufgrund eines eindeutigen Antwortmusters, bei dem für alle Fragen der gleiche Wert ausgewählt wurde. Damit lag die Anzahl der auswertbaren Datensätze bei N = 57. Alle Testpersonen machten Angaben zu ihrem Vorwissen im Bereich der Softwareentwicklung, wobei alle Testpersonen abgaben, mindestens geringes Vorwissen zu besitzen. 47,4 % der Testpersonen gaben an, gute Vorkenntnisse zu be-

sitzen, weitere 35,1 % schätzen ihre Vorkenntnisse als sehr gut ein. Es wurden also mit überwiegender Mehrheit Testpersonen mit fortgeschrittenen Kenntnissen in der Softwareentwicklung erreicht. Da auch geringes Vorwissen für die Verwendung des Prototyps ausreicht, wurden keine Datensätze aufgrund dieser Angabe eliminiert.

Tabelle 7 fasst die statistischen Kennzahlen der Umfrageauswertung zusammen. Der erste Fragenblock zur Funktionalität wurde anhand der Skala von 1 („Stimme überhaupt nicht zu“) bis 7 („Stimme voll zu“) ausgewertet, der zweite Fragenblock zur Benutzerzufriedenheit anhand der Skala von 1 („Gar nicht zufrieden“) bis 7 („Sehr zufrieden“). Neben dem Mittelwert werden zusätzlich auch der Median und der Modalwert angegeben. Diese Werte verdeutlichen, dass nicht nur im Durchschnitt, sondern auch für die anderen beiden Kennwerte generell hohe bis sehr hohe Werte für die Zustimmung beziehungsweise Zufriedenheit erreicht wurden. Auch für den geringsten Wert wurde mit 5,05 für Frage F5 immer noch ein Wert deutlich oberhalb des neutralen Punkts von 4 erzielt.

Fragenblock	Nr.	Frage zu:	Mittelwert	Median	Modalwert
Funktionalität	F1	Generieren von Vorschlägen für die Diagramme	6,28	7	7
	F2	Semantischer Analyse während der Eingabe	6,14	6	7
	F3	Direkter Verbindung zu Geschäftsprozessen über BPMN	5,95	6	6
	F4	Eingabe in Form von Text (in natürlicher Sprache)	6,02	6	6
	F5	Bearbeitung direkt im Browser	5,05	5	7
	F6	Insgesamt: Produktivität bei der Softwareentwicklung	5,70	6	6
	F7	Insgesamt: Qualität (z. B. Konsistenz und Nachverfolgbarkeit)	5,57	6	6
Benutzerzufriedenheit	B1	Benutzerfreundlichkeit der Oberfläche	5,56	6	6
	B2	Umfang der Funktionalität der Software insgesamt	5,56	6	6
	B3	Aufwand, sich in die grundlegende Bedienung einzuarbeiten	6,00	6	6
	B4	Schnelligkeit der Software	6,09	6	7
	B5	Fehlerfreiheit der Software	5,36	6	6

Tabelle 7: Statistische Kennzahlen der Umfrage

In einem Datensatz fehlten die Antworten auf die beiden Fragen F6 und F7, in einem anderen die auf B5. Da die weiteren Angaben verwertbar waren, wurden diese Datensätze in die Auswertung einbezogen, wobei die Ergebnisse der betreffenden Fragen auf Basis der verbleibenden Datensätze berechnet wurden.

Die Hauptfunktionalitäten der Generierung von Diagrammvorschlägen, der semantischen Analyse während der Eingabe, der direkten Verbindung zu Geschäftsprozessen und der Eingabe in natürlicher Sprache wurden mit Mittelwerten zwischen 5,95 und 6,28 bewertet. Damit kann für diese Funktionalitäten im Durchschnitt auf eine hohe Zustimmung bei den Testpersonen geschlossen werden. Die Abweichung zu dem deutlich niedrigeren Mittelwert von 5,05 für die direkte Eingabe im Browser könnte zum einen so verstanden werden, dass die Testpersonen zwar die Eingabe über den Browser präferieren, diese Präferenz gegenüber einer Desktop-basierten Lösung aber nicht so stark ausgeprägt ist. Zum anderen könnte die Bewertung auch darauf hinweisen, dass Bedenken bestanden. Eine Testperson gab als Antwort auf die offene Frage am Ende der Umfrage an, dass Sicherheitsbedenken bestehen für die Verwendung im Unternehmen. Da Anforderungsspezifikationen sensible Daten sind, müssen vor einem produktiven Einsatz entsprechende Sicherheitsmechanismen etabliert werden. Weiterhin wäre es denkbar, den Einsatzbereich auf das Intranet eines Unternehmens zu beschränken.

Insgesamt wurde der Beitrag des Konzepts und des Prototyps hinsichtlich der möglichen Produktivitäts- und Qualitätssteigerung mit einem Durchschnitt von 5,70 beziehungsweise 5,57 deutlich als positiv bewertet. Auch in den offenen Antworten wurde von einer Testperson explizit angesprochen, dass sie eine Zeitersparnis durch die gebotenen Funktionalitäten erwarten würde. Eine andere Testperson kritisiere die fehlende Round-Trip-Unterstützung. Diese Funktionalität wurde bereits oben angesprochen; sie ist wünschenswert, geht aber über die prototypische Implementierung hinaus. Eine weitere Funktionalität, die von einer Testperson vorgeschlagen wurde, war die, dass der zugehörige Satz aus der Anforderungsspezifikationen eingeblendet wird, wenn beispielsweise eine Klasse oder ein Attribut im Diagramm angeklickt wird. Dieser Vorschlag würde den Benutzer bei der Bearbeitung der Diagramme unterstützen und ist für eine Weiterentwicklung eine sinnvolle Erweiterung.

Die Mittelwerte für die Benutzerzufriedenheit lagen mit 5,36 bis 6,09 auf vergleichbarer Höhe mit denen der Funktionalität. Die Spanne zwischen den Mittelwerten ist insgesamt nicht sehr hoch, als besonders gut bewertet wurden die Schnelligkeit und der Aufwand für die Einarbeitung in die Bedienung. Auch die Benutzerfreundlichkeit der Oberfläche, der Umfang der Funktionalität und die Fehlerfreiheit wurden deutlich positiv bewertet. Daraus kann geschlossen werden, dass der Prototyp die Anforderungen an eine gute Benutzerfreundlichkeit insgesamt erfüllt. Als Kritikpunkte und Verbesserungsansätze traten hier in den Antworten auf die offene Frage hervor, dass eine Zoomfunktion für die Diagramme hilfreich wäre. Insbesondere für große Diagramme wäre diese eine sinnvolle Ergänzung; dies wurde von einer Testperson angesprochen. Von einer anderen Testperson wurde das Layout der BPMN-Diagramme kritisiert, von einer weiteren die Anzeige von Beziehungen im Klassendiagramm. Treten zwischen zwei Klassen mehrere Beziehungen auf, so werden diese übereinandergezeichnet. Beide Probleme sind auf die verwendete Graphikbibliothek zurückzuführen, da die Algorithmen für die Anordnung der Diagrammelemente durch diese vorgegeben werden. Ein Austausch der Bibliothek könnte diese Probleme beheben.

7.5 Zusammenfassung

Insgesamt wurden sowohl die Funktionalitäten als auch die Benutzerfreundlichkeit des Prototyps als positiv bewertet. Dies ist nicht nur an den Mittelwerten deutlich oberhalb des neutralen Punkts der Skala abzulesen, sondern auch an den hohen Ergebnissen für die Modal- und Medianwerte. Im Vergleich zu den anderen Funktionalitäten wurde die Browser-basierte Bearbeitung nicht ganz so gut bewertet.

Aus den Eingaben in die freien Eingabefelder am Ende der Umfrage waren berechtigte Kritikpunkte erkennbar. Aus diesen lassen sich sinnvolle Erweiterungen wie ein Sicherheitskonzept und die Bearbeitungsmöglichkeit in Form eines grafischen Editors ableiten. Aufgrund der geringen Anzahl an Nennungen bestimmter Aspekte innerhalb der offenen Antwortformate sollten jedoch keine Rückschlüsse auf den Zusammenhang zwischen der Begründung einzelner Testpersonen und der Bewertung einzelner Funktion erfolgen. Auch insgesamt betrachtet, lässt die Zahl der Testpersonen keine Generalisierungen zu.

8 Zusammenfassung

Die vorliegende Arbeit beantwortet die Frage danach, wie im Rahmen der Nachverfolgbarkeit semantische Technologien dazu eingesetzt werden können, die Trennung von natürlichsprachlichen Anforderungsspezifikationen und weiteren Artefakten zu überwinden. Der vorgestellte Lösungsansatz befasst sich dazu im Einzelnen mit ausgewählten Fragestellungen bezüglich der Auswahl der Artefakte, der Gestaltung der Nachverfolgbarkeitsbeziehungen, der Automatisierbarkeit der semantischen Analyse sowie der Präsentation der Ergebnisse. Als Artefakte einbezogen werden schriftliche Anforderungsspezifikationen in natürlicher Sprache, Klassendiagramme als Vertreter der Strukturdiagramme, Anwendungsfalldiagramme als Vertreter der Verhaltensdiagramme sowie BPMN-Diagramme zur Abbildung von Geschäftsprozessen. Der Lösungsansatz unterstützt bidirektionale Nachverfolgbarkeitsbeziehungen zwischen diesen ausgewählten Artefakten. Der erstellte Algorithmus erlaubt die Extraktion vorhandener Informationen aus den Anforderungsspezifikationen, sodass ausgewählte Elemente direkt für die verschiedenen Diagramme vorgeschlagen werden. Die Interaktion mit dem System erfolgt über den Web-Browser.

Im Folgenden werden die Ergebnisse zusammengefasst; anschließend wird der Beitrag der vorliegenden Arbeit zur Forschung beschrieben. Der Ausblick auf eine mögliche weitere Forschung schließt das Kapitel ab.

8.1 Ergebnisse

Die Basis dieser Arbeit bilden die Themenfelder der Nachverfolgbarkeit und der semantischen Technologien. Zunächst wurden die Grundlagen der Nachverfolgbarkeit herausgearbeitet, wobei vertieft auf die Artefakte eingegangen wurde. Dabei wurde die Bedeutung von schriftlichen Anforderungen und UML-Modellen als Quasi-Standard für die Softwaremodellierung im Rahmen der Softwareentwicklung verdeutlicht. Für die Unterstützung der Nachverfolgbarkeit wurden mehrere Werkzeuge vorgestellt, die die Verknüpfung von Artefakten untereinander ermöglichen. Anschließend wurden die semantischen Technologien als zweites Themenfeld eingeführt. Bedeutsam war es hier, herauszustellen, dass natürliche Sprache komplex und die Verarbeitung mit Herausforderungen verbunden ist. Gleichzeitig existiert auf diesem Gebiet eine Vielzahl an Forschungsarbeiten, die sich mit den verschiedenen Schritten der Verarbeitung befassen. Diese reichen von Technologien für die Textvorbereitung, wie die Tokenisierung, über basale Technologien, wie die Erkennung benannter Entitäten, bis hin zu fortgeschrittenen Technologien, wie die Analyse typisierter Beziehungen und die Koreferenzanalyse. Umgesetzt werden diese Konzepte bereits durch eine große Auswahl an Werkzeugen und mit einem breiten Spektrum an Einsatzmöglichkeiten.

Es wurde dann untersucht, inwiefern bereits semantische Technologien für die Nachverfolgbarkeit und insbesondere die Analyse von schriftlichen Anforderungsspezifikationen eingesetzt werden. Dabei stellte sich heraus, dass es durchaus Ansätze gibt, die bereits die Idee umsetzen, Informationen aus schriftlichen Anforderungen zu extrahieren und für eine kleine Anzahl an Diagrammelementen zu verwenden. Allerdings beruhen diese Ansätze entweder auf schwer automatisierbaren Regeln, oder sie verwenden nur sehr basale semantische Tech-

nologien. Des Weiteren werden Geschäftsprozesse bei der Betrachtung der Artefakte vernachlässigt. Als Konsequenz aus diesen Erkenntnissen wurde ein Lösungsansatz erarbeitet, der auf fortgeschrittenen semantischen Technologien beruht. Das Herzstück des Ansatzes ist das Beziehungskonzept. Dieses baut auf der Technologie der typisierten Beziehungen auf. Der Algorithmus kann auf dieser Grundlage zwischen syntaktischen Subjekten und Objekten differenzieren, die Zugehörigkeit von Adjektiven zu einem Substantiv erkennen sowie die weiteren Beziehungen von Wörtern im Satz untereinander bestimmen, wie solche zwischen einem Hauptverb und seinen syntaktischen Subjekten und Objekten. Als Ergebnis dieser Analyse werden beispielsweise nur syntaktische Subjekte als Handelnde in einem Satz als Aktoren für Anwendungsfalldiagramme vorgeschlagen. Weiterhin können Verben als Beziehung im Klassendiagramm identifiziert werden, wenn sie in Beziehung zu zwei Substantiven stehen. Außerdem werden mithilfe der Koreferenzanalyse Beziehungen über Satzgrenzen hinweg aufgedeckt. Wird in einer schriftlichen Anforderungsspezifikation mehrmals ein Substantiv verwendet, so wird es nur einmalig als Element vorgeschlagen, beispielsweise als Klasse im Klassendiagramm. Dies funktioniert auch für die Referenz durch ein Personalpronomen. Die grundlegende Idee des Lösungsansatzes ist es auch hier, die Beziehungen von Wörtern zu verwenden, damit die in den schriftlichen Anforderungsspezifikationen vorhandenen Informationen bestmöglich genutzt werden können. Das Ziel der Analyse ist es, dem Benutzer Vorschläge für Diagrammelemente anzubieten. Dazu werden direkt während der Eingabe einzelne Wörter im Text farblich hinterlegt. Dadurch können diese Vorschläge für die Diagrammelemente visualisiert werden. Es ist nicht erforderlich, den gesamten Text einer schriftlichen Anforderungsspezifikation fertigzustellen, bevor eine Analyse durchgeführt werden kann; die Analyse erfolgt direkt während der Eingabe. Der Benutzer kann bereits an dieser Stelle Anpassungen vornehmen oder wahlweise auch nach der Generierung eines Diagramms. An jeder Stelle ist dabei die Nachverfolgbarkeit sichergestellt, da Diagramme direkt zu der zugehörigen schriftlichen Anforderungsspezifikation zugeordnet werden können und vice versa.

Im Rahmen der Implementierung wurde gezeigt, dass das Konzept in Form eines Prototyps umsetzbar ist. Hierbei stand vor allem der Aspekt der Ergebnispräsentation im Vordergrund. In letzter Zeit ist ein Trend zu leichtgewichtigen (engl. lightweight) Lösungen erkennbar. Das Angebot einer Browser-nativen Oberfläche kann als solche bezeichnet werden (Thum et al. 2009), da weder eine Installation vor Ort noch eine Konfiguration notwendig ist. Damit die Analyse während der Eingabe dynamisch erfolgen kann, wurde Ajax eingesetzt. Dies ermöglicht es, dass die Oberfläche für den Benutzer verwendbar bleibt. Die Eingabe wird an den Server geschickt, der das Ergebnis berechnet. Der Browser zeigt das Ergebnis an, sobald es vorliegt, wobei der Benutzer seine Eingabe kontinuierlich fortsetzen kann, da die eben beschriebenen Prozesse im Hintergrund ablaufen. Damit können die farblichen Markierungen fortlaufend aktualisiert werden, und der Benutzer sieht direkt nach der Eingabe eines Wortes, welches Diagrammelement vom Algorithmus vorgeschlagen wird. Während der Verwendung wird damit eine schnelle Rückmeldung über die Anforderungsspezifikation ermöglicht.

Die Evaluation des Konzepts und des Prototyps zeigt, dass die Funktionalitäten und die Benutzerfreundlichkeit von den Testpersonen sehr positiv aufgenommen wurden. Für alle Teilfragen lagen die erreichten Werte sind deutlich oberhalb des neutralen Punkts der verwendeten Skala. Auch die Median- und die Modalwerte zeigten, dass fast immer die beste oder

zweitbeste der möglichen Skalenausprägung gewählt wurde. Eine Ausnahme bildete die Bearbeitung direkt im Browser, welche einen Wert deutlich unterhalb der anderen Fragen erreichte. Da der Wert über dem neutralen Punkt der Skala lag, stellt das Ergebnis keinesfalls eine Ablehnung dar. Auch insgesamt stimmten die Testpersonen in hohem Maße der Aussage zu, dass die Arbeit mit der Software die Produktivität und Qualität in der Softwareentwicklung verbessern kann.

8.2 Beitrag zur Forschung

Der hier erarbeitete Lösungsansatz basiert auf fortgeschrittenen semantischen Technologien und steht im Einklang mit den bisherigen Arbeiten auf dem Gebiet der Nachverfolgbarkeit. Der Beitrag zur Forschung zeichnet sich vor allem durch die Kombination der verschiedenen Technologien zu einem integrierten Lösungsansatz aus, der die Unterstützung der Nachverfolgbarkeit an entscheidenden Stellen verbessert. Hier ist zum Ersten die Analyse in Echtzeit zur Überwindung der Trennung der Artefakte zu nennen. Bisher werden Artefakte häufig räumlich getrennt. Zur Bearbeitung einer schriftlichen Anforderungsspezifikation und von Diagrammen werden häufig unterschiedliche Werkzeuge verwendet. Dabei besteht die Gefahr, dass die Artefakte die Nachverfolgbarkeitsbeziehungen zueinander verlieren. Es gibt bereits Werkzeuge, die den Benutzer bei der Pflege und Erstellung dieser Nachverfolgbarkeitsbeziehungen unterstützen. Jedoch überwindet keines davon die zeitliche Trennung, die zwischen schriftlichen Anforderungsspezifikationen und den weiteren Artefakten besteht. Mit der Erstellung der Artefakte der Anforderungsanalyse wird erst begonnen, wenn die Formulierung der Anforderungsspezifikationen abgeschlossen wurde. Dieses Vorgehen ist im Rahmen der bisherigen Ansätze auch sinnvoll, da die manuelle Extraktion von Informationen für die Diagrammerstellung aus Anforderungsspezifikationen aufwendig ist. Der hier vorgestellt Lösungsansatz unterstützt diese Informationsextraktion durch einen Algorithmus. Dieser stellt den höchsten Grad der Automatisierung dar, der mit dem derzeitigen Stand semantischer Technologien umsetzbar ist. Mithilfe des Algorithmus gewinnt der Benutzer einen Überblick über die in den Anforderungsspezifikationen enthaltenen Informationen, da Diagrammelemente ohne zusätzlichen Aufwand vorgeschlagen werden. In Verbindung mit der Unterstützung der Nachverfolgbarkeit kann der Benutzer damit schnell die Konsistenz der Artefakte untereinander prüfen und bei Bedarf Anpassungen auch in den Anforderungsspezifikationen vornehmen. All dies ist zu jedem Zeitpunkt möglich, also auch, bevor die Anforderungsspezifikationen abschließend fertiggestellt sind.

Damit ist die räumliche und zeitliche Trennung zwischen den schriftlichen Anforderungsspezifikationen und den weiteren Artefakten überwunden. Bisherige Ansätze fokussierten vor allem eine punktuelle Unterstützung der Transformation meist in Klassendiagramme. Die verwendeten Regeln waren nur sehr schwer automatisierbar und machten damit eine manuelle Umsetzung erforderlich. In den wenigen Arbeiten, die semantische Technologien verwenden, wurde vor allem auf der Wortart aufgebaut. Nach dem Stand der Recherche der Autorin gibt es zu diesem Zeitpunkt keine mit dem hier vorgestellten Lösungsansatz vergleichbare Arbeit. Das Beziehungskonzept und die Verwendung der Koreferenzanalyse ermöglichen die Analyse von Wörtern untereinander und stellen einen vollständig neuen Ansatz zur Analyse von Anforderungsspezifikationen dar. Darüber hinaus wird auch die bisher vernachlässigte Verbindung zu den Geschäftsprozessen umgesetzt. Deren Bedeutung für den Softwareentwicklungs-

prozess wird von Nordheimer et al. (2012) hervorgehoben. Der Lösungsansatz zeigt, wie gut sich Geschäftsprozesse als ein Artefakt in die gesamte Betrachtung der Nachverfolgbarkeit einbeziehen lassen.

Eingesetzt werden kann der Lösungsansatz für alle Entwicklungsprojekte, die die Nachverfolgbarkeit unterstützen. Das einfache Herstellen der Verbindungen von Artefakten untereinander erlaubt es, schnell einen Überblick über die Zusammenhänge zu bekommen und die Konsistenz zu überprüfen. Die Bedeutung der Nachverfolgbarkeit steigt mit der Komplexität des Systems (Ramesh und Jarke 2001); der Lösungsansatz kann auch für kleine und mittlere Projekte eingesetzt werden, da die Einstiegshürden so niedrig wie möglich gehalten wurden. Benutzer können miteinander kooperieren, beispielsweise können Benutzer Anforderungsspezifikationen ergänzen, die dann von einem erfahrenen Mitglied des Teams weiterbearbeitet werden. Allgemein bilden alle Anforderungsspezifikationen – egal ob in Form von Text oder Diagrammen – gemeinsam die Grundlage für die Kommunikation zwischen den Beteiligten (Daniels und Bahill 2004). Damit unterstützt der Lösungsansatz die Zusammenarbeit, da alle relevanten Informationen an einem zentralen Punkt erfasst werden. Dies ist besonders in verteilten Entwicklungsprojekten von großer Bedeutung, da die Möglichkeit zur asynchronen Kommunikation die Verständigung über verschiedene Zeitzonen hinweg erleichtert (Hildenbrand et al. 2007).

Insgesamt werden durch den Lösungsansatz Bruchstellen reduziert und damit die Möglichkeiten für das Auftreten von Fehlern reduziert. Ebenfalls wird der Prozess beschleunigt. Verglichen mit der manuellen Analyse der schriftlichen Anforderungsspezifikationen sowie der manuellen Erstellung und Pflege der Nachverfolgbarkeitsbeziehungen, erleichtert der Lösungsansatz dem Benutzer alle damit in Verbindung stehenden Aufgaben.

8.3 Ausblick

Eine vollständige Automatisierung der Analyse von schriftlichen Anforderungsspezifikationen mit dem Ergebnis einer Generierung vollständiger Diagramme ist aus zwei Gründen nicht zu erwarten. Erstens ist Software komplex, und die verschiedenen Artefakte besitzen allein deshalb jeweils ihre Daseinsberechtigung, da sie sich mit Absicht unterscheiden, weil sie anderenfalls in ein Artefakt zusammengefasst werden könnten (Brooks 1986).

Ein zweiter Grund liegt in der Komplexität natürlicher Sprache, wobei auf diesem Gebiet zurzeit intensiv geforscht wird. Beispielsweise werden Angebote für Übersetzungen immer weiter verbessert. Bei entsprechend hoher Qualität der Ergebnisse wäre es damit denkbar, den hier entwickelten Ansatz in der Art zu erweitern, dass schriftliche Anforderungsspezifikationen in mehreren Sprachen analysiert werden können. Es wäre zu prüfen, ob eine Hinübersetzung ins Englische mit anschließender Rückübersetzung ein akzeptables Ergebnis liefert und damit die Anpassung der Regeln wegfallen lässt. Andernfalls können, wie angesprochen, auch die Regeln für jede Sprache direkt angepasst werden. Ebenfalls wird an der direkten Verbindung zwischen natürliche Sprache und formalisierten Notationen gearbeitet. Es liegt eine aktuelle Veröffentlichung vor (Kushman und Barzilay 2013), die natürliche Sprache direkt in reguläre Ausdrücke überführt. Als übergeordnetes Ziel kann hier gesehen werden, dass irgendwann Programme direkt in natürlicher Sprache geschrieben werden können. Wie viel Arbeit auf diesem Gebiet allerdings noch notwendig ist, zeigt sich in der Tatsache, dass be-

reits Mitte der 1990er-Jahre erste Ansätze für die Formulierung von SQL-Anfragen in natürlicher Sprache vorgestellt wurden (z. B. Androutsopoulos et al. 1994). Durchgesetzt haben sich diese Ansätze bisher allerdings nicht.

Eine wichtige Forderung, die durch die vorliegende Arbeit unterstrichen werden soll, ist die, dass schriftliche Anforderungsspezifikationen mit den weiteren Artefakten verbunden werden sollen. Häufig wird der Prozess der Anforderungserhebung losgelöst von den weiteren Artefakten betrachtet, womit die Nachverfolgbarkeit entweder unberücksichtigt bleibt oder erst mit zusätzlichem Aufwand erstellt werden muss. Daher sollten von Anfang an Grundlagen für die sinnvolle Verbindung von Artefakten gelegt werden und, falls möglich und wie in der vorliegenden Arbeit gezeigt, diese Verbindungen so gut wie möglich durch semantische Technologien unterstützt werden.

Anhang

A Umfrage

Bewertung der Funktionalitäten

	Stimme **überhaupt nicht** zu						Stimme **voll** zu
Das **Generieren von Vorschlägen für die Diagramme** bietet einen Vorteil gegenüber der **manuellen** Erstellung von Diagrammen:	○	○	○	○	○	○	○
Die **semantische Analyse** während der Eingabe bietet einen Vorteil gegenüber einer Software, die keine Analysefunktion hat:	○	○	○	○	○	○	○
Die direkte **Verbindung zu Geschäftsprozessen** über BPMN bietet einen Vorteil gegenüber einer Software, die das nicht unterstützt:	○	○	○	○	○	○	○
Die **Eingabe in Form von Text** (also natürlicher Sprache) bietet einen Vorteil gegenüber einer Software, die das nicht unterstützt:	○	○	○	○	○	○	○
Die **Bearbeitung direkt im Browser** bietet einen Vorteil gegenüber einer **Desktop**-Anwendung:	○	○	○	○	○	○	○
Insgesamt kann das Arbeiten mit der Software die **Produktivität** bei der Softwareentwicklung steigern, verglichen mit der manuellen Anforderungsanalyse.	○	○	○	○	○	○	○
Insgesamt kann die Arbeit mit der Software zu einer höheren **Qualität** (z.B. Konsistenz und Nachverfolgbarkeit) beitragen, verglichen mit der manuellen Anforderungsanalyse.	○	○	○	○	○	○	○

Bewertung der Benutzerzufriedenheit

	Gar nicht zufrieden						**Sehr** zufrieden
Wie zufrieden bist Du mit der Benutzerfreundlichkeit der Oberfläche?	○	○	○	○	○	○	○
Wie zufrieden bist Du mit dem Umfang der Funktionalität der Software insgesamt?	○	○	○	○	○	○	○
Wie zufrieden bist Du mit dem Aufwand Dich in die grundlegende Bedienung einzuarbeiten?	○	○	○	○	○	○	○
Wie zufrieden bist Du mit der Schnelligkeit der Software?	○	○	○	○	○	○	○
Wie zufrieden bist Du mit der Fehlerfreiheit der Software?	○	○	○	○	○	○	○

B Umfragedaten

	Fragen Nr.											
Response ID	F1	F2	F3	F4	F5	F6	F7	B1	B2	B3	B4	B5
15	2	3	3	3	2	2	4	4	4	6	6	3
16	7	7	6	7	7	6	7	6	7	7	6	7
17	7	6	7	6	5	7	7	7	6	7	7	5
18	6	5	7	5	1	5	5	6	5	7	6	6
19	7	7	7	7	5			5	7	6	7	5
20	7	6	6	7	4	4	5	6	6	6	6	6
23	7	7	6	6	4	7	7	6	7	7	7	7
25	7	7	7	7	6	6	6	5	6	4	4	5
27	5	6	5	6	7	5	6	5	5	5	6	6
28	6	6	5	5	6	6	4	5	5	6	3	3
31	6	6	6	7	5	6	6	6	5	5	6	4
32	5	6	5	6	2	5	5	5	3	4	6	4
33	7	7	7	7	7	4	4	7	7	7	7	7
34	7	7	7	7	7	4	7	7	4	7	7	7
37	7	7	6	6	3	6	6	6	5	6	7	6
38	7	7	6	6	5	6	6	6	7	6	7	7
39	6	7	7	6	4	6	4	5	5	6	5	3
40	7	7	7	7	5	7	7	7	7	4	7	7
41	7	7	7	7	4	7	7	4	5	6	6	6
44	6	7	5	6	6	7	4	6	4	6	7	6
47	6	6	5	7	6	7	5	6	5	7	6	5
48	6	6	6	5	5	5	4	5	3	6	6	3
49	6	4	4	5	7	7	7	7	6	7	7	5
51	7	7	6	6	4	6	6	6	5	7	6	6
52	6	6	6	7	7	5	5	6	6	7	7	7
54	6	6	5	5	3	5	4	4	5	5	6	6
55	7	7	7	6	5	5	6	6	5	5	6	7
56	7	7	5	6	3	6	5	6	6	6	7	6
58	7	6	5	6	7	6	6	6	6	7	6	6
59	6	4	6	4	3	5	6	6	3	5	6	1
60	7	6	6	7	4	7	6	5	6	6	3	5
61	7	5	5	4	7	5	4	6	6	6	4	3
62	6	7	6	6	4	4	5	2	6	3	6	6
65	5	7	6	7	2	5	6	6	7	6	7	5
67	6	6	5	7	6	6	6	7	6	7	6	6
70	5	3	5	5	7	4	5	4	4	5	7	4
74	6	7	7	7	6	5	6	7	6	6	5	6
77	7	6	7	7	4	5	6	3	5	4	6	5
79	6	6	6	6	3	7	5	2	6	5	6	6
82	7	6	6	6	7	6	6	7	4	7	7	7
84	6	6	6	5	4	5	6	5	6	6	6	4
85	7	7	7	7	7	7	7	6	5	6	7	7
93	7	7	6	7	6	6	7	5	6	6	7	7
99	7	7	5	6	3	6	4	6	7	7	7	7
101	6	6	7	7	7	7	6	6	7	7	6	7
104	6	6	6	6	6	6	4	5	4	6	5	3

105	7	5	5	6	3	6	6	5	6	6	7	6
107	6	6	6	6	6	6	6	5	5	6	5	5
109	5	5	6	6	6	6	6	5	6	6	5	
110	7	7	7	4	2	7	6	5	6	7	7	5
111	7	7	7	7	7	7	7	5	5	7	5	4
114	7	7	6	6	7	6	6	6	7	6	7	6
116	7	5	7	5	7	6	4	7	7	6	6	5
117	6	5	6	6	7	6	6	6	6	6	5	6
119	7	7	7	7	7	7	7	7	7	7	7	3
122	5	7	5	6	4	5	5	7	6	7	7	7
123	3	4	4	3	4	3	3	5	5	5	5	3

Literaturverzeichnis

Abbott, R. (1983): Program design by informal English descriptions. In: Communications of the ACM. 26 (11), S. 882–894.

Acedański, S. (2010): A morphosyntactic Brill Tagger for inflectional languages. In: Proceedings of the 7th International Conference on Advances in Natural Language Processing. S. 3–14.

Agarwal, R. und Sinha, A. P. (2003): Object-Oriented Modeling with UML: A Study of Developers' Perceptions. In: Communications of the ACM. 46 (9), S. 248–256.

Aizenbud-Reshef, N., Nolan, B. T., Rubin, J. und Shaham-Gafni, Y. (2006): Model traceability. In: IBM Systems Journal. 45 (3), S. 515–526.

Aizenbud-Reshef, N., Paige, R. F., Rubin, J., Shaham-Gafn, Y. und Kolovos, D. S. (2005): Operational semantics for traceability. In: European Conference on Modelling Foundations and Applications, S. 1–8.

Allweyer, T. (2009): BPMN 2.0-Business Process Model and Notation: Einführung in den Standard für die Geschäftsprozessmodellierung. 2. Auflage. Books on Demand.

Almeida, J. B., Frade, M. J., Pinto, J. S. und Sousa, S. M. de (2011): Rigorous Software Development: An Introduction to Program Verification. London: Springer.

Almeida, J. P. A., Iacob, M.-E. und van Eck, P. (2007): Requirements traceability in model-driven development: Applying model and transformation conformance. In: Information Systems Frontiers. 9 (4), S. 327–342.

Ambriola, V. und Gervasi, V. (1997): Processing natural language requirements. In: Proceedings of the 12th IEEE International Conference on Automated Software Engineering. S. 36–45.

Ambriola, V. und Gervasi, V. (2006): On the Systematic Analysis of Natural Language Requirements with CIRCE. In: Automated Software Engineering. 13 (1), S. 107–167.

Amdouni, S., Abdessalem, B. W. K. und Bouabid, S. (2011): Semantic annotation of requirements for automatic UML class diagram generation. In: IJCSI International Journal of Computer Science Issues. 8 (3), S. 259–264.

Van Amstel, M. F., Van den Brand, M. G. J. und Engelen, L. J. P. (2011): Using a DSL and fine-grained model transformations to explore the boundaries of model verification. In: Secure Software Integration & Reliability Improvement Companion. S. 120–137.

Amtrup, J. W. (2010): Aspekte der Computerlinguistik. In: Carstensen, K.-U., Ebert, C., Ebert, C. et al. (Hrsg.) Computerlinguistik und Sprachtechnologie: Eine Einführung. 3. Auflage. Heidelberg: Spektrum, S. 1–17.

Androutsopoulos, I., Ritchie, G. D., Thanisch, P. und Road, M. (1994): Natural Language Interfaces to Databases – An Introduction. In: Journal of Natural Language Engineering. 1 (1), S. 29–81.

Anquetil, N., Kulesza, U., Mitschke, R., Moreira, A., Royer, J.-C., Rummler, A. und Sousa, A. (2010): A model-driven traceability framework for software product lines. In: Software & Systems Modeling. 9 (4), S. 427–451.

Antoniol, G., Canfora, G., Casazza, G., Lucia, A. De und Merlo, E. (2002): Recovering Traceability Links between Code and Documentation. In: IEEE Transactions on Software Engineering. 28 (10), S. 970–983.

Arkley, P. und Riddle, S. (2005): Overcoming the traceability benefit problem. In: 13th IEEE International Conference on Requirements Engineering. S. 385–389.

Assawamekin, N. (2010): An Ontology-Based Approach for Multiperspective Requirements Traceability between Analysis Models. In: 2010 IEEE/ACIS 9th International Conference on Computer and Information Science. S. 673–678.

Assawamekin, N., Sunetnanta, T. und Pluempitiwiriyawej, C. (2009a): Deriving Traceability Relationships of Multiperspective Software Artifacts from Ontology Matching. In: 2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing. S. 549–554.

Assawamekin, N., Sunetnanta, T. und Pluempitiwiriyawej, C. (2009b): MUPRET: An Ontology-Driven Traceability Tool for Multiperspective Requirements Artifacts. In: 2009 Eighth IEEE/ACIS International Conference on Computer and Information Science. S. 943–948.

Assawamekin, N., Sunetnanta, T. und Pluempitiwiriyawej, C. (2008): Resolving Multiperspective Requirements Traceability through Ontology Integration. In: 2008 IEEE International Conference on Semantic Computing. S. 362–369.

Asuncion, H., François, F. und Taylor, R. (2007): An end-to-end industrial software traceability tool. In: ESEC-*FSE'07*. New York, New York, USA: ACM Press, S. 115–124.

Asuncion, H. U., Asuncion, A. U. und Taylor, R. N. (2010): Software Traceability with Topic Modeling Categories and Subject Descriptors. In: International Conference on Software Engineering.

Atkinson, C. und Kühne, T. (2007): A tour of language customization concepts. In: Advances in Computers. 70, S. 105–161.

Atwell, E. (2013): The University of Pennsylvania (Penn) Treebank Tag-set. Leeds University. Abgerufen am 14.03.2013: http://www.comp.leeds.ac.uk/ccalas/tagsets/upenn.html.

Baclawski, K., Kokar, M. K., Kogut, P. A., Hart, L., Smith, J., Iii, W. S. H., Letkowski, J. und Aronson, M. L. (2001): Extending UML to Support Ontology Engineering for the Semantic Web. In: Lecture Notes in Computer Science. 2185, S. 342–360.

Baeza-Yates, R. und Ribeiro-Neto, B. (2011): Modern information retrieval. 2. Auflage. Edinburgh Gate: Pearson.

Bajwa, I. S., Samad, A. und Mumtaz, S. (2009): Object Oriented Software Modeling Using NLP Based Knowledge Extraction. In: European Journal of Scientific Research. 35 (1), S. 22–33.

Bashir, M. F. und Qadir, M. A. (2006): Traceability Techniques: A Critical Study. In: 2006 IEEE International Multitopic Conference. S. 265–268.

Berenbach, B. (2003): The automated extraction of requirements from UML models. In: Proceedings of the 11th IEEE International Requirements Engineering Conference. S. 287.

Berkling, K., Geisser, M., Hildenbrand, T. und Rothlauf, F. (2007): Offshore software development: transferring research findings into the classroom. In: Lecture Notes in Computer Science. 4716, S. 1–18.

Berztiss, A. T. (1997): Natural-language-based development of information systems. In: Data & Knowledge Engineering. 23 (June 1996), S. 47–57.

Bird, S. (2006): NLTK: The Natural Language Toolkit. In: Proceedings of the COLING/ACL on Interactive Presentation Sessions. S. 69–72.

Bird, S. und Loper, E. (2004): NLTK: The natural language toolkit. In: Proceedings of the ACL 2004 on Interactive Poster and Demonstration Sessions. S. 1–4.

Blaha, M. und Premerlani, W. (1998): Object-oriented Modeling and Design for Database Applications. Upper Saddle River, NJ: Prentice Hall.

Blumauer, A. und Pellegrini, T. (2006): Semantic Web und semantische Technologien: Zentrale Begriffe und Unterscheidungen. In: Pellegrini, T., Blumauer, A. (Hrsg.) Semantic Web: Wege zur vernetzten Wissensgesellschaft. Berlin, Heidelberg: Springer, S. 9–26.

Boehm, B. (2006): Value-based software engineering: Overview and agenda. In: Biffl, S., Aurum, A., Boehm, B. et al. (Hrsg.) Value-Based Software Engineering. Berlin; Heidelberg: Springer, S. 3–14.

Booch, G., Jacobson, I. und Rumbaugh, J. (2005): The unified modeling language user guide. 2. Auflage. Upper Saddle River, NJ: Addison-Wesley.

Booch, G., Maksimchuk, R. A., Engle, M. W., Young, B. J., Conallen, J. und Houston, K. A. (2007): Object-Oriented Analysis and Design with Applications. 3. Auflage. Upper Saddle River, NJ: Addison-Wesley.

Börstler, J. (1996): User-centered requirements engineering in record-An overview. In: Proceedings of the Nordic Workshop on Programming Environment Research. S. 1–8.

Brachman, R. J. und Levesque, H. J. (2004): Knowledge Representation and Reasoning. Amsterdam: Morgan Kaufmann.

Brants, S., Dipper, S., Hansen, S., Lezius, W. und Smith, G. (2002): The TIGER Treebank. In: Proceedings of the Workshop on Treebanks and Linguistic Theories. S. 1–18.

Brill, E. (1992): Some Advances in Transformation-Based Part of Speech Tagging. In: Proceedings of the Twelfth National Conference on Artificial Intelligence. S. 722–727.

Brooks, F. P. (1986): No Silver Bullet — Essence and Accident in Software Engineering. In: Proceedings of the IFIP Tenth World Computing Conference. Amsterdam: Elsevier Science, S. 1069–1076.

Brown, E., Epstein, E., Murdock, J. W. und Fin, T. (2013): Tools and Methods for Building Watson. IBM Research. Abgerufen am 14.02.2013: http://domino.research.ibm.com/library/cyberdig.nsf/papers/152EF31994BDC3DC85257B1F005DE78F

Bunt, H. und Palmer, M. (2013): Conceptual and Representational Choices in Defining an ISO Standard for Semantic Role Annotation. In: Workshop on Interoperable Semantic Annotation. S. 1–10.

Cahill, A. und Riester, A. (2012): Automatically acquiring fine-grained information status distinctions in German. In: Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue. S. 232–236.

Carstensen, K.-U. (2010): Nicht-sprachliches Wissen. In: Carstensen, K.-U., Ebert, C., Ebert, C. et al. (Hrsg.) Computerlinguistik und Sprachtechnologie: Eine Einführung. 3. Auflage. Heidelberg: Spektrum, S. 532–543.

Chen, P. (1983): English sentence structure and entity-relationship diagrams. In: Information Sciences. 29, S. 127–149.

Cook, J. E. und Wolf, A. L. (1999): Software process validation: quantitatively measuring the correspondence of a process to a model. In: ACM Transactions on Software Engineering and Methodology. 8 (2), S. 147–176.

Costa, M. und da Silva, A. R. (2007): RT-MDD Framework – A Practical Approach. In: Proceedings of ECMDA Traceability Workshop. Haifa, Israel, S. 17–26.

Dalianis, H. (1992): A method for validating a conceptual model by natural language discourse generation. In: International Conference on Advanced Information Systems Engineering. S. 425–444.

Daniels, J. und Bahill, T. (2004): The hybrid process that combines traditional requirements and use cases. In: Systems Engineering. 7 (4), S. 303–319.

Davis, M. (2006): Semantic Wave 2006-Part 1: Executive Guide to Billion Dollar Markets. Project10X Special Report, Washington.

Decker, B., Ras, E., Rech, J., Jaubert, P. und Rieth, M. (2007): Wiki-Based Stakeholder Participation in Requirements Engineering. In: IEEE Software. 24 (2), S. 28–35.

Dengel, A. (2012): Semantische Technologien: Grundlagen - Konzepte - Anwendungen. Heidelberg: Spektrum.

Dengel, A., Bernardi, A. und Elst, L. van (2012): Wissensrepräsentation. In: Dengel, A. (ed.) Semantische Technologien: Grundlagen. Konzepte. Anwendungen. Heidelberg: Springer, S. 21–72.

DeRose, S. (1988): Grammatical category disambiguation by statistical optimization. In: Computational Linguistics. 14 (1), S. 31–39.

Dori, D., Korda, N., Soffer, A. und Cohen, S. (2004): SMART: System Model Acquisition from Requirements Text. In: Lecture Notes in Computer Science. 3080, S. 179–194.

Ebersbach, A., Glaser, M., Heigl, R. und Warta, A. (2007): Wiki - Kooperation im Web. 2. Auflage. Berlin, Heidelberg: Springer.

Ebert, C., Schiehlen, M., Klabunde, R. und Evert, S. (2010): Semantik. In: Carstensen, K.-U., Ebert, C., Ebert, C. et al. (Hrsg.) Computerlinguistik und Sprachtechnologie: Eine Einführung. 3. Auflage. Heidelberg: Spektrum, S. 330–409.

Egyed, A. (2003): A scenario-driven approach to trace dependency analysis. In: IEEE Transactions on Software Engineering. 29 (2), S. 116–132.

Egyed, A. (2006): Tailoring Software Traceability to Value-Based Needs. In: Biffl, S., Aurum, A., Boehm, B. et al. (Hrsg.) Value based software engineering. Berlin, Heidelberg: Springer, S. 287–308.

Egyed, A., Biffl, S., Heindl, M. und Grünbacher, P. (2005): A Value-Based Approach for Understanding Cost-Benefit Trade-Offs During Automated Software Traceability. In: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering. Long Beach, California, USA, S. 2–7.

Fantechi, A., Gnesi, S., Lami, G. und Maccari, A. (2003): Applications of linguistic techniques for use case analysis. In: Requirements Engineering. 8 (3), S. 161–170.

Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M. und Moreschini, P. (1994): Assisting Requirement Formalization by Means of Natural Language Translation. In: Formal Methods in System Design. 4, S. 243–263.

Fernandes, E., Santos, C. dos und Milidiú, R. (2012): Latent structure perceptron with feature induction for unrestricted coreference resolution. In: Proceedings of the Joint Conference on EMNLP and CoNLL: Shared Task.

Finkelstein, A. (2012): Requirements und Relationships: A Foreword. In: Cleland-Huang, J., Gotel, O., Zisman, A. (Hrsg.) Software and Systems Traceability. London: Springer London, S. v–vi.

Florian, R., Ittycheriah, A., Jing, H. und Zhang, T. (2003): Named Entity Recognition through Classifier Combination. In: Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL. S. 168–171.

Fox, C. (1989): A stop list for general text. In: ACM SIGIR Forum. 24 (1-2), S. 19–21.

Frakes, W. und Baeza-Yates, R. (1992): Information retrieval: Data structures & algorithms. Upper Saddle River, NJ: Prentice Hall.

Freund, J. und Rücker, B. (2012): Praxishandbuch BPMN 2.0. 3. aktualisierte Auflage, München: Carl Hanser Verlag.

Froehlich, J. und Dourish, P. (2004): Unifying artifacts and activities in a visual tool for distributed software development teams. In: Proceedings of the 26th International Conference on Software Engineering. IEEE Computer Society, S. 387–396.

Geisser, M., Happel, H., Hildenbrand, T. und Seedorf, S. (2007a): Einsatzpotentiale von Wikis in der Softwareentwicklung am Beispiel von Requirements Engineering und Traceability Management. Working Paper.

Geisser, M., Hildenbrand, T. und Riegel, N. (2007b): Evaluating the Applicability of Requirements Engineering Tools for Distributed Software Development. Working Paper.

Gervasi, V. und Nuseibeh, B. (2002): Lightweight validation of natural language requirements. In: Software: Practice and Experience. 32 (2), S. 113–133.

Giesbrecht, E. und Evert, S. (2009): Is Part-of-Speech Tagging a Solved Task? An Evaluation of POS Taggers for the German Web as Corpus. In: Web as Corpus Workshop (WAC5). San Sebastian, Spanien.

Goddard, C. (1998): Semantic analysis. Oxford: Oxford University Press.

Goldsmith, J. A. (2010): Segmentation and Morphology. In: Carstensen, K.-U., Ebert, C., Ebert, C. et al. (Hrsg.) Computerlinguistik und Sprachtechnologie: Eine Einführung. 3. Auflage. Heidelberg: Spektrum, S. 365–393.

Gotel, O. C. Z., Marchese, F. T. und Morris, S. J. (2007): On Requirements Visualization. In: Second International Workshop on Requirements Engineering Visualization. S. 1–10.

Gotel, O. und Finkelstein, C. (1994): An analysis of the requirements traceability problem. In: Proceedings of the 1st International Conference on Requirements Engineering. S. 94–101.

Grace, T. P. L. (2009): Wikis as a knowledge management tool. In: Journal of Knowledge Management. 13 (4), S. 64–74.

Gregor, S. und Hevner, A. (2013): Positioning and presenting design science research for maximum impact. In: Management Information Systems Quarterly. 37 (2), S. 337–355.

Grouin, C., Rosset, S., Zweigenbaum, P., Fort, K., Galibert, O. und Quintard, L. (2011): Proposal for an extension of traditional named entities: From guidelines to evaluation, an overview. In: Proceedings of the Fifth Law Workshop, S. 92–100.

Gruber, T. (1993). A translation approach to portable ontology specifications. In: Knowledge Acquisition, (5), S. 199–220.

Gupta, V. (2008): Accelerated GWT. New York: Apress.

Hagen, M., Jungmann, B. und Lauenroth, K. (2007): Ein Prozessmodell für ein agiles und wiki-basiertes Requirements Engineering mit Unterstützung durch Semantic-Web-Technologien. In: Proceedings of the 1st Conference on Social Semantic Web. S. 119–138.

Hagenbruch, A. (2010): Flache Satzverarbeitung. In: Carstensen, K.-U., Ebert, C., Ebert, C. et al. (Hrsg.) Computerlinguistik und Sprachtechnologie: Eine Einführung. 3. Auflage. Heidelberg: Spektrum, S. 264–279.

Hajič, J. und Hladká, B. (1998): Tagging inflective languages: Prediction of morphological categories for a rich, structured tagset. In: Proceedings of the 17th International Conference on Computational Linguistics. S. 483–490.

Hapke, M., Jaszkiewicz, A. und Kowalczykiewicz, K. (2004): OPHELIA: Open Platform for Distributed Software Development. In: Proceedings of the Open Source International Conference. Malaga, Spain, S. 1–25.

Happel, H., Maalej, W. und Stojanovi, L. (2008): Team: towards a software engineering semantic web. In: Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering. S. 57–60.

Happel, H.-J. und Seedorf, S. (2006): Applications of ontologies in software engineering. In: Workshop on Sematic Web Enabled Software Engineering, S. 1–14.

Happel, H.-J., Maalej, W. und Seedorf, S. (2010): Applications of Ontologies in Collaborative Software Development. In: Mistrík, I., Grundy, J., Hoek, A. et al. (Hrsg.) Collaborative Software Engineering. Heidelberg: Springer, S. 109–129.

Harmain, H. M. und Gaizauskas, R. (2003): CM-Builder : A Natural Language-Based CASE Tool for Object-Oriented Analysis. In: Automated Software Engineering. 10, S. 157–181.

Hayashi, S., Yoshikawa, T. und Saeki, M. (2010): Sentence-to-Code Traceability Recovery with Domain Ontologies. In: 2010 Asia Pacific Software Engineering Conference. S. 385–394.

Hayes, J. H., Dekhtyar, A., Sundaram, S. K., Holbrook, E. A., Vadlamudi, S. und April, A. (2007): REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery. In: Innovations in Systems and Software Engineering. 3 (3), S. 193–202.

Heise (2013a): Watson wird Kundenberater. Abgerufen am 01.06.2013: http://heise.de/-1866262.

Heise (2013b): Yahoo kauft iPhone-Nachrichten-App Summly. Abgerufen am 31.05.2013: http://heise.de/-1829694.

Henderson, J. B. (2010): Artificial Neural Networks. In: Clark, A., Fox, C., Lappin, S. (Hrsg.) The handbook of computational linguistics and natural language processing. Malden, MA: Wiley.Blackwell, S. 221–237.

Hevner, A. R., March, S. T., Park, J. und Ram, S. (2004): Design science in information systems research. In: Mis Quarterly. 28 (1), S. 75–105.

Hildenbrand, T., Geisser, M. und Klimpke, L. (2007): Konzeption und Implementierung eines Werkzeugs für nutzenbasiertes Traceability- und Rationale-Management in verteilten Entwicklungsumgebungen. Working Paper.

Hildenbrand, T. (2008): Improving traceability in distributed collaborative software development: a design science approach. In: Informationstechnologie und Ökonomie. 33. C. Becker, W. Gaul, A. Heinzl, M. Schader & D. Veit (Hrsg.). Frankfurt am Main; Berlin; Bern; Wien [u.a.]: Peter Lang Verlag.

Hildenbrand, T., Geisser, M., Klimpke, L. und Acker, T. (2008): Designing and Implementing a Tool for Distributed Collaborative Traceability and Rationale Management. In: Proceedings of the PRIMIUM Subconference at the Multikonferenz Wirtschaftsinformatik. München.

Hildenbrand, T., Heinzl, A., Geisser, M., Klimpke, L. und Acker, T. (2009): A Visual Approach to Traceability and Rationale Management in Distributed Collaborative Software Development. In: Heinzl, A., Dadam, P., Kirn, S. et al. (Hrsg.) Lecture Notes in Informatics. 151. Mannheim, S. 161–178.

Hoard, J. E., Wojcik, R. und Holzhauser, K. (1992): An Automated Grammar and Style Checker for Writers of Simplified English. In: Computers and Writing. S. 278–296.

Hoenderboom, B. und Liang, P. (2009): A Survey of Semantic Wikis for Requirements Engineering. Technical report, University of Groningen.

Hofmann, T. (2001): Unsupervised Learning by Probabilistic Latent Semantic Analysis. In: Machine Learning. 42, S. 177–196.

Holzmann, G. J. (1997): The model checker SPIN. In: IEEE Transactions on Software Engineering. 23 (5), S. 279–295.

Hudaib, A., Hammo, B. und Alkhader, Y. (2007): Towards software requirements extraction using natural language approach. In: International Conference on Software Engineering, Parallel and Distributed Systems. S. 155–160.

Ibrahim, M. und Ahmad, R. (2010): Class Diagram Extraction from Textual Requirements Using Natural Language Processing (NLP) Techniques. In: Second International Conference on Computer Research and Development. S. 200–204.

Ide, N., Erjavec, T. und Tufiş, D. (2002): Sense discrimination with parallel corpora. In: Proceedings of the SIGLEX/SENSEVAL Workshop on Word Sense Disambiguation: Recent Successes and Future Directions. S. 54–60.

IEEE (1990): IEEE Standard Glossary of Software Engineering Terminology. Abgerufen am 02.12.2013: http://ieeexplore.ieee.org/servlet/opac?punumber=2238.

Iivari, J. (2007): A paradigmatic analysis of information systems as a design science. In: Scandinavian Journal of Information Systems. 19 (2), S. 39–64.

Jackson, M. (1995): Problems and requirements. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering. York, UK: IEEE Computer Society Press, S. 2–8.

Jayanthi, S. K. und Prema, S. (2011): Word Sense Disambiguation in Web Content Mining Using Brill's Tagger Technique. In: International Journal of Computer and Electrical Engineering. 3 (3), S. 358–362.

Jing, Y. und Croft, W. (1994): An association thesaurus for information retrieval. In: *Proceedings of Recherche d'Inf*ormation Assistée par Ordinateur. S. 1–15.

Jirapanthong, W. und Zisman, A. (2007): XTraQue: traceability for product line systems. In: Software & Systems Modeling. 8 (1), S. 117–144.

Jurafsky, D. und Martin, J. (2009): Speech and language processing. Upper Saddle River, NJ: Pearson.

Kaiya, H. und Saeki, M. (2005): Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach. In: Proceedings of the Fifth International Conference on Quality Software. S. 223–230.

Kaiya, H. und Saeki, M. (2006): Using Domain Ontology as Domain Knowledge for Requirements Elicitation. In: 14th IEEE International Requirements Engineering Conference. S. 189–198.

Kamalrudin, M., Grundy, J. und Hosking, J. (2010): Managing Consistency between Textual Requirements, Abstract Interactions and Essential Use Cases. In: 2010 IEEE 34th Annual Computer Software and Applications Conference. S. 327–336.

Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C. und Giannopoulou, E. (2007): Ontology visualization methods – A survey. In: ACM Computing Surveys. 39 (4), S. 10:1–10:43.

Kecher, C. (2011): UML 2: Das umfassende Handbuch. 4. Auflage. Bonn: Galileo Press.

Kitchenham, B. (1996): DESMET : A method for evaluating Software Engineering methods and tools. University of Keele.

Klein, D., Smarr, J., Nguyen, H. und Manning, C. D. (2003): Named entity recognition with character-level models. In: Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL. Morristown, NJ, USA: Association for Computational Linguistics, S. 180–183.

Kleinz, T. (2013): Neuer Wikipedia-Editor für alle. Abgerufen am 29.07.2013: http://www.heise.de/newsticker/meldung/Neuer-Wikipedia-Editor-fuer-alle-1925045.html.

Kleuker, S. (2009): Grundkurs Software-Engineering mit UML: der pragmatische Weg zu erfolgreichen Softwareprojekten. Wiesbaden: Vieweg & Teubner.

Kof, L. (2004): Natural Language Processing for Requirements Engineering: Applicability to Large Requirements Documents. In: Conference on Automated Software Engineering.

Koike, H. (1993): The role of another spatial dimension in software visualization. In: ACM Transactions on Information Systems. 11 (3), S. 266–286.

Kotonya, G. und Sommerville, I. (2004): Requirements engineering: Processes and techniques. Chichester: Wiley.

Kuhlmann, M. und Gogolla, M. (2012): From UML and OCL to relational logic and back. In: Model Driven Engineering Languages. S. 415–431.

Kunze, C. (2010): Lexikalisch-semantische Ressourcen. In: Carstensen, K.-U., Ebert, C., Ebert, C. et al. (Hrsg.) Computerlinguistik und Sprachtechnologie: Eine Einführung. 3. Auflage. Heidelberg: Spektrum, S. 504–514.

Kushman, N. und Barzilay, R. (2013): Using Semantic Unification to Generate Regular Expressions from Natural Language. in. Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.

Lai, H., Peng, R., Sun, D., Shao, F. und Liu, Y. (2012): A Survey of RE-specific Wikis for Distributed Requirements Engineering. In: 2012 Eighth International Conference on Semantics, Knowledge and Grids. S. 47–55.

Lee, H., Peirsman, Y., Chang, A., Chambers, N., Surdeanu, M. und Jurafsky, D. (2011): Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 shared task. In: Proceedings of the CoNLL-2011 Shared Task. S. 1–7.

Leuf, B. und Cunningham, W. (2001): The Wiki way: Quick collaboration on the Web. Amsterdam: Addison-Wesley Longman.

Liddy, E. D. (2007): Natural Language Processing. In: Encyclopedia of Library and Information Science. 2. Auflage, S. 2126–2136.

Lindvall, M. und Sandahl, K. (1996): Practical implications of traceability. In: Software Practice and Experience. 26 (10), S. 1161–1180.

Lohmann, S., Heim, P. und Ziegler, J. (2007): Semantic Integrator : Semi-Automatically Enhancing Social Semantic Web Environments. In: Proceedings of the 1st Conference on Social Semantic Web. S. 167–172.

Lucia, A., Fasano, F., Oliveto, R. und Tortora, G. (2007): Recovering traceability links in software artifact management systems using information retrieval methods. In: ACM Transactions on Software Engineering and Methodology. 16 (4), S. 1–50.

Lüdeling, A. (2009): Grundkurs Sprachwissenschaft. Universität Bremen. 4. Auflage. Stuttgart: Klett.

Luisa, M., Mariangela, F. und Pierluigi, N. I. (2004): Market research for requirements analysis using linguistic tools. In: Requirements Engineering. 9 (1), S. 40–56.

Maalej, W., Panagiotou, D. und Happel, H.-J. (2008): Towards Effective Management of Software Knowledge Exploiting the Semantic Wiki Paradigm. In: Software Engineering. S. 183–197.

Mädche, A. und Staab, S. (2001): Ontology learning for the semantic web. In: IEEE Intelligent Systems. 16 (2), S. 72–79.

Malouf, R. (2010): Maximum Entropy Models. In: Clark, A., Fox, C., Lappin, S. (Hrsg.) The Handbook of Computational Linguistics and Natural Language Processing. Malden, MA: Wiley.Blackwell, S. 133–153.

Mani, I. und Maybury, M. T. (1999): Advances in Automatic Text Summarization. Cambridge, MA: MIT Press.

Manning, C. D., Raghavan, P. und Schuetze, H. (2009): An Introduction to Information Retrieval. Cambridge: Cambridge University Press.

Manning, C. und Schütze, H. (1999): Foundations of statistical natural language processing. MIT Press.

Marcus, M., Marcinkiewicz, M. und Santorini, B. (1993): Building a large annotated corpus of English: The Penn Treebank. In: Computational Linguistics - Special issue on using large corpora: II. 19 (2), S. 313–330.

Markoff, J. (2011): Computer Wins on “Jeopardy!”: Trivial, It’s Not. The New York Times. Abgerufen am 17.2.2013: http://www.nytimes.com/2011/02/17/science/17jeopardy-watson.html?pagewanted=all&_r=0

Marneffe, M. und Manning, C. D. (2008): The Stanford typed dependencies representation. In: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation. S. 1–8.

Marneffe, M., MacCartney, B. und Manning, C. (2006): Generating typed dependency parses from phrase structure parses. In: Proceedings of Language Resources and Evaluation. S. 449–454.

Marneffe, M. und Manning, C. (2012): Stanford typed dependencies manual. Revised for Stanford Parser v. 2.0.4. Abgerufen am 21.08.2013: http://nlp.stanford.edu/downloads/dependencies_manual.pdf

Marshall, I. (1987): Tag selection using probabilistic methods. In: Garside, R., Sampson, G., Leech, G. (Hrsg.) The Computational Analysis of English: A corpus-based approach. S. 42–65.

Mencl, V. (2004): Deriving behavior specifications from textual use cases. In: Proceedings of Workshop on Intelligent Technologies. S. 1–11.

Mens, T. und Van Gorp, P. (2006): A Taxonomy of Model Transformation. In: Electronic Notes in Theoretical Computer Science. 152, S. 125–142.

Meth, H. (2013): A Design Theory for Requirements Mining Systems. Universtität Mannheim.

Meth, H., Li, Y., Mädche, A. und Mueller, B. (2012): Advancing Task Elicitation Systems - An Experimental Evaluation of Design Principles. In: Thirty Third International Conference on Information Systems. Orlando, FL, S. 1–20.

Meyer, B. (1985): On Formalism in Specifications. In: IEEE Software. 2 (1), S. 6–26.

Meyer, B. (1998): Object-oriented software construction. Science of Computer Programming. Santa Barbara: Interactive Software Engineering Inc.

Mich, L. (1996): NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA. In: Natural Language Engineering. 2 (2), S. 161–187.

Mikheev, A., Moens, M. und Grover, C. (1999): Named entity recognition without gazetteers. In: Proceedings of the ninth Conference on European chapter of the Association for Computational Linguistics. S. 1–8.

Miller, M. (2009): Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online. Que Publishing.

Miller, S. P., Whalen, M. W. und Cofer, D. D. (2010): Software model checking takes off. In: Communications of the ACM. 53 (2), S. 58–64.

Missikoff, M., Navigli, R. und Velardi, P. (2002): The usable ontology: An environment for building and assessing a domain ontology. In: The Semantic Web. S. 39–53.

Mueller, M. (2009): NUPOS: A part of speech tag set for written English from Chaucer to the present. Northwestern University. Abgerufen am 20.05.2013: http://morphadorner.northwestern.edu/morphadorner/documentation/nupos/.

Murdock, J. W., Tesauro, G. und Tj, I. B. M. (2012): Statistical Approaches to Question Answering in Watson. Abgerufen am 11.06.2013: http://www.mathaware.org/mam/2012/pdfs/StatQA_v3.pdf

Nadeau, D. und Sekine, S. (2007): A survey of named entity recognition and classification. In: Lingvisticae Investigationes. 30 (1), S. 3–26.

Navigli, R. (2009): Word sense disambiguation. In: ACM Computing Surveys. 41 (2), S. 1–69.

Nederhof, M.-J. und Satta, G.; Clark, A., Fox, C., Lappin, S. (2010): The Handbook of Computational Linguistics and Natural Language Processing. Wiley.Blackwell.

Neumann, G. (2010): Text-basiertes Informationsmanagement. In: Carstensen, K.-U., Ebert, C., Ebert, C. et al. (Hrsg.) Computerlinguistik und Sprachtechnologie: Eine Einführung. 3. Auflage. Heidelberg: Spektrum, S. 576–615.

Noll, R. P. und Ribeiro, M. B. (2007a): Enhancing traceability using ontologies. In: Proceedings of the 2007 ACM Symposium on Applied Computing. New York, New York, USA: ACM Press, S. 1496–1497.

Noll, R. P. und Ribeiro, M. B. (2007b): Ontological Traceability over the Unified Process. In: 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems. S. 249–255.

Nordheimer, K., Seedorf, S. und Thum, C. (2012): Semantic Wiki for Tracing Process and Requirements Knowledge in Small and Medium Enterprises. In: Mistrik, I., Tang, A., Bahsoon, R., Stafford, J. (Hrsg.) Aligning Enterprise, System, and Software Architectures. IGI Global, S. 23–38.

Olson, D. L. und Delen, D. (2008): Advanced Data Mining Techniques. Berlin; Heidelberg: Springer.

OMG (2012a): OMG Unified Modeling Language (OMG UML), Superstructure.

OMG (2012b): OMG Unified Modeling Language (OMG UML), Infrastructure.

Overmyer, S., Lavoie, B. und Rambow, O. (2001): Conceptual modeling through linguistic analysis using LIDA. In: Proceedings of the 23rd International Conference on Software Engineering. S. 401–410.

Page-Jones, M. (1999): Fundamentals of Object Oriented Design in UML. Reading, MA: Addison-Wesley.

Palmer, M., Gildea, D. und Kingsbury, P. (2005): The Proposition Bank: An Annotated Corpus of Semantic Roles. In: Computational Linguistics. 31 (1), S. 71–106.

Panagiotou, D. und Mentzas, G. (2009): A semantic wiki for software development. In: 13th Panhellenic Conference on Informatics. S. 93–102.

Pang, B. und Lee, L. (2008): Opinion mining and sentiment analysis. In: Foundations and Trends in Information Retrieval. 1 (2), S. 91–231.

Partsch, H. A. (2010): Requirements-Engineering Systematisch: Modellbildung für softwaregestützte Systeme. 2. überarbeitete Auflage, Heidelberg: Springer.

Pérez-González, H. und Kalita, J. K. (2002a): Automatically generating object models from natural language analysis. In: Companion of the 17th annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications. New York, New York, USA: ACM Press, S. 86–87.

Pérez-González, H. und Kalita, J. K. (2002b): GOOAL: A Graphic Object Oriented Analysis Laboratory. In: Object-Oriented Programming, Systems, Languages & Applications. S. 38–39.

Pérez-González, H., Kalita, J. K., Salvador Núñez Varela, A. und Wiener, R. S. (2005): GOOAL: an educational object oriented analysis laboratory. In: Object-Oriented Programming, Systems, Languages & Applications. S. 180–181.

Perkuhn, R., Keibel, H. und Kupietz, M. (2012): Korpuslinguistik. Paderborn: Wilhelm Fink.

Petrov, S., Das, D. und McDonald, R. (2011): A universal part-of-speech tagset. In: Proceedings of the 8th International Conference on Language Resources and Evaluation.

Pfister, B. und Kaufmann, T. (2008): Sprachverarbeitung - Grundlagen und Methoden der Sprachsynthese und Spracherkennung. Berlin, Heidelberg: Springer.

Pinheiro, F. A. C. (2000): Formal and Informal Aspects of Requirements Tracing. In: Workshop em Engenharia de Requisitos. S. 1–21.

Pinheiro, F. A. C. (2003): Requirements traceability. In: Sampaio do Prado Leite, J. C., Doorn, J. H. (Hrsg.) Perspectives on Software Requirements. Berlin: Springer, S. 91–113.

Piprani, B., Borg, M., Chabot, J. und Chartrand, É. (2008): An adaptable ORM metamodel to support traceability of business requirements across system development life cycle phases. In: Proceedings of the OTM Confederated International Workshops and Posters on the Move to Meaningful Internet Systems. Berlin: Springer, S. 728–737.

Pohl, K. (2008): Requirements Engineering. 2. korrigierte Auflage. Heidelberg: Dpunkt Verlag.

Pohl, K. und Rupp, C. (2011): Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level. 3. korrigierte Auflage. Heidelberg: Dpunkt Verlag.

Pooley, R. J. J. und Stevens, P. (1999): Using UML: Software Engineering with Objects and Rules. Boston: Addison-Wesley.

Raghunathan, K., Lee, H., Rangarajan, S., Chambers, N., Surdeanu, M., Jurafsky, D. and Manning, C. (2010): A Multi-Pass Sieve for Coreference Resolution. In: Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing. S. 492–501.

Raman, M. (2006): Wiki technology as a “free” collaborative tool within an organizational setting. In: Information Systems Management. 23 (4), S. 37–41.

Ramesh, B. und Jarke, M. (2001): Toward reference models for requirements traceability. In: Software Engineering, IEEE Transactions. 27 (1), S. 58–93.

Recker, J., Rosemann, M., Indulska, M. und Green, P. (2009): Business process modeling: A comparative analysis. In: Journal of the Association for Information Systems. 10 (4), S. 333–363.

Robillard, P. (1999): The role of knowledge in software development. In: Communications of the ACM. 42 (1), S. 87–92.

Rodewig, K. M. (2011): Webserver einrichten und administrieren. Bonn: Galileo Press.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. und Lorensen, W. (1991): Object-Oriented Modeling and Design. Englewood Cliffs, NJ: Prentice Hall.

Sarma, A., Noroozi, Z. und Hoek, A. van der (2003): Palantír: raising awareness among configuration management workspaces. In: Proceedings of the 25th International Conference on Software Engineering.

Saxena, V. und Kumar, S. (2012): Validation of UML Class Model through Finite-State Machine. In: International Journal of Computer Applications. 41 (19), S. 13–18.

Schaffert, S., Bry, F., Baumeister, J. und Kiesel, M. (2008): Semantic wikis. In: IEEE Software. (July/August), S. 8–11.

Schiller, A., Teufel, S. und Thielen, C. (1999): Guidelines für das Tagging deutscher Textcorpora mit STTS. Technical report.

Schmid, H. (2010): Decision Trees. In: Clark, A., Fox, C., Lappin, S. (Hrsg.) The Handbook of Computational Linguistics and Natural Language Processing. Malden, MA: Wiley.Blackwell, S. 180–196.

Schneider, G. und Volk, M. (1998): Adding manual constraints and lexical look-up to a Brill-tagger for German. In: Proceedings of the ESSLLI-98 Workshop on Recent Advances in Corpus Annotation. S. 1–7.

Seedorf, S. (2010). Ontologie-gestützte Entwicklung komponentenbasierter Anwendungssysteme. In: Informationstechnologie und Ökonomie. 41. C. Becker, W. Gaul, A. Heinzl, M. Schader & D. Veit (Hrsg.). Frankfurt am Main; Berlin; Bern; Wien [u.a.]: Peter Lang Verlag.

Seedorf, S., Nordheimer, K. und Krug, S. (2009): STraS: A framework for semantic traceability in enterprise-wide SOA life-cycle management. In: 13th Enterprise Distributed Object Computing Conference Workshops. S. 212–219.

Sendall, S. und Kozaczynski, W. (2003): Model transformation: The heart and soul of model-driven software development. In: IEEE Software. 20 (5), S. 42–45.

Shannon, C. (1951): Prediction and entropy of printed English. In: Bell System Technical Journal. 30 (1), S. 50–64.

Shinde, S., Bhojane, V. und Mahajan, P. (2012): NLP based Object Oriented Analysis and Design from Requirement Specification. In: International Journal of Computer Applications. 47 (21), S. 30–34.

Sommerville, I. (2007): Software engineering. 8. Auflage. Edinburgh: Pearson.

Sommerville, I. (2012): Software Engineering. 9. aktualisierte Auflage. München: Pearson.

Song, I., Yano, K., Trujillo, J. und Sergio Luján-Mora (2004): A taxonomic class modeling methodology for object-oriented analysis. In: Information Modeling Methods and Methodologies. S. 216–240.

Sowa, H., Radinger, W. und Marinschek, M. (2008): Google Web Toolkit. Heidelberg: Dpunkt Verlag.

Spanoudakis, G. und Zisman, A. (2005): Software traceability: A roadmap. In: Handbook of Software Engineering and Knowledge Engineering. S. 1–35.

Steinberg, D., Budinsky, F., Paternostro, M. und Ed Merks (2008): EMF: Eclipse Modeling Framework. 2. überarbeitete Auflage. Amsterdam: Addison-Wesley Longman.

Stock, W. G. und Stock, M. (2008): Wissensrepräsentation: Informationen auswerten und bereitstellen. München: Oldenbourg Wissenschaftsverlag.

Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R. und Wenke, D. (2002): OntoEdit: Collaborative ontology development for the semantic web. In: Horrocks, I., Hendler, J. (Hrsg.) Lecture Notes in Computer Science. Heidelberg: Springer, S. 221–235.

Teyseyre, A. R. und Campo, M. R. (2009): An overview of 3D software visualization. In: IEEE Transactions on Visualization and Computer Graphics. 15 (1), S. 87–105.

Thum, C. (2011): Enabling Lightweight Real-time Collaboration. In: Informationstechnologie und Ökonomie. 48. C. Becker, W. Gaul, A. Heinzl, M. Schader & D. Veit (Hrsg.). Frankfurt am Main; Berlin; Bern; Wien [u.a.]: Peter Lang Verlag.

Thum, C., Schwind, M. und Schader, M. (2009): SLIM – A lightweight environment for synchronous collaborative modeling. In: ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems. S. 1–15.

Torkar, R., Gorschek, T., Feldt, R., Svahnberg, M., Raja, U. A. und Kamran, K. (2012): Requirements traceability: A systematic review and industry case study. In: International Journal of Software Engineering and Knowledge Engineering. 22 (3), S. 385–434.

Trainer, E., Quirk, S., Souza, C. de und Redmiles, D. (2005): Bridging the gap between technical and social dependencies with Ariadne. In: Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange. 1 (212), S. 26–30.

Tufiş, D., Ion, R. und Ide, N. (2004): Fine-grained word sense disambiguation based on parallel corpora, word alignment, word clustering and aligned wordnets. In: Proceedings of the 20th International Conference on Computational Linguistics. S. 1312–1318.

Uschold, M. und Gruninger, M. (1996): Ontologies: Principles, methods and applications. In: Knowledge Engineering Review. 11 (2), S. 93–155.

Uzuner, O., Bodnari, A., Shen, S., Forbush, T., Pestian, J. und South, B. R. (2013): Evaluating the state of the art in coreference resolution for electronic medical records. In: Journal of the American Medical Informatics Association. 19 (5), S. 786–791.

Vaishnavi, V. und Kuechler, W. J. (2007): Design science research methods and patterns: innovating information and communication technology. Boca Raton, FL: Auerbach Publication.

Vanhooff, B., Van Baelen, S., Joosen, W. und Berbers, Y. (2007): Traceability as input for model transformations. In: Third European Conference on Modelling Foundations and Applications Traceability Workshop. Haifa, Israel: SINTEF, S. 37–46.

Varjú, Z., Littauer, R. und Ernis, P. (2012): Using Clojure in Linguistic Computing. In: Proceedings of the 5th European Lisp Symposium. S. 1–4.

Verma, K., Kass, A. und Vasquez, R. (2011): Using syntactic and semantic analyses to improve the quality of requirements documentation. In: Semantic Web.

Vinay, S., Aithal, S. und Desai, P. (2009): An Approach towards Automation of Requirements Analysis. In: Proceedings of the International MultiConference of Engineers and Computer Scientists. S. 1–6.

Visser, W. und Havelund, K. (2000): Model checking programs. In: The Fifteenth IEEE International Conference on Automated Software Engineering. S. 3–11.

Volter, M. (2006): Model-Driven Software Development. Chichester: John Wiley & Sons.

Voutilainen, A. (1999): Handcrafted rules. In: Halteren, H. (ed.) Syntactic Wordclass Tagging. Kluwer, S. 217–247.

Walderhaug, S. und Johansen, U. (2006): Towards a generic solution for traceability in MDD. In: European Conference on Modelling Foundations and Applications Traceability Workshop.

Watkins, R. und Neal, M. (1994): Why and how of requirements tracing. In: IEEE Software. 11 (4), S. 104–106.

Way, A. (2010): Maschine Translation. In: Clark, A., Fox, C., Lappin, S. (Hrsg.) The Handbook of Computational Linguistics and Natural Language Processing. Malden, MA: Wiley.Blackwell, S. 531–573.

Webber, B. und Web, N. (2010): Question answering. In: Clark, A., Fox, C., Lappin, S. (Hrsg.) The Handbook of Computational Linguistics and Natural Language Processing. Malden, MA: Wiley.Blackwell, S. 630–654.

Wilbur, W. J. und Sirotkin, K. (1992): The automatic identification of stop words. In: Journal of Information Science. 18 (1), S. 45–55.

Willett, P. (2006): The Porter stemming algorithm: then and now. In: Program: Electronic Library and Information Systems. 40 (3), S. 219–223.

Winkler, S. und Pilgrim, J. (2009): A survey of traceability in requirements engineering and model-driven development. In: Software & Systems Modeling. 9 (4), S. 529–565.

Winter, S. (2010): Formal Language Theory. In: Clark, A., Fox, C., Lappin, S. (Hrsg.) The handbook of computational linguistics and natural language processing. Malden, MA: Wiley.Blackwell, S. 11–42.

Witte, R., Zhang, Y. und Rilling, J. (2007): Empowering Software Maintainers with Semantic Web Technologies. In: Franconi, E., Kifer, M., May, W. (Hrsg.) The Semantic Web: Research and Applications. Berlin: Springer, S. 37–52.

Woychowsky, E. (2006): AJAX: Creating web pages with asynchronous JavaScript and XML. Upper Saddle River, NJ: Prentice Hall.

Yoshikawa, T., Hayashi, S. und Saeki, M. (2009): Recovering traceability links between a simple natural language sentence and source code using domain ontologies. In: IEEE International Conference on Software Maintenance. S. 551–554.

Yue, T., Briand, L. C. und Labiche, Y. (2011): A systematic review of transformation approaches between user requirements and analysis models. In: Requirements Engineering. 16 (2), S. 75–99.

Zhang, T. und Johnson, D. (2003): A Robust Risk Minimization based Named Entity Recognition System. In: Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL. S. 204–207.

Zhang, Y., Witte, R., Rilling, J. und Haarslev, V. (2006): An ontology-based approach for traceability recovery. In: 3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering. S. 1–15.

Zhang, Y., Witte, R., Rilling, J. und Haarslev, V. (2008): Ontological approach for the semantic recovery of traceability links between software artefacts. In: IET Software. 2 (3), S. 185–203.

Zheng, J., Chapman, W. W., Miller, T. A., Lin, C., Crowley, R. S. und Savova, G. K. (2013): A system for coreference resolution for the clinical narrative. In: Journal of the American Medical Informatics Association. 19 (4), S. 660–667.

Zowghi, D. und Gervasi, V. (2003): On the Interplay Between Consistency, Completeness, and Correctness in Requirements Evolution. In: Information and Software Technology. 45 (14), S. 993–1009.

LEBENSLAUF – SIMONE KRUG

AKADEMISCHER WERDEGANG

Oktober 2007 – April 2014	Promotion am Lehrstuhl Wirtschaftsinformatik III, Universität Mannheim
September 2002 – September 2007	Studium Diplom Wirtschaftsinformatik, Universität Mannheim
September 2004 – Juli 2005	Licence MIAGE (Maîtrise de l'informatique appliquée à la gestion de l'entreprise), Université Sophia Antipolis in Nizza
September 1997 – Juni 2002	Abitur, Otto-Hahn-Gymnasium Monheim am Rhein
September 1999 – Juni 2000	High School Diplom, Nashwauk-Keewatin-High School Nashwauk, USA

LEHR- & FORSCHUNGSAUFENTHALTE

April 2011 – Mai 2011	Lehraufenthalt an der San Francisco State University, College of Business
Juni 2010 – August 2010	Forschungsaustausch an der Queensland University of Technology, Brisbane

AUSZEICHNUNGEN & ZERTIFIKATE

Oktober 2010	Baden-Württemberg-Zertifikat für Hochschuldidaktik
Juni 2008	Zweifache Preisträgerin des „Theseus Talente“-Wettbewerbs des Bundesministeriums für Wirtschaft und Technologie
September 2004 – Juli 2005	SOKRATES Stipendium