FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Semantische Features in Quality Function Deployment basierter Anforderungsmanagementsoftware

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsingenieurwesen Informatik

eingereicht von

Paul Christian Weißenbach, BSc
Matrikelnummer 0326481

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.-Ing. Detlef Gerhard

Wien, 26.05.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Semantic Features in Quality Function Deployment Based Requirements Engineering Tools

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Engineering and Computer Science

by

Paul Christian Weißenbach, BSc
Registration Number 0326481

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Dipl.-Ing. Dr.-Ing. Detlef Gerhard

Vienna, 26.05.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Paul Christian Weißenbach, BSc
Goldschlagstraße 90-92/31-32, 1150 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

_____ _____

(Ort, Datum) (Unterschrift Verfasser)

Danksagung

Meinen Eltern und Prof. Gerhard.

Abstract

Designing successful products and services is difficult. One important part of such an engineering process is to find out what the customer wants, looks at and ultimately buys. The quality of this information and its intake strongly influences the projects outcome. The tasks done in early project stages are usually labeled as requirements engineering (RE) with the goal of RE being to create a consistent product specification.

After looking into state-of-the-art semantically supported requirements engineering and generally analyzing the capabilities of requirements engineering software, an approach to improve a development team's view on potential customers is presented. This approach facilitates semantic web technology and is integrated into the quality function deployment method (QFD), more specifically, a selected combination of houses of quality. This selection is intended to support the approache's goal to enable the QFD team to question and justify their picture of the customer, that is, their market perception whilst at the same time not forgetting about product realization feasibility.

An unexpected result of this thesis is that the presented approach can potentially help in linear development process models to identify the moment when to move from a requirements engineering to an implementation phase: a finding that is useful for a QFD team. Furthermore its simplicity makes it potentially accessible for project outsiders such as those supporting the management rather than the engineering function.

Kurzfassung

Die Entwicklung von erfolgreichen Produkten und Dienstleitungen ist schwierig. Eine wichtiger Teil eines Entwicklungsprozesses ist es herauszufinden, was der Kunde will, wie er danach sucht und vor allem was er letztendlich kauft. Die Qualität solcher Informationen und deren Aufnahme im Entwicklungsteam haben starken Einfluss auf das Ergebnis des Entwicklungsprojektes. Aufgaben die in diesen Bereich fallen werden üblicherweise als Anforderungsmanagement (engl. requirements engineering) bezeichnet. Die Kernaufgabe von Anforderungsmanagement ist die Erstellung von konsistenten Produkt-/ Dienstleistungsspezifikationen.

Nachdem der state-of-the-art von Anforderungsmanagement mit semantischer Technologieunterstützung begutachtet wird und die Fähigkeiten von Anforderungsmanagementsoftware generell und in Bezug zu semantischen Technologien analysiert werden, wird in dieser Arbeit ein Ansatz zur Verbesserung des vorherrschenden Kundenbildes vorgestellt. Dieser Ansatz bedient sich "semantic web" Technologien und integriert sich in die Quality Function Deployment Methode, genauer, in eine Auswahl von Houses of Quality. Diese Auswahl ermöglicht es dem QFD Team, sein Kundenbild zu hinterfragen und auch zu rechtfertigen.

Ein unerwartetes Ergebnis dieser Arbeit ist, dass der vorgestellte Ansatz bei linearen Entwicklungsvorgehensmodellen hilft, den Zeitpunkt festzustellen, wann von einer Anforderungs-/ Spezifikationsphase zu einer Implementierungsphase übergegangen werden kann. Eine für das Team selbst und auch für Projektaußenstehende, wie zum Beispiel das Management, hilfreiche Information.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

1.1 Problemstellung

Ohne Wissen darüber, was der Kunde benötigt, erwartet oder wünscht ist es schwer möglich, das Benötigte, Erwartete oder Gewünschte zur Verfügung zu stellen. Geschweige denn, ein Produkt, egal ob Software, Hardware, Dienstleistung herzustellen, das beim Kunden zu einer positiven Kaufentscheidung führt.

In Projekten mit mehreren Beteiligten und unterschiedlichen Hintergründen ist es zudem wichtig, eine möglichst gleiche Wahrnehmung der Problemstellung des Kunden und der zu erstellenden Lösung zu erreichen. Illustriert werden soll dies an der bekannten „Schaukeldarstellung“ in Abbildung 1.1. Sie zeigt, welche unterschiedlichen Lösungsideen und Probleminterpretationen bei einem Projekt unter den verschiedenen Stakeholdern vorherrschen können.

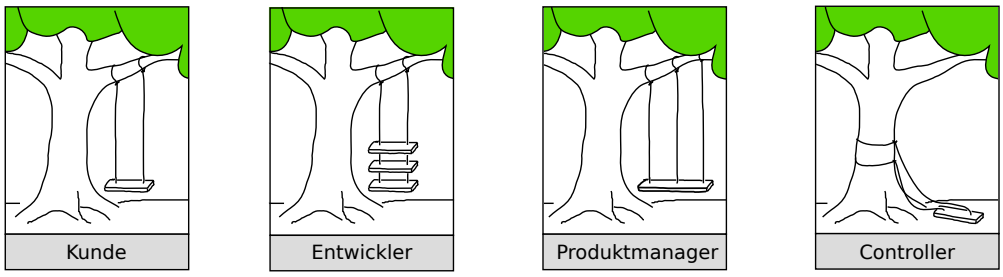

Abbildung 1.1: Verschiedene Vorstellungen von Lösungen

Verschiedene Studien unterstreichen die Notwendigkeit eines strukturierten Verwaltens von Anforderungen, wie sie das Anforderungsmanagement bietet. Die Standish Group bezifferte bereits 1995 [Gro95] die Beteiligung von Fehlern im Anforderungsmanagement oder das Nichtvorhandensein von Anforderungsmanagement mit 40% bei den Hauptursachen für Verzögerungen und das Scheitern von Projekten. Auch eine Studien der ESPITI (European Software Process Improvment Training Initiative) [ESP95] führte zu sehr ähnlichen Ergebnissen.

Der Artikel im QZ 56 (2011) [Ebe11] bezieht sich auf den Standish Group Report von 2009 und rezitiert, dass unzureichendes Requirements Engineering als ein Hauptgrund für den Misserfolg von Projekten im IT-Bereich ausgewiesen wird.

Fehlerhafte Anforderungen können den gesamten Entwicklungsprozess korrumpieren. Wobei das Ausbessern von solchen Fehlern in einer frühen Phase des Prozesses deutlich günstiger ist als später im Innovationsprozess bzw. noch später (nach Verkauf/Auslieferung). Die Abbildung 1.2 zeigt ein Beispiel aus dem Bereich Software Engineering. Der erste Verlustkostenhebel in Abbildung 1.3 a), bezieht sich auf mechanische Produkte. Abbildung 1.3 b) bezieht sich auf eine Studie der HOOD-Group, für dessen Erhebung sich ein Automobilhersteller verantwortlich zeigt.

Ganzheitlich gesehen sollen durch Anforderungsmanagement im Projekt (Unternehmen) weniger Fehler passieren und dadurch die aus Kundensicht bzw. Stakeholdersicht besseren Produkte erstellt werden. Mit dem Ergebnis, dass über den gesamten Produktlebenszyklus mehr Gewinn erwirtschaftet werden kann.

Eine in den sechziger Jahren entstandene und auf Qualitätssicherung und "Value Engineering" zurückgehende Methode ist Quality Function Deployment (QFD). QFD zielt, sehr ähnliche wie hier für das Anforderungsmanagement beschrieben, explizit darauf ab, dass alle Komponenten eines zu entwickelnden Systems auf Kundenbedürfnisse zurückgeführt werden können, und somit Wert für den Kunden erzeugen. Zudem liefert QFD unter anderem eine spezifische Darstellung für Anforderungen, genannt House of Quality (HoQ), und verbindet darin Anforderungen eingeteilt in Kunden- und Systemdimension.

1.2 Zielsetzung, Fragestellung

Ziel dieser Arbeit ist es, semantische Features von Anforderungsmanagement zu konzipieren, die einem Entwicklerteam helfen, den Kunden besser zu verstehen und mögliche Inkonsistenzen in Kundenwünschen aufzuzeigen. Dabei geht dieser Ansatz in seiner Grundhaltung davon aus, dass in einem solchen Team die Methode Quality Function Deployment bereits eingesetzt wird.

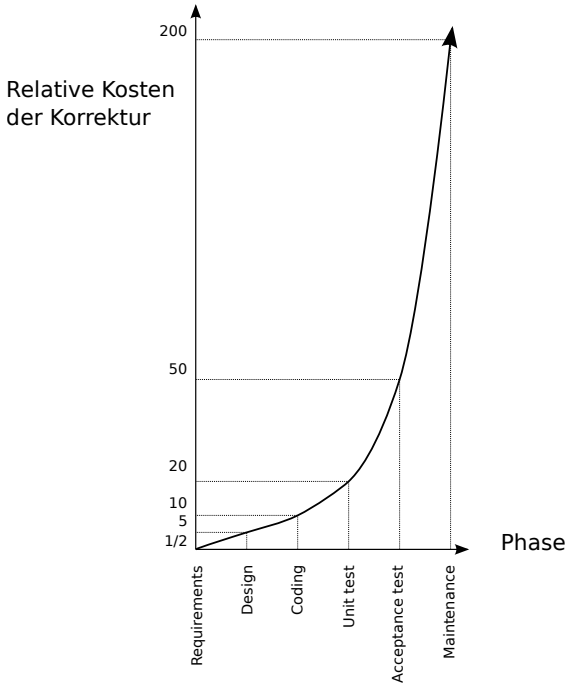

Abbildung 1.2: Relative Kosten für Anforderungskorrektur im Software Engineering vgl. [Sch02, S. 20]

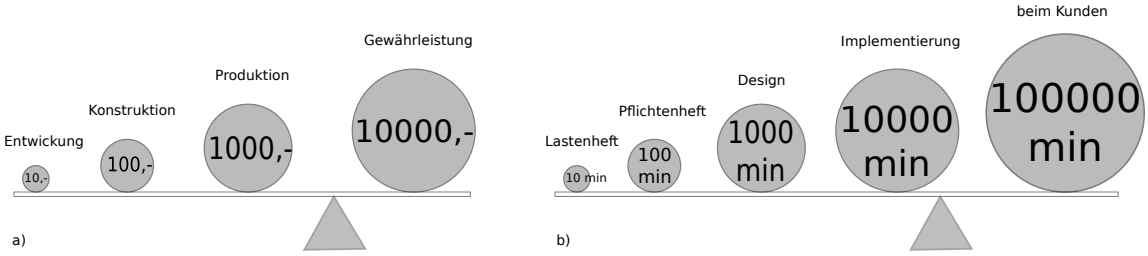

Abbildung 1.3: a) Verlustkostenhebel mit Kosten vgl. [SAKR05, S. 381]; b) mit Zeit aus [HW05, S. 28]

Es sollen, QFD entsprechend, Produkte und Dienstleistungsspezifikationen entstehen, welche aus Kundensicht größtmöglichen Wert besitzen, wobei zusätzlich Kaufentscheidungskriterien als Konzeptionierungsgrundlage mitberücksichtigt werden. Auch soll praktischen Problemen, wie steigende Unübersichtlichkeit und die damit mit einhergehende Fehleranfälligkeit bei wachsendem Umfang von Anforderungen durch semantische Unterstützung, entgegengewirkt werden. Eine weitere angestrebte Verbesserung durch den Einsatz semantischer Technologie sollte dazu führen, dass vorhandene HoQs leichter geändert werden, bzw. beim Ändern deren Implikationen unmittelbar abgeschätzt werden können.

Zusammenfassend sucht diese Arbeit folgende Frage zu beantworten: Welche Funktionen, die auf "semantic Web" Technologie basieren, können sinnvoll und ohne wichtige Methodenprinzipien zu verletzen, in QFD basierter Anforderungsmanagementsoftware hinzugefügt werden, um das Kundenverständnis in Produkt- und Dienstleistungsentwicklungsteams zu verbessern?

1.3 Scope, Abgrenzung

Diese Arbeit konzentriert sich auf Funktionen von Anforderungsmanagementsoftware die auf semantischen Technologien basieren. Zwar werden Vorgehensmodelle in Entwicklungsprojekten und deren Zusammenhang zu Anforderungen diskutiert, aufgezeigt werden die Funktionen jedoch nur innerhalb der Methode Quality Function Deployment.

1.4 Vorgehensweise, Methodik und Aufbau der Arbeit

Nach einer allgemeinen Einführung zum Anforderungsmanagement, den wichtigen Zusammenhängen zu Projektvorgehensweisen, der Vorstellung von üblichen Arten, Anforderungen zu modellieren und QFD, wird in Kapitel 3 anhand von Beispielen in semantische Technologien eingeführt. In weiterer Folge (Kapitel 4) werden die Funktionen von Anforderungsmanagementsoftware besprochen, um anschließend auf die Anwendung von semantischen Technologien darin einzugehen. Kapitel 5 markiert das Ende der Recherchetätigkeit. Darin wird eine auf QFD ausgerichtete OWL Ontologie erarbeitet, auf die SWRL Regeln angewandt werden können, die wiederum dem Entwicklungsteam bei der Erstellung eines aus Kundensicht konsistenten Produkts helfen sollen. Kapitel 6 diskutiert die Rechercheergebnisse und den praktischen Nutzen des mittels "semantic web" Technologien generierten Feedbacks.

KAPITEL 2

Anforderungsmanagement

2.1 Anforderungen

Es finden sich mehrere Definitionen zu *Anforderungen* in der Literatur. Da eine Formalisierung von Anforderungen hier später noch ausgiebig diskutiert wird, soll vorerst möglichst allgemein und im Sinne hoher Kundenorientiertheit mit dem Begriff Anforderung umgegangen werden, und zwar: Anforderungen sind Vermutungen über Bedürfnisse des Kunden. Auch wenn sie im Fall von auftragsbezogenen Entwicklungen vom Kunden selbst formuliert wurden.

Das *Anforderungsmanagement* sieht die Essenz seiner Aufgabe darin, sicherzustellen, dass in einem Innovationsprozess, wie zum Beispiel die Entwicklung eines Produktes oder ein Dienstleitung, die richtigen Anforderungen erfüllt werden.

Wie verschiedene formale Zugänge (Definitionen) für Anforderungen finden sich in der Literatur auch verschiedene Betrachtungswinkel und Erklärungen von den Hauptaufgaben des *Anforderungsmanagement*, die sich im Kern aber nicht unterscheiden.

Abbildung 2.1 zeigt den Blickwinkel von Schienmann und Abbildung 2.2 den von Siemens.

Bei der in Abbildung 2.2 dargestellten Struktur wird zwischen Requirements Development und Management unterschieden, was der CMMI [CMM10] Unterscheidung entspricht. Interessant ist dabei, dass Auer sich auf Schienenfahrzeuge bezieht, das CMMI, welches vom Software Engineering Inistut der Carnegie Mellon Universität[1] stammt, trotz seines allgemeinen Anspruches, ein Fokus auf Software jedoch nicht unerkennbar ist. Um die Unterteilung von Requi-

[1] `http://www.sei.cmu.edu/cmmi/`

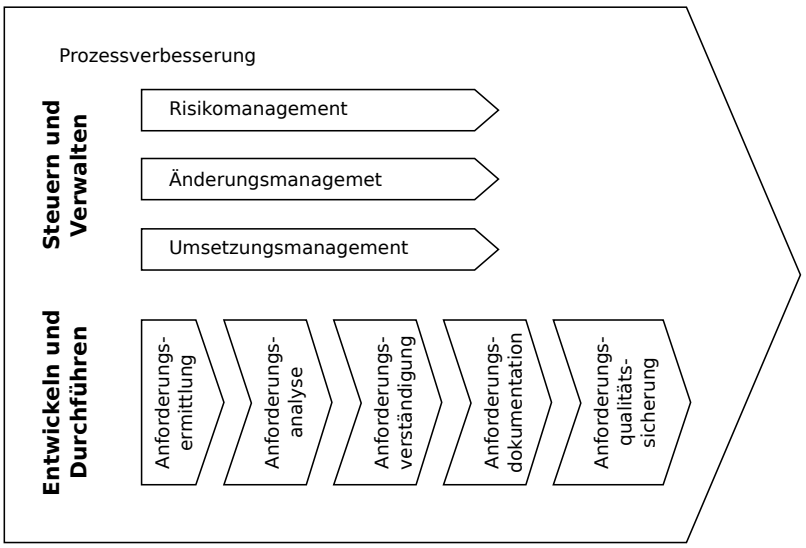

Abbildung 2.1: Anforderungsmanagement nach [Sch02, S. 33]

rements Development und Requirements Management zu verdeutlichen: Der Teil Requirments Development beginnt mit dem Angebotsprozess und die Tätigkeiten des Requirements Managements startet etwa mit der Übergabe des Lastenhefts and das Engineering. Das Requirements Development endet nach dieser Übergabephase.

Die Vorgehensweisen im Requirements Development sind im CMMI Standard in Version 1.5 ab Seite 325 nachzulesen. Für Requirements Management ab Seite 341.

Da die Software, die in dieser Diplomarbeit betrachtet wird, schwer durch diese verschiedenen Begriffe unterschieden werden kann, wird hier (konsequent) die Bezeichnung Anforderungsmanagement verwendet und es dem Leser überlassen, welchem Begriff er welche Funktion der beschriebenen Softwaresysteme zuordnet, da diese mehr von den Standards und Vorgehensweisen im Unternehmen abhängig sind, und weniger von den Funktionen, die ein semantisch unterstütztes Anforderungsmanagementsoftwaresystem mitbringen kann und soll.

Angemerkt wird, dass sich weitere Blickwinkel auf das Anforderungsmanagement finden lassen. Zum Beispiel in den Organisations- und Projektgestaltungsrichtlinien PRINCE2 [2] oder PMBOK[3].

[2] `http://www.prince-officialsite.com/`
[3] `http://www.pmi.org/`

Abbildung 2.2: Anforderungsmanagement bei Siemens vgl. [Aue09, S. 18]

Was im gegebenen Kontext mit Anforderungen gemeint ist, wird nach ausführlicher Behandlung der Formalisierung von Anforderungen sowie der Behandlung von Anforderungsmanagementsoftware und deren Aufgaben deutlicher.

2.2 Klassifizierung/Kategorisierung von Anforderungen

Natürlich gibt es eine Vielzahl von Möglichkeiten Anforderungen zu strukturieren, manche sind fachdomänenspezifisch, mache allgemein.

Fachspezifische Einteilungen sind zum Beispiel die Zuordung von Anforderungen zu Funktionsstruktur und/oder Hauptbaugruppen. Strukturierungsprinzipien für industrielle Systeme, Anlagen, Ausrüstungen und Industrieprodukte sind in der Norm DIN EN 81346-1 standardisiert.

In fast jeder Literatur zum Anforderungsmanagement wird zwischen funktionalen und nichtfunktionalen Anforderungen unterschieden. Wobei meistens gleich mit erwähnt wird, dass sich nicht alle Anforderungen exakt zuordnen lassen. Eine zu starke Einteilung in funktional/nicht

funktional wird aktuell kritisiert, wie zum Beispiel im iX-Artikel [FG12] ersichtlich. Die Autoren tretten darin für stärkere Verwendung von hierarchischen Strukturen ein.

Ein weiteres oft vorgestelltes Konzept zur Einteilung von Anforderungen ist jenes von Kano. Im Kanomodell (Abbildung 2.3) werden Anforderungserfüllungen mit ihren Auswirkungen auf die Kundenzufriedenheit verglichen. Das hat den Vorteil, dass bei einer weiterfolgenden Einteilung in Basis-, Funktion- und Begeisterungsanforderungen implizites Wissen in explizites umgewandelt wird, oder umgekehrt von den Eigenschaften auf die Kategorie geschlossen werden kann. Wird, zum Beispiel einer Anforderung in die Gruppe der Basisfaktoren eingeorndet, muss für diese Anforderung eine entsprechende Priorität gesetzt werden. Findet eine Anforderung bzw. ein Faktor in den meisten produktbezogenen Fachzeitschriften Erwähnung, kann dieser mit hoher Wahrscheinlichkeit zu den Leistungsfaktoren zugeordnet werden.

In Anlehnung an die Motivator-Hygiene-Theorie von Herzberg kann zu den drei bereits erwähnten Kategorien noch unerhebliche und Rückweisungsfaktoren hinzugefügt werden[4].

Es existieren noch weitere praktische Einteilungen. So zum Beispiel soll eine Einteilung in Kundenanforderungen und normspezifische Anforderungen folgendes Vorgehen erlauben: Ein Kunde möchte für zwei Länder das gleiche Produkt. Anforderungen die direkt auf den Kunden gehen und internationale Standards unterscheiden sich nicht und können gleich bleiben bzw. wieder verwendet werden. Lokale Richtlinien hingegen sollen einfach ausgetauscht werden können.

Eine letzte Einteilung, die an dieser Stelle noch erwähnt werden soll, ist das kontinuierliche Anforderungsmanagement für Software Unternehmen, wie es in [Sch02] vorgestellt wird. Dies soll unterstreichen, dass Anforderungsmanagement in klassischen Bereichen zumindest in sehr ähnlicher Form bereits existiert: Die Kernidee des kontinuierlichen Anforderungsmanagement ist eine Unterteilung in drei Gebiete: Kunden-, Produkt- und Projektanforderungsmanagement.

Gegenübergestellt werden können:

- Marketing, Produktmarketing übt ähnliche Aufgaben aus wie Kundenanforderungsmanagement,
- Produktmanagement wie Produktanforderungsmanagement und
- Projektmanagement wie Projektanforderungsmanagement.

Wobei wir hier sehen, das sich Anforderungen nicht immer auf Kunden/Stakeholder außerhalb des Unternehmens oder direkt auf das zu erstellende Produkt beziehen müssen. Beim kontinuierlichen Anforderungsmanagement werden auch Anforderungen an das Projekt sellbst gestellt.

[4] wie in `https://de.wikipedia.org/wiki/Kano-Modell` beschrieben

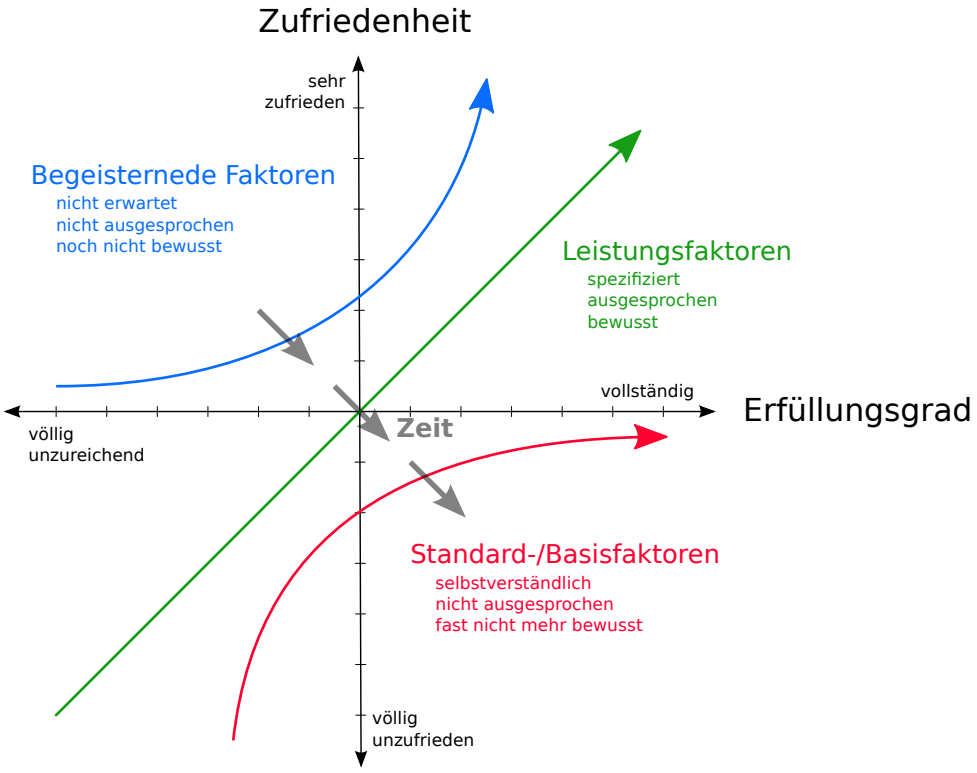

Abbildung 2.3: Kanomodell [SAKR05, S. 367]

Selten wird weiter gegangen und werden Anforderungen an die produkterstellende Organisation und dessen Management gestellt bzw. im Anforderungsmanagement berücksichtigt.

2.3 Anforderungsmanagement im Unternehmen

Anforderungsmanagement hat der Natur von Anforderungen entsprechend eine Schnittstellenfunktion. In Bezug auf Märkte eine zwischen Kunden und Unternehmen. Unternehmensintern zwischen verschiedenen Rollen bzw. Abteilungen.

Betrachtet man die Techniken des Anforderungsmanagement zur Anforderungsermittlung, wie sie von [Sch02] oder [Ebe10, S. 138] vorgestellt werden, muss nicht lange gesucht werden um andere Fachgebiete/Bereiche zu identifizieren, welche ähnliche oder sogar die gleichen Funktionen erfüllen müssen. In Abbildung 2.4 befindet sich eine Auszug von Methoden, wie man sie auch in Fachgebieten wie Marketing, Produktmanagement, Qualitätsmanagement und vor allem

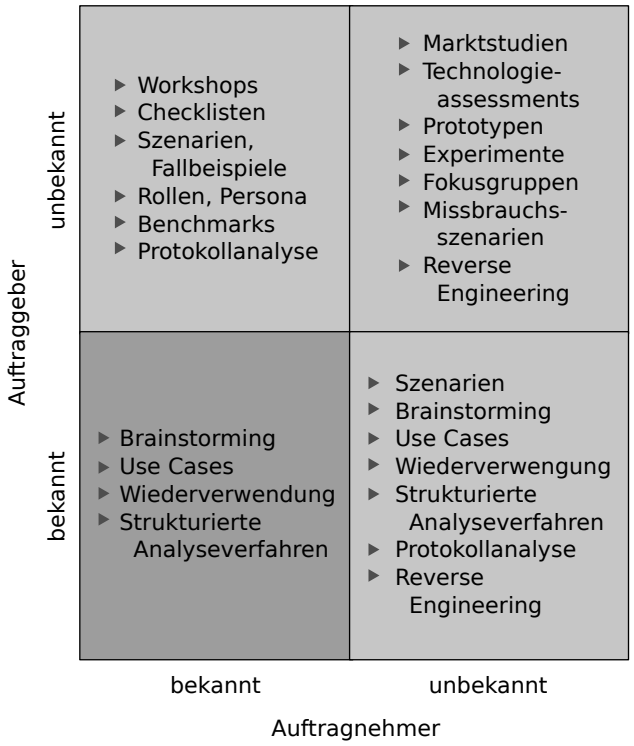

Abbildung 2.4: Methoden zur Ermittlung von Anforderungen [Ebe10, S. 135]

Marktforschung findet.

Anforderungen in verschiedenen Geschäftsarten

Teilt man Geschäftsfelder ähnlich wie bei Produktmarketing in „Einzelkunden“ und „anonymer Markt“ ein (diese Einteilung entspricht jener von Hofbauer in [HS06]) so ergibt sich der Unterschied, dass in anonymen Märkten der Kunde nicht für sich selbst sprechen kann. Typischerweise übernimmt in Entwicklungsprozessen eine Person oder Abteilung die Rolle des Kunden. Wie zum Beispiel ein Produktmanager, oder der Product Owner im agilem Vorgehensmodell Scrum.

Auch bei der Geschäftsart Einzelkunde und damit direktem Kontakt zum Kunden kann davon ausgegangen werden, dass die Anforderungen, die er/sie stellt, Schätzungen über den in Zukunft erwünschten Zustand sind.

Wobei als Basis für die Zusammenarbeit mit einem Einzelkunden rechtlich verbindliche Abnah-

mekriterien verhandelt werden (können).

Abnahmekriterien werden von Schienmann im Software Engineering als Ausgangssituation, Ereignis und erwartetes Ergebnis beschrieben. [Rup09, S. 277ff] sagt, dass Abnahmekriterien im Grunde Testfällen im Software Engineering entsprechen. In dieselbe Kerbe schlagend können hier TDD (Test Driven Development) und BDD (Behaviour Driven Development) erwähnt werden, die das Schreiben von Testfällen konsequent vor das Implementieren des jeweiligen Features legen und damit erreichen wollen, dass nur die gewünschten Funktionen implementiert werden um sogenanntes „Gold plating“ zu verhindern. Für eine Zusammenarbeit mit einem in Softwareentwicklung unversierten Kunden zielen Testframeworks, wie zum Beispiel Cucumber[5] ab, die das Ziel haben, das verlangte Verhalten in auch für Softwareentwicklungslaien verständlichem Klartext zu beschreiben. Um diese Beschreibung in weiterer Folge schrittweise in Testcode zu übersetzen. Folgendes Beispiel kommt vom Cucumber Projekt.

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario: Add two numbers
    Given I have enterd 50 into the calculator
    And I have entered 70 into the calculator
    When I press add
    Then the result schould be 120 on the screen
```

Abnahmekriterien/Spezifikationen werden typischerweise am Anfang einer Zusammenarbeit zur Kostenkalkulation bzw. Angebotserstellung festgelegt, gleichzeitig sollen sie aber das Ergebnis messbar auf ihre Erfüllung überprüfen können. Dies ist sehr schwierig, ohne den Lösungsraum auf ungewollte Art einzuschränken.

2.4 Anforderungen in verschiedenen Projektvorgehensweisen

Der Zeitbereich, in dem die Einschätzungen über die Anforderungen möglichst akkurat sein sollen, liegt in der Zukunft. Markteintritt und Marktaustritt begrenzen den Zeitraum. Bei auftragsbezogenen Projekten liegt die Verwendung des Produktes ebenfalls am Ende der Entwicklung.

[5] `http://cukes.info`

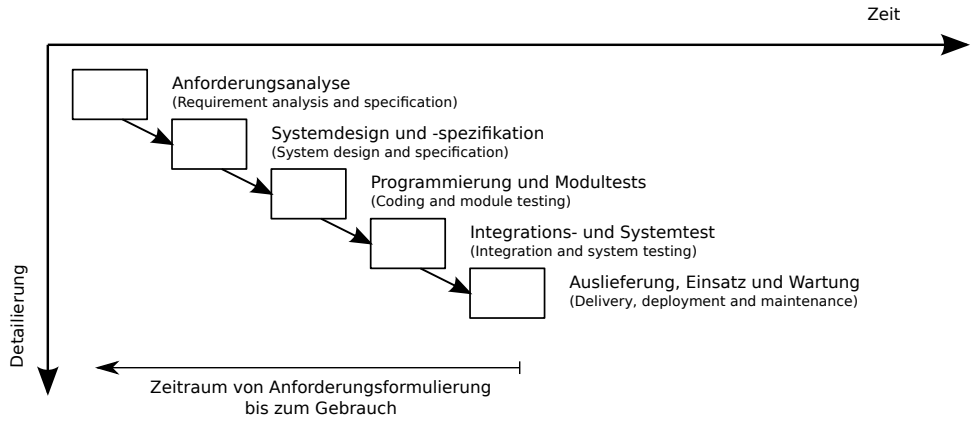

Abbildung 2.5: Wasserfallmodell

[Rup09, S. 363ff] behandelt, welche Zustände Anforderungen im Anforderungsmanagement durchlaufen und welche Transformationen sie dabei erfahren. [Rup09, S. 34] erwähnt den Unterschied zwischen wasserfallartigen, schwergewichtigen Vorgehensmodellen zu leichtgewichtigen, agilen, auf den in weiterer Folge etwas detailierter eingegangen wird. Dabei soll auch auf Entwicklungsprozesse ausserhalb der Softwareerstellung hingewiesen werden. Wo sinnvoll, sollen sie miteinander verglichen werden.

Davon ausgehend, dass Anforderungen umso schwieriger zu erfassen sind, je weiter der Zeitraum, in dem sie möglichst genau zutreffen sollten entfernt ist, können verschiedene Vorgehensmodelle anhand dieses simplen Kriteriums betrachtet werden. Es werden drei Vorgehensweisen untersucht:

- lineare Vorgehensmodelle
- iterative/inkrementelle Vorgehensweisen
- agile Vorgehensweisen

Lineare Vorgehensmodelle

Wasserfallmodelle (wie dargestellt in Abbildung 2.5) sind Vorgehensmodelle, in denen ein vordefinierter Schritt nach dem anderen ausgeführt wird. Am Projektstart befindet sich die Anforderungsanalyse und am Ende die Auslieferung des Produktes.

Der Begriff Wasserfallmodell wird hauptsächlich im Software Engineering verwendet. Strukturell ähnliche Vorgehensmodelle existieren auch außerhalb des Software Engineering, wobei Bezeichungen wie „Phasenmodell“ oder „Projektablaufmodell“ üblich sind.

Ein Beispiel ist die grundsätzliche Herangehensweise für mechanische Produkte, wie sie von Tjalve 2003 in [SAKR05, S. 255] beschrieben wird. Deren Phasen sind:

- Problemanalyse
- Hauptfunktionen
- Teilfunktionen und Funktionsträger
- Prinzipielle Struktur
- Quantitative Struktur
- Form des Gesamtsystems bzw. Form der Elemente

Weitere Beispiel existieren für Raumfahrt, technische Aufgaben, organisatorische Projekte, Investitionsprojekete [6].

Das V-Modell beschreibt wie die Wasserfallmodelle einen Ablauf von vordefinierten Schritten. Im Unterschied zu den vorher behandelten Vorgehensmodellen werden zuerst eine Dekomposition von Kundenanforderungen bis zur Zusammenbauebene durchgeführt, um danach von dieser untersten Detailierungsebene wieder zu integrieren und dabei deren Spezifikationen zu verifizieren und Anforderungen zu validieren. Die graphische Darstellung, wie in Abbildung 2.6 beschreibt ein „V“, deswegen auch der Name V-Modell.

Das V-Modell wurde ursprünglich zur Entwicklung von mechatronischen Systemen erstellt. Weitere Varianten sind zum Beispiel das V-Modell XT, welches unter anderem mehr Rücksicht auf Projektaspekte nimmt. Vorgehensweisen die mittels Automation von Verifizierungsschritten Zeit sparen, werden als „Y“-Modelle bezeichnet, ein Ansatz aus Model Driven Architecture.

Anforderungsdokumentation ist in linearen Vorgehensmodellen umfangreich. Es wird versucht, am Anfang bzw. in einer führen Phase des Projektes möglichst vollständig und konsistent die Anforderungen festzuhalten.

[6] urlhttp://de.wikipedia.org/wiki/Projektphase

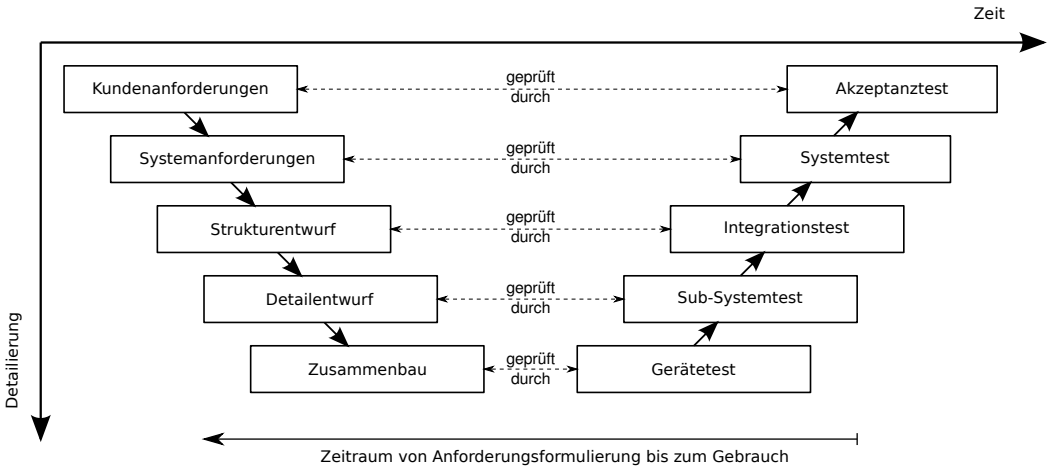

Abbildung 2.6: V-Modell vgl. [SAKR05, s. 255]

Änderungen und Korrekturen von Anforderungen bei linearem Vorgehen in späteren Phasen sind meist nur begrenzt möglich und im Normalfall sehr kostspielig. Siehe zum Beispiel Abbildung 1.2. Ändern sich Anforderungen bzw. findet man in späteren Phasen Fehler in Anforderungen, kann dies ein Zurück zu der ersten Projektphase bedeuten und kostet damit je nachdem in welcher Phase der Fehler auffällt, zusätzlich Zeit. Eine Möglichkeit das Risiko, Änderungen in späten Projektphasen umsetzen zu müssen, zu vermindern, ist eine intensivere und detailierte Beschäftigung mit Anforderungen, bevor zu weiteren Phasen übergegangen wird. QFD ist zum Beispiel eine Methode, die in der Anforderungsphase helfen kann nicht voreilig ein Produkt zu entwickeln.

Inkrementelle Entwicklungsprozesse

Davon ausgehend, dass Korrekturen mit Sicherheit auftreten, enthalten inkrementelle Vorgehensmodelle eine im vorhinein festgelegte Überarbeitung oder Wiederholung aller bzw. der produktrelevanten Projektphasen.

Die Phasen zwischen dem letzten Mal in dem die Anforderungen überarbeitet wurden und die Verwendung des Produktes durch den Kunden, sind nicht grundsätzlich unterschiedlich zu jenen in den linearen Vorgehensmodellen. Wobei zeitlich durch die gewonnene Erfahrung des Projektteams durch vorhergehende Durchläufe eine prinzipielle Reduzierung zu erwarten ist.

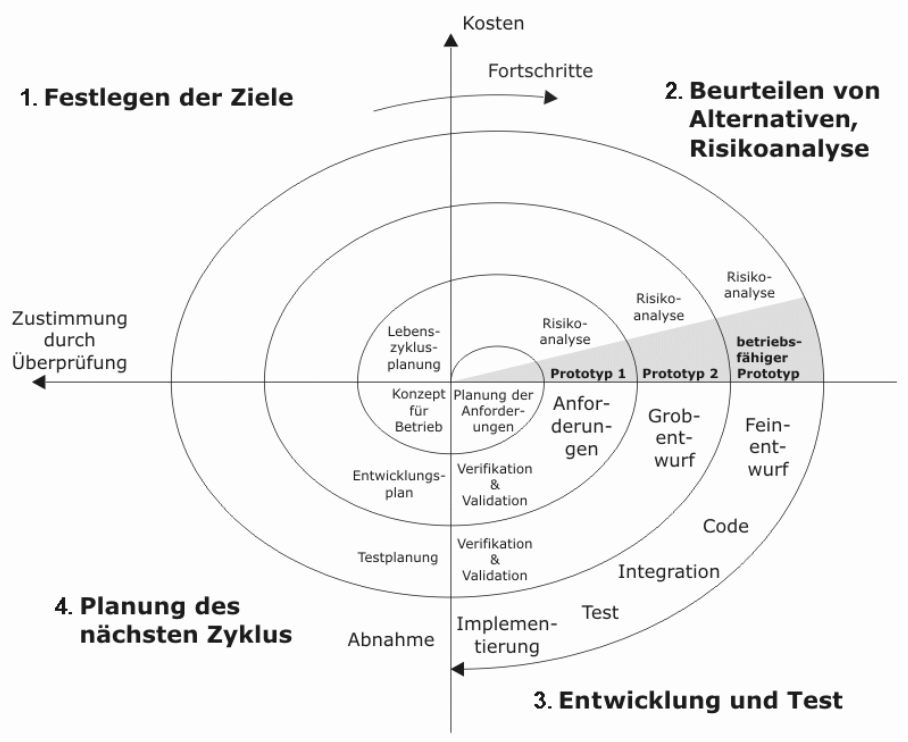

Abbildung 2.7: Spiralmodell nach Boehm. Quelle wikipedia de:User:Conny

Beispiel für ein solche inkrementelles Vorgehen ist das „Spiralmodell“ nach Barry W. Boehm, abgebildet in 2.7.

Agile Entwicklungsprozesse

Alle bisher beschriebenen Vorgehensmodelle verlangen, dass Anforderungen möglichst korrekt und vollständig vorliegen, bevor sie umgesetzt werden. Bei inkrementellen Vorgehensweisen geschieht dies wiederholt.

Agile Modelle (wie Scrum oder eXtremePrograming) unterscheiden sich davon dadurch, dass explizit auf eine vollständige Anforderungsspezifikation verzichtet wird. Vielmehr wird jeweils nur ein Teil der Anfordungen in kurzen Iterationen umgesetzt.

In Scrum, zum Beispiel, wird zwar eine Liste mit priorisierten Anforderungen (dem Produkt-Backlog) verwaltet. Welche Anforderungen davon in einem Iterationszyklus (Sprint) konkretisiert und umgesetzt werden entscheidet das Projektteam gemeinsam (festgehalten im Sprint-Backlog), wobei am Ende jedes Sprints ein funktionstüchtiges Produkt stehen soll. Abbildung 2.8 soll dies illustrieren.

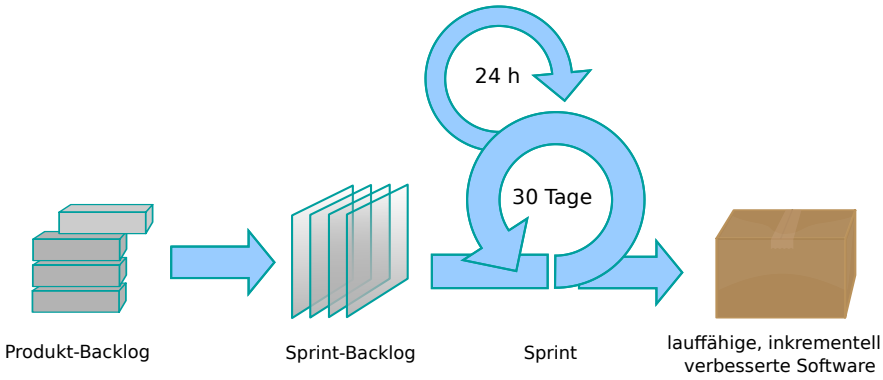

Abbildung 2.8: Scrum Prozess. Quelle wikipedia de:User:Sebastian Wallroth

Diese Methoden kommen aus dem Software Engineering und werden auch hauptsächlich dort eingesetzt. Es gibt jedoch Bestreben, diese auch für Hardware zu verwenden[7].

Einzelne Anforderungen für Features werden zum Beispiel in Scrum als User Stories formuliert, welche mit Akzeptanztests angereichert werden. Eine User Story mit Akzeptanztest wurde schon ohne es anzumerken auf Seite 11 unter Abnahmekriterien gezeigt.

Im „klassischen“ Engineering sind agile Methoden nicht verbreitet. Grund dafür könnte natürlich sein, dass sich Erstellung von Software fundamental von der Erstellug mechanischer Produkte unterscheidet, wobei durchaus schon agile Prinzipien in anderen Bereichen Beachtung finden[8].

Am nächsten kommen agilen Vorgehensmodellen, die in Vorentwicklungsprojekten angewandten, zum Beispiel in Projekten, in denen eine regelmäßige Fertigstellung von funktionstüchtigen Prototypen verlangt wird.

Zusammenhang zwischen Anforderungsarten und Vorgehensmodell

Eine ähnliche Aufarbeitung von Anforderungen in bezug zu verschiedenen Vorgehensmodellen findet sich in [Ebe10]. Wobei Ebert noch einen Schritt weiter geht und sinngemäß sagt, dass die Eigenschaften von Anforderungen die Vorgehensweise im Projekt vorgeben soll. Seine Zuordnung findet sich in Abbildung 2.9.

[7] `http://www.scrumhw.org/`
[8] [Goe11]

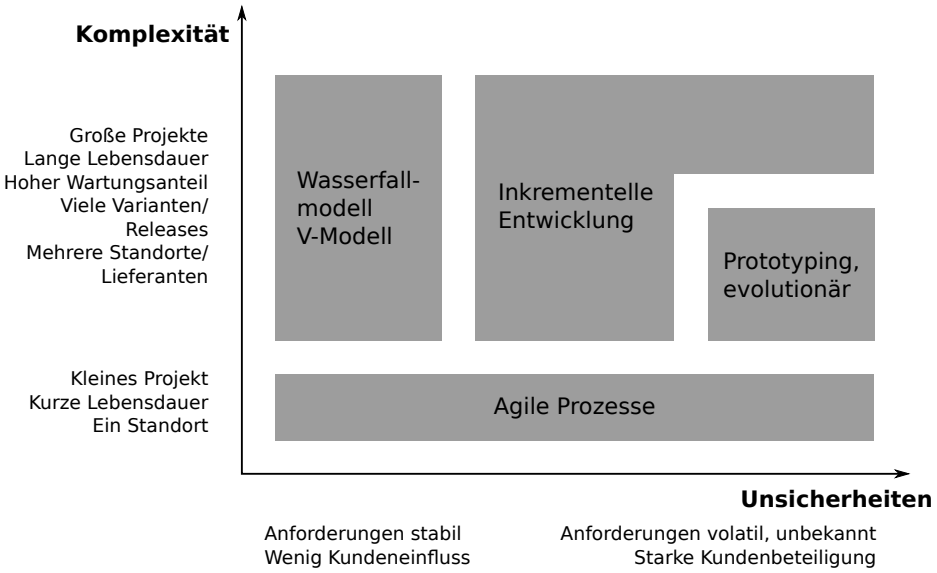

Abbildung 2.9: Auswahl von Vorgehensmodell vgl. [Ebe10, S. 61]

2.5 Formulierung und Formalisierung von Anforderungen

> „There is no such thing as a perfect software requirements specification."
> – [Dav93, S. 194]

Dies gilt wahrscheinlich allgemein für alle Spezifikationen, auch außerhalb des Software Engineerings.

Dennoch ist es notwendig, Anforderungen in ausreichender Qualität festzuhalten, um sie im Unternehmen und vor allem im Projekt kommunizieren und dokumentieren zu können. Dazu wird weitergehend besprochen, wie Anforderungen formuliert und formalisiert werden können. Zweck ist es, die Anforderungsqualität zu steigern und (nicht ausschließllich) Software basierte Werkzeuge zur Unterstützung dieser Aufgabe zu ermöglichen.

Anforderungen sind äußerst schwer qualitativ hochwertig festzuhalten, ohne den Lösungsraum einzuschränken. Eines der Qualitätskriterien von Schienmann [Sch02, S. 177] an Anforderungen ist die Umsetzbarkeit. Ebert [Ebe10, S. 240] bezeichnet dies als Realisierbarkeit. Dabei tritt das

selbe Problem wie für Abnahmekriterien auf: dass diese Kriterien nicht, ohne sich eine oder mehrere konkrete Umsetzungen vorzustellen, zu bewerten sind.

Im Folgenden einige Beispiele, wie Anforderungen in der unternehmerischen Praxis und Wissenschaft forumuliert und formalisiert werden:

- **Snowcards, Volere Atomic Requirement Templates**
  Snowcards sind im Idealfall vorgedruckte Karten und dienen als Hilfsmittel zu strukturierten Erfassung von Anforderungen. Tabelle 2.1 zeigt beispielhaft, welche Felder sich auf einer Snowcard befinden.

Attribut	**Beispielwert**
Nummer	`R932`
Anforderungstyp	`Funktional`
Anwendungsfall	`Sortierte Ausgabe`
Anforderung	`Das System soll (wenn gewünscht) eine Funktion zur Verfügung stellen die es erlaubt Benutzereingaben nach bestimmten Attributen zu sortieren.`
Priorität	`4`
Konflikte	`R432, R233`
Historie	`01.10.2012`

Tabelle 2.1: Attribute auf einer Snowcard

  Weitere Felder können sein: Auslöser, Abhängigkeiten, Quelle, Kundenzufriedenheit, Kundenunzufriedenheit, detaillierte Beschreibung.

- **Klassendiagramme**
  Klassendiagramme legen Attribute für Anforderungen und Methoden die auf diese Attribute ausgeführt werden fest. Attribute sind ähnlich oder gleich wie bei Snowcards: eindeutige Nummer, Anforderungstext etc.

  Der Natur von Klassendiagrammen entsprechend sind sie gedacht, um in Software umgesetzt zu werden. Ein Beispiel befindet sich in Abbildung 2.10.

- **Schablonen, Syntax für Anforderungen**
  Snowcards und das erwähnte Klassendiagramm enthalten „nur“ natürlichsprachliche Beschreibungen von Anforderungen, aber noch keine Regeln dazu, wie diese formuliert werden sollen. Da dies aber einen starken Einfluss auf die Qualität von Anforderungen hat, existeren verschiedene Vorschläge, wie natürlichsprachliche Anforderungen ausgedrückt werden sollen.

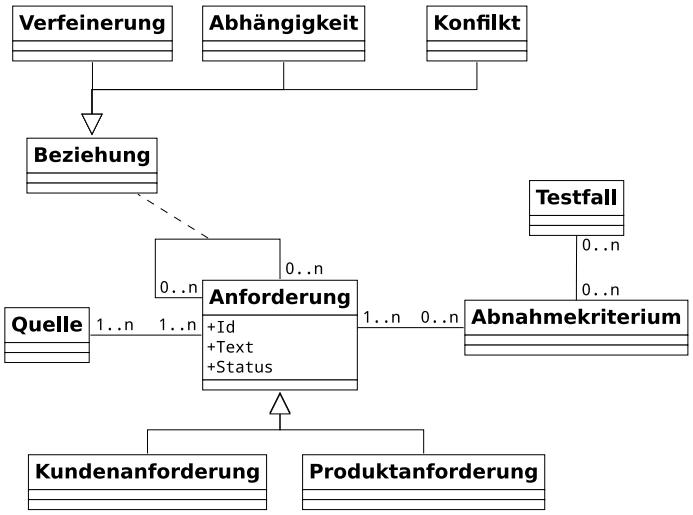

Abbildung 2.10: Teil des Klassendiagrammes aus [Sch02, S. 150]

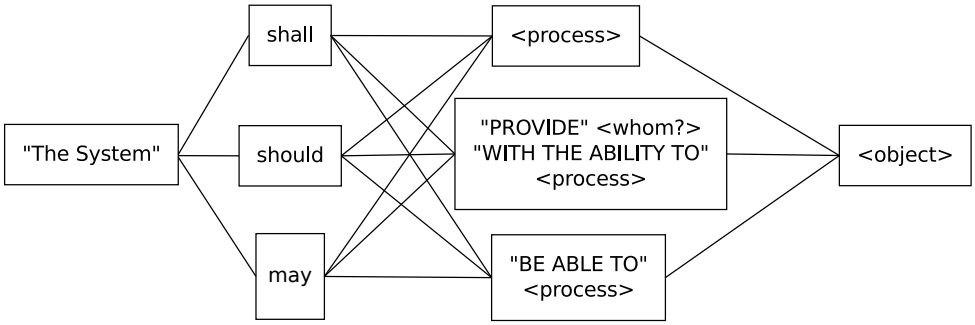

Abbildung 2.11: vlg. [Rup09, S. 166].

Ein weit verbreiteter Ansatz wird in [Rup09] für Deutsch und Englisch beschrieben. Dabei handelt es sich um Schablonen, wie sie in Abbildung 2.11 für eine einfache Anforderungen dargestellt sind. Weitere Schablonenen für die Definition von Glossareinträgen und erweiterte Form für Anforderungen unter bestimmten Bedingungen finden sich in erwähntem Buch. Angemerkt wird noch, dass dem RFC2119[9] folgend, "will" durch "may" ersetzt wurde.

[9] [Bra97]

Beispiele für eine solche natürlichsprachliche Anforderungensformulierung:

```
"The System shall BE ABLE TO sort the userinput"
```

In Deutsch:

```
"Das System muss FÄHIG SEIN die Benutzereingaben zu sortieren"
```

Eine andere Möglichkeit Syntax darzustellen ist die die erweiterte Backus-Naur from. Ein Beispiele wie dadurch eine auf den Schablone von Rupp aufbauenden Syntax für Anforderungen befindet sich in [Wag10, S. 66].

- **Modellbasierte Formalisierung**
  Eine modellbasierter Ansatz, wie er zum Beispiel in [GKBV11] angewandt wird, verwendet ein formal definiertes Modell, um daraufhin von den Erkenntnissen, die über solche Modelle bereits existeren, zu profitieren.

  Das erwähnte Beispiel basiert auf Prädikatenlogik erster Ordnung[10] und definiert eine Anforderung R als Tupel (P, S) mit P als Eigenschaft (Property) bzw. Eigenschaften und S als System, welche P erfüllt ($\forall s \in S : P(s)$).

  Das Modell $\mathcal{M}$ wird dann als Paar $(\mathcal{F}, \mathcal{P})$ beschrieben, wobei $\mathcal{F}$ eine Menge von Funktionssymbolen beschreibt und $\mathcal{P}$ eine Menge von Prädikatensymbolden.

  Zur zusätzlichen natürlichsprachlichen Beschreibung wird wieder auf die unter dem vorhergehende Punkt dargestellte Syntax zurückgegriffen.

- **Ontologiebasierte Formalisierung**
  Aufbauend auf modellbasierter Formalisierung und Webstandards funktioniert der semantische Web Ansatz sehr ähnlich wie der gerade beschriebene. Wobei nicht bis auf prädikatenlogische Ausdrücke zurückgegriffen, sondern auf eine höheren Abstraktionsebene eine Ontologie definiert wird.

  In diesem Zusammenhang ist eine Ontologie eine sprachlich gefasste und formal geordnete Darstellung einer Menge von Begrifflichkeiten, und dient dazu, Objekte und deren Beziehungen zueinander in einem bestimmten Wissensbereich zu definieren.[11]

  Dabei können verschiedenen Aussagemöglichkeiten modelliert werden. Zum Beispiel allgemein oder konkret über das zu entwicklende System:

[10] auch Logik erster Stufe oder nur Prädikatenlogik
[11] vgl. `http://de.wikipedia.org/wiki/Ontologie_(Informatik)`

```
"Systemkomponente" "stelltZuVerfügung" "Funktion" .

"Bildschirm" "ist" "Systemkomponente" .
"Ausgabe" "ist" "Funktion" .

"Bildschirm" "stelltZuVerfügung" "Ausgabe" .
```

2.6 Anforderungsmanagement mit Quality Function Deployment

QFD folgt einem praktischen Prinzip, nämlich des Aufteilens von relevanten Informationen in Kundenanforderungen und in technische Charakteristiken des zu entwickelnden Produktes und das gemeinschaftliche Gegenüberstellen dieser zwei Dimensionen. Gemeinschaftlich bezieht sich auf Markt- und Technologieexperten.

Damit unterstreicht QFD das gleichzeitige Behandeln mehrerer miteinander verbundener Aufgaben und Probleme. Aus Anforderungsmanagementsicht ist es jenes, das sicherstellt, dass sich in technischen Spezifikationen, Strukturen und Funktionen der Kundenwunsch widerspiegelt. Aus technischer Sicht, dass sich gleichzeitig keine Luftschlösser, also Produktspezifikationen ergeben, welche nicht realisierbar/machbar sind.

Wird die Methode richtig angewandt, so erfüllen die dadurch entstehenden technischen Spezifikationen/Anforderungen schon viele Qualitätskriterien. Betrachtet man die Liste von [Ebe10, S. 239] so trägt ein Befolgen der QFD-Methode dazu bei, dass bei den meisten dieser Qualitätskriterien ein gewisses Grundniveau erreicht wird. Besonders herauszunehmen ist neben der schon erwähnten Realisierbarkeit, die Konsistenz, Bewertbarkeit und vor allem die Relevanz.

Dieses Kapitel möchte die Fragestellung, welcher dieser Arbeit zugrunde liegt beantworten: Wie können die vorgestellten semantischen Technologien und die Anforderungsmanagementsoftware, die die QFD-Methode unterstützen, angepasst und erweitert werden, um ein besseres Entwicklungsergebnis zu erreichen?

Kern von QFD ist das sogenannte „House of Quality“ (HoQ), das aus 6 Haupkomponenten besteht. Abgebildet in 2.12. Unterteilt werden Anforderungen im HoQ in zwei Dimensionen. Die erste beschreibt kundennahe Anforderungen und wird im QFD-Jargon auch häufig als „WAS“, „WHATS“ oder selten als „WANTS“ bezeichnet, die zweite beschreibt technische Ausprägungen und werden als „WIE“, respektive als „HOWS“ bezeichnet.

Die einzelnen Komponenten des HoQ in der Reihenfolge, in der sie auch theoretischer Weise

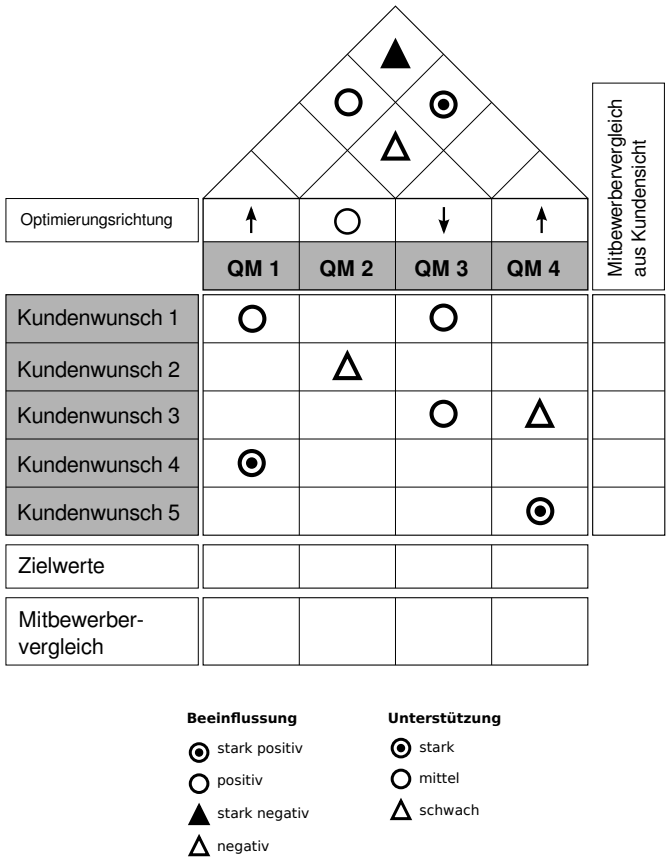

Abbildung 2.12: House of Quality (QM steht für Qualitätsmerkmal)

bearbeitet werden sollten[12], sind:

- **(1) WAS:** Liste von Kundenanforderungen/Kundenwünschen.
  Sammeln und Strukturieren von Kundenanforderungen, dies entspricht etwa dem Punkt Ermitteln und Herauslocken aus der Anforderungssoftwareevaluierung.
- **(2) WARUM:** Bewertung der Priorität von Kundenwünschen mit möglichem Mitbewerbervergleich.
  Die Aufgaben, welche hier unterstützt werden, fallen ebenfalls noch unter den Punkt Ermitteln und Herauslocken.

[12] vgl. [SAKR05, S.362]

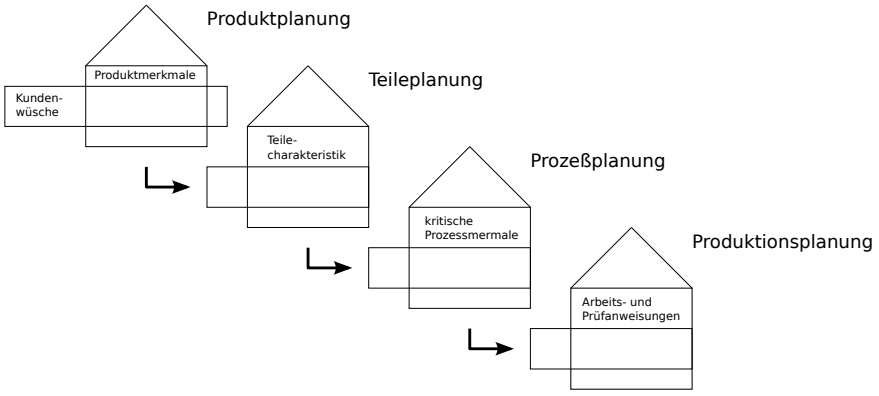

Abbildung 2.13: Vierstufiges QFD mit Stufen aus [BW10]

- **(3) WIE:** Technische Charakteristiken des zu entstehenden Produktes/Dienstleistung. Ähnlich wie die Punkte Spezifikation und Modellierung aus der Evaluierung.
- **(4) WIE & WIE:** Korrelation unter den technischen Charakteristiken (stark, schwach; negativ, positiv).
  Diese Gegenüberstellung soll den Zusammenhang zwischen technischen Charakteristika besser sichtbar machen und kann ebenfalls in die Kategorien Spezifikation und Modellierung aus der Anforderungssoftwareevaluierung eingeordnet werden.
- **(5) WAS & WIE:** Beschreibt die Unterstützung der technischen Charakteristika zur Erfüllung der Kundenwünsche (nicht, schwach, mittel, stark).
  Dieser Teil korreliert mit dem in der Evaluierung behandelten Funktionen zur Unterstützung der Anforderungsanalyse.
- **(6) WIEVIEL:** Angestrebte Ausprägung der technischen Charakteristiken bzw. technischer Mitbewerbervergleich.
  Dieses Gebiet lässt sich mit der Spezifikationsaufgaben von Anforderungen, beschrieben in Kapitel 2, vergleichen.

Wie in [BW10, S. 124] (allg. QM) und in [Wal01] (QM bezogen auf Software) beschreiben, werden für eine QFD mehrere HoQ kaskadiert. Abbildung 2.13 zeigt ein vierstufiges Vorgehen, wobei Art (Dimensionen) und Anzahl der HoQs sich in der Literatur und in der Praxis je nach zu erstellendem Produkttyp und Organisation unterscheiden.

KAPITEL 3

Basis semantischer Features in Anforderungsmanagementsoftware

Semantik, im Sinne der in der IT vorkommenden logischen Semantik, hat die Aufgabe, formale Sprachen zu interpretieren.[1]

Betrachten wir einen trivialen Ausdruck `"Variable1 + Variable2"`.

Mit Syntax beschreiben wir den Aufbau der formalen Sprache. Ein gültiger Ausdruck ist in diesem Fall, dass zuerst eine Variablenbezeichnung steht, dann ein Pluszeichen und daraufolgend wieder eine Variablenbezeichnung. Ein anderer (in der Algebra ungültiger) Ausdruck wäre zum Beispiel `"Variable3 + +"`.

Semantik im Unterschied zu Syntax beschreibt die Bedeutung dieser Sprache bzw., wie ein Ausdruck in dieser Sprache mathematisch interpretiert werden soll. In unserem Beispiel sollen für die Variablenbezeichnungen ihre Werte eingesetzt werden und diese dann miteinander addiert werden.

Dafür wird eine Interpretationsfunktion `I` verwendet. Mit dem Wert 3 und 5 hinter den Variablenbezeichungen `"Variable1"` und `"Variable2"` kann der Wert des gesamten Ausdruckes folgendermaßen berechnet werden.

```
I("Variable1 + Variable2")
= addiere(I("Variable1"), I("Variable2"))
```

[1] vgl. `http://de.wikipedia.org/wiki/Semantik`

```
= addiere(3, I("Variable2"))
= addiere(3, 5)
= 8
```

Im ersten Schritt wird das Pluszeichen durch eine die Funktion „addiere“ interpretiert und die zwei Parameter der binären Funktion werden durch die Interpretationen der jeweiligen Ausdrücke links und rechts des Plus ausgetauscht.

In den nächsten zwei Schritten werden die Interpretationen für die Variablenbezeichner eingesetzt, also die vorher festgelegten Werte 3 und 5.

Im letzten Schritt wird noch das Ergebnis berechnet. Hinter der Funktion `addiere` soll wirklich die mathematische Addition stehen.

Dieses sehr einfache Beispiel soll eine Idee davon geben, was in der Informatik unter Semantik verstanden wird. Für eine vollständige Beschreibung fehlt noch eine formale Definition eines Modells (der Symbole und Funktionen und deren Beziehung zueinander). Grundsätzlich sieht man, dass an zwei Dingen für Anforderungsformalisierung angesetzt werden kann: der formalen Sprache, mit der wir Anforderungen und deren Kontext beschreiben, und der Interpretationsfunktion: wie wir dieser Sprache berechnenbare Bedeutung geben.

3.1 Formalisierung von Anforderungen aufbauend auf Prädikatenlogik

Prädikatenlogik erster Stufe [2] ist eine Erweiterung der Aussagenlogik mit Quantoren ($\forall$ und $\exists$). Produktionsregeln, die beschreiben wie ein wohlgeformter Ausdruck in der Prädikatenlogik aussieht findet sich in [RN07, S. 311]. Eine sehr prägnante Zusammenfassung findet sich im Anhang von [GKBV11].

Ein prädikatenlogikischer Ausdruck kann drei unterschiedliche Elemente[3] enthalten, welche sind:

- **Objekte** stehen zum Beispiel für Systemkomponenten, Baugruppen, eine Anforderung oder der Stakeholder der sie stellt.

[2] wird auch als verkürzt als Logik erster Stufe oder einfach nur Prädikatenlogik bezeichnet. Auch ist die Abkürzung FOL gängig, welche vom englischen Ausdruck "first order logic" stammt

[3] vgl. [RN07, S. 305]

- **Relationen** beschreiben Beziehungen zwischen Objekten. Sie können unär sein, wie zum Beispiel `istEigenständigesProdukt`, aber auch allgemein n-äre. Z.B.: `istBestandteilVon`, welche sich auf ein Systemkomponenten und eine Baugruppe anwenden lässt.
- **Funktionen** sind Relationen, in denen es einen (Rückgabe-) Wert für eine bestimmte Eingabe gibt. Beispielweise `erstesBenachbartesBauteil`, welches ein Bauteilobjekt zurück gibt.

Diese Struktur findet sich auch in der Signatur von prädikatenlogischen Formeln wieder, welche aus folgenden Bestandteilen besteht.

- Konstantensymbole
- Prädikatensymbole
- Funktionssymbole

Formale Definition von Anforderungen

Die im folgenden besprochenen Beispiel wurden sinngemäß aus [GKBV11] übersetzt.

Es seien $R_1 = (P_1, S_1)$ und $R_2 = (P_2, S_2)$ zwei Anforderungen, wobei (wie im ersten Kapitel bereits erwähnt) eine Anforderung ein Tupel (P, S) ist. Hier steht P für Eigenschaft bzw. Eigenschaften und S für Systeme welche die Eigenschaft(en) P aufweisen. $\forall s \in S : P(s)$.

Vorstellen kann man sich P zum Beispiel als Relation *bietetSchnittstelle*, und wenn diese Relation das System S zu $wahr$ auswertet ($bietetSchnittstelle(S) = wahr$) verfügt das System über eine Schnittstelle.

Formale Definition von Relationen

Ein prädikatenlogischer Ausdruck, welcher eine Beziehung zwischen Anforderungen beschreibt, sieht jetzt zum Beispiel folgendermaßem aus:

$$\forall s \in S_1 : s \in S_2 \wedge \exists s \in S_2 : s \notin S_1$$

Dies liest sich: für alle Elemente s in S_1 gilt, s ist Element von S_2 *und* es existiert (mindestens) ein s aus S_2, für das gilt s ist kein Element von S_1. Obwohl in Allquantorausdruck und in Existenzquantorausdruck die selbe Variable s steht, besteht kein Zusammenhang zwischen den beiden Varibalen bis auf der konjunktiven Verknüpfung der Terme, die sie enthalten.

In bezug auf Anforderungen bedeutet dies: Sind alle Systeme in S_1 auch in S_2, aber es gibt mindestens ein System, dass nur in S_2 aber nicht in S_1 dann bedeutet das R_1 *requires* R_2.

Prinzipiell könnte man jetzt jedes Paar aus Anforderungen in diese Aussage einsetzen, an eine Interpretationsfunktion übergeben, damit den Wahrheitsgehalt berechnen und somit feststellen, welche Anforderung welche anderen „erfordern“. Ergebnis einer solchen Berechnung wäre für jedes Anforderungspaar ein Wahrheitswert $Wahr$ oder $Falsch$.

Allgemein kann in Logiken die „Bedeutung“ mit dem Wahrheitsgehalt von Aussagen (Sätzen) in Bezug auf ein Modell gleich gestellt werden.

Ein weiteres extensionales Beispiel, in denen die Eigenschaft P verwendet findet, ist die Anforderungsbeziehung *conflicts*:

R_1 steht in Konflikt mit R_2, wenn

$$\neg\exists s : (s \in S_1 \wedge s \in S_2 \wedge P_1(s) \wedge P_2(s))$$

Es existiert kein System s, für welches sich in den Mengen S_1 und S_2 befindet und die beiden Eigenschaften $P_1(s)$ und $P_2(s)$ erfüllen.

Ein Konflikt ist demnach vorhanden, wenn das Erfüllen einer Anforderungen das Erfüllen einer anderen Anforderung ausschließt.

Auch hier kann für alle Anforderungskonstellationen berechnet werden, ob sich der Ausdruck zu $Wahr$ oder $Falsch$ auswertet und damit die Beziehung zwischen den Anforderungen zutrifft oder nicht. Das Berechnen, ob solche Aussagen zutreffen ist zwar praktisch, aber noch nicht die gewünschte Funktion.

Formale Definition von prädikatenlogischen Modellen

Um in weiterer Folge (semi-automatisches) Schließen zu ermöglichen, ist es notwendig ein prädikatenlogisches Modell zu definieren. In diesem Fall ist das formale Modell $\mathcal{M}$ ein Paar aus $(\mathcal{F}, \mathcal{P})$, $\mathcal{F}$ ist eine Menge von Funktionssymbolen und $\mathcal{P}$ eine Menge von Prädikatensymbolen. Dazu kommt A als nicht leere Menge, die alle Werte im Modell beinhaltet.

Für jede Funktion f aus $\mathcal{F}$ bildet A^n auf A ab: $f^{\mathcal{M}} : A^n \rightarrow A$.

Jedes $P \in \mathcal{P}$ mit n Argumenten ist Teilmenge von A^n. $P^{\mathcal{M}} \subseteq A^n$

Die Prädikatensymbole sind $all\text{-}in\text{-}part^{\mathcal{M}}$, $all\text{-}in\text{-}whole^{\mathcal{M}}$, $some\text{-}implies\text{-}in^{\mathcal{M}}$, $all\text{-}implies\text{-}in^{\mathcal{M}}$ und $all\text{-}equals\text{-}in^{\mathcal{M}}$. Auch unsere Beispielbeziehung gehört in diese Menge $\mathcal{P}$: $bietetSchnittstelle^{\mathcal{M}}$.

Beispielhaft sollen hier noch zwei dieser Prädikatensymbole genauer betrachtet werden.

Dazu werden folgende Definitionen benötigt:

- xs und ys sind Formeln in konjunktiver Normalform aus A,
- $set^{\mathcal{M}}$ ist eine Funktion, welche eine Formel aus A auf die Menge ihrer Klauseln in konjunktiver Normalform abbildet
- $||^{\mathcal{M}}$ ist eine Funktion, welche ein Formel aus A auf die Anzahl an Klauseln abbildet
- $eval^{\mathcal{M}}$ ist ein Prädikat, welches eine Formel aus A aufnimmt und für deren Wahrheitswert steht.
- f ist eine Funktion, welche von $set(xs)$ zu $set(ys)$ abbildet
- $implies(xs, ys) =_{def} eval(xs) \rightarrow eval(ys)$
- $related(x, y) =_{def} implies(x, y) \vee equals(x, y)$, wobei $equals$ genau dann zu $Wahr$ ausgewertet wird, wenn x und y die selben Prädikatensymbole mit den selben Argumenten enthält.

Damit kann nun definiert werden:

$$all\text{-}in\text{-}part(xs, ys) =_{def} (\forall x \in set(xs) : related(x, f(x))) \wedge (|xs| < |ys|)$$

$$all\text{-}in\text{-}whole(xs, ys) =_{def} (\forall x \in set(xs) : related(x, f(x))) \wedge (|xs| = |ys|)$$

Für später wesentlich ist, dass die zwei beschriebenen Prädikate disjunkt sind ($=$ und $<$).

Für eine detailierte Erklärung sowie weitere Definitionen wird auf den Anhang B von [GKBV11] verwiesen.

Mapping zwischen Anforderungsbeziehungen und den theoretischen Modellbeziehungen

Die Beziehungen zwischen Anforderungen („requires“, „refine“, „containes“, etc.) lassen sich auf Aussagen im formalen Modell übersetzen.

Bezogen auf Systeme

$$(S_1 \subset S_2) \text{ iff } (R_1 \textit{ requires } R_2)$$

$$(S_1 \subset S_2) \text{ if } (R_1 \textit{ refines } R_2)$$

$$(S_1 \subset S_2) \text{ if } (R_1 \textit{ contains } R_2)$$

$$((S_1 \cap S_2) = \emptyset) \text{ iff } (R_1 \textit{ conflicts } R_2)$$

Desweitern können die Anforderungsrelationen auf Beziehungen zwischen den Eigenschaften P übersetzt werden. Beispielhaft:

$$all\text{-}in\text{-}whole(P_1, P_2) \wedge some\text{-}implies\text{-}in(P_1, P_2) \text{ iff } (R_1 \textit{ refines } R_2)$$

$$all\text{-}in\text{-}part(P_1, P_2) \wedge all\text{-}equals\text{-}in(P_2, P_1) \text{ iff } (R_1 \textit{ contains } R_2)$$

Weitere Mappings finden sich in der bereits mehrfach erwähnten Quelle.

Logisches Schließen

Gewünscht ist es, Erkenntnisse aus mehreren derart definierten Anforderungbeziehungen gewinnen zu können. Zum Beispiel inwiefern verschiedene Anforderungen auch indirekt (über andere Anforderungsbeziehungen) in Konflikt miteinander stehen. Dies fällt in das Gebiet "reasoning" bzw. "inferencing".

Inferenz

Den prinzipiellen Ablauf, um von explizit gegeben Beziehungen auf darin implizit enthaltene Beziehung zu schließen, soll folgendes kleine Beispiel zeigen.

Folgende Aussage:

$$(R_1 \textit{ refines } R_2) \wedge (R_2 \textit{ contains } R_3) \rightarrow (R_1 \textit{ requires } R_3)$$

Unter der Anwendung der Mappingregeln für die Menge an Systemen folgt:

$$= (S_1 \subset S_2) \wedge (S_2 \subset S_3)$$

unter Berücksichtigung der Transitivität des Mengenoperator folgt

$$= (S_1 \subset S_3)$$

Dies bedeutet, die obenstehende Aussage ist $wahr$. Ein Reasoner ist fähig, automatisch solche Aussagen zu finden und damit neue explizite Fakten zu generieren. Ein händisches Vorgehen, wie hier ist nicht mehr notwendig.

Inkonsistenzen

Nach dem Generieren neuer Fakten durch Inferenz ist eine weitere wichtige Aufgabe das Finden von Inkonsistenzen.

Folgendes Beispiel soll illustrieren, wie dies prinzipiell funktiniert. Wir haben zwei Aussagen (bzw. Anforderungsbeziehungen) über zwei Anfoderungen R_1 und R_2 festgelegt.

$$(R_1 \textit{ refines } R_2)$$

$$(R_1 \textit{ contains } R_2)$$

Als Mensch sieht man bei diesem Beispiel auf den ersten Blick, dass etwas nicht stimmt. Dies zu sehen wird aber mit steigender Anzahl von Anfoderungsrelationen äußerst schwierig und damit fehleranfällig. Dementsprechend ist eine automatisierbare Vorgehensweise wünschenswert.

Die beiden Aussagen konjunktiv verknüpft:

$$(R_1 \textit{ refines } R_2) \wedge (R_1 \textit{ contains } R_2)$$

Dann übersetzen wir den ersten Teil mittels Mapping in

$$\textit{all-in-whole}(P_1, P_2) \wedge \textit{some-implies-in}(P_1, P_2)$$

tun dies auch mit dem zweiten Teil des Ausdruckes

$$\textit{all-in-part}(P_2, P_1) \wedge \textit{all-equals-in}(P_2, P_1)$$

Da $\textit{all-in-whole}$ und $\textit{all-in-part}$ disjunkt sind, und $\textit{some-implies-in}(P_1, P_2)$ der symmetrischen $\textit{all-equals-in}(P_2, P_1)$ Beziehung widerspricht, kann der gesamte Ausdruck **nicht** zu $Wahr$ ausgewertet werden. Damit lernen wir, dass es in dieser kleinen Menge von Aussagen einen Widerspruch geben muss.

Weitere Eigenschaften von diesen theoretischen Beziehungen und eine Liste von Ableitungsregeln finden sich im Anhang B von [GKBV11].

Reasoner

Auch die Konsistenzprüfung wird nicht von Hand ausgeführt, sondern wird automatisch von sogenanten „Reasonern“ erledigt. Solche Reasoner basieren auf logischen Kalkülen, welche wiederum aus Axiomen und Ableitungsregeln bestehen. Bekannte Verfahren die dabei eingesetzt werden, sind das semantische Tableaux[4] oder die Resulotion[5].

[4] Theoretische Informatik und Logik Skriptum, Institut für Computersprachen, TU Wien

[5] vgl. [RN07, S. 368]

Eine wichtige Eigenschaft von implementierten Reasonern für den produktiven Einsatz ist deren Effizienz, welche wiederum stark von der zugrunde gelegten Sprache (bzw. Sprachklasse) abhängt. Eine Klassifizerung von Sprachen in diesem Bezug findet sich in [Baa07, S. 528]. Den Trade-Off zwischen Berechnungskomplexität und Ausdruckstärke[6] wird ausführlichst in den Kapiteln zwei und drei von [Baa07] behandelt.

[GKBV11] beschreibt zwar noch weitere Anforderungsrelationen und zeigt deren Mapping auf theoretische Beziehungen. Auch wird einen Software Prototyp vorgestellt[7], jedoch wird kein Reasoner implementiert, sondern für diese Funktionalität auf einen OWL-Reasoner zurückgegriffen. OWL ist eine Sprache aus den semantischem Web, dessen Grundlagen im nachfolgenden Abschnitt einführt wird.

3.2 Formalisierung von Anforderung mittels „Semantic Web“ Techniken

Semantik im Sinne von „semantischen Webs“ ist die Anreicherung von Dokumenten in einer für Algorithmen „verstehbaren“ Form. Semantische Web Technologien basieren ebenfalls auf den Prinzipien von Logiken und sind damit nicht grundlegend unterschiedlich von dem beschriebenen Ansatz mit Prädikatenlogik, wobei die Verwendung von semantischen Web Standards für die hier besprochenen Zwecke Vorteile mit sich bringen.

Zum Beispiel sind in Web Ontology Language (OWL) im Kontrast zu Prädikatenlogik erster Stufe Aussagen über Kardinaltitäten möglich. Weiters hat es den praktischen Vorteil, dass Konzeptbeziehungen nicht erneut formal eingeführt werden müssen, sondern auf bereits vorhandene einfache Beschreibungsmittel und Techniken zurückgegriffen werden kann. Diese Komponenten sind:

- **Literale** sind Datenwerte eines spezifischen Typs. Z.B. String, Number, Boolean etc. Es wird auf die primitiven Datentypen von XML-Schema (XSD)[8] zurückgegriffen.
- **Instanzen (Individuen)** sind konkrete Elemente in einer Ontologie (wie Objekte in der Prädikatenlogik). Beispiele zusätzlich zu den schon vorher genannten, wären ein konkretes System, auf das sich Anforderungen beziehen, die Kontaktadressen eines Stakeholders oder eine bestimmte Anforderung.

[6] Englisch: "computational complexitiy of reasoning and expressiveness of the language"
[7] `http://trese.cs.utwente.nl/tric/`
[8] http://www.w3.org/XML/Schema.html

- **Klassen (Konzepte, Typen)** sind Abstraktionen, um bestimmte Gruppen von Elementen zu beschreiben, die in irgendeiner Form eine Gemeinsamkeit teilen. Z.B. funktionale Anforderungen, firmeninterne Stakeholder.
- **Beziehungen (Properties, Eigenschaften)** beschreiben Verbindungen zwischen Elementen in einer Ontologie. Zum Beispiel, dass ein Stakeholder „Benutzer“ mit einer Anforderung „bessere Kollaborationsmöglichkeiten“ in Beziehung steht, die als „wünschtSich“ bezeichnet werden kann. Es können auch Relationen zwischen Klassen beschrieben werden. Typen der Klasse Stakeholder können mit Typen der Klasse Anforderung in der Beziehung „wünschtSich“ stehen.
- **Regeln** bieten die Möglichkeit Bedingungen, die von Elementen in einer Ontologie eingehalten werden müssen, zu beschreiben.

Die Standards für das semantische Web sind mehrstufig aufgebaut, mit jeder Stufe gewinnen sie an Ausdrucksstärke. Gleichzeitig steigt aber auch die Komplexität bzw. der Aufwand, um Aussagen darin zu berechnen. Das Fundament wird gebildet durch XML, welches selbst auf UNICODE Zeichensatz zurückgreift, und Uniform Resource Identifier (URI) bzw. deren internationalisierte Variante IRI, um Ressourcen eindeutig identifizieren zu können. Eine Übersicht gibt Abbildung 3.1.

Im Folgenden wird kurz auf die einzelnen Standards eingegangen:

- **RDF**[9]
  stehst für „Resource Description Framework“ und bezeichnet „eine technische Herangehensweise im Internet zur Formulierung logischer Aussagen über beliebige Dinge (Ressourcen)“ [10].

  Eine Aussage in RDF besteht immer aus einem Tripel: Subjekt, Prädikat und Objekt. Ressourcen werden global eindeutig mit URI[11] bzw. IRI identifiziert.

  Mit dem Standard RDF/XML [12] wird beschrieben, wie ein RDF-Graph mittels XML ausgedrückt werden soll.

  In dieser Arbeit wird in weiterer Folge Turtle[13] als Schreibweise verwendet. Grund dafür ist die leichtere Lesbarkeit.

[9] `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`
[10] vlg. `http://de.wikipedia.org/wiki/RDF`
[11] `http://www.w3.org/TR/2001/NOTE-uri-clarification-20010921/`
[12] `http://www.w3.org/TR/2004/REC-rdf-syntax-grammer-20040210/`
[13] `http://www.w3.org/TeamSubmission/turtle/`

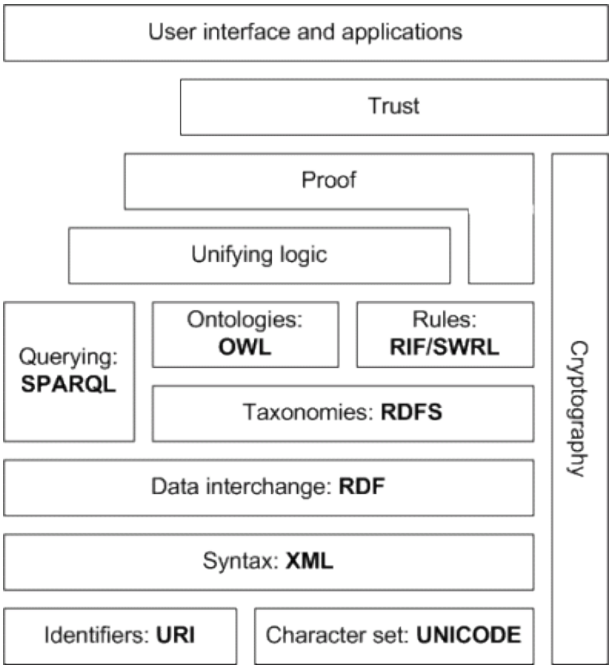

Abbildung 3.1: Semantic Web Stack

Ein Beispiel in Turtle:

```
@prefix  ex:   <http://www.example.org/ns#> .
@prefix  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns> .
@prefix  xsd:  <http://www.w3.org/2001/XMLSchema#> .

ex:Anf1  rdf:type         ex:Anforderungen .
ex:Anf1  ex:Bezeichnung   "SortierbareEingaben"^^xsd:string .
ex:Anf1  ex:Beschreibung  "sortierbare Benutzereingaben"@de .
ex:Anf1  ex:requires      ex:Anf2 .
```

Die ersten drei Zeilen, die jeweils mit `@prefix` beginnen, definieren Abkürzungen, um dann in weiter Folge nicht bei jeder Ressource die vollständige URI angeben zu müssen, d.h. es erlaubt statt `http://example.org/ns#Anf1 ex:Anf1` zu schreiben. Im XML Beispiel sieht man, dass es sich um eine Namespacedeklaration handelt.

Dann wird die Ressource `ex:Anf1` beschrieben. Zuerst, dass sie vom Typ `ex:Anforderungen` ist. Dann die Bezeichnung festgelegt und ein Beschreibung festgelegt. Die Letzeren unterscheiden sich dadurch, dass die Bezeichnung vom XML-Schema Datentyp String ist und die Beschreibung einen Text in deutscher Sprache festlegt.

Die letzte Zeile sagt noch aus, dass `ex:Anf1` mit einer weiteren Ressource `ex:Anf2` in der Relation `ex:requires` steht.

Im Vergleich dazu eine äquivalentem XML-Beschreibung:

```
<rdf:RDF
  xmlns:ex="http://example.org/ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Description rdf:about="http://example.org/ns#Anf1">
  <rdf:type rdf:resource="http://example.org/ns#Anforderungen">
  <ex:Bezeichnung
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    SortierbareEingaben</ex:Bezeichnung>
  <ex:Beschreibung xml:lang="de">sortierbare Benutzereingaben
  </ex:Beschreibung>
  <ex:requires rdf:resource="http://example.org/ns#Anf2">
</rdf:Description>
</rdf:RDFS>
```

- **RDFS**[14]

  steht für „RDF-Schema“ und ist eine Erweiterung zu RDF, was bedeutet, dass alle gültigen RDFS Dokumente auch gültige RDF Dokumente sind. Zusätzlich können noch Eigenschaften (sogenannte Properties) und Klassen genauer beschrieben werden.

  Für Properties lassen sich zum Beispiel Bild- und Wertebereich festlegen. Auch lassen sich mit RDFS bereits hierarchische Klassenbeziehungen modellieren.

  ```
  @prefix ex:   <http://www.example.org/ns#> .
  @prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns> .
  @prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
  @prefix rdfs: <http://www.w3.org/TR/rdf-schema/> .

  ex:funktionaleAnforderungen rdfs:subClassOf ex:Anforderungen .

  ex:requires rdfs:range  ex:Anforderungen ;
              rdfs:domain ex:Anforderungen .
  ```

  Die `@prefix`-Zeilen beschreiben im Grunde wieder Abkürzungen. Die Zeile mit `rdfs:subClassOf` drückt aus, dass die Ressourcen `ex:Kundenanforderungen` eine Unterklasse von `ex:Anforderungen` sind.

[14] urlhttp://www.w3.org/TR/rdf-schema/

Die letzten beiden Zeilen beschreiben die Beziehung `ex:requires` genauer. `rdfs:range` legt den Wertebereich fest und `rdfs:domain` den Definitionsbereich, welche in diesem Beispiel mit `ex:Anforderungen` übereinstimmen. Eine andere Beziehung `ex:hatAlter` hätte sinnvollerweise unterschiedliche Definitions- und Wertebereich. Als Definitionsbereich eine Person oder ein Objekt und als Wertebereich eine Datenrange eine nicht negative Zahl.

Für eine komplette Liste der Sprachkonstrukte kann auf die Standards des W3C verwiesen werden, bzw. mit deutschsprachiger Einführung in [HKRS08] Kapitel 3.

- **OWL**[15]
  steht für „Web Ontology Language“ und ist eine Erweiterung zu RDFS. Alles was in RDFS ausgedrückt werden kann, kann auch in OWL ausgedrückt werden. Jedoch kommen in OWL zusätzliche Ausdrucksmöglichkeiten dazu. Es lassen sich mit OWL Aussagen negieren. Damit in Zusammenhang stehend gibt es auch die Möglichkeit, disjunktive Aussagen über zwei Elemente zu treffen, zum Beispiel, dass eine Instanz nur in einer von zwei Klassen enthalten sein kann, aber nicht in beiden gleichzeitig.

  Des weiteren können im Vergleich zu RDFS Klassen als äquivalent markiert werden, es lassen sich Klassen duch Enumerierung ihrer Mitglieder definieren und OWL ermöglicht Aussagen über Kardinalitäten, zum Beispiel, dass ein Entwicklungsprojekt mehrere (>1) Entwickler haben muss, oder dass genau ein Verantwortlicher/Verantwortliche für das Projekt definiert sein muss ($=1$).

  Letztere Fähigkeit überschreitet auch die Ausdrucksfähigkeit von Prädikatenlogik erster Stufe, was in gewisser Hinsicht auch das „erster Stufe“ erklärt.

  Erweiterung des Beispiels mit Aussagen, die nur in OWL getätigt werden können:

```
@prefix ex:   <http://www.example.org/ns#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix owl:  <http://www.w3.org/TR/2009/REC-owl2-primer-20091027/> .

ex:Anforderungen rdf:type owl:Class .

ex:funktionaleAnforderungen rdf:type :Anforderungen ;
    owl:disjointWith ex:nichtfunktionaleAnforderungen .

ex:nichtfunktionaleAnforderungen rdf:type ex:Anforderungen .
```

[15] `http://www.w3.org/TR/2009/REC-owl2-overview-20091027/`

Zu beachten ist, dass in diesem Beispiel `ex:Anforderungen` vom Typ `owl:Class` ist, was im ersten Beispiel bei RDF noch unterschlagen wurde.

Interessant ist die Aussage `ex:funktionaleAnforderungen owl:disjointWith ex:nichtfuntionaleAnforderungen`, die zum Ausdruck bringt, dass eine Anforderung nur in einer der beiden Unterklassen sein kann.

OWL liegt mittlerweile in Version 2 vor und für OWL 2 existieren mit EL/QL/RL Profile, welche als Teil oder Untersprachen von OWL 2 verstanden werden können[16]. Sie geben die Ausdrucksstärke des gesamten Sprachstandards von OWL 2 auf, um im Gegenzug mehr Effizienz beim Reasoning zu erlangen.

- **SWRL**[17]
  steht für Semantic Web Rule Language. Ein Ausdruck in SWRL ist eine Implikation und besteht aus einem Körper und einer Konsequenz. Sind alle Bedingugnen im Körper erfüllt, wird auch die Konsequenz erfüllt.

  ```
  funktionaleAnforderungen(?f), nichtfunktionaleAnforderungen(?nf),
  requires(?f, ?nf)
  -> seltsameAnforderungsKombination(?f, ?nf)
  ```

  Bei der Auswertung dieses Beispiels würde für alle Anforderungen, bei denen eine funktionale Anforderung eine Nichtfunktionale erfordert, die Beziehung `seltsameAnforderungsKombination` hinzugefügt werden.

Reasoning

Wie für die Prädikatenlogik besprochen können Abbildungsregeln und Axiome genutzt werden, um über Aussagen in OWL formulierten Ontologien zu schlussfolgern. [HKRS08] beschreiben diese für RDFS und OWL und erklären einen Algorithmus welcher auf Tableauxverfahreren mit Blocking basiert. Eine Beschreibung für OWL 2 befindet sich in der aktuelleren englischsprachigen Ausgabe des Buches.

Beispiele für Axiome:

```
rdf:type rdf:type rdf:Property .
rdfs:subClassOf rdfs:domain rdfs:Class .
```

[16] `http://www.w3.org/TR-OWL2-profiles/`
[17] `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`

Die RDF Eigenschaft `rdf:type` ist selbst vom Typ `rdf:Property`. Der zweite Satz beschreibt, dass der Definitionsbereich von `rdfs:subClassOf` `rdfs:Class`. Axiome sind demnach Aussagen, die immer $Wahr$ sind (sein müssen).

Beispiel für eine Ableitungsregeln zur syntaktischen Schlussfolgerung:

$$(\texttt{rdfs 9})\frac{u\ \texttt{rdfs:subClassOf}\ x\ .\quad v\ \texttt{rdf:type}\ u\ .}{v\ \texttt{rdf:type}\ x\ .}$$

Mittels der Regel `rdfs 9`[18] kann geschlossen werden: Ist u `rdfs:subClassOf` x und ist v `rdf:type` u, dann ist v vom `rdf:type` x.

Prinzipielle Idee ist es eine komplexere Aussage (eine Wissensbasis) mithilfe von Ableitungsregeln auf Axiome zurückzuführen. Gelingt dies nicht, können Gegenbeispiele gefunden werden. Bzw. Informationen darüber, welcher Teil der Wissensbasis inkonsistent ist.

[18] [HKRS08, S. 112]

KAPITEL 4

Anforderungsmanagementsoftware-funktionen

Anforderungsmanagement ist fest mit der Organisation, in der sie angewandt wird, verknüpft. Folgt man den Konzepten und Best Practices aus der Literatur, werden dadurch viele Prozesse im Unternehmen festgelegt. Wobei es zu diesen meist passende Softwaresysteme gibt. Von Beschaffungsseite für ein Unternehmen mit bereits etablierten Prozessen, bedeutet dies, dass ein anzuschaffendes Softwaresystem gut mit den Arbeitsabläufen und den bereits vorhandenen Softwaresystememem zusammenarbeiten soll. Werden zum Beispiel schon Methoden wie QFD, FMEA und TRIZ angwendet, existieren dafür eigene Softwarepackete[1]. [Rup09, S. 418] gibt dem Ganzen eine konkrete Zahl und sagt, dass 80% des Anforderungsmanagementkonzeptes fertig gestellt sein soll, bevor mit einer Anforderungssoftwareevaluierung begonnen werden soll. Ansonsten werde die Software die Methoden festlegen.

Natürlich muss zur Verwaltung von Anforderungen nicht unbedingt ein spezielles Softwaresystem angeschafft werden. Es können auch einfache Tabellen mit Tabellenkalkulationsprogramm, Wikis, Workflow-Tools, Entwicklungsumgebungen und Modellierungswerkzeuge angewandt werden. Je nach Projektgröße und gewünschtem Anforderungsmanagement. [Ebe10, S. 321ff] geht auf diese Möglichkeiten detaillierter ein.

Für speziell auf Anforderungsmanagement ausgerichtete Software existeren mehrere Möglichkeiten, deren Features zu evaluieren. Zum Beispiel der sichtenbasierte Ansatz von Dr. Klaus Pohl, vorgestellt in [Sch02, S. 287] der die Beurteilung in folgende Sichten unterteilt:

[1] wie zum Beispiel `http://www.ideacore.com/` oder `http://qfdcapture.com`

- Produktsicht
- Benutzersicht
- Anbietersicht
- Projektsicht
- Prozesssicht
- technische Sicht
- betriebswirtschaftliche Sicht

Eine sehr ähnliche Vorgehensweise wird in [Rup09] vorgeschlagen, wobei auf einen Evaluierungsbogen[2] verwiesen wird, dessen Sichten sich nur gering von Vorhergehenden unterscheiden. Anstatt einer Projekt- und einer Prozesssicht gibt es eine Umsetzungssicht (Managementsicht) und auf eine Anbietersicht wird zugunsten einer Entwicklungssicht verzichtet.

Die hier nachfolgend detaillierter vorgestellte Analyse folgt der Norm ISO/IEC 24766 Information technology – Systems and Software Engineering – Guide for Requirements Engineering Tool Capabilities [ISO09] bzw. nach der in [dGNA+12] überarbeiteten Version. Zum Einen soll dadurch auf die aktuelleren Studienergebnisse zurückgegriffen werden können. Zum Zweiten bringt der Normungscharakter den Vorteil, dass mögliche zukünftige Evaluierungsstudien ebenfalls dieser Struktur folgen.

Die acht Hauptpunkte für Anforderungsmanagementsoftware sind:

- Ermittlung/ Herauslocken (Elicitation)
- Analyse (Analysis)
- Spezifikation (Specification)
- Modellierung (Modeling)
- Verifikation und Validierung (Verification and validation V&V)
- Management
- Nachvollziehbarkeit, Verfolgbarkeit (Traceability)
- Andere Features (Other capabilities)

[2] `http://www.sophist.de/fileadmin/SOPHIST/Downloadcontent/Homepage-Downloads/Bewertung_RM-Tools.rar`

Es sei noch erwähnt, dass in verschieden Bereichen nach den gleichen Funktionen gefragt wird. So ist zum Beispiel der Export und Import zu anderen Softwaresystem in verschieden Kategorien wichtig. Die Wiederholungen werden hier teilweise, wie auch weitere Funktionen, die nicht unmittelbar mit Anforderungsmanagment (zum Beispiel Rechtschreibprüfung) zu tun haben, unterdrückt. Dies bedeutet jedoch nicht, dass solche Funktionen für den alltäglichen Gebrauch von Anforderungsmanagmentsoftware sehr nützlich und damit für viele Anwender unverzichtbar sind.

4.1 Ermittlung/ Herauslocken (Elicitation)

In dieser Gruppe soll nach Features von Anforderungsmanagementsoftware gefragt werden, die ihren Fokus auf das Unterstützen beim Finden von Stakeholdern, Festhalten von Geschäfts-/Kundenanforderungen und Einteilung in passende Kategorien legen.

Fangen/Festhalten (Requirements Capture)

Hierbei geht es um Features die es dem Benutzer erlauben, Dokumente von Interviews, Workshops, Beobachtungen etc. zu speichern. Zudem soll die Anforderungsmanagementsoftware Stackholdern, erlauben Informationen (wie zum Beispiel Kontakte, Kommentare) zu verwalten. Des weiteren sollen hierachische Zusammenhänge von Anforderungen erzeugt und mit Begründung hinterlegt werden können.

Speichern und Handhaben von Anforderungsattributen die zum Klassifizieren und Kategorisieren dienen können bereits während der Anforderungsindentifikation unterstützt werden.

Zudem soll die Software Funktionen zur Suche, automatisierter Erstellung von Berichten (z.B.: Anforderungshistorie/Änderungshistorie oder Crystal Reports), Speichern von zusätzlichen Daten wie Bildern, Text-Dateien etc. zu Verfügung stellen.

Schablonen/Checklisten (Elicitation templates and checklists)

Um eine konstistente Struktur zum Festhalten von Anforderungsbeschreibungen und anforderungsrelevanten Informationen zu ermöglichen, kann Anforderungsmanagementsoftware Vorlagen und Checklisten zu Verfügung stellen. Dies können einfache Listen sein, aber auch komplexere Formulare, die z.B. zur Priorisierung von Anforderungen dienen.

Der für die Studie benutzte Fragebogen fragt konkret nach Schablonen für QFD (Quality Function Deployment - House of Quality).

Schnittstellen (Importing and exporting to and from other sources)

Anforderungsmanagementsoftware soll mit den im Unternehmen bestehenden Datenquellen zusammenarbeiten. Zum Beispiel direkt mit Daten aus Design-, Projektmanagement- und Bürosoftware, beziehungsweise mit standardisierten Dateiformaten, wie CSV (Comma Seperated Values) oder XML umgehen können. Weitergehend wird noch berücksichtigt, dass Anforderungsmanagementsoftware auch offene (ReqIF/RIF [OMG][3]) sowie nicht offene Formate, welche sich speziell auf Anforderungen beziehen, unterstützen können.

Dokumentation (Elicitation documentation)

Das Ergebnis der gesamten Anforderungsermittlung soll in verschiedenen Formaten editier- und abfragbar sein. Einerseites in textueller Form aber auch in Form von Bildern, Tabellen, Gleichungen, Dimensionierungsdokumenten etc.

4.2 Analyse (Analysis)

In dieser Gruppe wird nach Fähigkeiten von Anforderungsmanagementsoftware gefragt werden, die es erlauben, abstrakte Anforderungen besser in detaillierte zu übersetzen, Machbarkeit einzuschäten, Prioritäten auszuhandeln, Konflikte zu indentifizieren, Inkonsistenzen und Unvollständigkeiten zu beheben.

Qualitätsanalyse (Quality requirements analysis)

Sie erlaubt dem System, Anforderungen in unterschiedlicher Art zu verwalten. Zum Beispiel für verschiedene Attribute, Strategien/Regelwerke (Policies) und Einschränkungen (Constraints).

Machbarkeitsanalyse (Feasibility analysis)

Inwiefern unterstützt die Software den Benutzer beim Evaluieren unter bestimmten Randbedingungen, wie z.B. Kosten. Des weiteren soll das System dem Benutzer helfen, einzuschätzen, ob die geforderte Leistung technisch und ökonomisch erreicht werden können.

Attributanalyse (Attribute analysis)

Die Evaluierung fragt danach, ob die Software erlaubt, benutzerdefinierte Attribute ja nach Projektbedürfnissen hinzuzufügen. Attribute bei einzelnen Anforderungen sollen ermöglichen, dass

[3] Requirements Interchange Format (`http://www.omg.org/spec/ReqIF/`)

Anforderungen nach diesen Attributen sortiert und gruppiert werden können. Anforderungen, die bestimmte Attribute nicht aufweisen, sollen aufgefunden und markiert werden können. Zudem sollen Veränderungen von Attributen Regeln einhalten und eine Veränderungshistorie gespeichtert werden. Zum Beispiel welcher Benutzer wann welche Änderung in das System eingetragen hat.

Risikoanalyse und -managment (Risk analysis and management)

Das System soll beim Identifizieren und Festhatlen von Risiken helfen. Dies kann durch Anbieten von einfachen Schablonen oder komplexeren für gängige Vorgehensweisen (FMEA) erfolgen. Auch soll das System zum Austausch mit spezialisierter Risikoanalysesoftware fähig sein. Wobei der Einfluss von Anforderungen auf Risiken (ob steigernd oder mindernd) festgehalten werden können soll.

4.3 Spezifikation (Specification)

In dieser Gruppe soll nach Fähigkeiten gefragt werden, die sich darauf konzentrieren, die Funktion des erstellenden Systems und die Randbedingungen, die es berücksichtigen muss, festzuhalten.

Spezifikationsdokumentation (Requirements specification documentation)

Anforderungsmanagementsysteme sollen Berichte erstellen können, in denen funktionale Anforderungen und deren Ausnahmen aufgelistet werden. Dies soll in Standardformate (RTF/HTML) und Grafiken von Diagrammen (JPEG, PNG, SVG) exportierbar sein. Dabei ist auch eine Synchronisation zwischen Ausgabe und RE-Tool wünschenswert.

4.4 Modellierung (Modeling)

Funktionen in dieser Kategorie beziehen sich auf das Abstrahieren in Modellen und die Unterstützung, die Anforderungsmanagementsoftware dafür liefern kann.

Modellierungsanalyse (Modeling analysis)

Anfoderungsmanagementsoftware soll Funktionen zum Im- und Export von Modellierungstools bereitstellen. (Wolfram System Modeller ist ein Beispiel für einen solches Modellierungstool für Systeme auch außerhalb des Software Engineerings).

Zudem soll das Speichern und Verwalten von Schablonen für goal-oriented scenarios (z.B. Simulation von Geschäftsszenarien), strategischen Dinge oder benutzerdefinierte Szenarios ermöglicht werden. Solche Szenarien sollen benutzt werden können, um Anforderungen auf Basis von Geschäftsfällen zu evaluieren.

Modellierungs- und Spezifikationssprachen (Modeling ans specificatoion languages)

Unterstützt die Software auch nicht natürlichsprachliche Ausdrücke wie Kontextdiagramme, "Conceptional Domain Models" (z.B. Klassendiagramme), Business Process Model Notation (BPMN), Goal Models, SysML Artefacts, UML artefacts, DATA Flow Diagramm DFD (Flußdiagramm), Entity Relationshipt diagramm.

Hier wird es auch für semantische Technologien interessant. Zum Beispiel: Kann mittels Anforderungsmanagementsoftware eine Wissensbasis aufgebaut/erweitert werden, um danach (semi-) automatisch Aussagen über Inkonsistenzen zu tätigen?

4.5 Verifikation und Validierung (Verification and validation V&V)

In dieser Kategorie wird nach verschiedenen Fähigkeiten (von Anf.mg.software systemem) gefragt, die das Validieren und Verifizieren von Anforderungen an dem zu entstehenem/entstandenem Produkt/Dienstleistung unterstützen.

V&V

Anforderungsmanagementsoftware kann diese Aufgaben dadurch unterstützen, indem sie Möglichkeiten zur Verfügung stellt, um Verifikations- und Validierungsanweisungspläne und Vorgehensweisen zu speichern, Reviews und Verbesserungen von solchen zuzulassen. Dazu kommt noch das zu Verfügung stellen von Schnittstellen zu anderen Tools, wie zum Beispiel Testsoftware.

Weitere Funktionen können sein: Das Aufzeigen von Anforderungen ohne Verifikation oder Validierungpläne oder umgekehrt das Hinweisen auf V&V-Pläne ohne zugewiesenen Anforderungen.

4.6 Management

Baseline of the requirements

Aus verschiedenen Versionen von Anforderungen fasst die Baseline jene Anforderungen zusammen, auf die sich die Benuter und/oder Stakeholder geeinigt haben. Tagging bzw. ein Pflichtenheft kommt einer „Baseline“ sehr nahe.

Fähigkeiten, die Anforderungsmanagementsoftware in diesem Bereich zu Verfügung stellen kann, sind: Speichern und Verwalten von Baseline Dokumenten, Versionsverwaltung, Lese- und Schreibschutz und Darstellung von Unterschieden zwischen verschiedenen Baseline Versionen.

Änderungsmanagement (Requirements change management)

Verwalten von Versionsinformationen, wie Versionsnummern, Datum, Zeit der Erstellung und Änderung. Flexible Suchfunktion für alle Gegenstände, die von einer Änderung betroffen sind. Spezifische Vorgehensweise, um Anforderung im Baseline hinzuzufügen, zu ändern oder zu aktualisieren und den dazugehörigen Änderungs- / Anforderungsstatus mitzuverwalten.

Projektmanagement (Project Management)

Bringt die Anforderungsmanagementsoftware Funktionen mit, um den Status von Anforderungen über die Projektlaufzeit zu verfolgen. Bezogen auf Ressourcen, (Zeit-) Pläne und Einbringung. Aufgaben dieser Funktionen sind: das Aufzeichnen, Verfolgen und Berichten über den allgemeinen Status des Anforderungsmanagementprozess. Unterstützt werden soll möglichst auch exterene Projektmanagementsoftware.

Offenes oder geschlossenes Datenmodell (Open or closed data model)

Hat das Anfoderungsmanagementsystem ein offengelegtes (standardisiertes) Datenformat, erlaubt es externen Anwendungen den Zugriff ohne Verwendung eines kompletten Protokolles.

4.7 Nachvollziehbarkeit, Verfolgbarkeit (Traceability)

Die in diese Gruppe fallenden Funktionen legen ihren Fokus auf das Dokumentieren vom Leben einer Anforderung. Also das zu Verfügung stellen von Verknüfungsfunktionen zwischen bestimmten Anforderungen und das Tracking von Veränderungen.

Verfolgbarkeit (Traceability)

Anforderungsmanagementsoftware kann Verfolgbarkeit/Nachvollziehbarkeit mit folgenden Fähigkeiten unterstützen: Speichern und Verwalten von Rollen und Verantwortlichkeiten der Stakeholder. Durch Darstellen von „Traces“ in textueller und grafischer Form, sowie die Reduktion auf Baselineanforderungen. Das Verknüpfen von Dokumenten und das Verwalten dieser Verknüpfungen. Durch das Erstellen von Berichten, die aufzeigen, welche Dokumente sich verändert haben und die deren Änderungen darstellen.

Flexibles Verfolgen (Flexible tracing)

Hier geht es darum herauszufinden wie flexibel die Software ist. In Bezug darauf, ob sie vorwärts und rückwärts Verfolgung, eines zu vielen, vieles zu einem und viele zu vielen Traces ermöglicht. Als zweitens geht es hier darum, wie flexibl Inhalte sein können. Also Texte, Grafiken, Tabellen, Tabellenzellen etc.

Bidirektionales Verfolgen (Bi-directional tracing)

Das System soll helfen festzustellen, das alle Quellanforderungen berücksichtigt sind und alle detailierten/low-level Anforderungen einer Quellanforderung zugewiesen sind.

Verfolgbarkeitsanalyse (Traceabiltiy analysis)

Hier soll das System eine Einsicht in den Projektstatus liefern, indem es über verwaiste und fehlende Anforderungen berichtet. Ein solcher Bericht kann auch Änderungen von Anforderungen enthalten und beispielweise in Matrizenform gestaltet sein.

4.8 Andere Features (Other capabilities)

Diese Kategorie betrifft Funktionen, aus den Bereichen Administrative Informationen (RE tool administrative information), Grafische Benutzeroberfläche (Graphical user interface GUI) und Systemintegration (Data integration). Also Benutzerverwaltung, Weboberfläche oder native GUI's.

4.9 Semantische Funktionen

Der Fragebogen, der dieser Aufzählung zugrunde liegt, und jener empfohlen von [Rup09] stellen sehr ähliche Fragen, unterteilen diese jedoch in unterschiedliche Kategorien. Wobei in Beiden keine Fragen explizit nach semantischen Technologien gestellt werden.

Nachdem schon darauf eingangen wurde, wie Anforderungen für Maschinen (Algorithmen) „verstehbar“ formuliert werden können, noch zu den generellen Anwendungen von semantischen Technologien, wie sie folgendermaßen von [MWHB11] identifiziert werden:

- (semi-) automatisches Finden von Mappings
- explizite formale Repräsentation dieser Mappings
- Schließen unter Verwendung dieser Mappings

Diese Anwendungen sind aufgrund der Mächtigkeit der zugrundeliegenden Theorie entsprechend groß, im Kontext von Funktionen für Anforderungsmanagementsoftware können folgende drei Katergorien beschrieben werden.

Kategorisierung

Durch das (semi-) automatische Finden von Strukturen können semantische Technologien helfen, Anfoderungen zu kategoriesieren, bzw. (semi-)automatisch zu attributisieren. Beispielweise kann das Verwenden von der WordNet Ontologie[4] und der darin enthaltenen Hyponomie- und Synonymbeziehungen helfen, verschiedene Formulierungen des selben Sachverhalts zu berücksichtigen. Deutsches Pendant zu WordNet ist GermaNet[5]. Vorteil davon wäre eine Zeitersparnis im Gegensatz zu einem rein manuellen Vorgehen.

Tracing

Formale Repräsentationen können vielseitig genutzt werden. Auf die in Anfoderungsmanagementsoftware wichtige Funktion des Tracings bezogen, beispielweise durch Modellieren von Relationen wie „bestehtAus“, „istVorgängerVon“ und deren inversen Relationen „istTeilVon“, „istNachfolgerVon“. Vorteil hiervon könnte das einfachere (und schnellere) Erstellen von Modellen sein.

Konfliktanalyse

Unentdeckte Konflikte bzw. Inkonsistenzen in Anforderungsspezifikationen können, wie bereits im ersten Kapitel ausgeführt, zu erheblichen Mehrkosten Parentsführen. In weiterer Folge konzentriert sich diese Arbeit auf das Finden von Inkonsistenzen bzw. Hindeuten auf mögliche Inkonsistenzen durch logisches Schließen in Anfoderungswissensbasen.

[4] `http://wordnet.princeton.edu/`
[5] `http://www.sfs.uni-tuebingen.de/GermaNet/`

4.10 Anwendungsbeispiele

Die generellen Anwendungsgebiete finden sich auch auch in konkreten Umsetzungen wieder. Zusätzlich zu dem Einführungsbeispiel von auf Seite 25 ff. sollen werden noch zwei weitere Beispiele vorgestellt.

Beispiel SWORE2

SWORE ist ein Teil des Softwiki-Projekts des Instituts für Informatik der Universität Leipzig. Softwiki basiert auf OntoWiki und zielt darauf ab, die Zusammenarbeit bei der Anforderungserstellung in Softwareprojekten von großen und räumlich getrennten Stakeholdergruppen auf einfache Weise zu ermöglichen.

Hauptbestandteil neben der Wikisoftware ist die Ontologie SWORE, die bereits in der Version 2 existiert und im Folgenden etwas genauer besprochen werden soll.

Nachdem die Beziehungen zwischen Anforderungen in dieser Arbeit bereits im Zusammenhang mit prädikatenlogischem Modellen ausführlich besprochen wurden, wird hier der Fokus auf eine weitere große Stärke des „Semantic Web“-Ansatzes gelegt. Nämlich der Vernetzbarkeit mit anderen Wissenbasen bzw. deren Ontologien.

Die Abbildung 4.1 zeigt die SWORE2 Ontologie als Graph. Etwa in der Mitte der Grafik befindet sich die zentrale Klasse `req:Requirement`. Im oberen Teil befinden sich die Anforderungenbeziehungen.

Interessant ist, wie sich die Ontologie mit anderen integrieren lässt. Die eingefärbten Felder stehen jeweils für Elemente aus einer anderen Ontologie. Beispielhaft sind drei herausgenommen:

- **FOAF** steht für "Friend of a friend" und wird verwendet, um Personen und Relationen zwischen Personen zu beschreiben.
- **SIOC** steht für "Semantically-Interlinked Online Communities" und wird verwendet um, verschiedene Kommunikationskanäle miteinander zu verknüpfen.
- **DOAP** steht für "Description of a Project" und wird verwendet, um projektrelevante Informationen auszuzeichnen.

Der Vorteil von einem derartigen Ansatz für Anforderungsmanagementsoftware ist neben einem möglichen Reasoning über Anforderungsbeziehungen, einer der Hauptpunkte bei der Evaluie-

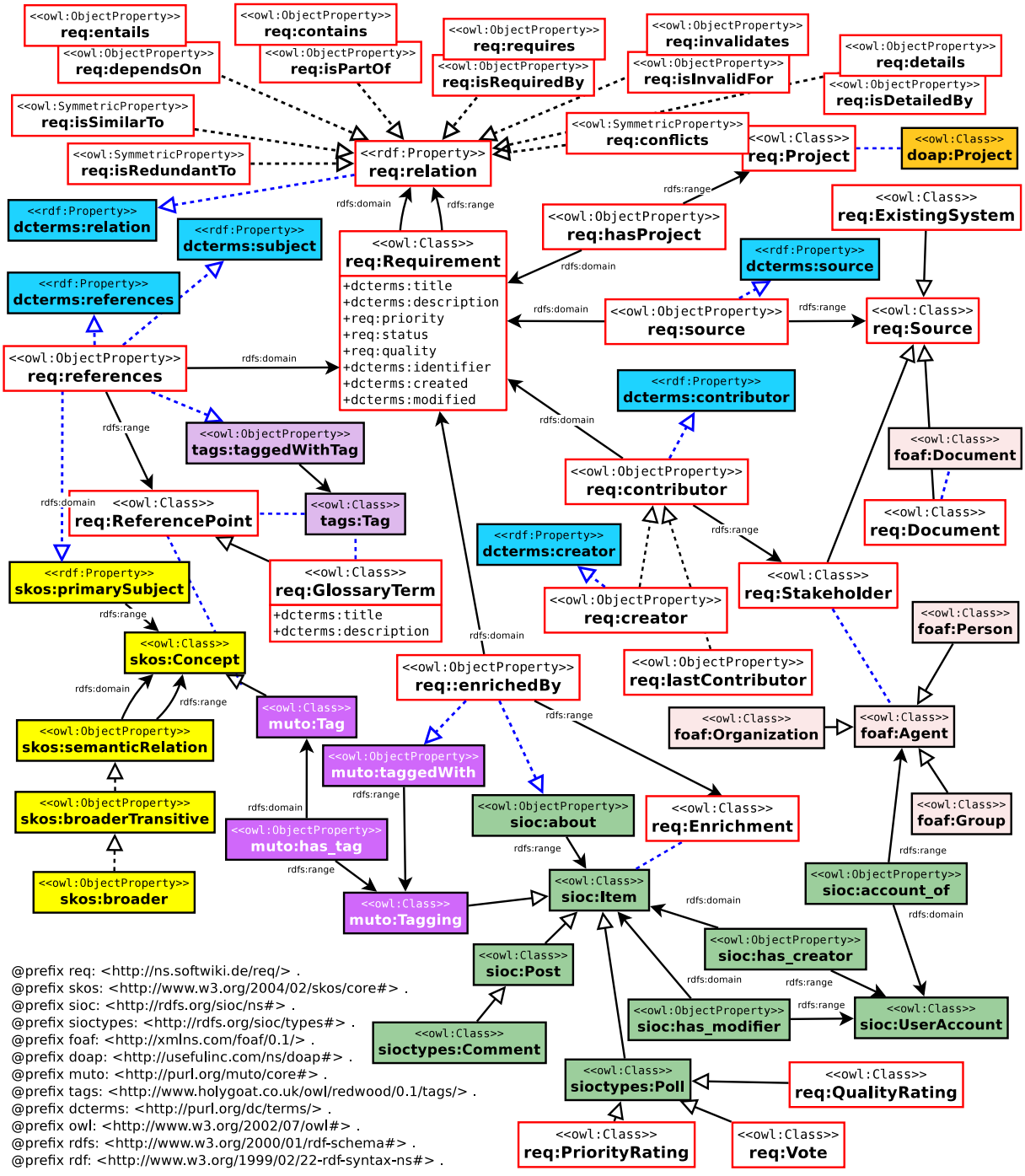

Abbildung 4.1: Ontologie SWORE 2 vom Softwiki Projekt [RLL+12]

rung: Schnittstellen zu anderer Software. Dieser Vorteil lässt sich durch die vorher beschrieben offenen Standards erklären.

Beispiel OntRep

Eine Möglichkeit, semantische Technologie auf Anforderungen anzuwenden, wurde hier noch nicht detaillierter ausgeführt. Dabei handelt es sich um die Frage, wie unter Zuhilfenahme der Fachdomäne auf hilfreiche Fakten geschlossen werden kann. Zum Beispiel, welche Anforderungen Probleme verursachen können. Dazu wird hier kurz auf einen Ansatz eingegangen, der in [MWHB11] bzw. [Wag10] beschrieben ist.

In diesem Ansatz wird folgendermaßen vorgegangen: Anforderungen, Businessconstraints, Glossar und Dokumentationsrichtlinien werden in einer festgelegten, in Kapitel 2 erwähnten Syntax geschrieben. Die Elemente dieser natürlichsprachlichen Aussagen werden dann möglichst automatisch in eine Ontologie übersetzt. Dies kann manuel überprüft, korrigiert und mit zusätzlichem Wissen angereichert werden.

Das Glossar, welches ebenfalls definiert werden muss, enthält systemrelevante fachspezifische Begriffe. Wie zum Beispiel die Benutzerrollen `TrustedPersons` und die Klasse `SecuredResource`.

Der wesentliche Teil folgt in Form von regelbasierend definierten Klassen, welche Bedingungen beschreiben, die von keinen Anforderungen eingehalten werden dürfen. In einer konsistenten Anforderungsbasis wären diese Klassen instanzenlos.

Eine Regel wäre zum Beispiel: keine Anforderung darf festhalten, dass eine unberechtige Person/Benutzerrolle auf geschützte Ressourcen zugreifen soll. Dazu wird die Klasse `ProcessingOfSecuredResourcesByUntrustedPersons` erstellt und mit folgender Regel befüllt.

```
SecuredResource(?s) ^ (not TrustedPersons)(?p) ^ processedBy(?s, ?p)
-> processingOfSecuredResourcesByUntrustedPersons(?p, ?s)
```

Lässt man den Reasoner den Inhalt der Klasse berechnen, bekommt man einen Hinweis auf inkonsistente oder falsche Anforderungen. In diesem Ansatz wurde das Regelwerk mittels Protegé erstellt und hat die From von Klassenrestriktionen, wobei dies für die grundsätzliche Vorgehensweise und Funktion keine Rolle spielt.

Die Autoren untersuchen desweiteren, inwieweit ein solches semantisch unterstütztes Vorgehen Vorteile gegenüber manueller Anforderungsanalyse hat. Ihre Tests kommen zum Ergebnis, dass ihr Vorgehen vielversprechend ist.

KAPITEL 5

Semantische Features für QFD basierte Anforderungsmanagementsoftware

Wie die vorhergehenden Beispiele, sowie die Einführung in modellbasiertes Reasoning gezeigt haben, existieren bereits Ansätze und Umsetzungen für die semantische Analyse von Tracing-Relationen (Gnokil), Ansätze die ihren Schwerpunkt auf Zusammenarbeit legen (SWORE) und welche, die Spezifikationen unter Zuhilfenahme der Fachdomäne kategorisieren und mit der Fachdomäne auf Inkonsistenzen bzw. Fehler schließen (OntRep).

Die Idee Inkonsistenzen in Spezifikationen mittels Reasoning zu finden soll nun unter Verwendung der Methode QFD auf Kundenanforderungen angewandt werden.

5.1 Semantischen House of Quality

Die grundsätzliche Vorgehensweise für die semantische Unterstützung soll aus [MWHB11] übernommen werden:

- Syntax $\rightarrow$ Ontologie
- Ontologie $\rightarrow$ Reasoner
- Reasoner $\rightarrow$ Feedback

Um diesem Plan zu folgen legen wir eine Syntax fest, entwickeln eine Ontologie, finden ein Übersetzung zwischen Syntax und Ontologie und beschreiben Regeln innerhalb der Ontologie,

die es uns in weiterer Folge erlauben Feedback zu geben. Dieses Feedback soll auf Inkonsistenzen bzw. mögliche Inkonsistenzen aufmerksam zu machen.

Syntax

Betrachtet man die Syntax für Anforderungen nach [Rup09] beziehungsweise die leicht angepasste Version in [Wag10], so fällt es schwer, in einer Menge von formulierten Anforderungen und Businessconstraints den Kundenwunsch herauszulesen.

Beispiele aus [Wag10, S. 90]:

```
23. The system shall offer a config information page.
15. The system shall use server side caching.
8.  The administrator shall have access to administrative panel.
```

`23.` Kein Aussage zu was, wer oder wozu konfiguriert werden soll.

`15.` Nicht klar ist, ob es darum geht dem Kunden schneller Daten auszuliefern, oder ob dies aus technischen, systemspezifischen Gründen erfolgen muss.

`8.` Bei diesem Businessconstraint ist ebenfalls schwer zu sagen, in welcher Form damit für den Kunden Wert generiert werden soll.

Gleiches gilt für die vorgestellte formale Anforderungsformalisierung ($R = (S, P)$). Weil in [GKBV11] ebenfalls auf die selbe Form zur natürlichsprachlichen Repräsentation von Anforderungen zurückgegriffen wird ("The system shall ...").

D.h., es werden geforderte Systemeigenschaften festgehalten und nicht explizit der Kundenwunsch, wie in QFD unter „WAS“ gedacht. Natürlich kann aus den beschriebenen Systemeigenschaften der Kundenwunsch heraus gelesen werden, dies erfordert jedoch eine Interpretation, welche sich zwischen beteiligeten Personen (Stakeholdern und Teammitgliedern) unterscheiden kann.

Der Grund warum beides, kundennahe und systennahe Anforderungen, festgehalten werden sollen, findet sich in den anfangs besprochenen Aufgaben des Anforderungsmanagements. Es soll helfen, unter allen Projektbeteiligten eine möglichst gleiche Wahrnehmung des Problems *und* der angestrebten Lösung zu kommunizieren.

- Konzentriert man sich nur sehr lösungs-/systemnahen Anforderungen
  ist der Systemzweck / Kundenwunsch interpretation.

- Werden umgekehrt Anforderungen nur sehr kundennah festgehalten, existieren verschiedene Vorstellungen von den Lösungen.

In QFD wird beides festgehalten und zusätzlich noch der Zusammenhang zwischen den Anforderungen auf den verschiedenen Abstraktionsgraden mit Hilfe der Unterstützungsmatrix erarbeitet.

Kategorisierungen, wie die zwischen Kundenwünschen und technischen Charakteristiken, sind nichts Neues im Gebiet des Anforderungsmanagements, weder im Softwarebereich noch in „klassischen“ Engineering. Im Kontext von Software vergleicht [BB10] high-level mit detaillierten Anforderungen folgendermaßen:

- High-level oder business Anforderungen sind relativ lesbar und geeignet für den Kunden. Sie werden in der Organisation von Marketingabteilung erstellt und enthalten Informationen basierend auf Kundengesprächen und Marktforschung[1].
- Detailierte Anforderungen bestehen aus vollständigen Einzelheiten, die darauf abzielen dem Entwickler präzise mitzuteilen wie das zu entwickelnde System auszusehen hat.

[BB10] sagt, um die Herausforderung zu bewältigen, Anforderungen über die Zeit (Projektzeitraum) konsistent zu halten, hilft es den Abstraktionsgrad von high-level und detaillierten Anforderungen groß zu halten. Er unterscheidet bei deren Formulierung nicht zwischen high-level und detailliert, aber eine derartige Unterscheidung erscheint sinnvoll.

Betrachtet man die bisher beschriebene Syntax (Schablonen) zum Festhalten von Anforderungen, so wird klar, dass diese besser dafür geeignet sind um das „WIE“ zu beschreiben, weniger das „WAS“.

Deswegen wird hier noch eine geeignetere Formulierung für Kundenwünsche vorgestellt.

Kundenwunsch (KW) Eine gut geeignete Formulierung kommt aus der agilen Softwareentwicklung. In Scrum und XP werden Kundenwünsche als User Stories formuliert.

User Stories[2] scheinen mit ihrem kurzen einfachen Aufbau gut geeignet zu sein, Kundenwünsche im House of Quality festzuhalten.

[1] solche Methoden finden sich auch in Abbildung 2.4 Seite 10

[2] `http://de.wikipedia.org/wiki/User_Story`

Aus [Wir11]:
"Als <Rolle> will ich <das Ziel> [, so dass <Grund für Ziel>]"

Aus Wikipedia:
"Als <Rolle> möchte ich <Ziel/Wunsch> [, um <Nutzen>]"

Bzw. Englisch:
"As a <Role>, I want <goal/desire> [, so that <benefit>]"

Wobei User Stories im Rahmen von agilen Softwareentwicklungsweisen zur Spezifikation von Anforderungen eingesetzt werden und demnach auch andere organisatorische und praktische Eigenschaften aufweisen sollten. So ist in den Regeln zu XP[3] hervorgehoben, dass User Stories in der Releaseplanung für Zeitabschätzungen verwendet werden und eine bedeutende Rolle in der Erstellung von Akzeptanztests einnehmen. An der Diskussion um User Stories[4] ist zu erkennen, dass der Begriff "User Story" in der Softwareindustrie nicht eindeutig verwendet wird, jedoch werden Eigenschaften vorgeschlagen, die User Stories haben sollten und die nicht zwangsläufig für den Einsatz in QFD benötigt werden. Hier die von [Wir11] [5] :

- Independend (untereinander unabhängig)
- Negotiable (mit dem Kunden verhandelbar)
- Valuable (bewertbar)
- Estimateable (abschätzbarer Aufwand)
- Small (klein)
- Testable (testbar)

So ist ein abschätzbarer Aufwand in QFD noch nicht zwangsweise nötig, da sich QFD eher für schwergewichtige Entwicklungsmodelle eignet, in denen oft andere Projektmanagementmethoden[6] eingesetzt werden. Auch ist die Unabhängigkeit für QFD nicht unbedingt erforderlich, da es ebenfalls Möglichkeiten anbietet, Zusammenhänge zwischen Kundenanforderungen zu verdeutlichen. Eine einfache und oft angewandte Möglichkeit ist eine Kategorisierung.

Folgt man inhaltlich [Wir11, S. 51] und unterdrückt zusätzlich jene Eigenschaften, welche aus praktischen Überlegungen in agilen Vorgehenmodellen verlangt werden, so liegt man wieder

[3] http://www.extremeprogramming.org/rules/userstories.html
[4] http://c2.com/cgi/wiki?UserStory
[5] Englische Ausdrücke, weil die Anfangsbuchstaben das einfach zu merkende Wort INVEST bilden.
[6] z.B. Target Costing

sehr nahe an einer Untermenge der bereits mehrfach erwähnten Qualitätskriterien für Anforderungen.

Auch die grundsätzliche Eigenschaft von User Stories, nämlich dass sie vage gehalten werden sollen, passt zur Methode QFD. Wirdemann argumentiert dass in einem agilen Vorgehensmodell wie Scrum (in dessem Kontext er User Stories behandelt) die Unvollständigkeit von Stories Teil des Konzeptes ist. Der Grund kann im Scrum-Manifest gefunden werden, in dem unter anderem festgehalten ist, dass

- Induviduen und Interaktionen über Prozesse und Werkzeuge und
- Zusammenarbeit mit Kunden über Verhandlungen von Verträgen

zu stellen sind, was nicht bedeut, dass Prozesse, Werkzeuge und Verhandlungen unwichtig wären, sondern nur weniger wichtig sind. Für Scrum bedeutet dies, dass die Konkretisierung der User Story erst bei ihrer Umsetzung im Sprint und in Zusammenarbeit mit dem Kunden bzw. dem Product Owner erfolgen soll.

Einen sehr ähnlich Prinzip folgt QFD, in QFD steht auch das gemeinsame Erarbeiten des HoQ im Vordergrund und damit die Kommunikation über Kundenanforderungen und deren Übersetzung in ein technisches System bzw. Spezifikation.

Als Konsequenz werden User Stories bzw. der Teil `<Ziel/Wunsch>` von Userstories im Kundenwuschbereich des HoQs verwendet.

Produktstruktur (PS), Qualitätsmerkmale (QM) In weiterer Folge wird ein dreistufiges QFD zur Erläuterung dieses Ansatzes verwendet. Dazu wird die abstrakte Kategorie technische Charakteristiken weiter in Produktstruktur (PS) und Qualitätsmerkmale (QM) unterteilt.

- Die **Produktstruktur**, beschrieben durch Produktkomponenten, hängt von dem zu entwickelnden System ab, den Vorlieben des Teams und natürlich von bereits existierenden Komponenten, die in einem zu entwickelnden Produkt Verwendung finden könnten.

  Da es sich bei Anwendung von QFD verstärkt um die Zusammenhänge zwischen verschiedenen Komponenten untereinander und deren Relation zu den Kundenwünschen geht, werden in weiterer Folge die komponenteninternen Anforderungen vernachlässigt.

  Für die Formulierung von Anforderungen bedeutet dies, dass sich die Schablone auf folgende Aussagen reduziert:

```
<Systemkomponente> (MUSS | KANN) <Wem?> DIE
FÄHIGKEIT <Fähigkeit> ZUR VERFÜGUNG STELLEN.
```

`<Wem?>` ist entweder ebenfalls eine Systemkomponente oder eine Kundenrolle.

Des weiteren soll beschrieben werden können, dass eine Fähigkeit von einer Systemkomponente genutzt wird.

```
<Wem?> (MUSS | KANN) DIE FÄHIGKEIT <Fähigkeit>
VON  <Systemkomponente> NUTZEN.
```

- Die **Qualitätsmerkmale** beschreiben Produktausprägungen und werden ohne spezielle Syntax durch Zielwerte beschrieben. Solche Zielwerte können Wahrheitswerte (Eigenschaft ist vorhanden) oder Zahlenwerte (z.B. Leistungswerte) sein.

Ontologie, Mappings

Dieser Abschnitt beginnt eine Ontologie zu deklarieren und dazu Mappings, wie die Aussagen in vorher beschriebener Syntax in diese Ontologie übersetzt werden kann.

Es gelten folgende Präfixe:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix : <http://example.org/qfd#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://example.org/qfd> .

<http://example.org/qfd> rdf:type owl:Ontology .
```

Um die Funktionalitäten hervorzuheben, welche sich auf das Finden von Inkonsistenzen konzentrieren, wird darauf verzichtet auf Elemente einzugehen, die hauptsächlich Versionierung und Management von Anforderungen betreffen. Dementsprechend finden sich keine `erstelltVon`, `hatStatus`, etc. Eigenschaften in dieser Ontologie. Für die Modellierung solcher Informationen wird auf das SWORE Projekt, vorgestellt in Abschnitt 4.10, verwiesen.

Kundenwunsch (KW) Einen Kundenwunsch, ausgedrückt in der Form „`Als <Rolle> möchte ich <Ziel/Wunsch> [, um <Nutzen>].`“, soll in folgende Ontologie übersetzt werden.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

:Wunsch rdf:type owl:Class .
:Rolle rdf:type owl:Class .

:wünscht rdf:type     owl:ObjectProperty ;
            rdfs:domain  :Rolle ;
            rdfs:range   :Wunsch .

:gewünschtVon owl:inverseOf :wünscht .

:Nutzen  rdf:type  owl:Class .

:um   rdf:type    owl:ObjectProperty ;
      rdfs:domain :Wunsch ;
      rdfs:range  :Nutzen .

# Verknüpfung mit anderen Ontologien FOAF
:nimmtEin rdf:type    owl:ObjectProperty ;
          rdfs:domain foaf:Person ;
          rdfs:range  :Rolle .

:eingenommenDurch owl:inverseOf :nimmtEin .
```

Das Mapping selbst ist denkbar einfach:

- <Rolle> in `:Rollen`,
- <Ziel/Wunsch> in `:Wünsche` und
- <Nutzen> in `:Nutzen`.

Bei zusätzlicher Information, welcher Kunde welche Rolle einnimmt kann diese(r) als "Friend of a Friend" Person modelliert werden.

Beispiel: Kunde1 wünscht sich: `Als Kundentyp1 möchte ich "schnelle Auftragsabwicklung"`. Welcher als Wunsch1 in folgenden Turtle Zeilen festgehalten kann.

```
:Wunsch1  rdf:type  :Wunsch ;
          rdf:value "schnelle Auftragsabwicklung"@de .

:Kundentyp1 rdf:type    qfd:Rolle ;
            :wünscht :Wunsch1 .

:Kunde1  rdf:type  foaf:Person ;
         :nimmtEin :Kundentyp1 .
```

Für die Abbildung der Rollen und Personen könnten auch andere Ontologien verwendet werden bzw. mit solchen verknüpft werden. PROTON (Proto Ontology), Top Module[7] hat zum Beispiel die Eigenschaft `roleHolder` oder der W3C Working Draft "An organization ontology"[8] hat `role`. Beispielsweise den `:Kunde1` mit einer Entität aus einer anderen Ontologie gleichstellen.

```
:Kunde1  owl:sameAs  <http://example.org/company1#department1> .
```

Qualitätsmerkmale (QM) Davon ausgehend, dass Qualitätsmerkmale durch einzelne Werte beziehungsweise durch Wertebereiche beschrieben werden können, wie zum Beispiel: Preis, Gewicht, Abmessungen, wurde für Qualitätsmerkmale keine eigene Syntax festgehalten. Eine sicherlich für den praktischen Einsatz relevante Fähigkeit zu Modellierung von Enumerationen kann unterschiedlich ermöglicht werden. Einerseites durch Modellierung in eine mit `owl:oneOf`, andererseits in XML-Schema (`xsd:enumeration`).

Die Klasse Merkmal hat vier Unterklassen, um verschiedene Merkmalsarten ausdrücken zu können. Des weiteren wird eine Eigenschaft eingeführt bei vorhandenem Zielwert bzw. Zielbereich diese abbilden kann.

Unterklasse `ZielwertMerkmal` wird dadurch limitiert, dass sie genau eine Angabe des Datenproperties erlaubt. Sowie `ZielbereichMerkmal` genau zwei solcher Angaben erlaubt.

Des weiteren kann ein Merkmal als zu maximierend bzw. zu minimierend klassifiziert werden. Die Angabe, dass die beiden Klassen miteinander disjunkt sind sorgt dafür, dass falls ein Merkmal beiden Unterklassen zugeordnet wird, ein Reasoner diese Inkonsistenzen aufzeigen würde.

```
:Merkmal rdf:type owl:Class .
```

[7] `http://proton.semanticweb.org/2005/04/protont`
[8] `http://www.w3.org/TR/vocab-org/#org:role`

```
:hatZielwert rdf:type owl:DatatypeProperty ;
             rdfs:range xsd:decimal .

:ZielwertMerkmal rdfs:subClassOf :Merkmal ,
                         [ rdf:type owl:Restriction ;
                           owl:onProperty :hatZielwert ;
                           owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                           owl:onDataRange xsd:decimal
                         ] .

:ZielbereichMerkmal rdfs:subClassOf :Merkmal ,
                         [ rdf:type owl:Restriction ;
                           owl:onProperty :hatZielwert ;
                           owl:qualifiedCardinality "2"^^xsd:nonNegativeInteger ;
                           owl:onDataRange xsd:decimal
                         ] .

:ZuMaximierendesMerkmal rdf:type owl:Class ;
                        rdfs:subClassOf :Merkmal ;
                        owl:disjointWith :ZuMinimierendesMerkmal .

:ZuMinimierendesMerkmal rdf:type owl:Class ;
                        rdfs:subClassOf :Merkmal .
```

Des weiteren sollen noch die Zusammenhänge von Merkmalen und damit ihre gegenseitige Beeinflussung ausgedrückt werden können. Für diesen Zweck werden die symmetrische Eigenschaft `:beeinflusst` und vier davon hierarchisch untergeordneten Eigenschaften eingeführt, welche die in HoQs üblicherweise verwendeten Beeinflussungssymbole abbilden.

Da diese nicht gleichzeitig auf dieselben Instanzen angewendet werden können sollen, wird noch folgendes allgemeines Axiom hinzugefügt, welches die Disjunktheit sicherstellt.

```
:beeinflusst rdf:type owl:ObjectProperty ,
                      owl:SymmetricProperty ;
             rdfs:range :Merkmal ;
             rdfs:domain :Merkmal .

:beeinflusstNegativ rdf:type owl:ObjectProperty ;
                    rdfs:subPropertyOf :beeinflusst .
```

```
:beeinflusstPositiv rdf:type owl:ObjectProperty ;
                    rdfs:subPropertyOf :beeinflusst .

:beeinflusstStarkNegativ rdf:type owl:ObjectProperty ;
                         rdfs:subPropertyOf :beeinflusst .

:beeinflusstStarkPositiv rdf:type owl:ObjectProperty ;
                         rdfs:subPropertyOf :beeinflusst .

[ rdf:type owl:AllDisjointProperties ;
  owl:members ( :beeinflusstNegativ
                :beeinflusstPositiv
                :beeinflusstStarkNegativ
                :beeinflusstStarkPositiv
              )
] .
```

Einfaches Beispiel mit zwei Merkmalen, welche sich stark positiv beeinflussen.

```
:Beschleuningung rdf:type :ZielbereichMerkmal ,
                          owl:NamedIndividual ;
                 :hatZielwert 54 ,
                              64 .

:Leistung rdf:type :ZielwertMerkmal ,
                   owl:NamedIndividual ;
          :hatZielwert 42.4 .

:Leistung :beeinflusstStarkPositiv :Beschleuningung.
```

Produktstruktur (PS) Vereinfachend können in dieser Ontologie Produktkomponenten ihre Fähigkeiten anderen Komponenten zur Verfügung stellen. Umgekehrt können Komponenten die Fähigkeiten Anderer auch nutzen. Für komplexeres Fälle, welche eine bessere Abbildung der Fachdomäne erfordern, wird auf den in [MWHB11] bzw. [Wag10] vorgestellten Ansatz verwiesen.

Für die Illustration von Schnittstellen zwischen den Komponenten wird die bereits vorgestellte Syntax verwendet.

```
<Systemkomponente> (MUSS | KANN) <Wem?> DIE
```

```
FÄHIGKEIT <Fähigkeit> ZUR VERFÜGUNG STELLEN.

<Wem?> (MUSS | KANN) DIE FÄHIGKEIT <Fähigkeit>
VON  <Systemkomponente> NUTZEN.
```

Die mit folgenden Teilen der Ontologie beschrieben werden kann:

```
:Komponente rdf:type owl:Class .
:Fähigkeit rdf:type owl:Class .

:bietet rdf:type owl:ObjectProperty ;
        rdfs:range :Fähigkeit ;
        rdfs:domain :Komponente .

:gebotenVon rdfs:inverseOf :bietet .

:kannBieten rdfs:subPropertyOf :bietet .
:mussBieten rdfs:subPropertyOf :bietet .

:bietetNicht rdf:type

:nutzt rdf:type owl:ObjectProperty ;
       rdfs:range :Fähigkeit ;
       rdfs:domain :Komponente .

:kannNutzen rdfs:subPropertyOf :nutzt .
:mussNutzen rdfs:subPropertyOf :nutzt .
```

Beispiel: Zwei Komponenten, von der eine die Fähigkeit der anderen nutzt.

- `Cache MUSS Controller DIE FÄHIGKEIT "schnell abfragbare Speicherung von Key-Value Paaren" ZUR VERFÜGUNG STELLEN.`
- `Controller MUSS DIE FÄHIGKEIT "schnell abfragbare Speicherung von Key-Value Paaren" VON Cache NUTZEN.`

Übersetzt in die Ontologie ergibt folgendes:

```
:Cache rdf:type :Komponente .
:Controller rdf:type :Komponente .
```

```
:Fähigkeit1 rdf:type :Fähigkeit ;
            rdf:value "schnell abfragbare Speicherung von
                       Key-Value Paaren"@de .

:Cache :mussBieten :Fähigkeit1 .
:Controller :mussNutzen :Fähigkeit1 .
```

Unterstützungsmatrix Es fehlt noch der im House of Quality wichtigste Teil: die Unterstützungsmatrix als Kernelement.

Eine Komponente bzw. ein Merkmal kann die Erfüllung eines Wunsches unterstützen. Im vorgestellten Fall existiert auch eine Unterstützungsmatrix in der eine Komponente ein Merkmal unterstützten kann.

Die Unterstützung kann in vier, voneinander disjunkten Abstufungen ausgedrückt werden: Nicht, Schwach, Mittel und Stark.

```
:unterstützt rdf:type owl:ObjectProperty ;
             rdfs:domain :Komponente ,
                         :Merkmal ;
             rdfs:range :Merkmal ,
                        :Wunsch .

:unterstütztVon rdfs:inverseOf :unterstützt .

:unterstütztNicht rdf:type owl:ObjectProperty ;
                  rdfs:subPropertyOf :unterstützt .

:unterstütztSchwach rdf:type owl:ObjectProperty ;
                    rdfs:subPropertyOf :unterstützt .

:unterstütztMittel rdf:type owl:ObjectProperty ;
                   rdfs:subPropertyOf :unterstützt .

:unterstütztStark rdf:type owl:ObjectProperty ;
                  rdfs:subPropertyOf :unterstützt .

[ rdf:type owl:AllDisjointProperties ;
  owl:members ( :unterstütztMittel
                :unterstütztNicht
```

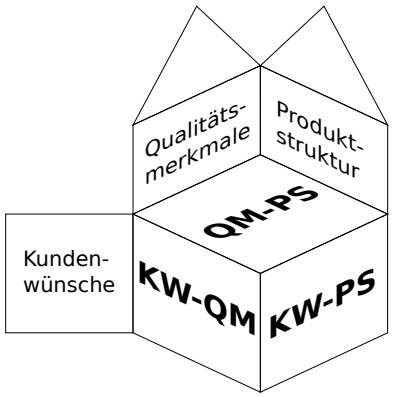

Abbildung 5.1: Die drei HoQs in dreidimensionaler Darstellung

```
                :unterstütztSchwach
                :unterstütztStark
            )
] .
```

5.2 Semantisches QFD

Beim Vorgehen nach QFD werden mehrere HoQ Matrizen verwendet, die üblicherweise wasserfallartig kaskadiert werden[9]. Um ein großes Potential an Inkonsistenzen aufzeigen zu können, werden für diese Arbeit folgende drei HoQs verwendet.

- Kundenwünsche (KW) - Qualitätsmerkmale (QM)
- Kundenwünsche (KW) - Produktstruktur (PS)
- Qualitätsmerkmale (QM) - Produktstruktur (PS)

Anstatt einer Wasserfalldarstellung wird eine dreidimensionale Gesamtrepräsentation gewählt, dies um möglichen HoQ übergreifende Inkonsistenzen besser darstellen zu können. Diese Darstellung ist in Abbildung 5.1 skizziert.

[9] Übersicht möglicher Matrizen für QFD `http://de.wikipedia.org/w/index.php?title=Datei:QFD_King_Matrix_der_Matrizen.PNG&filetimestamp=20040924213712`

Die Idee hinter dieser Auswahl ist folgende: Kundenwünsche sollen die echten Stimmen des Kunden zum Ausdruck bringen, auch wenn die Wünsche für unrealistisch oder widersprüchlich gehalten werden. Eine Liste von Kundenwünschen kann zum Beispiel aus einer Befragungsserie entstehen.

Die Qualitätsmerkmale hingegen sollen die wesentlichen Kriterien für die Kaufentscheidung abbilden, beziehungsweise die dem Kaufentscheidungsprozess zugrunde liegende Opportunität mitberücksichtigen. Beispielsweise jene Kriterien, die in einschlägigen Fachzeitschriften behandelt werden. Eine geeignete Methoden zu Priorisieren solcher Merkmale, neben der des HoQs selbst, ist die diskrete Conjoint Analyse.

Die Gegenüberstellung dieser Kriterien **KW–QM** in der Unterstützungsmatrix soll nun helfen, ein genaueres Bild vom Kunden und seinem Kaufentscheidungsprozess zu bekommen. Des weiteren kann ein Vergleich der aus der Unterstützungsmatrix berechneten Bedeutungswerte mit jenen aus einer Conjoint Analyse darauf hindeuten, wann das Bild, das in diesem HoQ vom Kunden beschrieben wird, ausreichend gut ist. Kriterium könnte sein, dass die Diskrepanz zwischen den einzelnen Werten ausreichend klein ist oder sich durch zusätzliche Erklärung begründen lassen kann.

Die Produktstruktur soll nun den Zusammenhang zwischen Kundenwünschen und dem zu erstellenden Produkt bzw. Dienstleistung herstellen.

Die Gegenüberstellung von Produktkomponenten und Kundenwünschen soll aufzeigen, welche Teile des System die Erfüllung des Kundenwunsches unterstützen. Nachdem auch die Qualitätsmerkmale ebenfalls mit den Produktkomponenten in einem HoQ gegenübergestellt werden, können die Bedeutungswerte, die durch Verketten von **KW–QM** und **QM–PS** mit denen von **KW–PS** entstehen, verglichen werden. Auftauchende Diskrepanzen deuten wiederum Inkonsistenzen im vorherrschenden Bild vom Kunden oder in der Vorstellung davon, wie das Produkt die Kundenwünsche erfüllt. Klärung solcher möglichen Missverständnisse ist die Hauptaufgabe des Anforderungsmanagements.

Ort im HoQ in denen diese Differenzen angezeigt werden können, ist unter den Punkten **?8** und **?9** in Abbildung 5.2 auf Seite 67 eingezeichnet.

Diese Zusammenstellung von HoQs hat grundsätzlich noch nichts mit semantischen Technologien zu tun. Deswegen soll im nächsten Abschnitt aufgezeigt werden, wie solche Technologie mit der eingeführten Ontologie auf den drei HoQs (im folgenden als "Tower of Quality" bezeichnet) unterstützend in Anforderungsmanagementsoftware angewandt werden kann. Da in der Praxis

üblicherweise umfangreiche HoQs bearbeitet werden und dadurch leicht Zusammenhänge übersehen werden können.

"Tower of Quality"

Bei der Erarbeitung der HoQs sollen folgende, als SWRL Regeln ausgedrückte, Fragen das QFD-Team unterstützten ihre Angaben möglichst konsistent und lückenlos zu tätigen. Dabei soll das Zeigen von „versteckten“ Beziehungen helfen eine bessere Produktstruktur zu finden. Die in Klammern angegeben und mit Fragezeichen beginnende Nummerierung markiert die entsprechende Darstellung der Regel im "Tower of Quality", abgebildet in 5.2.

Die ersten zwei Regeln beziehen sich ausschließlich auf Korrelationen innerhalb einer Dimension.

- **QM–QM (?1)**
  Die Frage, die folgende Regel stellt ist die: Wenn zwei Merkmale über ein drittes sich stark beeinflussen, warum beeinflussen sie sich nicht direkt gegenseitig?

  ```
  Merkmal(?qm1), Merkmal(?qm2), Merkmal(?qm3), beinflusstStark(?qm1, ?qm3),
  beinflusstStark(?qm2, ?qm3), beeinflusstNicht(?qm1, ?qm2)
  ->
  möglicheBeinflussung(?qm1, ?qm2)
  ```

- **PS–PS (?2)**
  Eine Komponente die auf nicht vorhandene Fähigkeiten zugreifen muss, soll markiert werden.

  ```
  Komponente(?pk1), Komponente(?pk2), Fähigkeit(?f), mussNutzen(?pk1, ?f),
  (gebotenVon = 0)(?pk2), bietet(?pk2, ?f)
  ->
  KomponenteNutztNichtVorhandeFähigkeit(?pk1)
  ```

Die nächsten Fragen greifen bereits auf Informationen aus der Korrelationsmatrix und der Unterstützungsmatrix in einem HoQ zurück.

- **KW–QM (?3)**
  Mit folgender Regel kann das System die Frage stellen: Wenn ein Merkmal den Kundenwunsch stark unterstützt und dieses Merkmal stark durch ein weiteres Merkmal beeinflusst wird, warum unterstützt letzteres Merkmal nicht ebenfalls den Kundenwunsch?

```
Wunsch(?kw), Merkmal(?qm1), Merkmal(?qm2), unterstütztStark(?qm1, ?kw),
beinflusstStark(?qm1, ?qm2), unterstütztNicht(?qm2, ?kw)
->
möglicheUnterstützung(?kw, ?qm2)
```

- **KW–PS (?4)**
  Die Software kann den Benutzern fragen, „Wenn eine Produktkomponente den Kundenwunsch unterstützt, diese von den Fähigkeit einer anderen abhängig ist, warum unterstützt diese andere nicht auch den Kundenwunsch?“.

```
Wunsch(?kw), Komponente(?pk1), Komponente(?pk2), Fähigkeit(?f),
unterstütztStark(?pk1, ?kw), mussNutzen(?pk1, ?f), bietet(?pk2, ?f),
untersütztNicht(?pk2, ?kw)
->
möglicheUnterstützung(?pk2, ?kw)
```

- **QM–PS**
  Die vorhergehende Regel kann auch auf das HoQ zwischen Qualitätsmerkmalen und Produktstruktur angewandt werden.

```
Merkmal(?qm), Komponente(?pk1), Komponente(?pk2), Fähigkeit(?f),
unterstütztStark(?pk1, ?qm), mussNutzen(?pk1, ?f), bietet(?pk2, ?f),
untersütztNicht(?pk2, ?qm)
->
möglicheUnterstützung(?pk2, ?qm)
```

- **QM–PS (?5)**
  Auch kann die Regel **?3** auf dieses HoQ umgesetzt werden.

```
Merkmal(?qm1), Merkmal(?qm2), Komponente(?pk1), unterstütztStark(?pk1, ?qm1),
beinflusstStark(?qm2, ?qm1), unterstützNicht(?pk1, ?qm2)
->
möglicheUnterstützung(?pk1, ?qm2)
```

Für die nachfolgenden Regeln kommen, HoQ übergreifend, Informationen aus mehreren Unterstützungsmatrizen zur Anwendung.

- **KW (?7)**
  Nachdem sich Kundenwünsche auch auf nicht produktbezogene Eigenschaften richten können, soll diese Regel auf Wünsche aufmerksam machen, die nicht durch Produktmerkmale oder Produktkomponenten unterstützt werden. Ein Beispiel wäre, dass sich ein Kunde wünscht, dass ein Sportgerät von einem erfolgreichen Profisportler verwendet wird.

```
Wunsch(?kw), (unterstütztVon = 0)(?kw)
->
UnunterstützterKundenWunsch(?kw)
```

- **KW–QM–PS (?6)**
  Wenn ein Kundenwunsch stark durch ein Merkmal unterstützt wird und dieses Merkmal selbst wiederum stark durch eine Produktkomponente unterstützt wird, warum unterstützt diese Komponente nicht direkt den Kundenwunsch?

  ```
  Wunsch(?kw), Merkmal(?qm), Komponente(?pk), unterstütztStark(?qm, ?kw),
  unterstütztStark(?pk, ?qm), unterstütztNicht(?pk, ?kw)
  ->
  möglicheUnterstützung(?pk, ?kw)
  ```

"Tower of Quality"

Ein vollständiger "Tower of Quality" wird in Abbildung 5.2 dargestellt. In dieser Abbildung ist zusätzlich zu den vorher besprochenen Fällen, noch die Möglichkeit weitere Informationen zu Integrieren dargestellt. Rechts neben den Mitbewerbervergleich wurde ein Feld für Akzeptanztests eingeführt, das anzeigen soll, wann das zu erstellende Produkt den Kundenwunsch erfüllt bzw. welche Mitbewerber diesen Test erfüllen. Letzteres dient in Kombination mit dem Mitbewerbervergleich dazu, dass Akzeptanztests auf ihre Validität überprüft werden können.

Am Boden wurde noch in Anlehnung an [BW10, S. 124] ein viertes House of Quality angefügt, das den Zusammenhang zwischen Prozessen zur Erzeugung des Produktes und den Komponenten beschreibt und auf die grundsätzliche Erweiterbarkeit eines solchen Ansatzes aufzeigen soll.

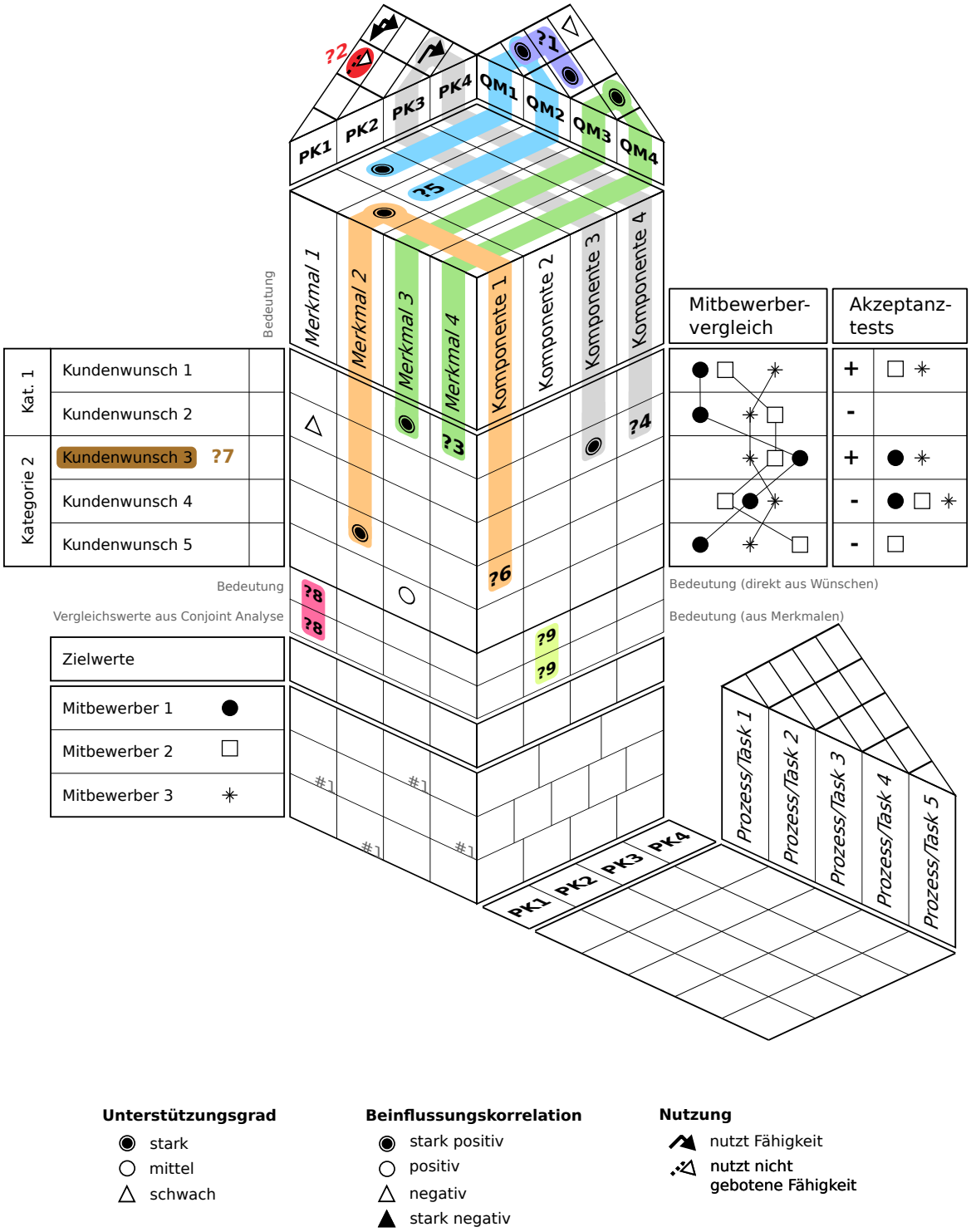

Abbildung 5.2: "Tower of Quality" mit eingezeichneten möglichen Inkonsistenzen

KAPITEL 6

Ergebnis

Regelwerk

Das vorgestellte Regelwerk hat seinen Nutzen vor allem bei Änderungen in HoQs, um in größeren Tabellen auf mögliche Inkonsistenzen hinzudeuten, die womöglich mit freiem Auge schwer zu sehen sind. Hingegen ist der Nutzen bei dem erstmaligen Ausfüllen der Unterstützungsmatrizen und Korrelationsdächer fragwürdig, da auf nicht unterstützte, nicht unterstützende Wünsche, Qualitätsmerkmale und Produktkomponenten aufmerksam gemacht werden.

Zwei Wege zur Komponentenwichtigkeit

Die ausgewählten HoQs, speziell deren Anordnung, erlauben es Komponentenbedeutung aus Kundensicht auf zwei unterschiedlichen Wegen zu berechnen. Dies einerseits direkt aus den gewichteten Kundenwünschen mit der Unterstützungsmatrix für Komponenten und andererseits mittels Umweg über die Qualitätsmerkmale. D.h. es werden zuerst die Bedeutungswerte für die Qualitätsmerkmale berechnet, welche dann wiederum mit der Unterstützungsmatrix zwischen Qualitätsmerkmalen und Komponenten in Komponentenbedeutungen umgewandelt werden können.

Diskrepanzen der Bedeutungswerte für eine Komponente deuten darauf hin, dass das vorherrschende Bild über den Kunden inkonsistent ist. D.h. die Auswahl der Kundenwünsche, der Qualitätsmerkmale oder ihre gegenseitigen Unterstützung beziehungsweise ihre Unterstützung durch das geplante Produkt noch Mängel aufweist. Dieses ungeplante Resultat hat potentiell die größere Bedeutung als das auf "sematic web" basierende Regelwerk, da es für wasserfallartige, schwergewichtige Vorgehensmodelle den Zeitpunkt markieren kann, in dem von der An-

forderungsanalysephase in die Systemimplementierungsphase übergegangen werden kann. Des weiteren hat die Betrachtung der Bedeutungswerte aus Managementsicht den Vorteil, dass auch ohne den gesamten Tower of Quality verstehen zu müssen auf das Kundenverständnis des Entwicklungsteam geschlossen werden kann.

Zusätzliche Informationen

Der Ansatz unterstützt auch die Einbringung von zusätzlichen Informationen. Zum Beispiel können, um das im Entwicklungsteam vorherrschende Bild des Kunden weiter zu hinterfragen, Informationen aus Quellen außerhalb der QFD Methodik, integriert werden. Zum Vergleich von Bedeutungswerten eignen sich zum Beispiel Werte aus einer Conjointanalyse. Damit entsteht ein Effekt für Qualitätsmerkmalbedeutungen, ähnlich wie bei dem vorhergehenden beschriebenen Vergleich von Komponentenbedeutungen.

Eine andere Art von Informationen, die in die vorgestellte Struktur eingebunden werden können, sind Akzeptanztests. Dem Prinzip von QFD folgend kann das Einbringen und Diskutieren von Akzeptanztests, angewandt auf Mitbewerber und das eigene Produktkonzept, helfen, Unstimmigkeiten zu den Vorstellungen vom Kunden bei den einzelnen Projektmitgliedern aufzudecken und zu beheben.

Semantic Web Technologie

Angemerkt sei noch: Damit ein Reasoner Regeln mit den Kardinalitäten $=$ und $<$ auswerten kann, muss die OWL Ontologie, die der "Open World Assumption" unterliegt, „geschlossen“ werden. Dies erfolgt dadurch, dass alle Individuen aufgezählt werden. Ohne dieses „Schließen“ kann ein OWL Reasoner nicht davon ausgehen, dass es keine weiteren Elemente gibt um die exakt und minimum Kardinalität Aussagen zu $Wahr$ auszuwerten.

Auch gilt in OWL keine "Unique Name Assumption". Es ist daher notwendig, für die vorgestellte Anwendung alle Individuen als unterschiedlich zu kennzeichnen. Dies kann einerseits durch eine `owl:AllDifferent` und `owl:distinctMembers` Deklaration erfolgen, oder dadurch, dass allen unterschiedlichen Individuen in der Ontologie mit dem funktionalen Property `hasKey` ein eindeutiger Schlüssel zugewiesen wird.

KAPITEL 7

Zusammenfassung und Ausblick

Im zweiten Kapitel wurden die wichtigen Punkte aus dem Anforderungsmanagement zusammengefasst, wobei drei Punkte betont wurden:

- Zusammenhang mit Vorgehensmodellen.
- Es ist schwer bis unmöglich, Anforderungen mit ausreichender Qualität zu formulieren ohne den Lösungsraum einzuschränken.
- Anforderungsmanagement hat Schnittstellenfunktion und viele der verwendeten Prozesse, Techniken und Werkzeuge finden sich in anderen Fachbereichen wieder.

In Zukunft wird es voraussichtlich interessant zu beobachten, ob sich die Verwendung der Einteilung funktional, nicht-funktional reduziert. Spannend kann auch werden, ob sich mit der immer stärkeren Verwendung von Software in mechanischen Produkten und bei ihrer Erstellung auch einen Wandel in den Vorgehensweisen in Richtung agilen Vorgehensmodelle entwickelt (hard scrum) bzw. eine Mischung ergibt (agil solange bis zu Konzeptphase und dann weiter mit vordefinierten Phasen).

Im dritten Kapitel wurde anhand von zwei semantischen Ansätzen im Anforderungsmanagement in die Grundlagen zu Semantischen Technologien eingeführt: Die Basis liegt in der Logik und die Standards kommen vom World Wide Web Consortium.

Im vierten Kapitel wurden Evaluierungsmethoden für Anforderungsmanagementsoftware vorgestellt und auf eine konkrete Studie aus dem Jahr 2012 eingegangen. Dabei wurden detailliert

auf Funktionen, die von solchen Systemen zu Verfügung gestellt werden können, sollen und müssen, eingegangen. Dabei wurde festgestellt, dass keine der bekannteren Evaluierungsfragebögen explizit nach semantischen Technologien fragt. Anschließend wurden weitere die Beispiele SWORE und OntRep betrachtet bereits existierende semantische Features zu beleuchten.

Im fünften Kapitel liegt der eigentlich Beitrag dieser Diplomarbeit. Hier wurde eine OWL Ontologie mit natürlichsprachlichem Syntaxmapping und Regelwerk vorgestellt, welche Teile der Methode "Quality Function Deployment" abbilden.

Ziel ist es durch ein semantisch angereichertes QFD, den Wert von Unterlagen, die mittels QFD entstanden sind, zu erhöhen. Respektive während der Erarbeitung unterstützende Funktionen anzubieten. Aufgedeckt wurden potentielle Funktionen, aber auch Nachteile, die die Verwendung von Semantischen Web Technologien für die beschriebenen Funktionen bringen. Nochmals hervorgehoben: ein praktischen Problem mit der OWL zugrunde liegenden "Open World Assumption" und der dadurch benötigtes „Schließen“ der Ontologie.

Literaturverzeichnis

[Aue09] J. Auer. Methodik für ein Requirements Engineering von Schienenfahrzeugen. Master's thesis, Technische Universität Wien, 2009.

[Baa07] F. Baader. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, 2007.

[BB10] E.J. Braude and M.E. Bernstein. *Software Engineering: Modern Approaches*. Wiley, 2010.

[Bra97] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119 (Best Current Practice), March 1997.

[BW10] F.J. Brunner and K.W. Wagner. *Qualitätsmanagement: Leitfaden für Studium und Praxis*. Praxisreihe Qualitätswissen. Hanser Fachbuchverlag, 2010.

[CMM10] Product Team CMMI. Cmmi for development version 1.3. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2010.

[Dav93] A.M. Davis. *Software requirements: objects, functions, and states*. PTR Prentice Hall, 1993.

[dGNA+12] Juan M. Carrillo de Gea, Joaquín Nicolás, José L. Fernández Alemán, Ambrosio Toval, Christof Ebert, and Aurora Vizcaíno. Requirements engineering tools: Capabilities, survey and assessment. *Information and Software Technology*, 54(10):1142 – 1157, 2012.

[Ebe10] C. Ebert. *Systematisches Requirements-Engineering: Anforderungen ermitteln, spezifizieren, analysieren und verwalten*. dpunkt-Verlag, 2010.

[Ebe11] C. Ebert. Riskante nachlässigkeit. *Qualität und Zuverlässigkeit (QZ)*, 56:20–23, 2011.

[ESP95] ESPITI. User survey report. Technical report, European Software Process Improvement Training Initative, 1995.

[FG12] Jörg Friedrich and Cornelia Gaebert. Inning verbunden - requirements engineering: Plädoyer für eine wende. *iX – Magazin für professionelle Informationstechnik*, page 98ff, August 2012.

[GKBV11] Arda Goknil, Ivan Kurtev, Klaas Berg, and Jan-Willem Veldhuis. Semantics of trace relations in requirements models for consistency checking and inferencing. *Software & Systems Modeling*, 10:31–54, 2011.

[Goe11] Clement Goebel. Agile methods influencing non-it projects, 2011. Accessed: 2012-19-10.

[Gro95] Standish Group. The scope of software development project failures. Technical report, Standish Group, 1995.

[HKRS08] P. Hitzler, M. Krötzsch, S. Rudolph, and Y. Sure. *Semantic Web: Grundlagen*. Springer London, Limited, 2008.

[HS06] G. Hofbauer and A. Schweidler. *Professionelles Produktmanagement: Der Prozessorientierte Ansatz, Rahmenbedingungen Und Strategien*. Publicis, 2006.

[HW05] C. Hood and R. Wiebel. *Optimieren von Requirements Management& Engineering:*. Xpert. press Series. Springer, 2005.

[ISO09] Information technology – Systems and software engineering – Guide for requirements engineering tool capabilities, 2009.

[MWHB11] Thomas Moser, Dietmar Winkler, Matthias Heindl, and Stefan Biffl. Requirements management with semantic technology: An empirical study on automated requirements categorization and conflict analysis. In Haralambos Mouratidis and Colette Rolland, editors, *Advanced Information Systems Engineering*, volume 6741 of *Lecture Notes in Computer Science*, pages 3–17. Springer Berlin Heidelberg, 2011.

[RLL+12] Thomas Riechert, Steffen Lohmann, Jens Lehmann, Kim Lauenroth, Philipp Heim, and Sebastian Tramp. Semantic Web Ontology for Requirements Engineering (SWORE). online, January 2012.

[RN07] S. Russell and P. Norvig. *Künstliche Intelligenz: Ein moderner Ansatz*. I - Informatik. Pearson Studium, 2. auflage. edition, 2007.

[Rup09] Chris Rupp. *Requirements-Engineering und -Management*. Hanser Fachbuchverlag, 5., Überarbeitete auflage. edition, 7 2009.

[SAKR05] B. Schäppi, M.M. Andreasen, M. Kirchgeorg, and F.J. Radermacher. *Handbuch Produktentwicklung*. Hanser, 2005.

[Sch02] B. Schienmann. *Kontinuierliches Anforderungsmanagement: Prozesse-Techniken- Werkzeuge*. Programmer's Choice. Addison-Wesley, 2002.

[Wag10] A. Wagner. Using semantic technology to support project reporting. Master's thesis, Technische Universität Wien, 2010.

[Wal01] E. Wallmüller. *Software-Qualitätsmanagement in der Praxis*. Hanser, 2001.

[Wir11] R. Wirdemann. *Scrum mit User Stories*. Hanser Fachbuchverlag, 2011.