Verwaltung und Prüfung natürlichsprachlicher Spezifikationen

Von der Fakultät Informatik der Universität Stuttgart
zur Erlangung der Würde eines Doktors der
Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

vorgelegt von

Ralf Melchisedech

aus Aachen

Hauptberichter: Prof. Dr. J. Ludewig
Mitberichter: Prof. Dr. M. Glinz

Tag der mündlichen Prüfung: 15.06.2000

Institut für Informatik der Universität Stuttgart

2000

Dank

Vielen ist es zu verdanken, daß diese Arbeit zu einem erfolgreichen Abschluß gekommen ist. Zuerst ist hier natürlich mein Doktorvater Professor Ludewig zu erwähnen, der mir in vielen Diskussionen neue Impulse und Denkanstöße für meine Arbeit und darüberhinaus gegeben hat und der durch seinen unkomplizierten Umgang mit seinen Mitarbeitern stets für eine sehr angenehme Arbeitsatmosphäre gesorgt hat.

Sehr gefreut hat mich, daß Professor Glinz den Mitbericht für diese Arbeit übernommen hat.

Meinen Kollegen Patricia Mandl-Striegnitz, Stefan Krauß, Ralf Reißing, Anke Drappa, Angela Georgescu, Adolf Veith-Willier, Max Schneider, Marcus Deininger, Jürgen Schwille, Horst Lichter, Helga Hoff, Jinhua Li und Ursula Mühlbayer möchte ich für das angenehme Arbeitsklima danken. Besonders hervorheben möchte ich hierbei Patricia und Stefan und ihnen für die vielen Diskussionen, konstruktive Kritik und guten Ratschläge, insbesondere in der Endphase der Arbeit, danken.

Vielen Dank auch an Nathalie Boissinot und Jochen Peschel für aufmerksames und gründliches sprachliches Korrekturlesen der Arbeit.

Danken möchte ich auch all den Studenten, die mich in meiner Arbeit unterstützt haben. Da wären zum einen Uwe Mindrup und Martin Richter, die in ihren Diplomarbeiten an der Grundkonzeption mitgearbeitet und eine erste Version des ADMIRE-Werkzeugs entwickelt haben, und zum anderen Markus Knauß, Stefan Opferkuch und Markus Badstöber, die als »Hiwis« für viele notwendige und »schöne« Erweiterungen zuständig waren.

Zuletzt möchte ich auch noch meinen Eltern danken, ohne deren langjährige Unterstützung diese Arbeit niemals entstanden wäre.

Inhaltsverzeichnis

Zusammenfassung 11

Kapitel 1: Einleitung 13

1.1 Motivation und Einführung 13
1.2 Ein Beispiel 14
1.3 Ergebnisse 15
1.4 Aufbau dieser Arbeit 17

Kapitel 2: Grundlagen des Requirements Engineering 19

2.1 Pflichtenheft 20
2.1.1 Zentrale Rolle des Pflichtenhefts 21
2.1.2 Abgrenzung zwischen Pflichtenheft und Entwurf 21
2.1.3 Eigenschaften eines guten Pflichtenhefts 22
2.1.4 Normen für Pflichtenhefte 24
2.2 Rollen 25
2.3 Prozeß 28
2.3.1 Tätigkeiten 29
2.3.2 Erheben von Informationen 31
2.3.3 Änderungsmanagement 34
2.3.4 Validierung und Verifikation (V&V) 35
2.3.5 Spezifikationsnotationen und -methoden 38
2.4 Bedeutung informaler Notationen 41

Kapitel 3: Stand der Praxis – eine Untersuchung 49

3.1 Untersuchungsgegenstand 49
3.1.1 Projektkenndaten 49
3.1.2 Systemkenndaten 50
3.2 Untersuchungsergebnisse - Dokumente 51
3.2.1 Aufbau der Dokumente 51
3.2.2 Klassifikation der Informationen 53
3.2.3 Verwendete Notationen 55
3.3 Untersuchungsergebnisse - Rollen 57
3.4 Untersuchungsergebnisse - Prozesse 58
3.4.1 Werkzeugeinsatz 58
3.4.2 Kommunikation mit den Kunden 58
3.4.3 Prüfung der Requirements-Engineering-Dokumente 58
3.5 Konsequenzen 59
3.5.1 Kommunikation mit den Auftraggebern und Benutzern 59
3.5.2 Änderungsmanagement 60
3.5.3 Fehlende Informationen 60
3.5.4 Informale oder formale Notationen? 60
3.6 Schlußfolgerung 61

Kapitel 4: Überblick über den ADMIRE-Ansatz 63

4.1 Einführung 63
4.2 Informationsmodell 65
4.3 Prozeßmodell 68
4.4 Prüfverfahren 69
4.5 Werkzeug 70
4.6 Anwendungsbeispiel: SESAM 71

Kapitel 5: Informationsmodell 73

5.1 Dokumente 73
5.2 Anforderungsdokumente 74
5.3 Inhaltselemente 77
5.3.1 Attribute von Inhaltselementen 79
5.3.2 Kategorisierung von Inhaltselementen 82
5.4 Annotationen 87
5.5 Systemmodelle 88
5.6 Glossar 91
5.7 Wortliste 94
5.8 Quellenkatalog 95
5.9 Mängelkatalog 97
5.10 Qualitätskriterien 99
5.11 Sinnvolle Vorbelegungen 105

Kapitel 6: Formales Modell 109

6.1 Formales Modell für informale Spezifikationen 109
6.1.1 Grundlegende Mengen 109
6.1.2 UML (Unified Modeling Language) 111
6.1.3 Prädikatenlogik 111
6.1.4 Prädikate und Funktionen auf Wörtern, Phrasen und Freitexten 113
6.2 Einträge 114
6.3 Quellenkatalog 116
6.4 Glossar 117
6.5 Systemmodelle 118
6.6 Inhaltselemente 119
6.7 Annotationen 121
6.8 Dokumente und Anforderungsdokumente 121
6.9 Mängelkatalog 123
6.10 Metaspezifikation 124
6.11 Informationsmodell 127
6.12 Hilfsmengen und -funktionen 128
6.13 Qualitätskriterien 130

Kapitel 7: Prozeßmodell 133

7.1 Phasen 133
7.1.1 Problemanalysephase 134
7.1.2 Spezifikationsphase 136
7.1.3 Spätere Projektphasen 137
7.1.4 Lebenszyklen der Anforderungsdokumente 138
7.2 Tätigkeiten 140
7.2.1 Bereite Dokumentation vor 141
7.2.2 Erhebe Informationen 141
7.2.3 Entwickle Lösungsideen 142
7.2.4 Integriere Anforderungssammlungen 143
7.2.5 Verifiziere Dokument 144
7.2.6 Validiere Dokument 145
7.2.7 Führe Entscheidungsverfahren durch 146
7.2.8 Überarbeite Anforderungsdokument 147
7.2.9 Erstelle Pflichtenheft 147
7.2.10 Bearbeite Änderungsmeldung 148
7.3 Validierungs- und Verifikationstabellen 149
7.3.1 Anforderungsdokument und Glossar 149
7.3.2 Quellenkatalog 154

Kapitel 8: Validierung und Verifikation 155

8.1 Prüfverfahren 155
8.1.1 Inhaltsprüfung 155
8.1.2 Fokussierte Inspektion 158
8.1.3 Statusprüfung 161
8.1.4 Inspektion 162
8.1.5 Ähnlichkeitssuche 163
8.2 Prüfung der Qualitätskriterien 167
8.2.1 Korrektheit 167
8.2.2 Vollständigkeit 170
8.2.3 Adäquatheit 175
8.2.4 Konsistenz 176
8.2.5 Externe Konsistenz 180
8.2.6 Redundanzfreiheit 180
8.2.7 Keine Überspezifikation 180
8.2.8 Realisierbarkeit 181
8.2.9 Überprüfbarkeit 182
8.2.10 Atomizität 182
8.2.11 Verständlichkeit 184
8.2.12 Eindeutigkeit 185
8.2.13 Präzision 188
8.2.14 Minimalität 189
8.2.15 Normkonformität 189

Kapitel 9: ADMIRE - Werkzeug 191

9.1 Überblick 191
9.1.1 Arbeitsumgebung 191
9.1.2 Wichtige Funktionen 192
9.1.3 Benutzungsschnittstelle 193
9.2 Realisierungsaspekte 196
9.3 Typische Benutzungsszenarien 197
9.3.1 Szenario 1: »Eingabe und Prüfung einer Anforderung« 197
9.3.2 Szenario 2: »Prüfung des Glossars auf Konsistenz« 202
9.3.3 Szenario 3: »Prüfung eines Anforderungsdokuments auf Vollständigkeit« 204
9.3.4 Szenario 4: »Prüfung eines Anforderungsdokuments auf Konsistenz« 206
9.3.5 Szenario 5: »Entscheidung über ein wichtiges Konzept« 209

Kapitel 10: Vergleich und Bewertung 213

10.1 Verwandte Ansätze 213
10.1.1 Werkzeuge 213
10.1.2 NLP-Ansätze 216
10.2 Ergänzende Ansätze 222
10.2.1 Szenarien/Use Cases 222
10.2.2 Notationszentrierte Werkzeuge 223
10.2.3 Ansätze zur Erstellung von Glossaren 223
10.2.4 Textverständlichkeit 225
10.2.5 Freitext-Analysen 227
10.3 Bewertung 228
10.3.1 Zusammenfassung: Stärken und Schwächen der Arbeit 228
10.3.2 Ausblick 232

Glossar 235

Index 249

Literatur 255

Zusammenfassung

Im Software Engineering wird gefordert, daß in einem Software-Projekt die Probleme und Anforderungen geklärt werden und eine Spezifikation des Systems erstellt wird, bevor mit der Realisierung begonnen wird.

Spezifikationsdokumente können in mehr oder weniger formalen Notationen geschrieben werden. In der industriellen Praxis wird vorwiegend natürliche Sprache eingesetzt. Mit ihrer Verwendung sind aber einige Probleme verbunden. Die Aussagen sind oft unpräzise, mehrdeutig oder widersprüchlich. In einigen Ansätzen wird deswegen vorgeschlagen, semi-formale oder formale Notationen einzusetzen. Eine in dieser Arbeit durchgeführte Untersuchung hat aber gezeigt, daß ein Umstieg in vielen Projektumgebungen nicht möglich ist.

In dieser Arbeit wird der ADMIRE-Ansatz vorgestellt, ein Ansatz für das Requirements Engineering, in dem die hauptsächliche Verwendung natürlicher Sprache als eine zentrale Rahmenbedingung betrachtet wird. Ziel des ADMIRE-Ansatzes ist es, den Requirements-Engineering-Prozeß zu verbessern, insbesondere die Erstellung, Verwaltung und Prüfung der Spezifikationsdokumente.

Die wesentlichen Ergebnisse dieser Arbeit sind ein Informationsmodell, ein Prozeßmodell, eine Menge von Prüfverfahren und ein Werkzeug.

In dem Informationsmodell werden alle für die natürlichsprachliche Spezifikation relevanten Informationstypen und deren Zusammenhänge beschrieben. Da sich Mängel in einem Requirements-Engineering-Prozeß oft nicht vermeiden oder sofort beheben lassen, läßt das Informationsmodell zu, daß bestimmte wünschenswerte Eigenschaften verletzt und Mängel dokumentiert werden.

Das Prozeßmodell beschreibt eine exemplarische, sinnvolle und auf das Informationsmodell abgestimmte Aufteilung des Requirements-Engineering-Prozesses in Phasen und Tätigkeiten.

Für jede »verletzbare« Eigenschaft des Informationsmodells werden visuelle und automatische Prüfverfahren zur Validierung und Verifikation eingeführt. Durch die Prüfverfahren können sowohl strukturelle Bedingungen als auch die Inhalte der natürlichsprachlichen Freitexte überprüft werden.

Das Werkzeug kann Instanzen des Informationsmodells verwalten, unterstützt die im Prozeßmodell beschriebenen Tätigkeiten und realisiert die Prüfverfahren.

Zur Validierung des Ansatzes wurde eine umfangreiche Spezifikation erstellt.

Abstract

Most software engineers agree that in a software project problems should be identified, requirements should be elicited, and the target system should be specified before implementation begins.

Requirements documents may be written in more or less formal notations. The main notation used in industrial software projects is natural language. This likely causes problems because requirements expressed in natural language are often imprecise, ambiguous, or inconsistent. That is why some approaches propose the use of semi-formal or formal notations. However, an investigation carried out by the author shows that a change to more formal notations is not possible in many typical project environments.

In this work the ADMIRE approach is presented, a requirements engineering approach which is primarily based on natural language. The aim of ADMIRE is to improve the requirements engineering process, especially the writing, management, validation, and verification of requirements documents.

The main results of this work are an information model, a process model, a set of validation and verification procedures, and a tool.

The information model describes information types and relations relevant to natural language specifications. Because defects cannot always be avoided or solved immediately in requirements processes, the information model allows that some desirable properties are currently not met and offers the possibility to document these defects.

The process model describes, suited to the information model, an exemplary dividing of the requirements process into phases and activities.

For each desirable property of the information model, a set of visual and automatic procedures for validation and verification is introduced. These procedures verify both structural constraints and the contents of the natural language texts.

The tool manages instances of the information model, supports the activities of the process model, and implements the validation and verification procedures.

The ADMIRE approach has been validated by writing an extensive specification document.

Kapitel 1

Einleitung

1.1 Motivation und Einführung

Software kann auf verschiedene Weisen entwickelt werden. Eine Möglichkeit besteht darin, »einfach darauf los« zu programmieren, also ohne vorhergehende Planung. Dieses Verfahren, auch *Code and Fix* genannt, wird in der Praxis sehr häufig angewandt, auch wenn es völlig ungeeignet ist. Ein zentrales Problem hierbei ist, daß mit der Realisierung des Software-Systems begonnen wird, bevor die Anforderungen geklärt sind. Die Gefahr, daß das entwickelte System nicht den Vorstellungen und Anforderungen der Kunden entspricht, ist beträchtlich.

Im Software Engineering wird deswegen gefordert, daß in einem Software-Projekt zuerst die Probleme und Anforderungen geklärt werden und eine Spezifikation des Systems erstellt wird, bevor mit der Realisierung begonnen wird. Die Forschungsdisziplin, die sich mit der Erhebung von Anforderungen und der Spezifikation von Systemen beschäftigt, wird *Requirements Engineering* genannt.

Der Begriff Software Engineering wurde 1968 von F.L. Bauer (Naur, Randell, 1969) geprägt. Die frühesten Arbeiten über das Requirements Engineering wurden Mitte der 70er Jahre veröffentlicht (z.B. Boehm, 1976). Seit Ende der 80er Jahre steigt sowohl im universitären als auch im industriellen Umfeld das Bewußtsein, daß Requirements Engineering für ein Software-Projekt nützlich und notwendig ist.

Spezifikationsdokumente können in mehr oder weniger formalen Notationen geschrieben werden. In der industriellen Praxis werden vorwiegend informale Notationen eingesetzt, insbesondere natürliche Sprache.

Mit der Verwendung natürlicher Sprache sind einige Probleme verbunden. Die Aussagen können unpräzise und mehrdeutig sein. Anforderungen werden oft weder klar als solche gekennzeichnet noch von anderen Anforderungen oder sonstigen Informationen getrennt. Solche Anforderungen können nur schwer als Vorlage für die Realisierung verwendet werden. Ebensowenig eignen sie sich als Vorlage zur Prüfung, ob das System seine Anforderungen erfüllt. Spezifikationsdokumente können sehr umfangreich werden; mehrere hundert Seiten sind keine Seltenheit. Inhaltlich zusammengehörende Informationen sind oft über weite Teile des Textes verstreut. Widersprüche können so nur schwer gefunden werden.

In vielen universitären Ansätzen wird vorgeschlagen, diese Probleme durch Einsatz semi-formaler oder formaler Notationen zu lösen oder zu vermindern. In den Kapiteln 2.4 und 3.5.4 und in Ludewig (1993, S. 288) wird jedoch gezeigt, daß es gute Gründe für die Verwendung natürlicher Sprache im Requirements Enginee-

ring gibt und der Umstieg auf formalere Notationen in vielen Projektumgebungen nicht möglich ist.

Aus diesen Gründen wird die hauptsächliche Verwendung natürlicher Sprache als eine zentrale Rahmenbedingung für den in dieser Arbeit vorgestellten ADMIRE[1]-Ansatz betrachtet. Der Einsatz natürlicher Sprache im Requirements Engineering wird in der Forschung relativ wenig berücksichtigt. Dieses Defizit soll durch *ADMIRE* vermindert werden.

1.2 Ein Beispiel

In Kapitel 1.1 wurde festgestellt, daß mit der Verwendung natürlicher Sprache eine Reihe von Problemen verbunden ist. An einem Auszug aus einem Originaltext sollen die Probleme verdeutlicht und einige Lösungsansätze aufgezeigt werden:

> »Die Reaktionszeiten an den Benutzeroberflächen liegen innerhalb der Interaktionszeit eines Benutzers (<1 sec). Bei längeren Verarbeitungszeiten während eines Benutzerdialogs wird die Wartezeit zur Anzeige gebracht (zum Beispiel durch Sanduhr). Besonderer Wert wird darauf gelegt, die Übersetzungs- und Generierungszeiten möglichst kurz zu halten, um eine optimale ›turn around‹-Zeit für den Anwender zu erreichen.« (Auszug aus dem Pflichtenheft von Projekt A, siehe Kapitel 3)

Mit diesem Absatz sind u.a. folgende Probleme verbunden:

- Dieser Absatz enthält mehrere Anforderungen und eine Begründung. Es ist sehr umständlich, von anderen Stellen des Spezifikationsdokuments oder von anderen Dokumenten aus auf die Einzel-Informationen zu verweisen. Wie kann z.B. ausgedrückt werden, daß die Anforderung bzgl. der Anzeige der Wartezeit (Satz 2) schon realisiert wurde, die Anforderung bzgl. der Reaktionszeit (Satz 1) aber noch nicht, ohne die Anforderungen ganz oder teilweise zu wiederholen oder die Sätze in dem Absatz zu zählen?
- Daß es sich bei den ersten beiden Sätzen um Anforderungen handelt, kann, streng genommen, nur vermutet werden. Die Sätze selbst geben hierfür keinen Anhaltspunkt. Satz 2 z.B. könnte auch so interpretiert werden, daß es einen (vorhandenen) Mechanismus gibt, der die Anzeige der Wartezeit bewerkstelligt.
- Streng genommen widersprechen sich die Anforderungen in Satz 1 und 2. In der ersten Anforderung wird gefordert, daß die Reaktionenzeiten kleiner eine Sekunde sind. In der zweiten Anforderung wird deutlich, daß hiervon auch Ausnahmen möglich sind. Das wäre nicht so offensichtlich, wenn die beiden Anforderungen nicht unmittelbar aufeinander folgen würden, sondern weit verstreut in dem Spezifikationsdokument wären.
- Die ersten beiden Anforderungen sind unpräzise, weil nicht angegeben ist, in welchen Fällen Ausnahmen von der vorgegebenen Reaktionszeit erlaubt sind. Nach Fertigstellung des Systems kann nicht objektiv entschieden werden, ob die Anforderung erfüllt wird.

---

1. ADMIRE ist ein Akronym und steht für *Ad*vanced *M*anagement of *I*nformal *Re*quirements.

- Ein ähnliches Problem liegt auch in der dritten Anforderung (Satz 3) vor. Was bedeutet »möglichst kurz«?
- Unklar ist zudem, ob es sich bei den »Benutzern« (Zeile 2) und den »Anwendern« (Zeile 6) um die gleichen Personen handelt.

Die hier beschriebenen Probleme können folgendermaßen gelöst oder zumindest vermindert werden:

- Der Absatz sollte in mehrere Absätze aufgespaltet werden. Jeder Absatz sollte dabei jeweils nur eine Anforderung oder Begründung enthalten. Um einen Absatz von anderen Stellen aus eindeutig referenzieren zu können, sollte er einen innerhalb des Dokuments eindeutigen Bezeichner erhalten.
- Die einzelnen Absätze sollten kategorisiert werden. Zu jedem Absatz sollte zumindest angegeben werden, ob es sich um eine »Anforderung«, eine »Begründung« oder ein »Beispiel« handelt.
- Wenn die Kategorisierung verfeinert würde, könnte sie genutzt werden, um inhaltlich zusammengehörende Informationen und Widersprüche auch dann noch aufzuspüren, wenn sie weit über das Dokument verstreut sind. Es wäre z.B. sinnvoll, Satz 1 und 3 jeweils als »Laufzeitanforderung« und Satz 2 als »Laufzeitanforderung« und »funktionale Anforderung« zu kategorisieren.
- Auch wenn erkannt wurde, daß die Anforderungen unpräzise und nicht prüfbar sind, ist es keineswegs trivial, sie zu präzisen und prüfbaren Anforderungen umzuformulieren. Das gilt insbesondere für nicht-funktionale Anforderungen, die oft umfassende Qualitäten oder Eigenschaften des Gesamtsystems beschreiben. Eine Möglichkeit, mit dem Problem umzugehen, besteht darin, Testfälle anzugeben. Die Kunden und Entwickler einigen sich darauf, daß das System die Anforderung erfüllt, wenn es alle angegebenen Testfälle erfüllt. Ein solcher Testfall könnte exemplarisch für einen typischen Arbeitsablauf vorgeben, welche Reaktionszeiten bei den einzelnen Teilschritten jeweils erlaubt sind.
- In vielen Fällen lassen sich unpräzise Aussagen in einer Anforderung an der Benutzung bestimmter Wörter (sog. »WeakWords«) festmachen. Ein solches WeakWord ist z.B. »kurz«. Liegt eine Liste von typischen WeakWords einer Sprache vor, können die natürlichsprachlichen Texte automatisch auf die Benutzung solcher Wörter geprüft werden.
- Die begrifflichen Unklarheiten können durch ein Glossar beseitigt werden, in dem die projektspezifischen Begriffe definiert werden.

1.3 Ergebnisse

Ziel des ADMIRE-Ansatzes ist es, den Requirements-Engineering-Prozeß zu verbessern, insbesondere die Erstellung, Verwaltung und Prüfung der Spezifikationsdokumente. Die Ergebnisse sollen in »normalen« Projekten anwendbar sein. Es müssen also die Rahmenbedingungen der Industrie berücksichtigt werden. Die wichtigste Rahmenbedingung ist, daß natürliche Sprache als Hauptnotation zur Spezifikation eingesetzt wird (siehe Kapitel 1.1).

Durch den ADMIRE-Ansatz sollen die Voraussetzungen dafür geschaffen werden, einige der in Kapitel 1.2 genannten und in einer Untersuchung des Stands der Pra-

xis (Kapitel 3) aufgedeckten Probleme zu beheben. Allgemeines Ziel ist es, qualitativ hochwertige Spezifikationsdokumente zu erstellen.

Die wesentlichen Ergebnisse dieser Arbeit sind:

- ein *Informationsmodell*, in dem alle für die natürlichsprachliche Spezifikation relevanten Informationstypen und deren Zusammenhänge beschrieben werden. Neben Anforderungen sind das z.B. Rahmenbedingungen, Erklärungen, Begründungen und Definitionen von Begriffen. Neben dem zu entwickelnden System wird auch die Arbeitsumgebung, in die das System eingebettet wird, berücksichtigt.

  Die Informationen können in natürlicher Sprache oder in sehr ähnlichen Notationen beschrieben werden. Darüber hinaus wird auch eine einfache graphische Notation unterstützt.

  Das Informationsmodell besteht aus einem informalen Teil, in dem die Typen eingeführt und deren Sinn begründet wird. Zur Präzisierung der informalen Aussagen wird das Informationsmodell formalisiert. Durch das Informationsmodell werden strukturelle Minimalbedingungen vorgegeben, die jede Instanz erfüllen muß. Für alle Qualitätsaspekte, die nicht durch die strukturellen Vorgaben abgedeckt werden, werden Qualitätskriterien definiert.

- ein *Prozeßmodell*, in dem eine exemplarische, sinnvolle und auf das Informationsmodell abgestimmte Aufteilung des Requirements-Engineering-Prozesses in Phasen beschrieben wird. Zu jeder Phase wird angegeben, welche Requirements-Engineering-Tätigkeiten durchgeführt werden sollen, welche Dokumente dabei erstellt oder geändert werden sollen, welche Qualitätskriterien die Dokumente erfüllen sollen und welche Prüfmaßnahmen durchgeführt werden sollen. Durch das Prozeßmodell wird beschrieben, wie eine Instanz des Informationsmodells sukzessive erzeugt, erweitert, geändert und geprüft werden kann.

  In dem Prozeßmodell werden alle Phasen berücksichtigt, in denen Requirements-Engineering-Tätigkeiten durchgeführt werden. Das sind neben der Problemanalyse- und Spezifikationsphase auch die späteren Projektphasen, in denen das Spezifikationsdokument verwendet und bei Bedarf geändert wird.

- eine Menge von visuellen und automatischen *Prüfverfahren* zur Validierung und Verifikation. Einige der automatischen Prüfverfahren basieren auf der Idee, daß ein Werkzeug nach Indizien für einen Mangel sucht, die Entscheidung, ob wirklich ein Mangel vorliegt, aber dem Benutzer überläßt. Für jedes im Informationsmodell definierte Qualitätskriterium werden Prüfverfahren angegeben.

- ein *Werkzeug*[1], das Instanzen des Informationsmodells verwalten kann. Neben Eingabe-, Verwaltungs- und Generierungsoperationen unterstützt das Werkzeug auch die konsistente Integration neuer Informationen in bestehende Spezifikationsdokumente sowie die Validierung und Verifikation. Die Spezifikationsdokumente können automatisch geprüft werden, soweit das bei natürlichsprachlichen Informationen möglich ist. Darüber hinaus wird auch die (visuelle) Prüfung durch Menschen unterstützt.

---

1. Weitere Informationen über das Werkzeug finden Sie auf den WWW-Seiten der Abteilung Software Engineering (http://www.informatik.uni-stuttgart.de/ifi/se/se.html) unter »Forschung«.

1.4 Aufbau dieser Arbeit

Grundlagen des Requirements Engineerings (Kapitel 2)

Dieses Kapitel enthält einen Überblick über die Forschungsdisziplin Requirements Engineering als Teilgebiet des Software Engineerings. Es werden die zu erstellenden Dokumente, die durchzuführenden Tätigkeiten und die Rollen der beteiligten Personen beschrieben. Zu den Tätigkeiten gehören die Erhebung von Informationen und die Dokumentation, Validierung und Verifikation der Ergebnisse. Die wichtigsten Spezifikationsnotationen und -methoden werden kurz vorgestellt. Es wird die Frage diskutiert, welche Vor- und Nachteile formale und informale Notationen für das Requirements Engineering haben.

Stand der Praxis – eine Untersuchung (Kapitel 3)

In diesem Kapitel wird eine Untersuchung des Stands der Requirements-Engineering-Praxis präsentiert, in der die Dokumente, beteiligten Personen und Prozesse von drei Projekten in drei Unternehmen genauer betrachtet wurden. Es werden Probleme identifiziert und Verbesserungsmöglichkeiten aufgezeigt.

Überblick über den ADMIRE-Ansatz (Kapitel 4)

Dieses Kapitel enthält einen Überblick über den ADMIRE-Ansatz. Die wichtigsten Eigenschaften des Informationsmodells, des Prozeßmodells, der Prüfverfahren und des Werkzeugs werden motiviert und kurz beschrieben. Am Ende des Kapitels wird das SESAM-Projekt eingeführt, das in den folgenden Kapiteln als durchgehendes Anwendungsbeispiel verwendet wird.

Informationsmodell (Kapitel 5)

In diesem Kapitel wird das ADMIRE-Informationsmodell durch Aufzählung seiner (strukturellen) Eigenschaften natürlichsprachlich beschrieben. Jede der Eigenschaften wird erklärt und motiviert. Am Ende dieses Kapitels werden die Qualitätskriterien definiert.

Formales Modell (Kapitel 6)

Um das Informationsmodell so präzise wie möglich zu beschreiben, wird es formalisiert. Dazu wird zuerst eine formale, UML-ähnliche Notation eingeführt und mit Prädikatenlogik erster Stufe kombiniert. Anschließend wird das Informationsmodell in dieser Notation formal beschrieben.

Prozeßmodell (Kapitel 7)

In diesem Kapitel wird das Prozeßmodell vorgestellt. Zuerst wird eine Aufteilung in Phasen vorgenommen. Anschließend werden die dabei durchzuführenden Requirements-Engineering-Tätigkeiten beschrieben. Zu jeder Tätigkeit wird angegeben, welche Dokumente jeweils zu Beginn vorliegen müssen und welche Dokumente erzeugt, geändert oder geprüft werden. Zuletzt wird durch Validierungs- und Verifikationstabellen für jedes Dokument vorgegeben, welche Qualitätskriterien es in welcher Phase erfüllen soll.

Validierung und Verifikation (Kapitel 8)

In diesem Kapitel werden die automatischen und visuellen Prüfverfahren vorgestellt. Zuerst werden die allgemeinen Prüfverfahren eingeführt. Anschließend wird gezeigt, durch welche konkreten Prüfverfahren die Erfüllung der Qualitätskriterien überprüft werden kann.

ADMIRE-Werkzeug (Kapitel 9)

In diesem Kapitel wird das ADMIRE-Werkzeug vorgestellt. Zuerst werden die allgemeine Funktionalität des Werkzeugs, die Benutzungsoberfläche und einige Realisierungsdetails beschrieben. Anschließend wird anhand eines ausführlichen Beispiels gezeigt, wie das ADMIRE-Werkzeug den Systemanalytiker bei seiner Arbeit unterstützen kann.

Vergleich und Bewertung (Kapitel 10)

In diesem Kapitel werden verwandte und ergänzende Ansätze aus den Forschungsbereichen Requirements Engineering, künstliche Intelligenz, Linguistik und Psychologie vorgestellt und mit dem ADMIRE-Ansatz verglichen. Abschließend wird der ADMIRE-Ansatz bewertet.

Kapitel 2

Grundlagen des Requirements Engineering

Im Software Engineering werden Projekte typischerweise in Phasen eingeteilt. Eine Beispielaufteilung wird in Abbildung 1 angegeben.

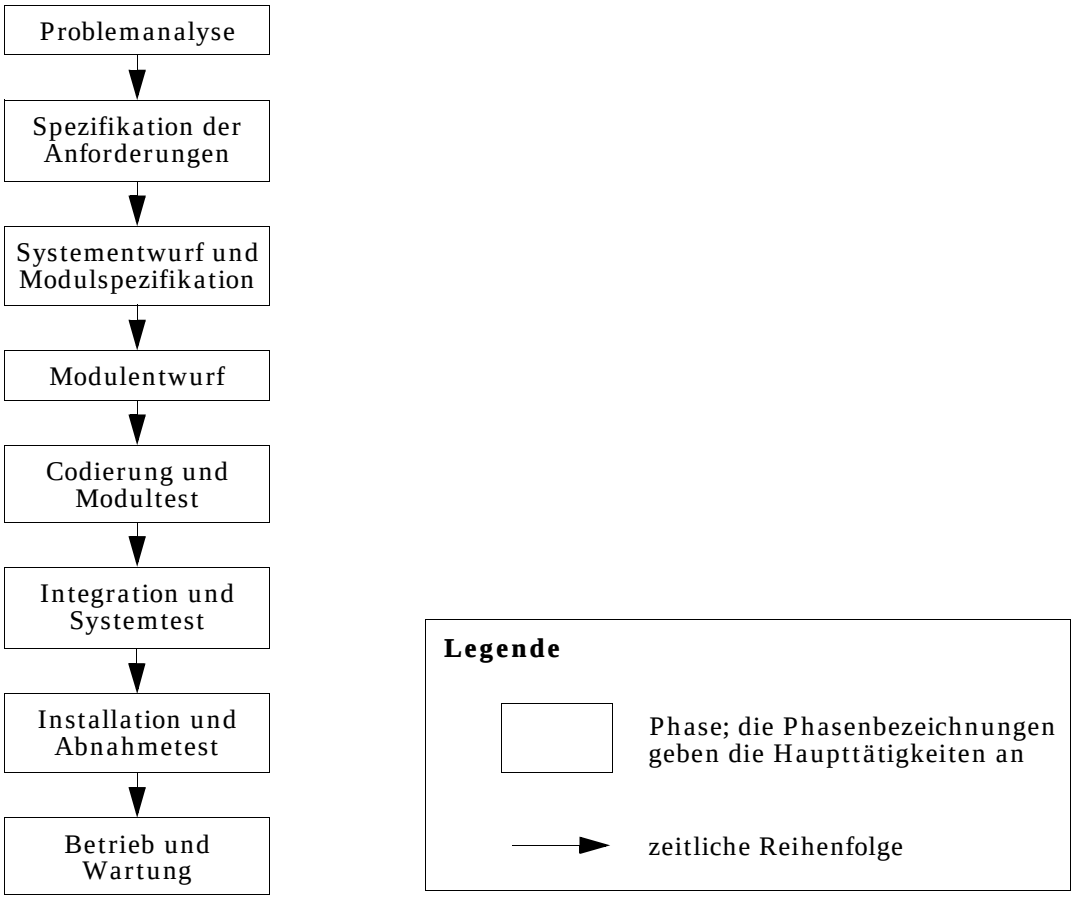


Abbildung 1: Phasenplan zur Software-Entwicklung

Die frühen Phasen bis einschließlich »Spezifikation der Anforderungen« werden auch als *Requirements-Engineering-Phase* bezeichnet. Ziel der frühen Phasen ist es, ausgehend von einer (vagen) Problemvorstellung oder einer gegebenen Aufgabenstellung, Anforderungen an das zu entwickelnde System, das sog. *Zielsystem*, zu erheben und eine Spezifikation zu erstellen. Ein Zielsystem ist in dieser Arbeit immer ein reines Softwaresystem.[1] Wenn im folgenden der Begriff »Zielsystem« verwendet wird, dann sind damit implizit auch immer die Fälle eingeschlossen, in denen mehrere Zielsysteme realisiert werden sollen.

Die Requirements-Engineering-Phase gilt als die wichtigste Phase in der Software-Entwicklung überhaupt. Welchen Nutzen hat es, ein nach Informatik-Gesichtspunkten perfektes System zu entwickeln, wenn es nicht das System ist, das die Kunden wünschen?

Dieses Kapitel vermittelt einen Überblick über das Requirements Engineering. In Kapitel 2.1 wird das Pflichtenheft als Hauptergebnis der Requirements-Engineering-Phase eingeführt. In Kapitel 2.2 werden die Rollen der typischerweise am Requirements Engineering beteiligten Personen beschrieben. Der Requirements-Engineering-Prozeß sowie die wichtigsten Methoden und Notationen werden dann in Kapitel 2.3 vorgestellt. Kapitel 2.4 schließlich behandelt die kontrovers diskutierte Frage, ob formale oder informale Notationen besser geeignet sind.

2.1 Pflichtenheft

Das zentrale Dokument der Requirements-Engineering-Phase ist die *Spezifikation* (requirements specification). Sie wird in IEEE (1990, S. 69) folgendermaßen definiert:

> »A document that specifies the requirements for a system or component. Typically included are functional requirements, performance requirements, interface requirements, design requirements, and development standards.«

Demzufolge enthält eine Spezifikation eine Menge von Anforderungen an ein zu realisierendes System oder eine zu realisierende Systemkomponente. Eine *Anforderung* wird in IEEE (1990, S. 62) folgendermaßen definiert:

> »1) A condition or capability needed by a user to solve a problem or achieve an objective. 2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document. 3) A documented representation of a condition or capability as in (1) or (2).«

Im folgenden gilt, daß alle Aussagen, die die Wünsche der Kunden und sonstiger Personen (siehe Kapitel 2.2) oder die zu realisierenden Aspekte des Zielsystems betreffen, als Anforderungen bezeichnet werden. Daß eine Anforderung sowohl die Kundensicht als auch die Systemsicht betreffen kann, wird von Macaulay (1996, S. 4) besonders klar herausgestellt. Sie definiert »Anforderung« folgendermaßen:

> »Something, which a customer needs, something, which needs to be designed.«

Häufig wird zwischen *funktionalen* und *nicht-funktionalen* Anforderungen unterschieden. Durch funktionale Anforderungen (*behavioral requirements*, *operational requirements)* wird das Verhalten des Zielsystems beschrieben. Durch nicht-funktionale Anforderungen (*non behavioral* oder *non operational requirements*) werden allumfassende Eigenschaften und Qualitäten des Zielsystems beschrieben. Solche Qualitäten sind z.B. Zuverlässigkeit, Effizienz, Sicherheitsaspekte, Wartbarkeit und Portierbarkeit.

---

1. Die Forschungsdisziplin, die sich mit der Realisierung von kombinierten Hardware- und Software-Systemen, sog. eingebetteten Systemen, beschäftigt, heißt System Engineering (Sailor, 1990, Dorfman, 1990).

Der Begriff »Spezifikation« ist mehrdeutig. Nagl (1990, S. 8) z.B. versteht unter einer Spezifikation ein Entwurfsdokument. Oft wird nicht unterschieden, ob es sich bei »Spezifikation« um eine Phase, eine Tätigkeit oder ein Dokument handelt. Deswegen wird von jetzt an der Begriff »*Pflichtenheft*« verwendet, wenn das Ergebnisdokument der Spezifikationsphase gemeint ist.

2.1.1 Zentrale Rolle des Pflichtenhefts

Das Pflichtenheft nimmt eine zentrale Rolle in einem Software-Projekt ein. Es dient als Grundlage für alle folgenden Phasen der Software-Entwicklung. Aufbauend auf das Pflichtenheft wird der *Entwurf* erstellt und implementiert. Aus dem Pflichtenheft können *Testfälle* für den *Abnahmetest* abgeleitet werden. Es dient als Grundlage für das *Benutzungshandbuch*. Auch für die *Wartung*, *Erweiterung* oder *Reimplementierung* eines Zielsystems ist ein Pflichtenheft nützlich.

Laut IEEE (1998, S. iii) bietet ein »gutes« Pflichtenheft u.a. folgenden Nutzen:

- Es dient als Grundlage für die Vereinbarungen zwischen Kunden und Entwicklern. Häufig ist es Teil des Vertrags. Ohne Pflichtenheft besteht die Gefahr, daß Anforderungen unklar oder ungeklärt bleiben.
- Durch die klare Vereinbarung zwischen Kunden und Entwicklern wird die Gefahr für eine Entwicklung des falschen Systems und damit der Aufwand für spätere Änderungen deutlich reduziert.
- Aufbauend auf das Pflichtenheft können realistische Kosten- und Zeitschätzungen für das Projekt durchgeführt werden.
- Es kann als Grundlage für Validierung, Verifikation und Abnahme verwendet werden. Ohne Pflichtenheft gibt es kein objektives Kriterium, um Streitigkeiten mit den Kunden zu klären.

2.1.2 Abgrenzung zwischen Pflichtenheft und Entwurf

Ein Problem im Requirements Engineering ist die genaue Abgrenzung zwischen Spezifizieren und Entwerfen. Oft wird diese Grenze folgendermaßen gezogen: Aussagen darüber, *was* das System können soll, gehören in das Pflichtenheft, Aussagen, die beschreiben, *wie* das System realisiert werden soll, gehören in das Entwurfsdokument (oder in die Implementierung).

Auch wenn diese Trennung auf den ersten Blick einleuchtend erscheint, ist sie in der Praxis dennoch nicht einfach durchzuführen. Die Ansicht, ob ein Dokument gerade das *was* oder *wie* beschreibt, hängt von der Rolle des Betrachters ab.

Der Kunde hat ein Problem, das gelöst werden soll. Aus diesem Problem heraus ergeben sich Anforderungen. Diese Anforderungen beschreiben das *was* aus Sicht des Kunden. Aufbauend auf die Anforderungen wird ein Pflichtenheft erstellt. Dieses beschreibt aus Sicht des Kunden die Lösung seines Problems (*wie*-Sicht des Kunden); aus Sicht des Entwerfers enthält es Vorgaben (*was*-Sicht des Entwerfers). Das Entwurfsdokument wiederum beschreibt die Systemarchitektur (*wie*-Sicht des Entwerfers), die als Vorgabe für den Codierer dient (*was*-Sicht des Codierers). Dieses Gedankenspiel läßt sich beliebig weit fortsetzen (siehe *»What versus how«-Dilemma* in Davis (1993, S. 17f)).

Im folgenden gilt, daß das Pflichtenheft als Abschlußdokument der Requirements-Engineering-Phase für den Kunden die Lösung seines Problems beschreiben und für den Entwerfer als Vorgabe dienen soll.

Ein anderes Abgrenzungsproblem besteht darin, daß die Requirements-Engineering- und Entwurfstätigkeiten nicht streng sequentiell durchgeführt werden können. Der Entwurf kann bei der Erstellung des Pflichtenhefts nicht ignoriert werden (Swartout, Balzer, 1982, Ludewig, 1982, S. 13f), weil Entwurfsentscheidungen Rückwirkungen auf das Pflichtenheft haben können.

Angenommen, Datensicherheit spielt in einem Software-Projekt eine wichtige Rolle, dann kann der Entwerfer (u.a.) eine der folgenden Möglichkeiten wählen:

- Er benutzt eine Datenbank mit entsprechenden Datensicherungsmechanismen.
- Das Zielsystem speichert die Daten regelmäßig in einer Datei.

Werden die Daten nur vom Zielsystem verwendet, dann liegt es im Ermessen des Entwerfers, welche Möglichkeit er wählt. Wählt er die zweite Möglichkeit, dann sollten im Pflichtenheft Angaben über die Häufigkeit, mit der die Daten gesichert werden sollen, enthalten sein. Das wiederum bedeutet aber, daß die zweite Möglichkeit explizit im Pflichtenheft berücksichtigt werden mußte.

Um dieses Problem zu lösen, gibt es nur zwei Möglichkeiten. Entweder geben die Spezifizierer Lösungsmöglichkeiten vor, treffen also Realisierungsentscheidungen, oder sie formulieren die Datensicherheitsanforderungen so allgemein, daß deren Erfüllung bei beliebigen Datensicherungsmechanismen überprüft werden kann. Letzteres ist sehr schwierig.

Die Konsequenz daraus ist, daß die klare Trennung zwischen Pflichtenheft und Entwurf, wie sie in der reinen Lehre oft gefordert wird, in der Praxis nicht beibehalten werden kann, weil im Pflichtenheft oft Fragen beantwortet werden müssen, die sich erst durch den Entwurf stellen, und weil ein Pflichtenheft durchaus Realisierungsvorgaben enthalten darf. Häufig werden z.B. die zu benutzenden Programmiersprachen und Bibliotheken vorgegeben. Ein Pflichtenheft sollte nur solche Realisierungsvorgaben enthalten, die die Kunden wirklich verlangen. Die Freiheit des Entwerfers sollte so wenig wie möglich eingeschränkt werden.

2.1.3 Eigenschaften eines guten Pflichtenhefts

In (Davis, 1993, S. 181ff), (Davis et al., 1993), (IEEE, 1998, S. 4ff), (Robertson, Robertson, 1999, S. 181ff) und (Roman, 1985, S. 16) werden Eigenschaften beschrieben, die ein Pflichtenheft soweit wie möglich erfüllen sollte.

- *korrekt*: Die Korrektheit ist hoch, wenn die enthaltenen Informationen die Probleme und Wünsche jeweils mindestens eines Kunden beschreiben.
- *relevant*: Die Relevanz ist hoch, wenn die enthaltenen Informationen sinnvoll zur Beschreibung oder Lösung der Probleme beitragen.
- *vollständig*: Die Vollständigkeit ist hoch, wenn alle Anforderungen der Kunden und alle für die Kunden relevanten Funktionen, Qualitäten und sonstigen Eigenschaften des Zielsystems beschrieben werden.
- *konsistent*: Die Konsistenz ist hoch, wenn die enthaltenen Informationen keine Widersprüche aufweisen.

- *extern konsistent*: Die externe Konsistenz ist hoch, wenn die enthaltenen Informationen keine Widersprüche zu vorhandenen Projektdokumenten oder sonstigen für das Projekt relevanten Dokumenten aufweisen.
- *adäquat*: Die Adäquatheit ist hoch, wenn die Probleme und Wünsche aus Sicht der Kunden und bezogen auf die Problemstellung angemessen beschrieben werden.
- *realisierbar*: Das Pflichtenheft ist realisierbar, wenn es eine Implementierung gibt, die alle enthaltenen Anforderungen gleichzeitig erfüllt.
- *nicht redundant*: Die Redundanz ist niedrig, wenn wenige Sachverhalte an mehr als einer Stelle beschrieben werden.
- *eindeutig*: Die Eindeutigkeit ist hoch, wenn die Anforderungen so formuliert sind, daß sie für alle zu erwartenden Leser die gleiche Bedeutung haben.
- *überprüfbar*: Die Überprüfbarkeit ist hoch, wenn die Anforderungen so formuliert sind, daß kosteneffektiv und mit vertretbarem Aufwand geprüft werden kann, ob das Zielsystem die Anforderungen erfüllt.
- *präzise*: Die Präzision ist hoch, wenn die Anforderungen keine ungenauen Aussagen enthalten und wenn, soweit möglich, numerische Werte verwendet werden.
- *verständlich*: Die Verständlichkeit ist hoch, wenn die Anforderungen so geschrieben sind, daß die zu erwartenden Leser ihre Bedeutung leicht erfassen können.
- *annotiert*: Die Annotiertheit ist hoch, wenn der Leser leicht herausfinden kann, welche Anforderungen am wichtigsten für die Kunden sind, welche Anforderungen sich am wahrscheinlichsten ändern werden und welche Anforderungen in welcher Produktversion erfüllt werden müssen.
- *entwurfsunabhängig*: Die Entwurfsunabhängigkeit ist hoch, wenn das Pflichtenheft wenige Entwurfs- und Implementierungsvorgaben enthält.
- *vorwärtsgerichtet nachvollziehbar*: Die vorwärtsgerichtete Nachvollziehbarkeit ist hoch, wenn die Anforderungen eindeutig gekennzeichnet sind, so daß von anderen Dokumenten aus auf sie Bezug genommen werden kann.
- *rückwärtsgerichtet nachvollziehbar:* Die rückwärtsgerichtete Nachvollziehbarkeit ist hoch, wenn zu den Anforderungen die Ursprünge ersichtlich sind.
- *verwendbar während Einsatz und Wartung*: Die Verwendbarkeit ist hoch, wenn das Pflichtenheft so beschaffen ist, daß es nicht nur während der Entwicklungsphase gebraucht werden kann, sondern auch später, wenn das Zielsystem eingesetzt wird oder Änderungen am Zielsystem vorgenommen werden.
- *änderbar*: Die Änderbarkeit ist hoch, wenn Stil und Struktur des Pflichtenhefts so sind, daß Änderungen einfach, vollständig und konsistent durchgeführt werden können.
- *strukturiert*: Die Strukturiertheit ist hoch, wenn das Pflichtenheft so aufgebaut ist, daß der logische Zusammenhang zwischen zusammengehörenden Teilen klar erkennbar ist und daß einzelne Informationen leicht gefunden werden können. Es sollte also zumindest ein Inhaltsverzeichnis und einen Index enthalten.

- *querverwiesen*: Die Querverwiesenheit ist hoch, wenn Querverweise zwischen zusammengehörenden Informationen enthalten sind, insbesondere zwischen redundanten oder sich ergänzenden Informationen und zwischen abstrakteren und detaillierteren Beschreibungen der gleichen Information.
- *elektronisch gespeichert*: Das Pflichtenheft ist elektronisch gespeichert, wenn alle Teile in rechnerlesbarer Form vorliegen.
- *ausführbar*: Das Pflichtenheft ist ausführbar, wenn es ein Software-Werkzeug gibt, das das Pflichtenheft als Eingabe akzeptiert und dynamisch das Verhalten des Zielsystems simulieren kann.
- *wiederverwendbar*: Die Wiederverwendbarkeit ist hoch, wenn die enthaltenen Abschnitte so formuliert sind, daß sie leicht an andere Gegebenheiten angepaßt werden können.
- *knapp*: Die Knappheit ist hoch, wenn das Pflichtenheft keine unnötigen oder kürzer formulierbaren Informationen enthält.

Da sich die Eigenschaften zum Teil widersprechen, können in der Regel nicht alle gleichzeitig erfüllt werden (siehe Tabelle 1).

Eigenschaften	Beschreibung
Konsistenz ↔ Korrektheit	Die Anforderungen mehrerer Kunden, die, jede für sich alleine betrachtet, korrekt sind, enthalten Widersprüche.
Entwurfsunabh. ↔ Vollständigkeit	Die Kunden machen Realisierungsvorgaben (siehe Kapitel 2.1.2).
nicht redundant ↔ Verständlichkeit	Oft enthalten Erklärungen, die zur Verbesserung der Verständlichkeit notwendig sind, redundante Informationen.
nicht redundant ↔ querverwiesen	Jeder Querverweis ist redundant.
Knappheit ↔ Verständlichkeit	Wenn Informationen zu knapp präsentiert werden, sind sie oft nicht mehr verständlich. Bis zu einem gewissen Grad kann Knappheit aber sinnvoll sein, insbesondere weil knapp formulierte Aussagen oft prägnant formuliert sind.
Ausführbarkeit ↔ Verständlichkeit	Damit ein Pflichtenheft ausführbar ist, muß es in einer formalen Notation geschrieben sein. In Kapitel 2.4 wird gezeigt, daß formale Notationen schwer verständlich sind.

Tabelle 1: Mögliche Widersprüche zwischen den Eigenschaften

2.1.4 Normen für Pflichtenhefte

Es gibt viele verschiedene *Normen* für Pflichtenhefte (Dorfman, Thayer, 1990). Fast jede größere Organisation verwendet ihre eigene Norm. Grob lassen sich die Normen in zwei Kategorien einteilen:

- *Produktnormen*, z.B. IEEE Std. 830-1998 (IEEE, 1998), beschreiben Inhalt und Aufbau eines Pflichtenheftes,

- *Prozeßnormen* beschreiben die Tätigkeiten, die bei der Software-Entwicklung durchgeführt werden sollen. Auf diese Weise soll gewährleistet werden, daß das resultierende Produkt (Zielsystem) bestimmten Qualitätsanforderungen genügt. Prozeßnormen werden in diesem Kapitel nicht weiter behandelt.

Eine Produktnorm läßt sich vor allem für folgende Zwecke einsetzen:

- als Checkliste, um zu prüfen, ob alle notwendigen Informationen vorhanden sind, und
- als Gliederungsgrundlage für das zu erstellende Pflichtenheft.

Nach IEEE Std. 830-1998 (IEEE, 1998, S. 10ff) muß ein Pflichtenheft folgende Teile enthalten (siehe auch Abbildung 2):

- eine Einführung, in der ein Überblick über Zweck und Funktionsumfang des Zielsystems gegeben wird und die wichtigsten Begriffe definiert werden,
- eine allgemeine Beschreibung, in der die Einbettung des Zielsystems in seine Umgebung, seine Schnittstellen, seine wichtigsten Funktionen, die Charakteristika der späteren Benutzer, Entwurfseinschränkungen, Annahmen und Abhängigkeiten beschrieben werden, und
- eine detaillierte Beschreibung aller Anforderungen. Diese wiederum enthält:
  - Anforderungen an die Schnittstellen des Zielsystems,
  - funktionale Anforderungen,
  - Leistungsanforderungen (nicht-funktionale Anforderungen bzgl. der Effizienz; hierzu gehören z.B. Anforderungen bzgl. Datenmenge, Datenkomplexität, Anzahl gleichzeitiger Benutzer und Reaktionszeiten),
  - Entwurfsvorgaben, also alle Anforderungen, die die Freiheiten des Entwerfers einschränken, und
  - Attribute (sonstige nicht-funktionale Anforderungen; hierzu zählen z.B. Angaben über Verfügbarkeit, Sicherheit, Zuverlässigkeit, Wartbarkeit und Portierbarkeit).

Der IEEE Std. 830-1998 (IEEE, 1998, S. 21ff) macht mehrere Vorschläge für die Gliederung des 3. Kapitels, abhängig davon, ob z.B. Betriebsmodi, Daten oder Funktionen im Vordergrund stehen. Die Norm ist notationsunabhängig, d.h. innerhalb der einzelnen Kapitel können beliebige Notationen verwendet werden.

Ein Pflichtenheft nach IEEE Std. 830-1998 (IEEE, 1998, S. 10) enthält ausschließlich Anforderungen an das Zielsystem. Im Gegensatz dazu enthält ein Pflichtenheft nach »British Standard Guide to Specifying User Requirements for a Computer-Based System« (British Standards Institution, 1986, S. 42) auch Anforderungen an das Projekt (z.B. »Das Projekt muß bis zum 31.12.1999 abgeschlossen sein.«).

2.2 Rollen

An den Requirements-Engineering-Tätigkeiten sind in der Regel viele verschiedene Personen beteiligt. Eine wichtige Position nehmen die *Systemanalytiker* ein. Sie

Table of Contents

1. Introduction
   - 1.1. Purpose
   - 1.2. Scope
   - 1.3. Definitions, Acronyms, and Abbreviations
   - 1.4. References
   - 1.5. Overview
2. Overall Description
   - 2.1. Product Perspective
   - 2.2. Product Functions
   - 2.3. User Characteristics
   - 2.4. Constraints
   - 2.5. Assumptions and Dependencies
3. Specific Requirements
   - 3.1. External Interface Requirements
     - 3.1.1.User Interfaces
     - 3.1.2.Hardware Interfaces
     - 3.1.3.Software Interfaces
     - 3.1.4.Communication Interfaces
   - 3.2. Functional Requirements
     - 3.2.1.Mode 1
       - 3.2.1.1.Functional Requirement 1
       - ...
       - 3.2.1.n. Functional Requirement n
     - ...
     - 3.2.m. Mode m
       - ...
   - 3.3. Performance Requirements
   - 3.4. Design Constraints
   - 3.5. Software System Attributes
   - 3.6. Other Requirements
4. Appendixes
5. Index

Abbildung 2: Struktur eines zu IEEE-Std. 830-1998 konformen Pflichtenhefts

führen die meisten der in Kapitel 2.3 beschriebenen Tätigkeiten selbst durch oder sorgen dafür, daß sie durchgeführt werden. Es ist ihre Aufgabe, die in Kapitel 2.1 beschriebenen Dokumente zu erstellen und zu pflegen.

Die Anforderungen und sonstigen Informationen, die in das Pflichtenheft eingetragen werden sollen, sind keine Erfindungen der Systemanalytiker, sondern müssen erhoben werden. Alle Personen, die von dem Zielsystem positiv oder negativ betroffen sein können und alle Personen und Dokumente, die für die Entwicklung und den Gebrauch des Zielsystems relevante Informationen liefern können, sind potentielle *Informationsquellen*. Die Personen-Informationsquellen werden auch *Stakeholder* (Macaulay, 1996, S. 32) oder *Informanten* genannt.

Bei Informanten sollte zwischen Personen und Rollen unterschieden werden. Wenn eine Anforderung von Peter Müller kommt, dann ist einerseits die konkrete

Person von Interesse, andererseits aber auch die Rolle, die Peter Müller in dem Requirements-Engineering-Prozeß einnimmt, z.B. »Benutzer des Zielsystems«, »Experte des Anwendungsbereichs« oder »Auftraggeber, der das Projekt finanziert«. Zwei Informanten nehmen die gleiche Rolle ein, wenn sie für das Projekt ähnliche Informationen liefern können, eine ähnliche Sichtweise vertreten oder über ähnliche Befugnisse verfügen. Die Menge aller Informanten, die die gleiche Rolle einnehmen, wird *Partei* genannt.

Abbildung 3 gibt einen Überblick über typische Parteien in einem Requirements-Engineering-Prozeß. Wenn im Projektverlauf eine konkrete Person nicht mehr zur Verfügung steht, kann die Parteizugehörigkeit verwendet werden, um einen Ersatzansprechpartner zu ermitteln, der voraussichtlich ähnliche Informationen liefern kann.

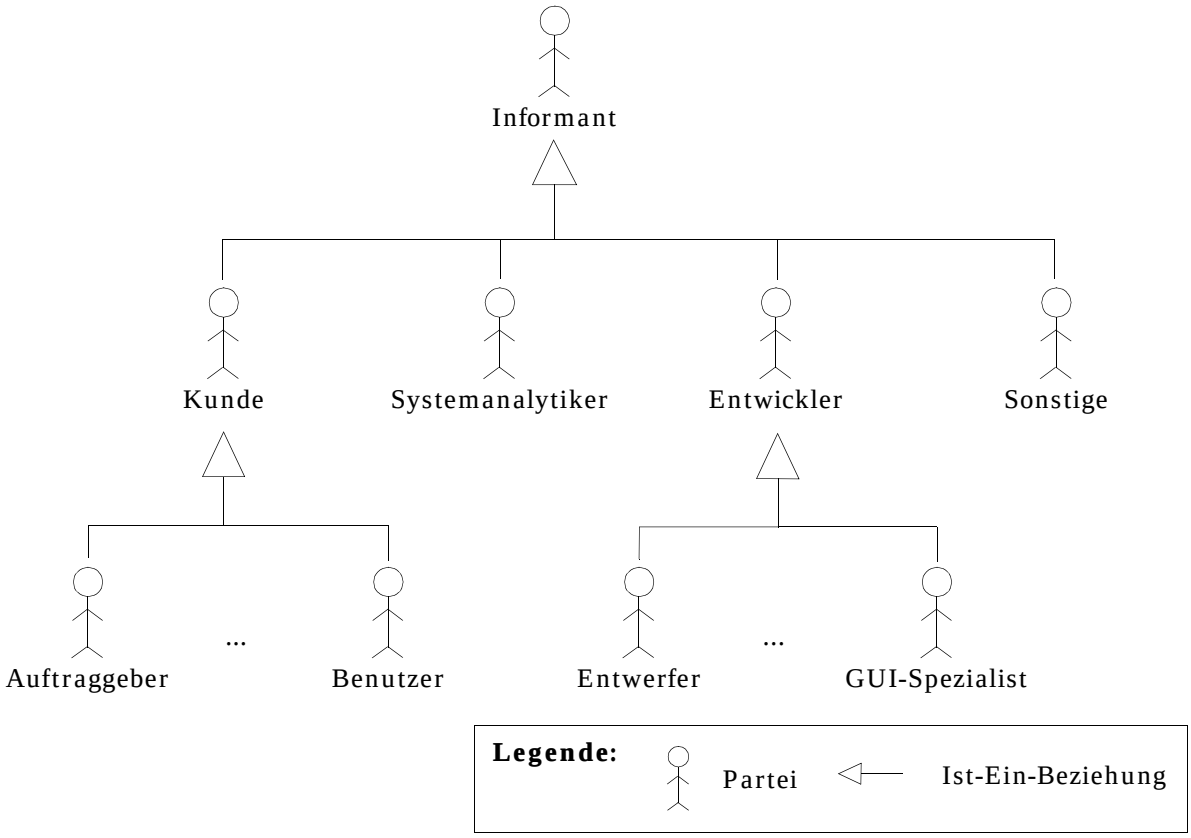


Abbildung 3: Typische Parteien in einem Software-Projekt

In jedem Projekt gibt es *Kunden* und *Entwickler*. Die Kunden sind diejenigen, die Bedarf für ein neues oder geändertes System haben. Die Entwickler sind diejenigen, die das neue System realisieren sollen. Kunden und Entwickler können durchaus dieselben Personen sein. Darüber hinaus können sonstige Informanten, z.B. Rechtsexperten, Sicherheitsexperten oder Mitarbeiter von Behörden, Versicherungen oder Zertifizierungsstellen, für die Requirements-Engineering-Tätigkeiten relevant sein.

In vielen Projekten können die Kunden unterteilt werden in Auftraggeber und Benutzer. *Auftraggeber* sind die Personen, mit denen der Vertrag geschlossen wird. Sie legen die Rahmenbedingungen fest und sind befugt, in Zweifelsfällen Entscheidungen zu treffen. *Benutzer* sind diejenigen, die später mit dem Zielsystem arbeiten werden. Das können »normale Anwender« sein ebenso wie Operateure oder

Wartungspersonal. Auch die Entwickler können bei größeren Softwareprojekten weiter unterteilt werden, z.B. in Entwerfer, Codierer, GUI-Spezialist und Tester.

Die Systemanalytiker nehmen eine zentrale Position ein (Abbildung 4). Sie kommunizieren mit den Informanten, um Anforderungen zu ermitteln oder Entscheidungen zu treffen. Sie nehmen außerdem eine Vermittlerrolle ein. Wenn sie auf Widersprüche stoßen, müssen sie deren Lösung initiieren. In der Regel dürfen sie aber die Entscheidung, wie der Widerspruch zu lösen ist, nicht selbständig treffen.

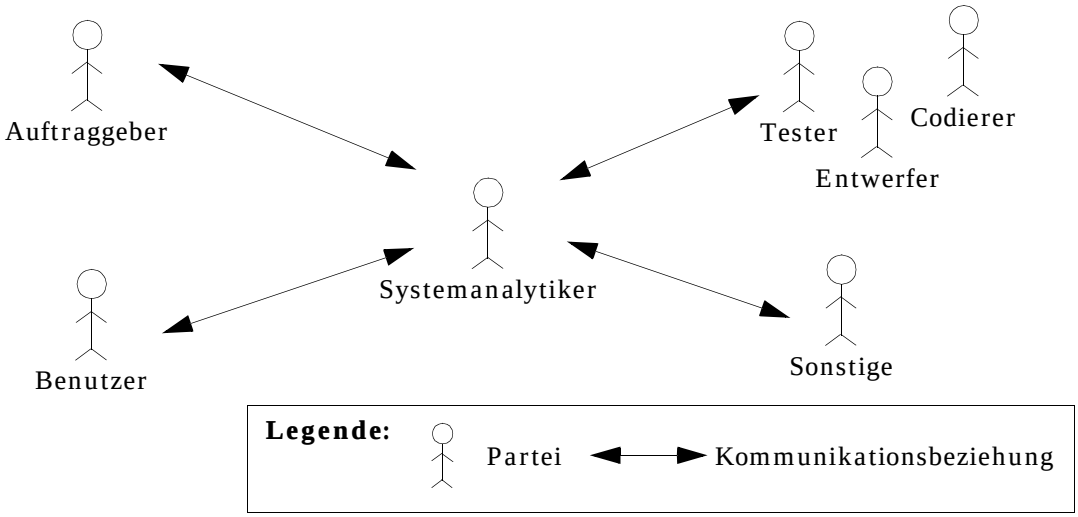


Abbildung 4: Zentrale Rolle der Systemanalytiker

2.3 Prozeß

Die Requirements-Engineering-Phase kann grob unterteilt werden in die *Problemanalysephase* und die *Spezifikationsphase* (Davis, 1993, S. 20f). In jeder der Phasen entsteht ein Dokument als Hauptergebnis: das Problemanalysedokument oder *Lastenheft* in der Problemeanalysephase, die Spezifikation oder das *Pflichtenheft* in der Spezifikationsphase.

Im Pflichten- und Lastenheft stehen unterschiedliche Aspekte im Vordergrund. Ziel der Problemanalysephase ist es, die Anforderungen und Rahmenbedingungen aus Sicht der Kunden zu erfassen. Es dürfen durchaus noch Inkonsistenzen und Vagheiten vorhanden sein. In der Spezifikationsphase steht die vollständige, konsistente und präzise Beschreibung eines Zielsystems im Vordergrund. Die vorhandenen Vagheiten und Inkonsistenzen müssen im Pflichtenheft beseitigt werden.

Ein Beispiel für eine solche Vagheit ist die genaue Grenze zwischen Zielsystem und seiner *Arbeitsumgebung*. Welche Aufgaben sollen von dem Zielsystem und welche von den Benutzern oder anderen Systemen übernommen werden? Angenommen, das Zielsystem soll Bestellungen verwalten, Zahlungen überwachen und feststellen, welche Käufer ihre Bestellungen schon bezahlt haben, dann gibt es u.a. folgende Lösungsmöglichkeiten:

- Das Zielsystem überwacht über eine Telefonverbindung regelmäßig den Zahlungsverkehr auf dem Firmenkonto und führt automatisch einen Abgleich durch.

- Der Sachbearbeiter erhält regelmäßig einen Kontoauszug von der Bank und gibt diese Informationen über ein Terminal an das Zielsystem weiter.

Im zweiten Fall übernimmt das Zielsystem von der Aufgabe, die Zahlungen mit den Bestellungen abzugleichen, nur einen Teil. Zur Lösung des Problems müssen die Sachbearbeiter ebenfalls einen Teil beitragen.

Ein anderes Beispiel für eine solche Vagheit ist die konkrete Gestalt der Schnittstellen des Zielsystems zu den Benutzern. Oftmals hat der Kunde nur vage Vorstellungen, z.B. »Die Benutzungsschnittstelle soll einfach zu bedienen sein.« oder »Die Benutzungsschnittstelle soll das typische Windows-Look-and-Feel haben.«. Über das konkrete Aussehen der Oberfläche kann der Kunde keine genauen Aussagen machen. Spätestens bei der Implementierung wird die konkrete Gestaltung der Benutzungsoberfläche festgelegt. Es ist aber aus folgenden Gründen sinnvoller, wenn dies schon in der Requirements-Engineering-Phase geschieht:

- Die Benutzer sind von der Benutzungsoberfläche direkt betroffen, so daß es sinnvoll ist, mit ihnen darüber zu diskutieren (und zu verhandeln), bevor die Realisierung beginnt. Die Erfahrung zeigt, daß die Akzeptanz eines Zielsystems erheblich von seiner Benutzungsoberfläche abhängt (siehe auch Kapitel 3.5.1).
- Für Systemanalytiker und Benutzer spielen Realisierungsdetails keine so wichtige Rolle, so daß sie mehr um eine logisch und einfach zu bedienende Benutzungsoberfläche bemüht sind als Entwerfer und Codierer, die eher an einer einfachen Realisierung interessiert sind.

2.3.1 Tätigkeiten

Nach (Sommerville, Sawyer, 1997, S. 11) muß ein Requirements-Engineering-Prozeß zumindest folgende Tätigkeiten beschreiben oder vorschreiben:

- Informationen erheben (elicitation): Möglichst alle relevanten Informationsquellen werden identifiziert. Von den Informanten werden Informationen erhoben. Die Dokumente werden ausgewertet. Informationserhebungstechniken werden in Kapitel 2.3.2 beschrieben.
- Informationen analysieren/verifizieren: Die erhobenen Informationen werden integriert und beispielsweise auf Widersprüche hin untersucht.
- Informationen validieren: Die erhobenen Informationen werden den Informanten zur Prüfung vorgelegt. Validierungs- und Verifikationstechniken werden in Kapitel 2.3.4 beschrieben.
- Verhandeln (negotiation): Sind in Teilen der erhobenen Informationen Widersprüche enthalten, müssen sie gelöst werden. Dafür müssen die »richtigen« Informanten involviert werden. Verhandlungstechniken werden z.B. in (Boehm, Horowitz, Lee, 1995) beschrieben.

Neben den bisher beschriebenen Tätigkeiten kann es, je nach Projekt, sinnvoll sein, weitere Tätigkeiten durchzuführen. Im folgenden werden die Tätigkeiten Domänenanalyse, Ist-Analyse, Zielanalyse und Task-Analyse kurz vorgestellt.

Domänenanalyse

Häufig ist es sinnvoll, nicht nur die Anforderungen für ein spezielles Zielsystem zu sammeln, sondern darüber hinaus auch anwendungsbereichspezifisches Wissen.

Das Sammeln und Modellieren von Anforderungen für einen Anwendungsbereich wird *Domänenanalyse* (Lam, McDermid, 1997, Iscoe 1993) genannt. Die Domänenanalyse liefert ein *Domänenmodell*, das für alle (bzw. viele) Systeme aus dem Anwendungsbereich Gültigkeit haben soll. Ein Domänenmodell enthält typischerweise Daten und *generische Anforderungen*, die für konkrete Systeme vervollständigt oder angepaßt werden müssen.

Das Domänenmodell kann als Referenzwissen und Ausgangsbasis für die Diskussion über neue Systeme aus dem gleichen Anwendungsbereich benutzt werden. Durch das Domänenmodell wird eine gemeinsame Basis geschaffen, durch die die Kommunikation zwischen verschiedenen Personen mit unterschiedlichem Kenntnisstand über den Anwendungsbereich erleichtert werden kann.

Ist-Analyse

In den meisten Fällen soll durch ein Projekt eine Verbesserung des aktuellen Zustands erreicht werden: Manuelle Tätigkeiten sollen automatisiert werden. Ein schon vorhandenes (»legacy«) System soll ersetzt oder erweitert werden. Werden Informanten nach ihren Anforderungen befragt, so treffen sie oft Aussagen der Art »so wie bisher, aber ...«. Damit diese Aussagen richtig verstanden werden können, muß die aktuelle Situation bekannt sein.

Die Tätigkeit, in der die aktuelle Situation (*Ist-Zustand*) in einem Unternehmen analysiert wird, wird *Ist-Analyse* (Ludewig, 1997, S. 15-1ff, Yeh, Ng, 1990, S. 454) genannt. Zur Beschreibung des Ist-Zustands gehören Aussagen über die aktuelle Arbeitsumgebung inkl. eines evtl. vorhandenen zu ersetzenden Systems, über die Arbeitsabläufe und die daran beteiligten Personen (bzw. deren Rollen) und über vorhandene Probleme.

Zielanalyse

In der Requirements-Engineering-Phase werden häufig falsche Annahmen getroffen bzgl. der Politik des Managements, Mißtrauen und Widerstand gegen neue Systeme, eingeführter und bewährter Praktiken usw. In der *Zielanalyse* (Yeh, Ng, 1990, S. 454) sollen die langfristigen Ziele einer Firma ermittelt werden, sowie die Bedingungen, die ein Erreichen der Ziele unterstützen oder behindern können, um möglichst früh Konflikte zu erkennen.

Task-Analyse

In der *Task-Analyse* wird untersucht, wie das Zielsystem in die Arbeitsumgebung eingebettet werden soll. Welche Teile der zur Lösung des übergeordneten Problems notwendigen Arbeitsabläufe soll das Zielsystem übernehmen? Inwieweit dürfen die bisherigen Arbeitsabläufe durch das neue System geändert werden? Hierbei wird das Zielsystem nur als eine Komponente von vielen betrachtet.

2.3.2 Erheben von Informationen

Eine zentrale Aufgabe der Systemanalytiker ist die Informationserhebung. Dieser Aspekt wird in vielen Requirements-Engineering-Methoden vernachlässigt, insbesondere in Methoden, die im Wesentlichen eine Notation bereitstellen, z.B. OMT (Rumbaugh et al., 1991). Die Autoren dieser Methoden gehen von einer gegebenen Aufgabenstellung oder einer informalen Beschreibung des Problems aus. In der Praxis kann in den meisten Fällen nicht von einer gegebenen Aufgabenstellung ausgegangen werden. Es kann nicht einmal vorausgesetzt werden, daß das Problem verstanden ist oder alle Informanten die gleiche Sicht auf das Problem haben.

Im folgenden werden zuerst Probleme beschrieben, die typischerweise bei der Kommunikation mit Informanten auftreten. Danach werden einige Techniken zur Informationserhebung kurz vorgestellt.

Klärung des Problems

Bevor Anforderungen niedergeschrieben werden können, müssen zuerst die zu lösenden Probleme geklärt werden (Davis, 1993). Die Probleme liegen in der Arbeitsumgebung der Kunden vor und können in der Terminologie der Kunden beschrieben werden.

Aussagen, die die zu lösenden Probleme betreffen, werden von Jackson (1995, S. 193ff) »Anforderungen« genannt. Werden konkrete Aussagen über das Zielsystem und die Einbettung des Zielsystems in seine Arbeitsumgebung getroffen, spricht er von »Spezifikationen«[1]. Aspekte, die nur das Zielsystem unabhängig von seiner Arbeitsumgebung betreffen, bezeichnet er als »Programme«. Jackson argumentiert, daß die Unterscheidung zwischen Anforderungen, Spezifikationen und Programmen sehr wichtig ist. Zuerst müssen die Anforderungen, also das Problem, formuliert werden, erst dann sollte durch Spezifikationen konkret beschrieben werden, wie das Problem gelöst wird. Die Formulierung der Programme geschieht dann in der Entwurfs- und Implementierungsphase.

Jackson unterstreicht durch die spezielle Auslegung des für das Requirements Engineering zentralen Begriffs »Anforderung« die Bedeutung, die der Problemsicht im Gegensatz zur Systemsicht zukommt.

Probleme bei der Kommunikation

Weinberg (1988, S. 56ff) stellt ein anschauliches Gedankenspiel vor (das sog. »Railroad-Paradoxon«), das zeigt, wie leicht Fehler bei der Informationserhebung gemacht werden können:

> »Die Bewohner einer kleinen Industriestadt in den USA müssen zum Einkaufen in die nächstgelegene Großstadt fahren. Die günstigste Zeit dafür wäre der frühe Nachmittag. Zu der Zeit hält jedoch kein Zug am Bahnhof der Industriestadt. Allerdings fährt um 14.30h ein Fernverkehrszug ohne anzuhalten an dem Bahnhof vorbei. Deswegen baten einige Bewohner die Bahngesellschaft, den Zug halten zu lassen. Bevor die Bahngesellschaft eine dauerhafte Änderung in Betracht zog, sollte geprüft werden, ob wirklich Bedarf vorhan-

1. Hierbei handelt es sich um eine sehr spezielle Sichtweise auf die Begriffe »Anforderung« und »Spezifikation«. Sie werden in den anderen Kapiteln dieser Arbeit nicht so verwendet.

> den war. Deswegen wurde in den folgenden Tagen jeweils um 14.30h überprüft, ob jemand am Bahnsteig steht, um mitzufahren. Da der zusätzliche Halt nicht im Fahrplan stand, war niemand da. Daraus wurde geschlossen, daß für einen zusätzlichen Halt anscheinend doch kein Bedarf besteht.«

Die Geschichte zeigt folgendes Problem: Wenn ein Produkt für bestimmte Benutzer unbrauchbar ist, wird es von ihnen auch nicht benutzt. Weil sie es aber nicht benutzen, werden sie oft auch nicht als potentielle Benutzer für ein verbessertes Produkt in Betracht gezogen und nicht in den Requirements-Engineering-Prozeß mit einbezogen. Deswegen bleiben ihre Anforderungen unberücksichtigt.

Ein Problem sind *implizite Annahmen*, sowohl auf Seiten der Systemanalytiker als auch auf Seiten der Kunden und Informanten. Für die eine Seite sind bestimmte Sachverhalte selbstverständlich, so daß sie nicht extra erwähnt werden müssen. Für die andere Seite, die oft aus einem ganz anderen Fachbereich kommt, sind diese Sachverhalte absolut nicht selbstverständlich. Die Systemanalytiker müssen versuchen, diese impliziten Annahmen zu entdecken und explizit zu formulieren, um potentielle Mißverständnisse so früh wie möglich zu erkennen.

Ein weiteres Problem ist der Zeitaufwand, den die Kunden in der Problemanalyse- und Spezifikationsphase erbringen müssen. Daß die *Mitwirkung des Kunden* bei der Informationserhebung und der Validierung eine wichtige Voraussetzung für den Erfolg des Projekts ist und sich durch nichts ersetzen läßt, ist nicht allen Kunden bewußt. Eine in der Abteilung Software Engineering der Universität Stuttgart durchgeführte Untersuchung (Nicklas, 1999) ergab, daß viele Kunden nicht bereit sind, viel Zeit in die Kommunikation oder ganz allgemein in das Projekt zu investieren.

Oftmals tritt der Effekt auf, daß den Kunden ihre eigenen Bedürfnisse nicht bewußt sind oder sie sie so ohne weiteres nicht formulieren können. Viele Kunden sind nicht mit modernen Rechneranlagen vertraut und haben keine klare Vorstellung, was (software-)technisch machbar ist. Hier ist es die Aufgabe der Systemanalytiker, mögliche Lösungsvorstellungen gemeinsam mit den Kunden zu entwikkeln, sich in den Anwendungsbereich einzuarbeiten, den Kunden bei ihren Arbeitsabläufen zuzuschauen (Goguen, 1996, S. 105), mitzuarbeiten oder sich gar als eine Art Lehrling in den Anwendungsbereich einführen zu lassen (Beyer, Holtzblatz, 1995). Requirements Engineering wird oft als *Lernprozeß* aufgefaßt (Jarke et al., 1994), in dem Kunden und Systemanalytiker gemeinsam versuchen, die Probleme zu verstehen und Lösungsmöglichkeiten zu konzipieren.

Bei der Informationserhebung treten nach Scharer (1981, S. 18) oft folgende Effekte auf:

- Der Kunde weiß nicht genau, was er will. Er möchte das jedoch verschleiern und stellt sehr viele, teilweise irrelevante Anforderungen.
- Der Kunde weiß, daß für ihn nur einige Anforderungen wirklich relevant sind. Er gibt jedoch viel mehr an, um über einen Verhandlungsspielraum zu verfügen.
- Der Kunde fordert von dem neuen System, daß es die gleichen Aufgaben wie das alte System erfüllt, nur besser. Er bezieht sich bei seinen Beschreibungen also auf die aktuelle Situation (»Ist-Zustand«).

Probleme bereiten oft auch die unterschiedlichen Erwartungshaltungen, die Systemanalytiker und Kunden an die zu definierenden Anforderungen haben.

Nach Scharer (1981, S. 17f) erwarten Systemanalytiker von Anforderungen,

- daß sie funktional definiert sind, z.B. in Form von Prozessen, Eingaben, Ausgaben und Datenstrukturen,
- daß sie präzise und unmißverständlich sind,
- daß sie vollständig sind, daß also keine weiteren Anforderungen hinzukommen,
- daß sie stabil sind, daß sie sich also nicht mehr ändern, und
- daß ein »gutes«, »elegantes« System realisiert wird.

Dagegen erwarten die Kunden von den Anforderungen,

- daß sie qualitativ definiert werden, daß also die allgemeinen Eigenschaften des Systems beschrieben werden,
- daß sie noch interpretiert werden müssen,
- daß auch nach Abschluß der Requirements-Engineering-Phase weitere Anforderungen eingebracht und bestehende Anforderungen geändert werden können, und
- daß ein lauffähiges System realisiert wird.

Methoden zur Informationserhebung

Im folgenden werden die wichtigsten Methoden zur Erhebung von Informationen kurz vorgestellt. Sie können unterteilt werden in allgemeine und Requirements-Engineering-spezifische Informationserhebungsmethoden.

Zu den allgemeinen Methoden gehören Brainstorming und Interviews. Wenn es darum geht, allgemeine Ideen oder Lösungsvorstellungen zu sammeln, eignet sich *Brainstorming*. Eine typische Brainstorming-Sitzung wird in zwei Phasen unterteilt. In der ersten Phase werden Ideen gesammelt. Die Teilnehmer sollen dabei ihrer Vorstellungskraft freien Lauf lassen. Auch ungewöhnliche oder abstruse Ideen werden notiert. Kritik ist in dieser Phase strengstens untersagt. In der zweiten Phase, die idealerweise einen Tag später stattfindet, so daß die Teilnehmer die Ideen in Ruhe reflektieren können, werden die Ideen bewertet. Die Ideenmenge wird reduziert. Ergebnis ist eine Menge von Ideen und oft noch vagen Anforderungen.

Sind die Vorstellungen des Systemanalytikers schon so weit gereift, daß er konkrete Fragen formulieren kann, können *Interviews* eingesetzt werden. Hierbei handelt es sich wahrscheinlich um die im Requirements Engineering am häufigsten angewandte Informationserhebungsmethode. Wenn der Systemanalytiker den »richtigen« Informanten die »richtigen« Fragen stellt, so erhält er in kurzer Zeit sehr viele und wertvolle Informationen. Das Problem liegt jedoch darin, die richtigen Informanten und die richtigen Fragen zu kennen.

Requirements-Engineering-spezifische Methoden können in folgende Kategorien unterteilt werden (Macaulay, 1996):

- Ansätze, in denen soziale, politische und firmenspezifische Aspekte eine wichtige Rolle spielen: In diese Kategorie fallen *Soft Systems Methodology* (*SSM*), *ETHICS* und *Eason's Approach to User Centered System Design*. Sie alle unterstützen die Definition von Produktkonzepten und die Durchführung von Problemanalysen, Machbarkeitsstudien und Kosten-Nutzen-Analysen. SSM

unterstützt explizit den Umgang mit verschiedenen Sichten und bietet Konfliktlösungsstrategien an, in ETHICS und Easons Ansatz steht das Zusammenspiel zwischen technischen und sozialen Systemen im Vordergrund.

- Gruppensitzungsansätze: Hierzu gehören *Joint Application Design* (*JAD*), ein von IBM entwickeltes Sitzungsformat, um gemeinsam Systeme zu entwerfen, und *Quality Function Deployment* (*QFD*). Sie werden eingesetzt, um die Kommunikation zu verbessern, Informationen zu verbreiten und Entscheidungen zu fällen. QFD ist ein qualitätsorientierter Ansatz, der ursprünglich für die japanische Autoindustrie entwickelt wurde, inzwischen aber auch in der Software-Entwicklung angewandt wird. Der Schwerpunkt von QFD liegt in der Ermittlung von Kundenanforderungen. Es werden einige Standarddokumente vorgegeben, u.a. auch das sog. House of Quality (HoQ): Die von den Kunden geforderten Qualitäten (z.B. »Das Produkt X muß leicht tragbar sein.«) werden in meßbare Teile zerlegt (z.B. Gewicht, Größe). Diese Werte werden in einer Matrix angeordnet. Auf diese Weise wird der Zusammenhang zwischen geforderten Qualitäten und technischen Anforderungen dokumentiert.
- Interaktive Ansätze, in denen Systemanalytiker und Kunden zusammenarbeiten. Im »*designer-as-apprentice*«-Ansatz übernimmt der Kunde die Rolle des Lehrers, der Systemanalytiker die Rolle des Lehrlings, der in den Anwendungsbereich eingeführt wird. Beim »*focus groups*«-Ansatz diskutieren Informanten in Anwesenheit eines Systemanalytikers über für sie relevante Themengebiete. Der Systemanalytiker soll dabei mehr über das Umfeld, in dem das Zielsystem eingesetzt werden soll, lernen. Durch »*future workshops*« soll eine gemeinsame Zukunftsvision geschaffen werden. Zu den interaktiven Ansätzen gehören (u.a.) auch einige Prototyping-Ansätze, z.B. »*cooporate prototyping*«, bei dem Systemanalytiker und Kunden zusammen einen Prototyp entwickeln.

2.3.3 Änderungsmanagement

Viele Requirements-Engineering-Ansätze gehen von einer präzise definierbaren und für die Dauer des Projekts stabilen Problemstellung aus. Diese Annahme ist in der Regel falsch. Jones (1998, S. 12) z.B. berichtet, daß eines der größten Probleme im Requirements Engineering ein kontinuierlicher Strom von Anforderungsänderungen (*requirements creep*) ist. Je nach Systemtyp ändern sich zwischen 1% und 3,5% der Anforderungen pro Monat.

Die Änderungen können folgende Ursachen haben:

- Viele Probleme werden spät erkannt. Erst wenn das System schon zu einem gewissen Grad entworfen oder implementiert ist, fallen bisher nicht berücksichtigte Aspekte auf. Häufig werden Mängel im Pflichtenheft erst bei der Abnahme entdeckt, also zu dem Zeitpunkt, an dem der Kunde das erste Mal mit dem fertigen System konfrontiert wird.
- Software-Projekte können mehrere Jahre dauern. Es kann vorkommen, daß sich die Wünsche der Kunden und sogar die zu lösenden Probleme wegen technologischer oder organisatorischer Änderungen im Unternehmen ändern.

Die wichtigste Konsequenz hieraus ist, daß Änderungen in den Requirements-Engineering-Tätigkeiten berücksichtigt werden müssen. Änderungen sind nicht unbedingt auf Fehler der Systemanalytiker zurückzuführen, sondern sie sind oftmals

unvermeidlich. Sie können in der Problemanalyse-, in der Spezifikationsphase, in den späteren Projektphasen und nach Abschluß des Projekts vorkommen.

Wenn es im weiteren Projektverlauf zu Diskrepanzen zwischen Pflichtenheft und Realisierung kommt, müssen zwei Fälle unterschieden werden:

- Wenn das realisierte System den Anforderungen der Kunden entspricht, dann muß das Pflichtenheft entsprechend angepaßt werden.
- Wenn das realisierte System von den Anforderungen der Kunden abweicht, dann sollte das Pflichtenheft unverändert gelassen werden. Die Abweichungen sollten aber dennoch (in einem eigenen Dokument) beschrieben werden.

Lehman (1980) argumentiert, daß sich nach Fertigstellung eines Systems die Anforderungen fast zwangsläufig ändern müssen: Wenn ein System fertig ist, wird es eingesetzt und verändert dadurch seine Umgebung und damit die Voraussetzungen, unter denen die Anforderungen ursprünglich erhoben wurden.[1]

Die eingesetzte Requirements-Engineering-Methode sollte also ein *Änderungsmanagement* vorsehen. Wenn der Kunde eine (wesentliche) Änderung wünscht, müssen die Auswirkungen der Änderung auf das laufende Projekt untersucht werden (impact analysis) und es müssen Entscheidungen getroffen werden. Eventuell muß sogar mit dem Kunden neu verhandelt werden.

2.3.4 Validierung und Verifikation (V&V)

Mängel, die im Pflichtenheft enthalten sind, haben große (und teure) Auswirkungen auf das Projekt. Angenommen, ein Mangel wird in der Requirements-Engineering-Phase gemacht, aber erst bei der Abnahme entdeckt, dann kann die Behebung des Mangels um bis zu Faktor 100 teurer sein, als wenn der Mangel in der Requirements-Engineering-Phase entdeckt und behoben wurde (Boehm, 1984, S. 75f). Laut Tom DeMarco (zitiert in Tavolato, Vincena, 1984) können ca. 56% aller in einem Projekt gefundenen Mängel auf Mängel des Pflichtenhefts zurückgeführt werden. Es ist also sinnvoll, Pflichtenhefte so früh wie möglich zu prüfen.

Es werden zwei Arten von Prüftechniken unterschieden: *Verifikation* und *Validierung*. In »IEEE Standard Glossary of Software Engineering Terminology« (IEEE, 1990, S. 80f) werden die Begriffe folgendermaßen definiert:

> »Verification: The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.«
>
> »Validation: The process of evaluating a system or component during or at the end of the software development process to determine whether it satisfies specific requirements.«

Da in der Requirements-Engineering-Phase noch kein realisiertes System vorliegt, muß das Pflichtenheft geprüft werden. Durch die Validierung soll geklärt werden, ob das richtige, von den Kunden gewünschte System spezifiziert wurde. Durch die Verifikation soll geklärt werden, ob das Pflichtenheft gegebenen Kriterien genügt.

---

1. Die hier erwähnten Programme sind sog. E-Programme. Es gibt noch S- und P-Programme. Lehman erläutert in seinem Artikel aber, daß bis auf wenige Ausnahmen alle Programme E-Programme sind.

Solche Kriterien sind z.B. Konsistenz und Verständlichkeit (siehe auch Kapitel 2.1.3). Ziel der Validierung und Verifikation ist es, Mängel zu finden, nicht aber, sie zu beheben.

Im folgenden werden die wichtigsten Techniken zur Validierung und Verifikation von Pflichtenheften kurz vorgestellt.

Die einfachste Vorgehensweise, ein Dokument zu prüfen, ist, es zu *lesen* (Boehm, 1984, S. 80). Diese Technik kann ohne große Vorbereitung durchgeführt werden. Trotzdem können viele Mängel gefunden werden. Andererseits ist es sehr wahrscheinlich, daß viele Mängel übersehen werden. Diese Technik eignet sich gut als Vorstufe zur eigentlichen Prüfung, um sicherzustellen, daß das zu prüfende Dokument überhaupt »prüfungswürdig« ist.

Zur Unterstützung der Prüfer können *Checklisten* verwendet werden (Boehm, 1984, S. 80f). Sie geben z.B. vor, welche Inhalte in einem Pflichtenheft enthalten sein sollten oder welche Eigenschaften es erfüllen sollte. Sie basieren oft auf Erfahrungen aus ähnlichen Projekten. Checklisten weisen die Prüfer auf bestimmte Punkte hin, die sie bei der Prüfung beachten sollen. Die Prüfung hängt sehr stark von der Qualität der Checkliste ab. Aspekte, die die Checkliste nicht abdeckt, werden in der Regel auch nicht geprüft.

Einen Schritt weiter geht folgender Ansatz: Statt Checklisten werden *Fehlererkennungsprozeduren*[1] (Porter, Votta, 1994) vorgegeben. Jede beschreibt eine Vorgehensweise, wie Mängel einer ganz spezifischen Art gefunden werden können. Pro Mangelart muß eine Fehlererkennungsprozedur vorhanden sein. Hierbei tritt, ähnlich wie bei Checklisten, das Problem auf, daß Mängel, für die es keine Fehlererkennungsprozedur gibt, in der Regel auch nicht gefunden werden.

Einige Prüfansätze basieren auf der Idee, daß die Prüfer mit dem Pflichtenheft arbeiten. Sie erstellen z.B. Querverweistabellen oder überführen das Pflichtenheft in ein formales Modell. Liegt ein solches Modell vor, kann das (Nicht-) Vorhandensein bestimmter Eigenschaften bewiesen werden. Eventuell kann das Pflichtenheft sogar durch ein Werkzeug *animiert* (ausgeführt) werden. Animation bedeutet, daß der Prüfer mit dem (animierten) Modell interagiert, indem er z.B. Nachrichten verschickt oder den internen Zustand verändert und prüft, wie es darauf reagiert. So kann der Prüfer feststellen, ob das Verhalten des Modells dem gewünschten Verhalten des Zielsystems entspricht. Diese Technik ist effektiv, aber auch sehr aufwendig und schwierig (siehe auch Kapitel 2.4).

Zur Validierung der Interaktion zwischen Zielsystem, Personen und anderen Systemen eignen sich *Szenarien* (Use Cases, siehe z.B. Weidenhaupt et al., 1998). Hierbei steht die Sicht der Benutzer im Vordergrund. Szenarien können auch von Nicht-IT-Fachleuten geprüft werden, wenn sie in einer ihnen verständlichen Notation (z.B. natürliche Sprache) verfaßt sind. Falls das Pflichtenheft keine Szenarien enthält, können sie auch nachträglich erstellt werden. Im Vergleich zur Erstellung formaler Modelle ist der Aufwand deutlich geringer.

Oft ist es sinnvoll, die Kunden frühzeitig mit einem ausführbaren Programm zu konfrontieren, das bestimmte Aspekte des Zielsystems abdeckt. Auf diese Weise erhält der Kunde einen Eindruck vom Zielsystem und kann entsprechend Kritik äußern. Ein solches Programm wird *Prototyp* genannt. Prototypen werden oft zur

1. Fehlererkennungsprozeduren werden in (Porter, Votta. 1994) auch Szenarien genannt.

Validierung der Benutzungsoberfläche eingesetzt. Sie können aber auch zur Prüfung bestimmter Machbarkeitsaspekte, z.B. der Realisierbarkeit zeitkritischer Algorithmen, verwendet werden. Wenn der Prototyp als *Wegwerfprototyp* und nicht als Vorversion des Zielsystems konzipiert wird, kann er schnell und kostengünstig (und entsprechend unsauber) realisiert werden.

Eine der wichtigsten Validierungs- und Verifikationstechniken ist das *Review*, ein Prozeß zur Prüfung von (beliebigen) Dokumenten. Gause und Weinberg (1989, S. 227ff) unterscheiden zwischen formalen und informalen Reviews und zwischen technischen Reviews und Projekt-Reviews. Informale Reviews finden innerhalb der Arbeitsgruppe statt, an formalen Reviews nehmen auch andere Personen teil. Beim *technischen Review* werden Inhalt und Qualität eines Dokuments geprüft, beim Projekt-Review dagegen wird das Projekt geprüft. Im folgenden werden nur technische Reviews betrachtet.

In (Frühauf, Ludewig, Sandmayr, 2000, S. 99ff) wird ein Prozeß für technische Reviews vorgeschlagen. Folgende Dokumente müssen vorhanden sein:

- der *Prüfling* (d.h. das zu prüfende Dokument) und
- *Referenzunterlagen*, die eine Beurteilung des Prüflings zulassen, z.B. Vorgaben, Richtlinien, Fragen- und Aspektkataloge.

Das Review wird von einem *Manager* oder Projektleiter initiiert. Er greift nicht aktiv in den Reviewprozeß ein, trägt aber die Verantwortung für die Durchführung des Reviews und die endgültige Freigabe des Prüflings.

Folgende Personen sind typischerweise an einem Review beteiligt:

- ein *Autor*, der an der Erstellung des Prüflings beteiligt war,
- ein *Moderator*, der das Review organisiert, leitet, für den ordnungsgemäßen Ablauf sorgt und für diesen auch verantwortlich ist,
- vier bis sechs *Gutachter*, die den Prüfling mittels ihres Sachverstands (und gesunden Menschenverstands) anhand der Referenzunterlagen beurteilen, und
- ein *Sekretär*, der das Protokoll führt.

Die eigentliche Reviewtätigkeit wird von dem Moderator geleitet und besteht aus drei Teilschritten:

- Vorbereitung: Die Gutachter prüfen den Prüfling anhand der ihnen zugeteilten Aspekte.
- Durchführung der Review-Sitzung: Die Gutachter tragen ihre Befunde vor. Jeder Befund wird diskutiert, gewichtet und abschließend protokolliert.
- Nachbereitung: Der Autor überarbeitet den Prüfling gemäß der Befunde.

Darüber hinaus gibt es noch die Teilschritte »Planung« und »Analyse«, die lange vor bzw. nach dem eigentlichen Review durchgeführt werden. Sie sollen sicherstellen, daß die benötigten Ressourcen (Zeit, Personen) im Projektplan berücksichtigt werden und daß die Erfahrungen in weitere Entwicklungen, insbesondere in die Verbesserung des Reviewprozesses selbst, einfließen.

In (Gause, Weinberg, 1989, S. 233ff) werden neben dem technischen Review noch folgende, nicht ganz so strenge Review-Typen beschrieben:

- *Vanilla Review*: Hierbei handelt es sich um die unstrukturierteste Review-Variante. Der Ablauf der Sitzung ist im Voraus nicht festgelegt. Er kann je nach Situation angepaßt werden. Diese Flexibilität erfordert einen fähigen Moderator.
- *Inspektion*: Während der Sitzung soll eine (nicht zu umfangreiche) vorher genau festgelegte Menge von Fragen beantwortet werden. Auf diese Weise können auch umfangreiche Dokumente in einer Sitzung geprüft werden.
- *Walkthrough*: Der Prüfling wird von jemandem, der mit ihm vertraut ist, präsentiert. Dieser Review-Typ eignet sich besonders, wenn ein allgemeiner Ansatz überprüft werden soll. Er ist ungeeignet für die Prüfung von Details.
- *Round Robin Review*: Hierbei ist jeder Teilnehmer für den Ablauf eines Teils des Reviews zuständig. Die Zuständigkeiten wechseln zyklisch. Diese Vorgehensweise eignet sich für Situationen, in denen mehrere Teilnehmer mit unterschiedlichen Perspektiven zusammen ein Review durchführen sollen, z.B. Benutzer und Entwickler, die zusammen ein Pflichtenheft prüfen sollen.

Hollocker (1990) schlägt vor, verschiedene Techniken zu kombinieren. Durch einen Walkthrough werden die Kernanforderungen geprüft. Durch ein technisches Review werden die Anforderungen in ihrer Gesamtheit auf Konsistenz und Vollständigkeit geprüft. Durch eine Inspektion wird die Durchführbarkeit geprüft.

2.3.5 Spezifikationsnotationen und -methoden

Einige der in der Requirements-Engineering-Literatur am häufigsten diskutierten Fragen beschäftigen sich mit den *Notationen*, in denen Pflichtenhefte geschrieben werden sollen. Die Fachwelt ist sich hierbei nicht einig.

Eine Möglichkeit, Notationen zu klassifizieren, besteht darin, sie anhand ihres Formalisierungsgrads zu unterscheiden (Tabelle 2, Ludewig, 1997, S. 17-1, Zave 1990, S. 197).

Formalisierungsgrad	Definition	Beispiele
formal (operational, ausführbar)	Syntax und Semantik sind definiert, das Pflichtenheft kann ausgeführt werden	endlicher Automat; Statecharts; Petri-Netz
formal (mathematisch)	Syntax und Semantik sind definiert, das Pflichtenheft ist nicht unbedingt ausführbar	Z; Prädikatenlogik
semi-formal	die Syntax ist definiert, die zugehörige Semantik höchstens teilweise	ER-/Klassendiagramm; Datenflußdiagramm; Sequenzdiagramm
informal	Syntax und Semantik sind nicht oder höchstens teilweise definiert	natürliche Sprache; Use Case; Pseudocode

Tabelle 2: Klassifikation von Spezifikationsnotationen anhand ihres Formalisierungsgrads

Im folgenden werden die wichtigsten Notationen für die Problemanalyse und Spezifikation vorgestellt. Diese Aufzählung ist weit davon entfernt, vollständig zu sein. Es soll jedoch ein Eindruck von den vielfältigen Möglichkeiten zur Formulierung von Anforderungen vermittelt werden.

Natürliche Sprache ist eine der wichtigsten und am häufigsten verwendeten Spezifikationsnotationen. Hierbei handelt es sich um eine informale, textuelle Notation, die nur durch den Wortschatz und die Orthographie- und Grammatikregeln der jeweiligen Sprache (z.B. Deutsch) eingeschränkt wird. Sie ist universell, d.h. sie kann in jedem Anwendungsbereich eingesetzt werden. Mittels natürlicher Sprache kann jeder Sachverhalt beschrieben werden. Einige Sachverhalte wie z.B. der Aufbau eines komplexen Datums lassen sich jedoch nur umständlich durch sie beschreiben.

Um komplexe Datenstrukturen zu beschreiben, werden teils graphische Notationen, z.B. *ER-Diagramme* (Chen, 1977) und *Klassendiagramme* (Coad, Yourdon, 1991), teils textuelle Notationen, z.B. *kontextfreie Grammatiken* oder die dazu sehr ähnlichen *(erweiterten) Backus-Naur-Formen* (*(E)BNF*, Wirth, 1985, S. 8f) verwendet. Mittels ER- und Klassendiagrammen können beliebig aufgebaute Datenstrukturen beschrieben werden. Eine solche Datenstruktur besteht aus Entitäten (Klassen), die über Relationen (Assoziationen), Aggregationen und Vererbungsbeziehungen miteinander verbunden sein können. Assoziationen können Kardinalitäten enthalten, die beschreiben, wieviele Instanzen der einen Entität mit wievielen Instanzen der anderen Entität verbunden sein dürfen oder müssen. Kann eine Datenstruktur eher durch einen Baum oder eine Hierarchie beschrieben werden, wie das oft bei Programmiersprachen der Fall ist, dann eignen sich kontextfreie Grammatiken und (E)BNFs besser.

Sollen die Zustände und die möglichen Zustandsänderungen eines Systems beschrieben werden, so bieten sich *endliche Automaten* bzw. *Zustandsübergangsdiagramme* an. Zwei Zustände gelten als unterschiedlich, wenn es mindestens ein externes, internes oder zeitliches Ereignis gibt, auf das das System unterschiedlich reagiert. Sind die Zustandsräume sehr komplex, werden oft *Statecharts* (Harel, 1987) verwendet, in denen z.B. mehrere Zustände zu einem sog. Superstate zusammengefaßt oder parallele Zustandsdiagramme erstellt werden können. Durch Statecharts können große Zustandsräume übersichtlich dargestellt werden. Die Ausdrucksmächtigkeit ist die gleiche wie bei endlichen Automaten.

Um komplexe Abläufe zu beschreiben, bieten sich *Use Cases* oder *Szenarien* an (Jacobson et al., 1995, S. 156ff), in denen die Interaktionen zwischen einem Zielsystem und seiner Arbeitsumgebung in (strukturierter) natürlicher Sprache beschrieben werden. Der gleiche Aspekt kann auch semi-formal durch *Sequenzdiagramme* bzw. *Interaktionsdiagramme* (Booch, Rumbaugh, Jacobson, 1998) ausgedrückt werden. Sequenzdiagramme werden auch zur Beschreibung systeminterner Abläufe eingesetzt.

Zur Beschreibung der Einbettung eines Systems in seine Arbeitsumgebung können *Kontextdiagramme* verwendet werden (Jackson, 1995, S. 35ff). Ein Kontextdiagramm besteht aus Knoten, die Zielsysteme, Systeme der Arbeitsumgebung und Personen (genauer gesagt deren Rollen) repräsentieren, und Kanten, die die möglichen Kommunikationsverbindungen aufzeigen. Eine spezielle Variante der Kontextdiagramme wird in der Methode SA (siehe unten) eingesetzt. Statt (ungerichteter) Kommunikationsverbindungen werden dort Datenflüsse modelliert.

Sollen Algorithmen beschrieben werden, kann *PDL* (*Program Design Language*, Caine, Gordon, 1975) verwendet werden. Die Abläufe werden durch natürliche Sprache beschrieben, wobei einige Schlüsselwörter (*if, then, else*) eine besondere Bedeutung haben. Eine Alternative zu PDL ist *Pseudocode*. Hängen die Abläufe von sehr vielen Einzelbedingungen und deren möglichen Kombinationen ab, können *Entscheidungstabellen* oder *Entscheidungsbäume* (Chvalovsky, 1983, Moret, 1982) verwendet werden.

Petri-Netze (Peterson, 1977) werden eingesetzt, wenn Synchronisationsfragen eine wichtige Rolle spielen. Sie bestehen aus Stellen und Transitionen. Stellen repräsentieren typischerweise parallel laufende Prozesse. Marken werden mittels der Transitionen aus Stellen entfernt und neue Marken in andere Stellen eingefügt. Die gängige Interpretation eines Petri-Netzes ist, daß nur solche Prozesse aktiv sind, in deren Stellen gerade eine Marke liegt.

Stehen weniger die Abläufe als die Datentransformationen im Vordergrund, können *Datenflußdiagramme* (*DFD*, DeMarco, 1982) verwendet werden. Darin wird das System in Prozesse unterteilt, die Eingabedaten erhalten, verarbeiten, dabei lesend und schreibend auf den internen Zustand zugreifen können und Ausgaben produzieren. DFDs werden insbesondere im kaufmännischen Bereich eingesetzt.

In vielen Requirements-Engineering-Methoden werden einige der »Basis«-Notationen kombiniert, um möglichst viele relevante Aspekte ausdrücken zu können. Meistens steht dabei eine Notation klar im Vordergrund.

In *Structured Analysis* (*SA*, DeMarco, 1982, McMenamin, Palmer, 1984) werden Datenflußdiagramme, EBNFs und Pseudocode kombiniert. In den Echtzeiterweiterungen (Ward, Mellor, 1985, Hatley, Pirbhai, 1987) kommen Zustandsübergangsdiagramme hinzu. Hierbei stehen die Datenflußdiagramme aber ganz klar im Vordergrund. Die EBNFs werden verwendet, um die Datenflüsse und internen Speicher genauer zu beschreiben. Pseudocode wird verwendet, um die Prozesse zu beschreiben.

In der *Unified Modeling Language* (*UML*, Booch, Rumbaugh, Jacobson, 1998) werden sehr viele verschiedene Notationen verwendet: Klassendiagramme, Sequenzdiagramme, Use-Case-Diagramme, Zustandsübergangsdiagramme usw. Wie auch in den meisten anderen objektorientierten Spezifikationsnotationen ist hier die Hauptnotation das Klassendiagramm.

Neben den bisher betrachteten informalen und semi-formalen Notationen gibt es einige formale Notationen, die auf mathematischen Konzepten wie z.B. der allgemeinen Mengenlehre, *temporaler Logik* (Kang, Ko, 1995) oder *Prädikatenlogik* erster Stufe (Zave, Jackson, 1993, S. 382ff) basieren.

Eine der bekanntesten formalen Spezifikationsnotationen ist *Z* (Norris, 1986). Grundlage ist der mathematisch wohldefinierte Begriff der Menge. Aufbauend auf einige als gegeben vorausgesetzte Mengen können weitere Mengen definiert werden. Relationen und Funktionen werden ebenfalls als Mengen aufgefaßt. Der Zustandsraum und das Verhalten des Zielsystems werden durch Schemata beschrieben. Ein Schema faßt Mengen, Vor- und Nachbedingungen und Invarianten zusammen. Wird nur eine bestimmte Teilmenge der umfangreichen Syntax benutzt, können Z-Modelle ausgeführt (animiert) werden.

2.4 Bedeutung informaler Notationen

In der Literatur gibt es einen regelrechten Streit über die Rollen, die informale und formale Notationen im Requirements Engineering einnehmen und einnehmen sollen. Die einzelnen Positionen werden in diesem Kapitel vorgestellt. Viele der folgenden Aussagen über formale Notationen gelten auch für semi-formale Notationen.

Auf der einen Seite stehen die Autoren, für die die natürliche Sprache (als wichtigster Repräsentant der informalen Notationen) eine zentrale Rolle im Requirements Engineering einnimmt und weiterhin einnehmen wird, z.B.:

> »For the most part, the state of the practice is that requirements engineering produces one large document, written in a natural language.« (Hsia, Davis, Kung, 1993, S. 75)

> »Natural language can be ambiguous, surprisingly opaque and is often misunderstood. Nevertheless, everyone can read natural language, so requirements will continue to be written in this way for the foreseeable future.« (Sommerville, Sawyer, 1997, S. 142)

Andere Autoren sagen, daß die Nachteile, die durch die Verwendung natürlicher Sprache entstehen, überwiegen:

> »Informal specifications are often nearly incomprehensible because of their size, ambiguity, incompleteness, and lack of structure. It is also extremely difficult to teach how to write a good specification in English, or to evaluate the result.« (Zave, 1990, S. 199)

Einige Autoren wollen formale und informale Notationen kombinieren, um so das Beste aus den jeweiligen Ansätzen herausholen zu können:

> »We in no way advocate formal specifications as a *replacement* for natural language requirements; rather, we view them as a *complement* to natural language descriptions.« (Meyer, 1985, S. 6)

In den folgenden Abschnitten werden die Vor- und Nachteile formaler und informaler Notationen diskutiert.

Übergang vom Informalen zum Formalen

Durch ein Projekt soll immer ein in der realen (informalen) Welt vorliegendes Problem gelöst werden. Die Lösung des Problems, das Zielsystem, ist immer ein formales System. Irgendwann zwischen Erkennen des Problems und Fertigstellung des Zielsystems muß ein Übergang vom Informalen zum Formalen stattgefunden haben. Die Frage ist, wann dieser Übergang stattfindet oder stattfinden soll.

Eine Möglichkeit besteht darin, ein formales Modell des Zielsystems zu erstellen. Eine andere Möglichkeit besteht darin, diesen Übergang erst in den Realisierungsphasen (Entwurf, Implementierung) zu vollziehen. In den Abbildungen 5 und 6 werden die beiden Vorgehensweisen dargestellt.

Eine wichtige Aufgabe der Problemanalyse besteht darin, das (informale) Problem zu beschreiben. Zur Beschreibung unstrukturierter Probleme ist die natürliche Sprache konkurrenzlos gut geeignet (Ludewig, 1993, S. 289). Es ist sehr schwer, informale Aspekte zu formalisieren (Jackson, 1995, S. 103f). Durch welchen For-

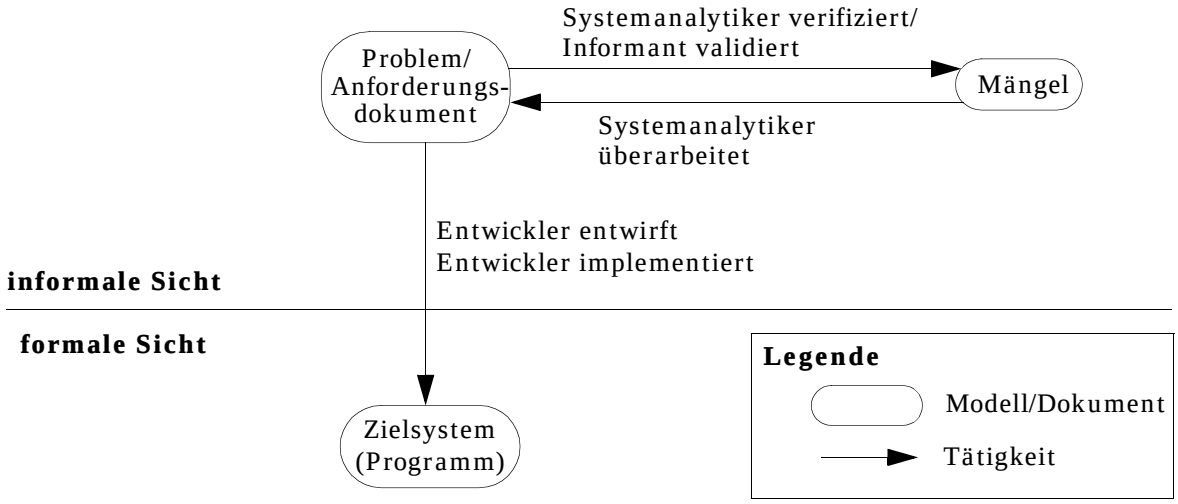


Abbildung 5: Software-Entwicklung ohne formales Modell

malismus lassen sich z.B. »Benutzer« sinnvoll beschreiben? Formale Notationen können nur dann sinnvoll eingesetzt werden, wenn das Problem vollständig verstanden wurde. In den meisten Fällen kann also nicht direkt mit formalen Notationen begonnen werden. Wird ein Konflikt entdeckt, dann muß dieser Konflikt auf der informalen Seite gelöst werden. Das heißt, wird der Konflikt in einem formalen Pflichtenheft entdeckt, muß er implizit oder explizit paraphrasiert werden. Insofern sind informale Notationen für das Requirements Engineering von großer Bedeutung.

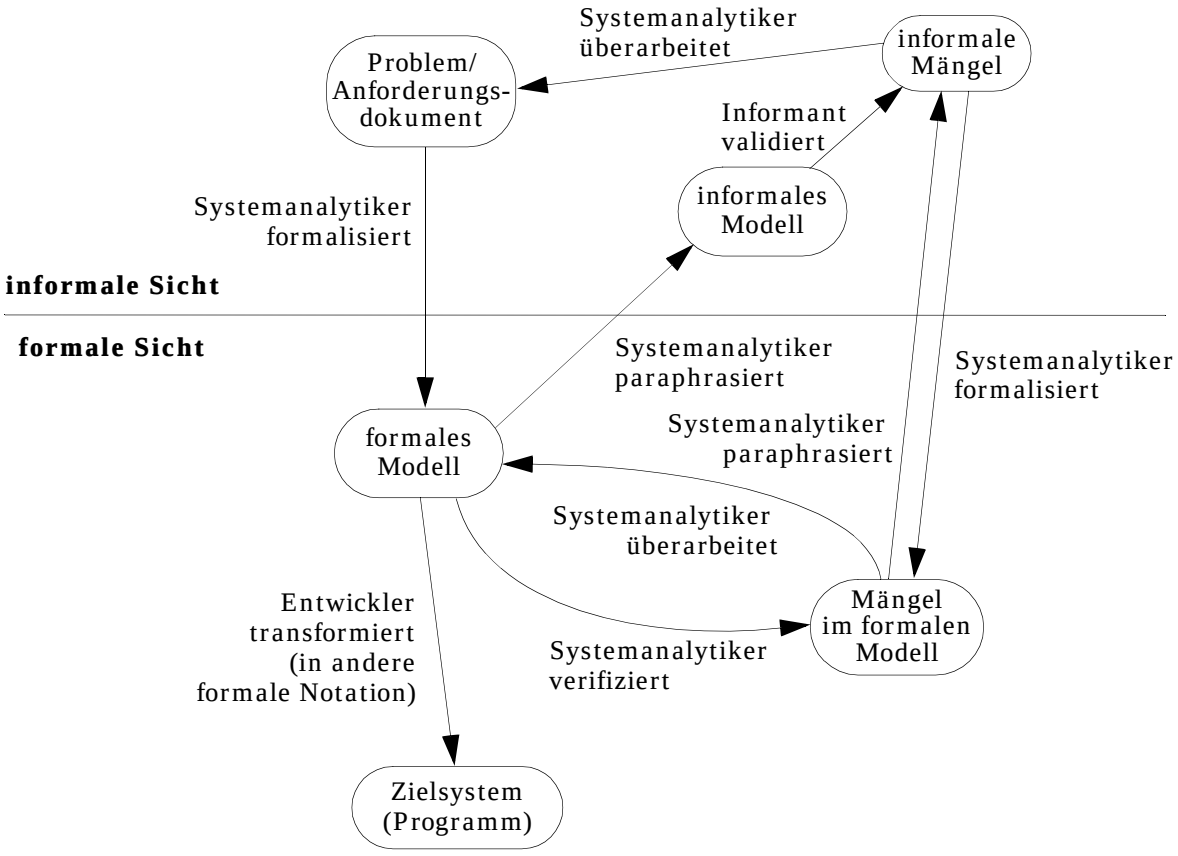


Abbildung 6: Software-Entwicklung mit formalem Modell

Unterschied zwischen Spezifizieren und Programmieren

Einige Autoren stellen sich die Frage, inwieweit sich formales Spezifizieren und Entwerfen bzw. Programmieren unterscheiden. Ist es sinnvoll, in einem Projekt verschiedene Formalismen zu verwenden, also zum einen die formale Spezifikationsnotation, zum anderen die formale Programmiersprache (LeCharlier, Flener, 1998, S. 291)? Faulk (1997, S. 94f) sieht zudem bei operationalen formalen Notationen die Gefahr, daß das Pflichtenheft ein (sehr abstraktes) Programm ist. Trifft das zu, so wäre das Pflichtenheft Teil der Realisierung des Zielsystems und würde seinen eigentlichen Zweck nicht erfüllen.

Inkonsistenzen

Der Einsatz einer formalen Notation setzt voraus, daß die Systemanalytiker eine präzise Vorstellung des Zielsystems haben, denn sonst könnten sie es nicht durch den Formalismus beschreiben. Diese präzise Vorstellung ergibt sich in der Regel erst im Laufe des Projekts. Requirements Engineering ist ein Prozeß, in dem Systemanalytiker und Informanten voneinander lernen, sich gegenseitig beeinflussen und miteinander verhandeln, bis sie eine gemeinsame Sicht entwickelt haben. Dabei müssen Inkonsistenzen erlaubt sein und dokumentiert werden können. Nur wenige formale Notationen bieten diese Möglichkeit (Balzer, 1991).

Kosten

Laut Sommerville und Sawyer (1997, S. 348) steigt die Dauer der Requirements-Engineering-Phase um 10 - 20%[1], wenn formale Notationen eingesetzt werden, im Vergleich zur ausschließlichen Verwendung informaler Notationen. Die Befürworter formaler Methoden argumentieren, daß der Mehraufwand in den früheren Phasen sich kostenreduzierend auf das Gesamtprojekt (inkl. Wartung) auswirkt (Hall, 1990, S. 17), insbesondere weil Mängel früher gefunden werden. Empirische Untersuchungen, die die Kostenreduktionen in industriellen Projekten belegen, sind mir nicht bekannt.

Die Einführung von formalen Notationen in ein Unternehmen ist sehr teuer, weil sie in der Regel mit einem umfangreichen Schulungsprogramm und evtl. der Einstellung von hochbezahlten Notationsspezialisten verbunden ist. Es müssen nicht nur die Personen geschult werden, die die Pflichtenhefte erstellen sollen, sondern auch die Leser. Der Leserkreis eines Pflichtenhefts kann sehr groß sein und Personen umfassen, die nicht zu dem Unternehmen gehören, die also nicht unbedingt geschult werden können oder wollen (Sommerville, Sawyer, 1997, S. 346).

Eindeutigkeit

In (Gause, Weinberg, 1989, S. 94ff) wird ein Beispiel angegeben, in dem gezeigt wird, daß Mißverständnisse und Mehrdeutigkeiten in vermeintlich einfachen natürlichsprachlichen Aussagen enthalten sein können.

Man betrachte folgenden einfachen, aus einem bekannten Kinderlied entnommenen Satz: »Mary had a little lamb«. Auf den ersten Blick scheint der Satz unmiß-

---

1. Diese Schätzung beruht auf Erfahrungen der Autoren. Konkrete Angaben, wie sie die Zahlen erhoben oder validiert haben, werden nicht gemacht.

verständlich zu sein. Gause und Weinberg versuchten mittels folgender Techniken, mögliche unterschiedliche Bedeutungen herauszufinden:

- Technik 1: Durch Betonung einzelner Wörter kann der Sinn des Satzes variieren. Betont man das Wort »had«, so bedeutet der Satz wohl eher, daß Mary ein Lamm hatte, aber nicht mehr hat. Betont man statt dessen »little«, so könnte der Satz bedeuten, daß das Lamm überraschend klein ist.
- Technik 2: Die einzelnen Wörter werden im Lexikon nachgeschlagen. Alle möglichen Bedeutungen werden miteinander kombiniert. Das Wort »have« hat in einigen Fällen die Bedeutung »have dinner«. Damit könnte der Satz auch bedeuten, daß Mary das Lamm gegessen hat.

Problematisch hierbei ist, daß die Mehrdeutigkeiten nicht durch unbekannte Wörter verursacht werden, sondern im ersten Fall durch die Art und Weise, wie ein Satz gelesen wird, und im zweiten Fall durch eine eher selten verwendete Bedeutung des Worts »have«. Es ist ziemlich unwahrscheinlich, daß das Wort »have« in dem Glossar eines Pflichtenhefts definiert wird.

Die o.g. Beispiele sind speziell konstruiert, um das Problem der Mehrdeutigkeiten möglichst gut zu verdeutlichen. Es gibt aber auch reale Fälle, in denen geläufige Wörter mit spezieller Bedeutung verwendet werden. Der Referent eines von mir besuchten Vortrags, der über seine Tätigkeiten in einem Rechenzentrum berichtete, benutzte regelmäßig das Wort »drucken«. Erst gegen Ende des Vortrags stellte sich heraus, daß er mit »drucken« eigentlich »formatieren« meinte. Somit konnte auch ohne Drucker »gedruckt« werden, z.B. in eine Datei oder auf den Bildschirm.

In einer formalen Notation sind die Syntax und Semantik definiert. Demzufolge hat jeder in der formalen Notation formulierte Sachverhalt eine klare Bedeutung. Dennoch sind Mehrdeutigkeiten nicht ganz auszuschließen, weil die Grundelemente der formalen Notation mit Sachverhalten der realen Welt in Beziehung gesetzt (d.h. definiert) werden müssen. Ein Beispiel hierfür sind die vorgegebenen Mengen in Z. Die Definition erfolgt in der Regel durch natürliche Sprache und kann demzufolge Mehrdeutigkeiten enthalten.

Verständlichkeit und Validierung

An den Requirements-Engineering-Tätigkeiten sind nicht nur Systemanalytiker und Entwickler beteiligt, die evtl. durch ihre Ausbildung in formalen Notationen trainiert sind, sondern auch Informanten aus anderen Fachbereichen. Letztere sind in der Regel nicht in diesen Notationen ausgebildet und können sie somit nicht direkt verstehen (Ludewig, Glinz, Matheis, 1985, S. 194). Auch viele semiformale Notationen wie z.B. SA oder UML sind nicht intuitiv verständlich (Herzog-Tabar, 1996, S. 47).

Zur Validierung eines Pflichtenhefts ist es aber unbedingt erforderlich, daß es die Informanten verstehen. Ist das nicht möglich, müssen die Systemanalytiker andere Lösungen finden: Sie können das Pflichtenheft z.B. in eine für die Informanten verständliche Notation paraphrasieren oder animieren (Hall, S. 18). Das ist teuer, und es muß sichergestellt werden, daß der paraphrasierte Text oder der Prototyp mit dem Pflichtenheft konsistent ist (Sommerville, Sawyer, 1997, S. 285).

Einige Autoren behaupten, daß die den formalen Notationen zugrundeliegenden mathematischen oder prädikatenlogischen Kalküle nicht sehr komplex seien

(Hall, 1990, S. 16), also leicht erlernbar und nicht sehr schwer zu verstehen seien (Meyer, 1985, S. 24). Finney (1996) hat jedoch in einem informalen Experiment gezeigt, daß selbst viele Informatikstudenten mit Vorkenntnissen in formalen Notationen nicht in der Lage sind, ein einfaches Z-Modell zu lesen und zu verstehen. Insofern kann das von Personen aus anderen Fachbereichen auch nicht erwartet werden.

Parnas (in Bowen et al. 1996, S. 28) meint, daß die vorhandenen formalen Methoden umständlich und von ihrer Syntax her sehr schwer zu verstehen sind, so daß sie für den praktischen Einsatz nicht angemessen sind.

Hofstadter (1991, S. 53ff) bietet noch ein weiteres Argument für die Unverständlichkeit formaler Notationen. Ein formales Modell an sich hat noch keine Bedeutung. Erst dadurch, daß eine Isomorphie zwischen Konzepten der Wirklichkeit und Konzepten des formalen Modells identifiziert wird, erhält es eine Bedeutung. Hiermit sind zwei Schwierigkeiten verbunden:

- Die Isomorphie ist nicht immer offensichtlich. Sie muß erst vom Leser erkannt werden. Da die Validierung aber kein Detektivspiel sein soll, muß der Verfasser eines formalen Modells die Isomorphie durch (informale) Kommentare erklären.
- Der Verfasser muß aufpassen, daß wirklich eine Isomorphie zwischen realer Welt und formalem Modell vorliegt. Nur durch die Isomorphieeigenschaft ist sichergestellt, daß Schlüsse, die auf Grund des formalen Modells gezogen werden, in die reale Welt übertragen werden können.

Hierzu ein Beispiel: In der Mathematik gilt 1+1=2. Dieses Konzept läßt sich jedoch nicht auf beliebige Sachverhalte der realen Welt übertragen. Wenn z.B. zwei Wassertropfen zusammenfließen, entsteht ein Wassertropfen. Er hat zwar ein größeres Volumen, es ist aber dennoch ein Wassertropfen. Die Gleichung 1 Wassertropfen + 1 Wassertropfen = 2 Wassertropfen ist also im Allgemeinen falsch. Die Abbildung zwischen Wassertropfen und Zahlen ist kein Isomorphismus.

Abbildung 7 enthält ein Beispiel für eine in einer formalen Notation geschriebene, algebraische Spezifikation (entnommen aus Nagl, 1992), die formal korrekt und absolut präzise die Datenstruktur Schlange (Queue) beschreibt. Sie ist aber völlig unverständlich. Erst durch die Verwendung intuitiver Bezeichner kann der Leser erfassen, was gemeint ist (siehe Abbildung 8). Dieses Beispiel zeigt, daß formale Modelle erklärt werden müssen. Ansonsten sind selbst an sich einfache formale Modelle nicht verständlich.

Aber auch natürlichsprachliche Pflichtenhefte sind nicht immer einfach zu verstehen. So können Sätze beliebig kompliziert und verschachtelt sein. Zusammengehörende Informationen können über weite Teile des Pflichtenhefts verstreut sein, ohne daß der Zusammenhang erkennbar wäre. Erschwert wird das durch einen Effekt, der von Meyer (1985, S. 7, 9f) »Rauschen« genannt wird: Viele Aussagen innerhalb eines Textes haben keinerlei Bedeutung, sie blähen den Text nur unnötig auf und verringern dadurch die Verständlichkeit.

Über den Punkt Knappheit gibt es unterschiedliche Meinungen. So behauptet Meyer (1985, S. 24), daß ein informales Pflichtenheft deutlich umfangreicher ist als ein inhaltsgleiches formales Pflichtenheft. LeCharlier und Flener (1998, S. 242) argumentieren dagegen, daß formale Modelle an sich unverständlich sind (siehe oben) und daß sie deswegen durch informale Erklärungen ergänzt werden müssen. Rechnet man die Erklärungen zum Umfang dazu, wird er insgesamt größer.

```
type MYSTERY
    functions       delete:      {} → MYSTERY
                    follow:      MYSTERY × CHAR → MYSTERY
                    say:         MYSTERY → MYSTERY
                    make:        MYSTERY → CHAR
                    is_cold:     MYSTERY → BOOL
                    destroy:     MYSTERY × MYSTERY → MYSTERY
    axioms for m, n ∈ MYSTERY, c ∈ CHAR let
                    is_cold(delete) = true
                    is_cold(follow(m,c)) = false
                    say(delete) = delete
                    say(follow(m,c)) =
                            if is_cold(m) then delete
                                          else follow(say(m),c)
                    make(delete) = undefined
                    make(follow(m,c)) =
                            if is_cold(m) then c else make(m)
                    destroy(m,delete) = m
                    destroy(m,follow(n,c)) =
                                follow(destroy(m,n),c)
end type MYSTERY
```

Abbildung 7: Eine unverständliche algebraische Spezifikation

Technologietransfer von formalen Notationen

Auch wenn es einige Firmen gibt, in denen erfolgreich formale Notationen eingesetzt werden (Craigen, Gerhart, 1995), so sind diese Firmen doch in der Minderheit.

Parnas (1998) identifiziert als ein Problem des Technologietransfers, daß ein sehr großer Anteil selbst der Fachkräfte nicht im Umgang mit formalen Spezifikationsnotationen geschult ist. Die Informatik-Fachwelt ist sich nicht einig darüber, ob und in welchem Umfang formale Spezifikationsnotationen im Studium gelehrt werden sollen.

Ein weiteres Problem ist der negative Ruf formaler Notationen. Es kann Jahre dauern, bis sie von den Mitarbeitern eines Unternehmens wirklich akzeptiert werden (Sommerville, Sawyer, 1997, S. 285), zumal viel Praxis und Erfahrung vorhanden sein müssen, bevor sie produktiv eingesetzt werden können. In vielen

```
type QUEUE
    functions       empty_queue: {} →QUEUE
                    enqueue:     QUEUE × CHAR → QUEUE
                    dequeue:     QUEUE → QUEUE
                    front:       QUEUE → CHAR
                    is_empty:    QUEUE → BOOL
                    concat:      QUEUE × QUEUE → QUEUE
    ...
end type QUEUE
```

Abbildung 8: Die gleiche algebraische Spezifikation (nur functions), diesmal mit verständlicheren Bezeichnern

Unternehmen gibt es Zweifel, ob die Notationen und zugehörigen Werkzeuge die notwendige Reife für große Projekte haben (Gaudel, 1994, S. 225), oder ob sie für ihre Projekte prinzipiell nützlich sind (Melchisedech, 1998, S. 97). Was fehlt, sind für den jeweiligen Anwendungsbereich angemessene Werkzeuge und praxisrelevante Beispiele (Holloway, in Bowen et al., 1996, S. 25).

Formale Notation als »Best Practice«

Formale Notationen werden weitgehend als »best practice« angesehen. Einer Firma, in der formale Notationen eingesetzt werden, wird zugetraut, daß sie den Software-Entwicklungsprozeß beherrscht. Das kann sich positiv auswirken, wenn sie auf Ausschreibungen reagiert, wenn sie wegen Produktversagen verklagt wird oder wenn sie ihre Produkte oder Prozesse zertifizieren lassen möchte (Sommerville, Sawyer, 1997, S. 282). Dem negativen Ruf, den die formalen Notationen bei den Personen haben, die sie anwenden sollen, steht ein positiver Ruf bei den Personen gegenüber, die Aufträge vergeben oder Unternehmen beurteilen wollen.

(Automatische) Verifikation

Ein wichtiger Vorteil formaler Notationen ist, daß ein formales Pflichtenheft mittels mathematischer Methoden und evtl. sogar automatisch auf bestimmte Aspekte der Vollständigkeit und Konsistenz geprüft werden kann (Sommerville, Sawyer, 1997, S. 281). Bestimmte Mängel können so sehr früh und ohne großen Aufwand gefunden werden. Es können aber nicht alle Mängel gefunden werden: Es kann z.B. nicht festgestellt werden, ob das spezifizierte System den Wünschen der Informanten entspricht oder ob alle Wünsche der Informanten berücksichtigt wurden (LeCharlier, Flener, 1998, S. 292).

Ein weiterer Vorteil ist, daß (zumindest prinzipiell) bewiesen werden kann, daß eine Implementierung das gegebene formale Pflichtenheft in allen Punkten erfüllt. Ein Problem hierbei ist, daß die Beweise praktisch nicht immer durchführbar sind (LeCharlier, Flener, 1998, S. 291) und daß die Beweise, wenn sie manuell durchgeführt werden, fehlerhaft sein können (Hall, 1990, S. 13).

Auch wenn die automatische Verifikation nicht alle Mängel findet oder nicht immer durchführbar ist, ergeben sich durch die Formalisierung einige Vorteile:

- Der Systemanalytiker muß die erhobenen (informalen) Informationen sehr genau analysieren. Allein dadurch wird er wahrscheinlich viele Mängel finden (Sommerville, Sawyer, 1997, S. 281).
- Hat der Systemanalytiker beim Formulieren natürlichsprachlicher Aussagen eine spätere Formalisierung im Hinterkopf, versucht er (wahrscheinlich), typische Problemfälle von vorneherein zu vermeiden. Oft reicht auch allein die Kenntnis formaler Notationen aus, um diesen Effekt zu erzielen (Gehani, 1982, S. 433).

Ausdrucksmächtigkeit

In der Requirements-Engineering-Phase werden Informationen vieler unterschiedlicher Arten erhoben. Einige davon lassen sich nur schwer oder umständlich durch formale oder semi-formale Notationen ausdrücken, z.B. Einleitungen, Zu-

sammenfassungen, Begriffsdefinitionen, Erklärungen, Begründungen, Beispiele, bestimmte nicht-funktionale Anforderungen oder Benutzerprofile.

Formale Notationen eignen sich gut, um das Verhalten und den Zustandsraum eines Systems und bestimmte nicht-funktionale Anforderungen (z.B. zeitliche Anforderungen) zu beschreiben.

Theoretisch wäre eine formale Notation denkbar, durch die alle Informationen beschrieben werden können. So könnte das Benutzerprofil für »Kunde« und »Systemadministrator« durch Prädikatenlogik folgendermaßen formuliert werden:

normale_Computer_Kenntnisse (Kunde)

geringes_anwendungsspezifisches_Wissen (Kunde)

großes_anwendungsspezifisches_Wissen (Systemadministrator)

Für jede denkbare Eigenschaft, die ein Benutzer des Zielsystems aufweisen kann, müßte es ein entsprechendes Prädikat geben. Da es nicht möglich ist, alle denkbaren Eigenschaften im Voraus zu bestimmen, muß die Möglichkeit bestehen, neue Prädikate zu definieren. Die Definition kann entweder durch logische Kombination vorhandener Prädikate oder natürlichsprachlich erfolgen.

Auch wenn der obige formale Ansatz vollständig durchgeführt würde, scheint es mir sehr zweifelhaft, ob viel gewonnen wäre. Die Prädikate können zwar in Formeln verwendet werden und es können prädikatenlogische Auswertungs- und Prüfverfahren automatisch ausgeführt werden. Die Bedeutung der Prädikate bleibt jedoch für ein automatisches Auswertungssystem verborgen. Die Prüfungen können nur dann sinnvolle Ergebnisse liefern, wenn entsprechende Regeln vorhanden sind. So könnte z.B. automatisch geprüft werden, ob gleichzeitig die Prädikate *normale_Computer_Kenntnisse (X)* und *keine_Computer_Kenntnisse (X)* erfüllt sind. Dafür müßte das Auswertungssystem aber »wissen«, daß sich die Prädikate widersprechen.

Schlußfolgerung

Trotz der Probleme im Umgang mit natürlicher Sprache (siehe auch Kapitel 1.2) wird sie meiner Meinung nach in absehbarer Zukunft eine zentrale Rolle im Requirements Engineering spielen. Hauptgründe hierfür sind,

- daß viele Informationen, die in Pflichtenheften vorhanden sein sollten, (sinnvoll) nur durch natürliche Sprache beschrieben werden können,
- daß der Requirements-Engineering-Prozeß nicht mit formalen Notationen beginnen kann, weil einige Schritte auf jeden Fall mit informalen Notationen durchgeführt werden müssen, und
- daß der Ausbildungsstand der Projektmitarbeiter in der Industrie und der Fachleute aus anderen Anwendungsbereichen, aber auch der von IT-Fachleuten, derart ist, daß sich formale Notationen nicht in großem Maßstab durchsetzen können.

In dieser Arbeit werden einige Überlegungen angestellt, wie die Qualität von im wesentlichen auf natürlicher Sprache basierenden Pflichtenheften verbessert werden kann.

Kapitel 3

Stand der Praxis – eine Untersuchung

In diesem Kapitel werden die Ergebnisse einer Untersuchung von industriellen Projekten vorgestellt (siehe Melchisedech, 1998). Ziel der Untersuchung war es, einen Einblick in den *Stand der Requirements-Engineering-Praxis* zu erhalten. Gegenstand der Untersuchung waren die Requirements-Engineering-Dokumente, die beteiligten Personen bzw. deren Rollen und der Requirements-Engineering-Prozeß von drei Projekten in drei Unternehmen. Die Dokumente wurden auf Aufbau, Inhalt und verwendete Notationen hin untersucht. Bei der Untersuchung der Prozesse standen der Werkzeugeinsatz, die Kommunikation mit den Kunden und die Prüfung der Dokumente im Vordergrund. Soweit wie möglich wurden die Informationen quantitativ erhoben.

3.1 Untersuchungsgegenstand

Im folgenden werden die Projekte und die Zielsysteme kurz charakterisiert. Aus Vertraulichkeitsgründen können in diesem Rahmen weder die Unternehmen genannt noch die Projekte im Detail beschrieben werden. Die Projekte werden im folgenden als Projekt A, B und C bezeichnet.

3.1.1 Projektkenndaten

Im Projekt A wurde eine Cross-Plattform-Entwicklungsumgebung für eingebettete Software realisiert. Das Entwicklungssystem läuft auf IBM PCs unter Windows. Bei der Ablaufumgebung, in der die von dem System erzeugte Software ausgeführt wird, handelt es sich um ein Spezialbetriebssystem für Echtzeitanwendungen. Die Benutzerinteraktion erfolgt über eine menügesteuerte, textuelle Schnittstelle. Mit dem Entwicklungssystem können Programme geschrieben werden, die einem vorgegebenen, auf Wiederverwendbarkeit ausgerichteten Programmierparadigma folgen. Dieses wurde im Rahmen einer Vorstudie entwickelt und stand zu Projektbeginn fest. Das Projekt wurde inzwischen abgeschlossen. Das Zielsystem wird von den Kunden eingesetzt.

Im Projekt B wurde ein graphischer Editor für Netzwerktopologien entwickelt. Ein Netzwerk besteht aus Symbolen und Verbindungen zwischen diesen Symbolen. Jedes Symbol kann durch ein Netzwerk verfeinert werden. Die Anwendung soll an eine bereits vorhandene Datenbank angebunden werden. Ein altes System, das bisher die Aufgaben erfüllt hat, soll durch das neue System teilweise ersetzt werden. Das Projekt wurde inzwischen abgeschlossen. Das entstandene System wird in Kundenprojekten zu Evaluierungszwecken eingesetzt.

Projektkenndaten	Projekt A (Entw.-System)	Projekt B (graph. Front-End)	Project C (Online-System)
Dauer des Projekts	26 Monate	29 Monate	(noch nicht beendet)
Aufwand des Projekts	48 Mann-Monate	155 Mann-Monate	(noch nicht beendet)
Anzahl Projektmitarbeiter	2-7	8-13	2-3
Dauer der RE-Phase	5 Monate	4 Monate	12 Monate
Aufwand der RE-Phase	13 Mann-Monate	2,5 Mann-Monate	8 Mann-Monate
Anzahl Systemanalytiker	2-3	2	1
Finanzierung	Festpreisprojekt	Eigenfinanzierung	Eigenfinanzierung
Prototyp erstellt?	GUI-Prototyp	Vorversion	Prototyp
Wartung	Kunde	Entwickler	Entwickler

Tabelle 3: Vergleich der Projektkenndaten der untersuchten Projekte

Im Projekt C wird ein System entwickelt, das im Bankenbereich eingesetzt wird. Angestellte sollen von ihrem Terminal aus Anfragen an eine zentrale Datenbank eines Drittanbieters durchführen können. Die Anfragen sollen »online« erfolgen. Innerhalb weniger Sekunden sollen die Antworten verfügbar sein. Das System soll in die schon vorhandene Arbeitsumgebung der Bankangestellten eingebunden werden. Die Anfragen, die bei dem Drittanbieter bisher manuell durchgeführt wurden, sollen durch das System automatisiert werden. Das Projekt wurde noch nicht beendet, die Spezifikationsphase ist jedoch schon abgeschlossen.

In Tabelle 3 werden die wichtigsten Projektkenndaten aufgelistet.

3.1.2 Systemkenndaten

Um besser abschätzen zu können, inwieweit die untersuchten Systeme Ähnlichkeiten aufweisen, wurden sie anhand der in Tabelle 4 angegebenen Kriterien klassifiziert.

Für das Kriterium Datenkomplexität muß die Komplexität der einzelnen Daten geschätzt werden. Es macht durchaus einen Unterschied, ob es sich bei den Daten um viele, einfach strukturierte Einzeldaten oder um einige wenige, dafür aber sehr komplex aufgebaute Daten handelt.

Unter *Metadaten* werden Daten zur Beschreibung von Daten verstanden. Ein System mit einer hohen Metadatenkomplexität verfügt über eine *Datenbeschreibungssprache,* durch die Datenstrukturen definiert werden können. Bei einem System mit niedriger Metadatenkomplexität sind die Datentypen vorgegeben und können nur eingeschränkt angepaßt werden.

Aufgrund der vorliegenden Dokumente wurde geschätzt, inwieweit die Kriterien auf die einzelnen Projekte zutreffen. Hierbei handelt es sich nicht um absolute Angaben. Die Angaben dienen lediglich dazu, die Systeme untereinander vergleichen zu können. In Tabelle 4 werden die Kenndaten der drei Systeme gegenübergestellt.

Gemeinsam ist allen Systemen, daß sie keine harten Zeitanforderungen[1] haben. Es handelt sich also nicht um Echtzeitsysteme.

Systemkenndaten	Projekt A (Entw.-System)	Projekt B (graph. Front-End)	Project C (Online-System)
Datenkomplexität	hoch	hoch	niedrig
Metadatenkomplexität	hoch	niedrig	niedrig
Datenmenge	niedrig	niedrig	mittel
Anzahl Funktionen	mittel	mittel	niedrig
Komplexität der Algorithmen	hoch	mittel	mittel
Komplexität der Benutzungsschnittstellen	mittel	hoch	niedrig
Komplexität der Systemschnittstellen	mittel	mittel	hoch
Zeitanforderungen	nicht hart	nicht hart	nicht hart
Anzahl gleichzeitiger Benutzer	viele	viele	sehr viele

Tabelle 4: Vergleich der Systemkenndaten der untersuchten Projekte

Das Entwicklungssystem (Projekt A) und das graphische Front-End (Projekt B) weisen einige Ähnlichkeiten auf. Die wesentlichen Unterschiede liegen darin, daß es im Entwicklungssystem eine komplexe Datenbeschreibungssprache gibt und daß die Algorithmen komplexer sind. Im graphischen Front-End hingegen ist die Benutzungsschnittstelle komplexer. Das Online-System (Projekt C) unterscheidet sich in fast allen Kriterien von den anderen beiden Systemen.

3.2 Untersuchungsergebnisse - Dokumente

In diesem und den folgenden zwei Unterkapiteln werden die Untersuchungsergebnisse beschrieben: Dokumente in Kapitel 3.2, Rollen der beteiligten Personen in Kapitel 3.3 und Prozesse in Kapitel 3.4.

3.2.1 Aufbau der Dokumente

In Projekt A wurden zwei Requirements-Engineering-Dokumente erstellt: ein *Lastenheft* und ein *Pflichtenheft*. Ihr Aufbau wird in Tabelle 5 beschrieben.

Hierbei ist bemerkenswert, daß in beiden Dokumenten die gleichen Kapitel vorkommen. Fast alle Kapitel sind im Pflichtenheft ausführlicher als im Lastenheft. Bei dem Lastenheft scheint es sich um eine Vorversion des Pflichtenhefts zu handeln. Ausnahmen sind das Kapitel »Gesamtübersicht« im Pflichtenheft, das nur Verweise auf das gleiche Kapitel im Lastenheft enthält, und das Kapitel »Datenmodell«, das nur im Lastenheft auftaucht. Im Pflichtenheft wird das Datenmodell durch die Angabe von Dateiformaten präzisiert. Das Datenmodell wird also aus zwei verschiedenen Blickwinkeln betrachtet: Im Lastenheft werden die Ideen und der prinzipielle Aufbau beschrieben, im Pflichtenheft die zugehörigen Dateischnittstellen.

1. Bei Projekt A beziehen sich die Zeitanforderungen auf das Entwicklungssystem, nicht auf die damit zu entwickelnde Software. Für letztere gelten in der Regel harte Zeitanforderungen.

Kapitel	Seiten (LH)	Seiten (PH)
Allgemeines (Abkürzungen, Definitionen, Literatur)	3	4
Projektkenndaten (Probleme, Lieferumfang, geplante Versionen)	2	2
Gesamtübersicht (Produktstruktur)	3	1
Vorgaben und Randbedingungen (Entwurfsvorgaben, nicht-funktionale Anforderungen, QS- und CM-Plan)	3	8
Beschreibung des Datenmodells (Metadaten)	11	-
Funktionsbeschreibung der Werkzeuge	16	40
Schnittstellen (Datenbank, Datei, Verzeichnis, Betriebssystem)	1	59
Benutzungsschnittstellen	1	43
Projektanforderungen	1	1

Tabelle 5: Struktur von Lastenheft (LH) und Pflichtenheft (PH) des Entwicklungssystems

In Projekt B (graphisches Front-End) wurden ebenfalls zwei Requirements-Engineering-Dokumente erstellt: eine *Requirements Specification* und eine *Functional Specification*.

Dokument	Kapitel	Seiten
Requirements Specification	Einleitung (Zweck, Leserkreis, Struktur des Dokuments, Referenzen auf andere Dokumente, Vereinbarungen/Abkürzungen)	2
	Ziele (Probleme, Kernfunktionen)	3
	Anforderungen (funktionale Anforderungen, Leistungsanforderungen, Entwurfseinschränkungen, Produktions- und Wartungsaspekte, sonstige Einschränkungen)	19
	Kostenschätzung	2
	Projektplanung	1
	Anhänge (Anforderungen an zukünftige Versionen, Masken für die Benutzungsschnittstelle, Aufwand für typische Produktanpassungen)	12
Functional Specification	Einleitung (Zweck, Leserkreis, Struktur des Dokuments, Referenzen auf andere Dokumente)	2
	Einführung (Überblick, allg. Eigenschaften/Einschränkungen)	3
	Benutzerinteraktionen (Szenarien)	24
	Datenmodell (Klassenbeschreibungen)	32
	Leistungsanforderungen	1
	Systemattribute (Verfügbarkeit, Wartbarkeit)	1
	Anhänge (Attribute der Klassen des Datenmodells)	25

Tabelle 6: Struktur der Requirements Specification und der Functional Specification des graphischen Front-Ends

Sie beschreiben das System aus unterschiedlichen Blickwinkeln. Die Requirements Specification enthält die Anforderungen aus Benutzersicht. In der Functional Specification wurde ein Modell des Systems erstellt; die Funktionalität wird durch Szenarien beschrieben, das Datenmodell durch Klassen. Der Aufbau der Dokumente wird in Tabelle 6 beschrieben.

Im Gegensatz zu den beiden anderen Projekten, bei denen je ein Dokument als Ergebnis der Problemanalyse- und Spezifikationsphase erstellt wurde, sind in Projekt C sehr viele kurze Requirements-Engineering-Dokumente entstanden. Jedes Dokument beschreibt (hauptsächlich) einen Aspekt. In diesen Dokumenten wird sehr viel Wert auf die Beschreibung der Arbeitsumgebung und der Einbettung des Systems in die Organisation gelegt. Es werden verschiedene Lösungsideen betrachtet und es wird untersucht, wie gut diese Lösungsideen zu den Zielen passen. Neben technischen Zielen werden auch finanzielle und soziale Ziele berücksichtigt. Der Ist-Zustand wird beschrieben (sowohl die technische Ausstattung als auch die bisherige Lösung des Problems). Für den Ist-Zustand und alle Lösungsalternativen werden Kosten-/Nutzenaufstellungen angegeben.

Querverweise zwischen den einzelnen Dokumenten wurden teilweise durch Matrizen hergestellt. So gibt es z.B. eine Matrix, in der Ziele und Lösungen einander gegenübergestellt werden. Zu jeder Lösung wurde angegeben, welche der Ziele durch diese Lösung erreicht werden.

In Tabelle 7 werden die erzeugten Dokumente aufgelistet.

Phase	Dokument	Seiten
Problemanalyse	Betroffene Organisationseinheiten	1
	Ziele	3
	Probleme	1
	Ist-Situation	1
	Funktionale und nicht-funktionale Anforderungen	4
	Lösungsansätze	3
	Kosten und Nutzen	5
Spezifikation	Dienste des Systems	18
	Kommunikationsprotokoll	2
	Szenarien und Benutzungsschnittstelle	9
	DV-Ist-Zustand	4
	Daten- und Zugriffsschutz	1
	Datenmodell	28

Tabelle 7: Requirements-Engineering-Dokumente des Online-Systems

3.2.2 Klassifikation der Informationen

Eine weitere Frage der Untersuchung war, Informationen welcher Arten in welchem Umfang in den Requirements-Engineering-Dokumenten enthalten sind. Um diese Frage quantitativ zu beantworten, wurde ein Kategorisierungsschema (siehe Tabelle 8) für Informationsarten aufgestellt und auf die Dokumente angewandt. Grundlage für das Kategorisierungsschema sind die Requirements-Engineering-Normen aus (Dorfman, Thayer, 1990) und (IEEE, 1998).

In Tabelle 8 wird angegeben, welchen Umfang (in Zeilen und in Prozent vom Gesamtumfang) die einzelnen Informationsarten in den Dokumenten einnehmen. Die

Information	Projekt A (Entw.-System)		Projekt B (Graph. Front-End)		Projekt C (Online-System)	
	Zeilen	%	Zeilen	%	Zeilen	%
Begriffsdefinitionen, Abkürzungen	178	2,4	29	0,7	20	1,1
Probleme, Ziele	134	1,8	116	2,6	116	6,1
Ist-Zustand	-	-	41	0,9	194	10,2
Lösungsideen, -alternativen	-	-	-	-	65	3,4
Arbeitsumgebung	46	0,6	20	0,5	50	2,6
Schnittstellen	3540	48,6	677	15,3	334	17,6
interne Struktur	1077	14,8	-	-	-	-
interner Zustand, Daten	213	2,9	1126	25,4	455	24,0
funktionale Anforderungen	1306	17,9	1861	42,1	510	26,9
nicht-funktionale Anforderungen	77	1,1	67	1,5	35	1,8
Entwurfsvorgaben	101	1,4	43	1,0	-	-
Zukunft des Systems	18	0,2	47	1,1	-	-
Projektvorgaben	134	1,8	133	3,0	106	5,6
Meta-Informationen (d.h. Informationen über die Dokumente selbst)	467	6,4	265	6,0	8	0,4

Tabelle 8: Kategorisierung der Informationen in den Requirements-Engineering-Dokumenten

Ergebnisse wurden folgendermaßen ermittelt: Das Dokument wurde in Bereiche (üblicherweise ein Absatz) aufgeteilt. Die Aufteilung erfolgte so, daß die in dem Bereich enthaltenen Informationen den gleichen Kategorien zugeordnet werden konnten. Für jeden Bereich wurde die Anzahl Zeilen gezählt, die der Bereich eingenommen hat, wobei unvollständige Zeilen (z.B. am Ende eines Absatzes) immer als ganze Zeilen gezählt wurden. Wurde ein Bereich mehreren Kategorien zugeordnet, so wurde jeder Kategorie ein entsprechender Anteil angerechnet. Tabellen und andere textuelle Notationen wurden genauso wie natürlichsprachliche Bereiche gezählt. Bei graphischen Notationen wurde die eingenommene Fläche in Zeilen umgerechnet.

Hierbei fällt auf, daß die Beschreibung der funktionalen Anforderungen, der internen Zustände und der Schnittstellen bei allen drei Systemen jeweils den größten Teil des Umfangs einnehmen (68,1%, 82,6% und 68,3%). Beim Entwicklungssystem (Projekt A) spielt zusätzlich die interne Struktur eine wichtige Rolle (14,8%). Das System wird also nicht als eine Black-Box aufgefaßt, deren interne Struktur »reine Entwurfssache« ist. Im Online-System (Projekt C) nimmt die Beschreibung des Ist-Zustands mit 10,2% einen wesentlichen Anteil ein.

3.2.3 Verwendete Notationen

Allen untersuchten Requirements-Engineering-Dokumenten ist gemeinsam, daß hauptsächlich natürliche Sprache als Notation eingesetzt wurde. Andere Notationen wurden nur in geringerem Umfang verwendet. In diesem Unterkapitel wird

Notation	Projekt A: (Entw.-System)			Projekt B (graph. Front-End)			Projekt C: (Online-System)		
	Anzahl	Seiten	%	Anzahl	Seiten	%	Anzahl	Seiten	%
natürliche Sprache	(-)	160,30	73,6	(-)	59,75	43,9	(-)	27,53	34,4
Aufzählung mit Schlüsselwörtern	-	-	-	31	25,00	18,3	18	18,00	22,5
hierarchische Aufzählung	32	10,75	4,9	-	-	-	-	-	-
Tabelle	30	8,75	4,0	85	50,25	36,9	80	31,50	39,4
Skizze	14	4,25	2,0	4	1,25	0,9	1	0,25	0,3
Bildschirmabzug	-	-	-	-	-	-	3	1,50	1,9
Baumdiagramm	22	9,00	4,1	-	-	-	1	0,25	0,3
Datenfluß-diagramm	4	4,00	1,8	-	-	-	1	0,50	0,6
Flußdiagramm	1	1,00	0,5	-	-	-	-	-	-
Pseudocode	13	19,25	8,8	-	-	-	-	-	-
EBNF	1	0,50	0,2	-	-	-	-	-	-
ER-Diagramm	-	-	-	-	-	-	1	0,50	0,6

Tabelle 9: Verwendete Notationen in den Requirements-Engineering-Dokumenten

beschrieben, welche Notationen in den Dokumenten eingesetzt (Tabelle 9) und welche Informationen durch die Notationen ausgedrückt wurden (Tabelle 10).

In Tabelle 9 werden die in den Dokumenten verwendeten Notationen aufgezählt. Für jedes Projekt wird angegeben, wie oft die Notation verwendet wird[1] und welchen Umfang (in Seiten und Prozent des Gesamtumfangs) die in der Notation abgefaßten Inhalte einnehmen. Dabei wurden *numerierte* und *punktierte Aufzählungen* und Überschriften ebenfalls als natürliche Sprache gezählt. Folgende Notationen müssen noch kurz erläutert werden:

- In einer *Aufzählung mit Schlüsselwörtern* beginnt jeder Absatz mit einem Schlüsselwort.
- In einer *hierarchischen Aufzählung* wird durch ein Einrückungsschema eine Baumstruktur nachgebildet.

1. Bei der natürlichen Sprache kann die Anzahl nicht angegeben werden, da diese Angabe keinen Sinn ergibt. Es kann z.B. gezählt werden, wieviele Tabellen oder Skizzen verwendet werden, aber nicht, an wievielen Stellen natürliche Sprache verwendet wird.

- Durch *Baumdiagramme* werden Hierarchien graphisch dargestellt.
- *Flußdiagramme* werden zur graphischen Darstellung von Abläufen benutzt.
- ER-Diagramme, Pseudocode, EBNFs und Datenflußdiagramme werden in Kapitel 2.3.5 beschrieben.

Notation	Informationen, die durch die Notation ausgedrückt werden
Aufzählung mit Schlüsselwörtern	Klassen interner Zustand
hierarchische Aufzählung	hierarchischer Aufbau von Dialogfenstern
Tabelle	Pläne: Aufwandsschätzung, Projektplan, CM-, QS-Plan Problem-, Ziel- und Ist-Analyse Definition von Begriffen, Abkürzungen Arbeitsumgebung Lösungsideen (nicht-) funktionale Anforderungen, Mengengerüst absehbare Änderungen von Anforderungen Schnittstellen, Aufbau von (Objekt-)Menüs, Liste von Dialogfenstern Zusammenhang zwischen Szenarien und Klassen Plausibilitätsregeln für Attribute
Skizze	Beispiele für die graphische Eingabe Prinzipieller Aufbau der graphischen Ausgabe Architekturvorgaben Funktionale Anforderungen
Bildschirmabzug	Benutzungsschnittstelle
Baumdiagramm	Hierarchien: Ziele, Funktionen, Programm-, Modell-Komponenten Verzeichnisstruktur, Aufbau von Dateien Zusammenhang zwischen Programmkomponenten und Funktionen Zusammenhang zwischen Funktionen und Programmkomponenten
Datenflußdiagramm	Zusammenspiel zwischen Zielsystem und Arbeitsumgebung
Flußdiagramm	Algorithmen
Pseudocode	Aufbau einer Datei (in Form eines unvollständigen Beispiels)
EBNF	Aufbau einer Datei
ER-Diagramm	Interner Zustand

Tabelle 10: Notationen und Informationen, die durch die Notationen ausgedrückt werden

In Tabelle 10 wird beschrieben, wozu die in Tabelle 9 aufgelisteten Notationen verwendet wurden, also welche Informationen damit ausgedrückt wurden.

Hierbei fällt auf, daß in den Requirements-Engineering-Dokumenten des Entwicklungssystems (Projekt A) sehr viele verschiedene Notationen eingesetzt wurden. Teilweise wurden unterschiedliche Notationen verwendet, um die gleichen Arten von Informationen zu beschreiben (z.B. EBNF und Pseudocode zur Beschreibung des Aufbaus von Dateien). Andererseits gibt es Notationen, die für viele verschiedene Zwecke eingesetzt wurden (z.B. Baumdiagramme). Die Requirements-Engineering-Dokumente des graphischen Front-Ends (Projekt B) enthalten mit

Ausnahme von vier Skizzen nur natürliche Sprache (bzw. sehr ähnliche Notationen wie Freitext mit Schlüsselwörtern und Tabellen). Das gilt auch für die Beschreibung von Klassen und Szenarien. In den Requirements-Engineering-Dokumenten für das Online-System spielen Tabellen eine sehr große Rolle. Tabellen wurden verwendet, um natürlichsprachliche Aussagen zu strukturieren und damit klarer lesbar zu machen.

3.3 Untersuchungsergebnisse - Rollen

Ein wichtiger Aspekt der Untersuchung waren die an den Requirements-Engineering-Tätigkeiten beteiligten Personen, insbesondere deren Rollen in dem Projekt.

Die zentrale Rolle in einem Requirements-Engineering-Prozeß nimmt der Systemanalytiker ein. In Projekt C gab es genau einen Systemanalytiker. In Projekt B gab es zwei Systemanalytiker, einer schrieb die Requirements Specification, der andere die Functional Specification. In Projekt A arbeiteten zwei Systemanalytiker gleichzeitig am Lastenheft und drei gleichtzeitig am Pflichtenheft.

Die Systemanalytiker in den Projekten A und C kannten sich schon zu Projektbeginn mit den Werkzeugen und Dokumentvorlagen aus. Projekt B jedoch war das erste Projekt in der Firma, in dem Szenarien eingesetzt wurden. Laut Aussagen der Projektverantwortlichen hat sich der Ansatz bewährt, auch wenn z.B. beim Management der Szenarien Verbesserungen denkbar sind.

In allen Projekten blieben die Systemanalytiker nach Ende der Requirements-Engineering-Phase weiterhin in dem Projekt tätig. Die anderen Projektmitarbeiter konnten also die Systemanalytiker fragen, wenn bestimmte Aspekte in den Requirements-Engineering-Dokumenten unklar waren.

Auftraggeber und Benutzer in Projekt A waren zwei verschiedene Abteilungen der gleichen Firma. Trotzdem wurde es wie ein Festpreisprojekt abgerechnet. Projekt B und C wurden eigenfinanziert, d.h. es gab keine externen Auftraggeber.

Die Benutzer in den Projekten A und B waren räumlich sehr weit von den Entwicklerteams und Systemanalytikern entfernt. In Projekt A waren es mehrere hundert Kilometer innerhalb Deutschlands, in Projekt B saßen die Benutzer in einem anderen Land. In Projekt C gab es einen Spezialisten aus dem Fachbereich, der die Benutzer repräsentieren sollte. Die eigentlichen Benutzer, die Bankangestellten, wurden nicht an dem Projekt beteiligt.

In Projekt C wurden noch weitere Informanten in das Projekt einbezogen:

- Ein *Datenmodellierer*, dessen Hauptaufgabe die Verwaltung und Pflege des unternehmensweiten Datenmodells (und der zugehörigen Datenbank) ist. Er sollte prüfen, ob Zielsystem und Datenbank zusammenpassen.
- Ein *Drittanbieter*, aus dessen Datenbank das Zielsystem Informationen abfragen wird.

Eine Besonderheit von Projekt A ist, daß die Wartung nach Abschluß des Projekts nicht von den Entwicklern, sondern von den Benutzern übernommen wurde.

3.4 Untersuchungsergebnisse - Prozesse

Die Requirements-Engineering-Prozesse wurden auf folgende Aspekte untersucht: Einsatz von Werkzeugen (Kapitel 3.4.1), Kommunikation mit den Kunden (Kapitel 3.4.2) und Prüfung der Requirements-Engineering-Dokumente (Kapitel 3.4.3).

3.4.1 Werkzeugeinsatz

In jedem Projekt wurden Textverarbeitungsprogramme eingesetzt, um die Requirements-Engineering-Dokumente zu erstellen und zu verwalten. In Projekt C wurde zusätzlich eine Datenbank für einen Teil der Dokumente eingesetzt.

Textverarbeitungsprogramme bieten keine Unterstützung darin, die Arbeit zwischen verschiedenen Personen am gleichen Dokument zu koordinieren. Im Projekt A zum Beispiel arbeiteten bis zu drei Personen gleichzeitig an einem Dokument. Schreib-/Lesekonflikte wurden dadurch verhindert, daß jeder Bearbeiter für einen Teil des Dokuments zuständig war und in den anderen Teilen keine Änderungen vornehmen durfte.

In allen Projekten wurden Vorlagen (Templates) eingesetzt, in denen festgelegt wird, in welcher Reihenfolge welche Kapitel vorkommen sollen, welche Informationen in welche Kapitel gehören und welche Notationen verwendet werden sollen. Dabei wurde deutlich, daß die Firma, in der Projekt B durchgeführt wurde, einen Hardware-Hintergrund hat. Die Vorlagen sehen z.B. ein Kapitel »Produktionsaspekte« vor, was im Software-Bereich eher vernachlässigbar ist.

3.4.2 Kommunikation mit den Kunden

In allen Projekten spielte die Kommunikation mit den Kunden eine wichtige Rolle. Es gab jeweils einen klar definierten Ansprechpartner auf Kundenseite.

In Projekt A war der Kunde an der Planung, der Erstellung des Lastenhefts und des Pflichtenhefts beteiligt, zumindest an den Prüfungen. Es gab insgesamt zehn zweitägige Treffen am späteren (mehrere hundert Kilometer vom Entwicklungsort entfernten) Einsatzort. Darüber hinaus war der Kunde auch in den späteren Phasen per E-Mail erreichbar.

In Projekt B war der Kunde an der Planung, an der Problemanalyse und am Abnahmetest beteiligt. Der Aufwand für die Kommunikation mit dem Kunden wird auf 5 - 10h pro Woche geschätzt. Der Gesamtkommunikationsaufwand der Projektmitglieder mit dem Kunden wird auf maximal 10% des Gesamtaufwands geschätzt. Die Kommunikation mit dem Kunden erfolgte ausschließlich über den Projektleiter.

In Projekt C erfolgte die Kommunikation über den Ansprechpartner aus dem Fachbereich, da die eigentlichen Benutzer, die Bankangestellten, nicht verfügbar waren.

3.4.3 Prüfung der Requirements-Engineering-Dokumente

Das Lastenheft von Projekt A wurde nicht formal geprüft. Für das Pflichtenheft wurde ein Review veranstaltet, an dem vier Personen aus dem Anwendungsbe-

reich und das Entwicklerteam beteiligt waren. Die Benutzungsoberfläche wurde von den gleichen Personen anhand des Prototypen geprüft. Nach dem Review wurde das Pflichtenheft entsprechend angepaßt. Alle Fehler, die danach in dem Pflichtenheft gefunden wurden, wurden in Form von Änderungsmeldungen festgehalten. Das Pflichtenheft selbst wurde nicht mehr geändert. Das aktuelle Pflichtenheft liegt also nicht mehr in Form eines einzelnen Dokuments vor, sondern als Dokument plus einer Menge von Änderungsmeldungen. Für jemanden, der nicht an dem Projekt beteiligt war, ist es sehr schwer, den aktuellen Stand zu ermitteln. Interne Widersprüche können nur sehr schwer gefunden werden. Laut Aussage des Projektleiters war dies jedoch für die Projektbeteiligten kein größeres Problem.

Die Requirements Specification von Projekt B wurde von den Kunden geprüft. Da sich die meisten Prüfer in anderen Ländern aufhielten, wurde das Dokument an die Prüfer verschickt. Die Befunde wurden an das Entwicklungsteam zurückgeschickt. Die Functional Specification wurde durch ein Review des Entwicklerteams ohne Beteiligung der Kunden geprüft. Bei den Prüfungen wurden eine Reihe von Fehlern gefunden. Diese wurden jedoch nicht behoben. Ebensowenig wurden Anforderungsänderungen, die im weiteren Projektverlauf aufgetreten sind, in den Dokumenten berücksichtigt.

Jedes in Projekt C erstellte Requirements-Engineering-Dokument wurde von dem Ansprechpartner aus dem Fachbereich geprüft, das ER-Datenmodell wurde zusätzlich von dem Datenmodellierer abgenommen. Bei den Prüfungen handelte es sich um informale Prüfungen und Abnahmen, die parallel zur Erstellung der Dokumente vorgenommen wurden. Nach Abschluß der Spezifikationsphase wurden die Dokumente nicht mehr geändert.

3.5 Konsequenzen

3.5.1 Kommunikation mit den Auftraggebern und Benutzern

Obwohl in allen Projekten eine klare Kommunikationsschnittstelle zwischen dem Entwicklerteam, den Systemanalytikern und den Kunden definiert war, sind Schwierigkeiten aufgetreten. So gab es z.B. in Projekt A das Problem, daß eine Entwicklungsumgebung realisiert werden sollte, die ein bestimmtes Programmierparadigma unterstützt und auch erzwingt. Die Konsequenzen, die sich dadurch auf die Arbeitsprozesse der Benutzer ergaben, waren nicht gründlich durchdacht. Darüber hinaus gab es Akzeptanzprobleme bei der Auslieferung. So war z.B. im Pflichtenheft eine textuelle Benutzungsschnittstelle vereinbart, mit der die Benutzer trotz des akzeptierten Prototypen nicht zufrieden waren. Hieraus folgt, daß die Kunden noch stärker in den Requirements-Engineering-Prozeß einbezogen werden müssen. Ein Indiz dafür, daß sich die Kunden nicht ausreichend mit den Inhalten der Dokumente auseinandergesetzt haben, ist, daß in dem Review viele formale, aber nur wenige inhaltliche Fehler gefunden wurden.

3.5.2 Änderungsmanagement

Eines der größten Probleme bei der Projektdurchführung war, daß sich auch nach Abschluß der Spezifikationsphase Anforderungen änderten und neue Anforderungen hinzukamen. Trotzdem wurden die Requirements-Engineering-Dokumente nicht an die Änderungen angepaßt. Zwar wurden in Projekt A Änderungsmeldungen gesammelt, sie wurden aber nicht in die Dokumente eingearbeitet.

Die Konsequenz daraus ist, daß die realisierten Systeme nur unzureichend durch die Requirements-Engineering-Dokumente beschrieben werden. Mit der Zeit wurden die Requirements-Engineering-Dokumente bedeutungslos für die fortschreitenden Projekte.

Durch folgende Maßnahmen können die Probleme vermindert werden:

- Es muß ein Verfahren zur Initiierung und Durchführung von Änderungen an den Requirements-Engineering-Dokumenten und zur Behandlung von evtl. daraus resultierenden Konflikten eingeführt werden.
- Es muß ein Konfigurationsmanagement eingeführt werden. Dies ist insbesondere dann notwendig, wenn es sich bei den Dokumenten um viele einzelne Dokumente handelt, die in vielen Versionen vorliegen.

3.5.3 Fehlende Informationen

Informationen über folgende Aspekte fehlten in den Requirements-Engineering-Dokumenten ganz oder wurden zu knapp behandelt:

- Definitionen von Begriffen und Abkürzungen. Die Definitionen sollten in einem Kapitel gesammelt und nicht erst an den Stellen eingetragen werden, an denen die Begriffe zum ersten Mal benutzt werden.
- Nicht-funktionale Anforderungen. In den Requirements-Engineering-Dokumenten von Projekt B fehlen z.B. Aussagen über die Komplexität der zu verwaltenden Daten. Sie wurde von den Entwicklern zu Beginn deutlich unterschätzt. Dies führte zu ernsthaften Effizienz-Problemen.
- Beschreibung der Arbeitsumgebung des Zielsystems.
- Beschreibung des Ist-Zustands.
- Ursprungsangaben für die erhobenen Informationen.
- Beschreibungen offener Punkte und sonstiger Mängel.

3.5.4 Informale oder formale Notationen?

In den untersuchten Requirements-Engineering-Dokumenten konnten Effekte beobachtet werden, die typisch für natürlichsprachliche oder informale Dokumente sind: unpräzise Aussagen, Widersprüche in scheinbar nicht zusammenhängenden, weit auseinanderliegenden Textstellen, keine klare Trennung zwischen einzelnen Anforderungen und zwischen Anforderungen und anderen Informationen wie z.B. Erklärungen und Begründungen.

Eine Möglichkeit, diese Probleme zu beheben, besteht darin, semi-formale oder formale Notationen zu verwenden. Jedoch kam auf die Frage, ob sich die Projektbe-

teiligten einen verstärkten Einsatz semi-formaler oder formaler Notationen vorstellen könnten, ein klares »Nein« als Antwort. Hauptgründe dagegen waren:

- sehr hoher Ein- bzw. Umstiegsaufwand,
- sehr hoher Schulungsaufwand,
- fehlende Vorstellung, inwiefern sie von Nutzen sein können (Projekt A),
- kein Bedarf – die bisherige Vorgehensweise ist geeignet (Projekt C).

In einem Unternehmen (Projekt A) wurden OOA (Coad, Yourdon, 1991) und SA (DeMarco, 1982) evaluiert, mit dem Ergebnis, daß beide Methoden für diesen Anwendungsbereich ungeeignet sind.

3.6 Schlußfolgerung

In dieser Untersuchung wurden die Requirements-Engineering-Dokumente, die Rollen der beteiligten Personen und die Requirements-Engineering-Prozesse von drei Projekten miteinander verglichen. Die Projekte waren sehr unterschiedlich. Bei einer Gegenüberstellung der zu realisierenden Systeme anhand vorgegebener Kriterien hat sich gezeigt, daß zwei der Systeme Gemeinsamkeiten aufweisen und daß sich das dritte System erheblich von den anderen beiden unterscheidet. Die Auswertung hat keine Indizien dafür geliefert, daß sich diese Ähnlichkeiten bzw. Unterschiede in der Struktur der Dokumente, in den enthaltenen Informationen oder in den verwendeten Notationen widerspiegeln.

Bei allen Requirements-Engineering-Dokumenten stand die Beschreibung der funktionalen Anforderungen, der Schnittstellen und der internen Zustände im Vordergrund. Nicht-funktionale Anforderungen, Begriffsdefinitionen, Probleme und die Arbeitsumgebung wurden jeweils zu knapp beschrieben.

Als Hauptnotation für die Spezifikation wurden in allen drei Projekten natürliche Sprache oder sehr ähnliche Notationen (z.B. Freitext mit Schlüsselwörtern, Tabellen) eingesetzt. Es wurden auch graphische Notationen eingesetzt, meistens jedoch ohne Definition der Syntax und Semantik.

Als die dringlichsten Probleme bei der Projektdurchführung wurden die Kommunikation mit den Kunden und die Verwendung und Änderung der Requirements-Engineering-Dokumente im weiteren Projektverlauf identifiziert.

Auch wenn in vielen wissenschaftlichen Publikationen formale oder semi-formale Methoden und Notationen eine sehr wichtige Rolle spielen, so liefert diese Untersuchung ein Indiz dafür, daß sie in der industriellen Praxis die bisherigen informalen Vorgehensweisen nicht ablösen werden. Die natürliche Sprache wird auch weiterhin eine zentrale Rolle spielen. Demzufolge sollte die Erstellung von natürlichsprachlichen Requirements-Engineering-Dokumenten durch Methoden und Werkzeuge unterstützt werden. Der im folgenden beschriebene ADMIRE-Ansatz ist ein Schritt in diese Richtung.

Kapitel 4

Überblick über den ADMIRE-Ansatz

In diesem Kapitel werden die wichtigsten Grundideen des ADMIRE-Ansatzes vorgestellt. ADMIRE ist ein Akronym und steht für »*Ad*vanced *M*anagement of *I*nformal *Re*quirements«. Ziel dieses Ansatzes ist es, den Requirements-Engineering-Prozeß zu verbessern, insbesondere die Erstellung, Verwaltung und Prüfung natürlichsprachlicher Spezifikationsdokumente. ADMIRE besteht aus folgenden Komponenten:

- einem *Informationsmodell*, in dem alle für die natürlichsprachliche Spezifikation relevanten Informationstypen, deren Attribute und Beziehungen beschrieben werden. Eine Instanz des Informationsmodells wird *Dokumentation* genannt.
- einem *Prozeßmodell*, in dem eine exemplarische, sinnvolle und auf das Informationsmodell abgestimmte Aufteilung des Requirements-Engineering-Prozesses in Phasen beschrieben wird.
- einer Menge von visuellen und automatischen *Prüfverfahren* zur Validierung und Verifikation.
- einem *Werkzeug*, das Dokumentationen verwalten kann, das die im Prozeßmodell beschriebenen Tätigkeiten unterstützt und das die Prüfverfahren realisiert.

Kapitel 4.1 enthält eine allgemeine Einführung in den ADMIRE-Ansatz. Anschließend werden die wichtigsten Ideen und Konzepte der einzelnen Komponenten vorgestellt und begründet: Informationsmodell (Kapitel 4.2), Prozeßmodell (Kapitel 4.3), Prüfverfahren (Kapitel 4.4) und Werkzeug (Kapitel 4.5). Zur Veranschaulichung der Konzepte und Ideen von ADMIRE wird ein durchgehendes Anwendungsbeispiel (»SESAM«) verwendet. Dieses wird in Kapitel 4.6 eingeführt.

Die ADMIRE-Komponenten werden in den folgenden Kapiteln im Detail beschrieben: das Informationsmodell in den Kapiteln 5 und 6, das Prozeßmodell in Kapitel 7, die Prüfverfahren in Kapitel 8 und das Werkzeug in Kapitel 9.

4.1 Einführung

Die Diskussion in Kapitel 2.4 und die Untersuchung in Kapitel 3 zeigen, daß in vielen Projektumgebungen natürliche Sprache als Hauptnotation zur Spezifikation eingesetzt wird und auch weiterhin eingesetzt werden soll. Im ADMIRE-Ansatz können alle Informationen durch natürliche Sprache oder sehr ähnliche Notationen beschrieben werden. Diese werden durch eine einfache graphische Notation für Systemmodelle (Kontextdiagramme) ergänzt.

Ziel des ADMIRE-Ansatzes ist es, die Systemanalytiker und Informanten bei der Erstellung qualitativ hochwertiger Dokumentationen zu unterstützen. In Kapitel 2.1.3 werden Eigenschaften aufgezählt, die ein »gutes« Pflichtenheft und somit auch eine »gute« Dokumentation so weit wie möglich aufweisen soll: »korrekt«, »relevant«, »vollständig«, »adäquat«, »realisierbar« »konsistent«, »extern konsistent«, »nicht redundant«, »eindeutig«, »überprüfbar«, »präzise«, »verständlich«, »annotiert«, »entwurfsunabhängig«, »nachvollziehbar« (vorwärts- und rückwärtsgerichtet), »verwendbar während Einsatz und Wartung«, »änderbar«, »strukturiert«, »querverwiesen«, »elektronisch gespeichert«, »ausführbar«, »wiederverwendbar« und »knapp«.

Das Informationsmodell macht (strukturelle) Minimalvorgaben, die eine Dokumentation zu jedem Zeitpunkt erfüllen muß. Diese Vorgaben sind hart, d.h. eine Verletzung der Vorgaben wird vom ADMIRE-Werkzeug nicht zugelassen. Das Informationsmodell wurde so entworfen, daß die o.g. Eigenschaften, mit Ausnahme von »ausführbar«, prinzipiell erfüllt werden können.

Ein typischer Effekt in Requirements-Engineering-Prozessen ist, daß Dokumentationen zumindest zeitweise einige der Eigenschaften nicht erfüllen. Während der Erstellung ist eine Dokumentation zwangsläufig unvollständig. Wenn sich die Informationen verschiedener Informanten widersprechen, liegen Inkonsistenzen vor. Das Informationsmodell läßt deswegen zu, daß die meisten der o.g. Eigenschaften verletzt werden. Für jede »verletzbare« Eigenschaft wird ein Qualitätskriterium definiert.

Für einige der oben genannten Eigenschaften werden keine Qualitätskriterien definiert, weil ihre Erfüllung durch das ADMIRE-Werkzeug oder das Prozeßmodell sichergestellt oder ermöglicht wird. Durch das ADMIRE-Werkzeug wird sichergestellt, daß die Informationen »elektronisch gespeichert« werden und daß sie leicht »geändert« und bis zu einem gewissen Grad auch »wiederverwendet« werden können. Durch das Prozeßmodell werden die Requirements-Engineering-Phase und alle nachfolgenden Phasen, in denen Requirements-Engineering-Tätigkeiten durchgeführt werden, berücksichtigt, also auch »Einsatz und Wartung des Zielsystems«.

Für die Eigenschaft »ausführbar« wird kein Qualitätskriterium definiert, weil es wegen des hohen Anteils natürlicher Sprache nicht erfüllt werden kann.

Das Prozeßmodell macht einen sinnvollen Vorschlag, welche Qualitätskriterien die Anforderungsdokumente, das Glossar und der Quellenkatalog in welchen Phasen jeweils erfüllen sollen und durch welche Prüfmaßnahmen (Validierung oder Verifikation) die Erfüllung der Qualitätskriterien überprüft werden soll. Die Vorgaben sind nicht hart. Sie können je nach Projekt angepaßt werden.

Ist ein Qualitätskriterium nicht erfüllt, liegt ein Mangel vor. Auch wenn ein Mangel erkannt ist, kann er nicht immer sofort behoben werden, weil oft noch Entscheidungen getroffen oder Verhandlungen darüber geführt werden müssen, wie der Mangel zu beheben ist. Deswegen bietet das Informationsmodell die Möglichkeit, Teile einer Dokumentation als mangelhaft zu markieren.

Durch die Prüfverfahren werden die Systemanalytiker dabei unterstützt, Mängel zu finden. Die den Prüfverfahren zugrundeliegenden Bedingungen verschärfen diejenigen des Informationsmodells, dürfen aber verletzt werden.

4.2 Informationsmodell

Das Informationsmodell beschreibt den strukturellen Aufbau einer Dokumentation. Beim Entwurf des Informationsmodells standen folgende Ziele im Vordergrund:

- Alle wichtigen Informationen, die während eines Requirements-Engineering-Prozesses erhoben werden, sollen dokumentiert werden können.
- Das Informationsmodell soll ermöglichen, qualitativ gute Dokumentationen zu erstellen, die möglichst viele der in Kapitel 2.1.3 beschriebenen Eigenschaften erfüllen.
- Für alle »verletzbaren« Eigenschaften (siehe Kapitel 4.1) sollen Qualitätskriterien definiert werden.
- Da Mängel nicht vermeidbar sind, soll das Informationsmodell Mängel zulassen und die Möglichkeit bieten, mangelhafte Einträge zu markieren und Mängel zu verwalten.

Zu den wichtigsten Informationen, die erhoben werden, gehören:

- *Probleme*, die in dem Projekt gelöst werden sollen,
- *Anforderungen*, die beschreiben, was genau realisiert werden soll,
- *Fakten*, die den aktuellen (Ist-) Zustand und die gegebenen Randbedingungen, unter denen die Anforderungen realisiert werden sollen, beschreiben, und
- *Zusicherungen*, die beschreiben, was spätestens für den Einsatzzeitpunkt des Zielsystems als gegeben betrachtet wird, was aber nicht realisiert werden soll.

Fakten und Zusicherungen unterscheiden sich dadurch, daß Fakten immer erfüllt sind, während das von Zusicherungen nur angenommen oder gefordert wird.

Probleme, Anforderungen, Fakten und Zusicherungen werden auch als *Inhaltselemente* bezeichnet. Die Kerninformation eines Inhaltselements ist ein natürlichsprachlicher Freitext.

Weitere wichtige Inhalte einer Dokumentation sind Erklärungen, Begründungen und Beispiele, also Informationen, die dazu beitragen, daß ein Leser die Dokumentation besser versteht. Solche Angaben werden *Annotationen* genannt.

Zur klaren Beschreibung der Einbettung des Zielsystems in seine Arbeitsumgebung und zur klaren Unterscheidung zwischen dem zu realisierenden Zielsystem und seiner als gegeben zu betrachtenden Arbeitsumgebung werden *Systemmodelle* (Kontextdiagramme) verwendet. Sie werden in einer einfachen graphischen Notation dargestellt.

Die bisher genannten Informationen werden in *Anforderungsdokumenten* organisiert. Folgende Arten von Anforderungsdokumenten werden dabei unterschieden:

- *Anforderungssammlung*, in der die Wünsche und Rahmenbedingungen aus Sicht einzelner Informanten beschrieben werden.
- *Ist-Analysedokument*, in dem der Ist-Zustand beschrieben wird.

- *Lastenheft*, in dem die Wünsche und Rahmenbedingungen aus Sicht aller befragten Informanten und die projektrelevanten Informationen aller ausgewerteten externen Dokumente dokumentiert werden,
- *Pflichtenheft*, in dem das Zielsystem spezifiziert wird.

Die Anforderungsdokumente sind strukturell prinzipiell gleich aufgebaut. Jedes Anforderungsdokument enthält eine Kapitelhierarchie und kann ein Systemmodelle enthalten. Jedes Kapitel kann Inhaltselemente enthalten. Annotationen können an Kapitel, Inhaltselemente und Systemmodelle gebunden werden. Um sicherzustellen, daß Anforderungsdokumente gegebenen Normen (d.h. strukturellen Vorgaben) entsprechen, können *Vorlagen* definiert werden.

Jedes Projekt verfügt über eine projektspezifische Terminologie, d.h. über Wörter und Phrasen, die in dem Projekt eine spezielle Bedeutung haben. Außenstehende sind mit ihr üblicherweise nicht vertraut. Für ein Projekt ist es notwendig, daß alle beteiligten Personen die projektspezifische Terminologie verstehen und verwenden. Sie sollen alle die »gleiche Sprache sprechen«. Die Terminologie wird im *Glossar* definiert.

Die Informationen in einer Dokumentation sind keine Erfindungen der Systemanalytiker, sondern werden durch Befragung von Personen, Lesen von Dokumenten oder Analyse der Arbeitsumgebung gewonnen. Diese Personen und Dokumente werden *Informationsquellen*[1] genannt. Alle relevanten Informationen über die Informationsquellen werden im *Quellenkatalog* verwaltet.

Wie am Anfang dieses Kapitels erwähnt, lassen sich *Mängel* in einem Requirements-Engineering-Prozeß nicht vermeiden. Auch wenn ein Mangel bekannt ist, ist es nicht immer möglich, ihn sofort zu beheben. Mängel können durch *Mangeleinträge* dokumentiert werden. Mangeleinträge werden im *Mängelkatalog* verwaltet. Für jedes Qualitätskriterium (Tabelle 11, S. 67) gibt es eine entsprechende Mangelkategorie.

Durch das Informationsmodell soll es dem Systemanalytiker ermöglicht werden, die geforderten Eigenschaften für »gute« Dokumentationen zu erfüllen. Dafür werden folgende Möglichkeiten geboten:

- Um das Vermischen einzelner Informationen zu vermeiden, können diese soweit wie möglich getrennt werden. Für jede Art von Information gibt es einen eigenen Eintragstyp. Jeder Eintrag erhält einen eindeutigen Bezeichner, der weder geändert noch jemals an einen anderen Eintrag der gleichen Dokumentation vergeben wird. Somit besteht die Möglichkeit, von anderen Einträgen (oder von externen Dokumenten) aus auf den Eintrag zu verweisen (vorwärtsgerichtet nachvollziehbar).
- Alle Einträge verfügen über allgemeine *Attribute* (z.B. Autor und Änderungsdatum) und über für den jeweiligen Eintragstyp spezifische Attribute (z.B. Zustand, Priorität und Stabilität bei Anforderungen). Die Attribute werden durch das Informationsmodell fest vorgegeben. Die Wertemengen einiger Attribute können geändert werden, um sie an projektspezifische Besonderheiten anzupassen.

---

1. Wenn die Arbeitsumgebung analysiert wird, dann wird nicht die Arbeitsumgebung als Informationsquelle angegeben, sondern die Personen oder Dokumente, die die Informationen über die Arbeitsumgebung geliefert haben, oder die Personen, die die Analyse durchgeführt haben.

- Inhaltselemente und Glossareinträge können *kategorisiert* werden, Inhaltselemente z.B. als »Funktion«, »Qualität«, »Datum« oder »Ereignis«. Durch die Kategorisierung besteht die Möglichkeit, inhaltlich zusammengehörende Informationen schnell aufzufinden. Es können projektspezifische *Kategorisierungshierarchien* erstellt werden.
- Um nachvollziehen zu können, woher eine Information ursprünglich kam, besteht die Möglichkeit, von Einträgen aus auf andere Einträge und insbesondere auf Einträge des Quellenkatalogs zu verweisen. Diese *Ursprungsinformationen* (rückwärtsgerichtet nachvollziehbar) können verwendet werden, um z.B. im Falle von Konflikten die betroffenen Informanten zu ermitteln.
- Zu Anforderungen, Fakten und Zusicherungen können *Testfälle* angegeben werden. Das ist immer dann sinnvoll, wenn die Anforderungen, Fakten und Zusicherungen nicht überprüfbar formuliert werden können (was z.B. bei nichtfunktionalen Anforderungen häufig vorkommt, siehe Kapitel 1.2). Kunden und Systemanalytiker einigen sich darauf, daß das Inhaltselement erfüllt ist, wenn alle angegeben Testfälle erfüllt sind (überprüfbar).
- Zwischen Einträgen können *Querverweise* eingetragen werden. Neben den o.g. Ursprungsverweisen gibt es z.B. Verweise zwischen synonymen und hyponymen Glossareinträgen, zwischen einer Annotation und den Einträgen, auf die sich die Annotation bezieht und zwischen einem Mangeleintrag und den Einträgen, die den Mangel aufweisen.

Folgende Qualitätskriterien werden im Informationsmodell definiert:

Qualitätskriterium	Zusammenhang mit den Eigenschaften aus Kap. 2.1.3
korrekt	Hierunter werden die Eigenschaften »korrekt« und »relevant« zusammengefaßt. Das Informationsmodell bietet die Möglichkeit, die Relevanz eines Eintrags durch ein Zustandsattribut (»aktuell«, »verworfen«) zu beschreiben. Relevanz kann somit auf die Korrektheit des Wertes für das Zustandsattribut zurückgeführt werden.
vollständig	Hierunter werden die Eigenschaften »vollständig«, »annotiert«, »querverwiesen« und »rückwärtsgerichtet nachvollziehbar« zusammengefaßt. Das Informationsmodell sieht für die letzten drei Eigenschaften bestimmte Attribute und Beziehungen vor. Wenn sie mit TBD belegt sind oder fehlen, liegt eine Unvollständigkeit vor.
adäquat	adäquat
konsistent	konsistent
extern konsistent	extern konsistent
nicht redundant	nicht redundant
nicht überspezifiziert	Wird statt »entwurfsunabhängig« verwendet, weil eine Dokumentation unter bestimmten Voraussetzungen Entwurfsvorgaben enthalten darf (siehe Kapitel 2.1.2).

Tabelle 11: Qualitätskriterien

Qualitätskriterium	Zusammenhang mit den Eigenschaften aus Kap. 2.1.3
realisierbar	realisierbar
überprüfbar	überprüfbar
atomar	Wird statt »vorwärtsgerichtet nachvollziehbar« verwendet. Jeder Eintrag hat einen eindeutigen Bezeichner, kann also eindeutig referenziert werden. Der Systemanalytiker muß allerdings dafür sorgen, daß die Einträge atomar sind, daß z.B. ein Inhaltselement genau eine Anforderung enthält.
verständlich	verständlich
eindeutig	eindeutig
präzise	präzise
minimal	entspricht »knapp«
normkonform	Wird statt »strukturiert« verwendet, weil die Beurteilung, ob etwas gut strukturiert ist, sehr subjektiv ist.

Tabelle 11: Qualitätskriterien

4.3 Prozeßmodell

Das Prozeßmodell beschreibt eine exemplarische, sinnvolle und auf das Informationsmodell abgestimmte Aufteilung des Requirements-Engineering-Prozesses in Phasen. Drei Zeiträume werden unterschieden: die Problemanalysephase, die Spezifikationsphase und die späteren Projektphasen. Zu jeder Phase wird angegeben, welche Tätigkeiten durchgeführt werden sollen, welche Dokumente an ihrem Ende vorliegen sollen, welche der im Informationsmodell definierten Qualitätskriterien sie erfüllen sollen und durch welche Prüfmaßnahmen die Erfüllung der Qualitätskriterien überprüft werden soll.

Ziel der *Problemanalyse* ist es, die Wünsche und Rahmenbedingungen aus Sicht der Kunden zu ermitteln. Dabei werden Informationserhebungstätigkeiten durchgeführt, deren Ergebnisse in Anforderungssammlungen, ins Glossar und in den Quellenkatalog eingetragen werden. Das Ist-Analysedokument und das Lastenheft entstehen durch Integration der Anforderungssammlungen. Hauptergebnisse dieser Phase sind das Lastenheft, das Glossar und der Quellenkatalog.

Die Anforderungsdokumente, die in der Problemanalysephase erstellt werden, sollen folgende Qualitätskriterien erfüllen: »vollständig«[1], »korrekt«, »verständlich«, »eindeutig«, »minimal« und »normkonform«. Für das Lastenheft gilt zusätzlich: »atomar« und »nicht redundant«. Für das Ist-Analysedokument, das nach Ende der Problemanalysephase nicht weiter verändert wird, gelten die gleichen Qualitätsanforderungen wie für Pflichtenhefte in der Spezifikationsphase mit Ausnahme von »realisierbar« und »nicht überspezifiziert«.

1. Das Qualitätskriterium »vollständig« ist abhängig von der Art des Dokuments definiert. Eine Anforderungssammlung z.B. soll vollständig sein bzgl. der Sichtweise einzelner Informanten. In einem Lastenheft hingegen müssen alle Informanten berücksichtigt worden sein.

Ziel der *Spezifikationsphase* ist es, ein Zielsystem zu spezifizieren, das den ermittelten Wünschen und Rahmenbedingungen genügt. Dazu müssen Mängel, insbesondere Widersprüche, die in der Problemanalysephase erlaubt waren, beseitigt werden. Darauf aufbauend wird das Pflichtenheft erstellt. Hauptergebnisse der Spezifikationsphase sind die erste Version des Pflichtenhefts, das aktualisierte Glossar und der aktualisierte Quellenkatalog.

In der Spezifikationsphase muß das überarbeitete Lastenheft zusätzlich folgende Qualitätskriterien erfüllen: »adäquat«, »konsistent« und »extern konsistent«. Für das Pflichtenheft schließlich gelten folgende zusätzliche Qualitätskriterien: »nicht überspezifiziert«, »realisierbar«, »überprüfbar« und »präzise«.

Ziel der Tätigkeiten in den *späteren Projektphasen* ist es, das Pflichtenheft, das Glossar und den Quellenkatalog aktuell zu halten. Wenn neue Informationen oder Wünsche auftauchen, wenn sich vorhandene Einträge ändern oder wenn Mängel erst in späteren Phasen entdeckt werden, werden die Dokumente angepaßt. Es gelten die gleichen Qualitätskriterien wie in der Spezifikationsphase.

Folgende Tätigkeiten werden im Prozeßmodell beschrieben: »bereite Dokumentation vor«, »erhebe Informationen«, »entwickle Lösungsideen«, »integriere Anforderungssammlungen«, »verifiziere Anforderungsdokument«, »validiere Anforderungsdokument«, »überarbeite Anforderungsdokument«, »führe Entscheidungsverfahren durch«, »erstelle Pflichtenheft« und »bearbeite Änderungsmeldung«. Zu jeder Tätigkeit wird angegeben, welche Dokumente vorliegen müssen und welche Dokumente erzeugt, geändert oder geprüft werden.

4.4 Prüfverfahren

Ziel des ADMIRE-Ansatzes ist es, qualitativ hochwertige Dokumentationen zu erstellen. Dennoch können sie Mängel enthalten. Durch visuelle und automatische Prüfverfahren werden die Systemanalytiker bei der Validierung und Verifikation unterstützt. Folgende Prüfverfahren werden beschrieben: Inhaltsprüfung, fokussierte Inspektion, Statusprüfung, Inspektion und Ähnlichkeitssuche.

Ziel der *Inhaltsprüfung* ist es, automatisch Indizien für Mängel in einzelnen Einträgen oder Teilen einer Dokumentation zu finden und Mangeleinträge zu erzeugen. Wenn z.B. in dem Freitext einer Anforderung Wörter wie »manchmal« oder »TBD« benutzt werden oder wenn der Freitext leer ist, deutet das auf eine unpräzise Aussage oder auf eine Unvollständigkeit hin. Wenn in einer aktuellen Anforderung ein nicht aktueller Quellenkatalogeintrag als Ursprung angegeben wird, liegt möglicherweise eine Inkonsistenz vor. Die Inhaltsprüfung erfolgt anhand fest vorgegebener *Prüfkriterien*. Da eine Inhaltsprüfung »nur« Indizien für Mängel finden kann, müssen die Systemanalytiker die endgültige Entscheidung darüber treffen, ob die Mängel wirklich vorliegen.

Die Idee der *fokussierten Inspektion* ist, daß der Systemanalytiker beim Schreiben oder Lesen eines Freitextes eine direkte visuelle Rückkopplung erhält. Wörter und Phrasen, die eine besondere Bedeutung haben, werden in dem Freitext markiert. Solche Phrasen sind z.B. Begriffe (Terme) des Glossars oder bestimmte, vorher festgelegte verbotene Wörter. Der Systemanalytiker kann aus einer Markierung oder dem Ausbleiben einer Markierung Rückschlüsse auf Mängel ziehen und evtl. schon während der Formulierung eines Freitextes Korrekturen vornehmen. Hier-

bei werden zwar keine Mangeleinträge erzeugt, der Systemanalytiker wird aber dabei unterstützt, Mängel zu erkennen oder zu vermeiden.

Die Idee der *Statusprüfung* ist, daß das Werkzeug dem Systemanalytiker auf übersichtliche Weise mitteilt, welche Informationen in einem Anforderungsdokument mit welcher Häufigkeit enthalten sind. So wird z.B. für jede Kategorie der Kategorisierungshierarchie angezeigt, wieviele Inhaltselemente entsprechend kategorisiert sind. Wird eine Kategorie gar nicht oder nur sehr selten verwendet, könnte das auf eine Unvollständigkeit hindeuten. Die Entscheidung, ob ein Mangel vorliegt, muß vom Systemanalytiker getroffen werden.

Einige Aspekte können nur durch *Inspektion* geprüft werden. Der ADMIRE-Ansatz unterstützt Inspektionen auf zweierlei Weisen:

- Die im Prozeßmodell geforderten Qualitätskriterien können als Checkliste für die Validierung oder Verifikation verwendet werden.
- Es können Filter definiert und angewandt werden, um die für die Prüfung relevanten Teile der Dokumentation zu extrahieren. Filter sind Bedingungen an die Eintragstypen und Beziehungen des Informationsmodells. Einige Filter sind vordefiniert. Es können auch eigene Filter definiert werden.

Die eigentliche Prüfung muß von Menschen durchgeführt werden.

Die Grundidee der *Ähnlichkeitssuche* ist, daß zu einem gegebenen Inhaltselement oder einer gegebenen Annotation die »ähnlichsten« Einträge gesucht und angezeigt werden. Der Systemanalytiker kann diese Informationen nutzen, um Inkonsistenzen oder Redundanzen zu erkennen. Folgende »semantischen« Informationen werden bei der Suche verwendet:

- die in den Freitexten benutzten Begriffe (Terme) des Glossars und
- die den Inhaltselementen zugeordneten Kategorien.

Die Ähnlichkeitssuche kann z.B. eingesetzt werden, wenn neue Einträge konsistent in ein bestehendes Anforderungsdokument eingetragen werden sollen.

Für jedes der im Informationsmodell definierten Qualitätskriterien wird angegeben, welche der o.g. Prüfverfahren sinnvoll angewandt werden können.

4.5 Werkzeug

Das ADMIRE-Werkzeug erfüllt zwei Hauptfunktionen: Es verwaltet Dokumentationen und bietet die Möglichkeit, die beschriebenen Prüfverfahren anzuwenden.

Einzelne Einträge können über eine graphische Oberfläche eingegeben, verändert, mit anderen Einträgen in Beziehung gesetzt und gelöscht werden. Für jeden Eintragstyp gibt es ein eigenes, speziell angepaßtes Fenster. Während ein Eintrag bearbeitet wird, dürfen die Bedingungen des Informationsmodells verletzt sein. Erst wenn der Systemanalytiker den neuen oder geänderten Eintrag speichern möchte, werden die Bedingungen geprüft. Nur wenn alle Bedingungen erfüllt sind, wird die Dokumentation geändert.

Der Systemanalytiker hat die Möglichkeit auszuwählen, wann welche Prüfverfahren auf welche Teile der Dokumentation angewandt werden sollen. Die Ergebnisse

werden entweder in einem Fenster angezeigt oder, bei Inhaltsprüfungen, optional in den Mängelkatalog eingetragen.

Darüber hinaus bietet das Werkzeug die Möglichkeit, Dokumente und durch Filter erzeugte Sichten als HTML-Dokumente auszugeben, so daß sie von einem WWW-Browser dargestellt oder ausgedruckt werden können.

Intern werden die Dokumentationen in einer relationalen Datenbank verwaltet.

4.6 Anwendungsbeispiel: SESAM

In dieser Arbeit werden Lösungsmöglichkeiten für einige Probleme des Requirements Engineerings präsentiert. Zur Veranschaulichung wird ein durchgehendes Beispiel verwendet: das SESAM[1]- Projekt, ein an der Universität Stuttgart in der Abteilung Software Engineering durchgeführtes Forschungsprojekt.

Das SESAM-Projekt eignet sich deswegen besonders gut,

- weil es hinreichend komplex ist, so daß viele der behandelten Requirements-Engineering-Probleme auch wirklich auftreten,
- weil es aktiv weiterentwickelt wird: es gibt »echte« Kunden[2] mit »echten« Anforderungen, die ein »echtes« Interesse an dem Projekt haben, und
- weil es, obwohl es sich um ein Forschungsprojekt handelt, viele Merkmale aufweist, die auch typisch für industrielle Software-Projekte sind: Es wird kein von Grund auf neues System realisiert. Stattdessen wird ein vorhandenes System erweitert. Die Kunden haben unterschiedliche Vorstellungen davon, wie das System weiterentwickelt werden soll. Die Mitarbeiterressourcen sind beschränkt, so daß nicht alle Anforderungen realisiert werden können. Es gibt Zeitvorgaben, bis wann bestimmte Teilsysteme fertiggestellt werden müssen. Teilaufgaben werden an externe Mitarbeiter[3] übergeben, die zu einem großen Teil nach Ende des Teilprojekts nicht mehr zur Verfügung stehen.

Ausführliche Informationen über das SESAM-Projekt finden sich in (Ludewig, 1994, Melchisedech et al., 1996, Drappa et al., 1995) und im Internet auf den WWW-Seiten der Abteilung Software Engineering der Universität Stuttgart[4].

Überblick über das SESAM-Projekt

Ziel des SESAM-Projekts ist es, die Projektleiter-Ausbildung zu verbessern. Es wird ein Simulator für Software-Projekte entwickelt. Ähnlich wie ein angehender Pilot durch einen Flugsimulator seine Flugfähigkeiten trainieren und verbessern

---

1. SESAM steht für *S*oftware *E*ngineering *S*imulation durch *A*nimierte *M*odelle.
2. Kunden sind der Leiter und die Mitarbeiter der Abteilung Software Engineering sowie ein Kooperationspartner aus der Industrie. Der Kooperationspartner soll hier nicht genannt werden. Deswegen wird hierfür die fiktive Firma SOFIE verwendet.
3. Die externen Mitarbeiter sind Studenten, die Teilsysteme in Praktika in Gruppen von drei bis neun Studenten oder in Studien- oder Diplomarbeiten realisieren.
4. Zum Zeitpunkt der Veröffentlichung dieser Dissertation sind die Informationen unter folgender Adresse abrufbar: http://www.informatik.uni-stuttgart.de/ifi/se/research/sesam.

kann, soll ein angehender Projektleiter am SESAM-Simulator seine Fähigkeiten zur Leitung von Software-Projekten trainieren und verbessern. Das SESAM-System simuliert alle Aspekte eines Software-Projekts, z.B. Personen und Dokumente, mit Ausnahme des Projektleiters.

Zu Beginn der Simulation, hier auch Spiel genannt, erhält der angehende Projektleiter, im folgenden Spieler genannt, eine textuelle Beschreibung des Projekts und der ihm zur Verfügung stehenden Ressourcen. Dann kann er den simulierten Personen über eine textuelle Schnittstelle Anweisungen erteilen, indem er sog. Benutzerkommandos absetzt. Beispiele für solche Benutzerkommandos sind: »schreibe Spezifikation«, »teste Module XY«, »prüfe Benutzungshandbuch«, »behebe gefundene Fehler im Systementwurf«. Die simulierten Personen reagieren auf die Benutzerkommandos, indem sie dem Spieler textuelle Anworten geben oder die simulierten Dokumente ändern. Dabei schreitet die simulierte Zeit fort. Am Ende des Spiels erhält der Spieler Rückmeldung darüber, wie gut er das simulierte Projekt geleitet hat, welche Qualität die Resultate haben und wie gut er die Budget- und Zeitvorgaben eingehalten hat.

Da die Effekte eines Software-Projekts sehr komplex sind, reicht das alleinige Spielen nicht aus. Wenn der Spieler Fehler macht, müssen sie ihm nach Spielende erklärt werden. Dafür gibt es den Tutor, eine reale Person, die über eine dem Spieler nicht zugängliche Schnittstelle Informationen über das simulierte Projekt und den Projektverlauf erfragen und somit den Spielverlauf analysieren kann.

Die Abläufe des Simulators werden durch ein sog. SESAM-Modell gesteuert. In dem Modell sind die Zusammenhänge eines Software-Projekts codiert. Der aktuelle Zustand des Simulators ist ein Graph, der aus attributierten Knoten und Kanten besteht. Er wird durch im SESAM-Modell definierte Regeln geändert. Immer, wenn die aktuelle Simulationszeit ein Vielfaches der Simulationsschrittweite annimmt oder überschreitet, wird ein Regelauswertungszyklus gestartet. Alle Regeln, deren Vorbedingungen erfüllt sind, werden dabei ausgeführt.

Die Modellbildung selbst ist ein eigener Forschungsbereich, der hier nicht weiter betrachtet werden soll. Die Modellbeschreibung liegt in Form einer Datei vor, die zu Beginn des Spiels vom Simulator geladen wird. Das Modell kann von einem Modellierer also leicht geändert werden.

Stand des Projekts

Das SESAM-Projekt startete 1990. 1994 wurde das in Smalltalk-80 geschriebene Pilotsystem SESAM-1 fertiggestellt. Im Oktober 1995 startete das Anschlußprojekt SESAM-2, in dem insbesondere die Modellbeschreibungssprache und die Benutzungsschnittstellen in wesentlichen Punkten überarbeitet wurden. Das SESAM-System wurde u.a. aus Effizienzgründen in Ada 95 reimplementiert.

Bis Februar 1998 ist in die Entwicklung des SESAM-2-Systems ein Aufwand von ca. 4 Mannjahren geflossen. Der Quellcode enthält ca. 49400 LOC (Programmzeilen ohne Leerzeilen und Kommentarzeilen). Zusätzlich gibt es ca. 500 Seiten Dokumentation. In der Spezifikationsphase wurden hauptsächlich natürlichsprachliche Pflichtenhefte mit einem Gesamtumfang von ca. 300 Seiten erstellt (siehe auch (Krauß, Drappa, 1998)).

In den folgenden Kapiteln werden Auszüge aus der Requirements-Engineering-Phase des SESAM-2-Projekts als Beispiele verwendet.

Kapitel 5

Informationsmodell

In der Requirements-Engineering-Phase und in den späteren Projektphasen, in denen Requirements-Engineering-Tätigkeiten durchgeführt werden, fallen viele Informationen unterschiedlicher Arten an, die dokumentiert werden sollen. Im ADMIRE-Ansatz wird ein Schema bzw. ein Muster für diese Informationen bereitgestellt. Hierbei handelt es sich um ein Modell auf Typebene. Dieses Modell wird *ADMIRE-Informationsmodell* (kurz *Informationsmodell*) genannt. Eine Instanz des Informationsmodells wird *Dokumentation* genannt.

Das Informationsmodell wird in diesem Kapitel durch Aufzählung seiner Eigenschaften E1 ... E130 beschrieben.

5.1 Dokumente

Die Informationen einer Dokumentation werden in Dokumenten organisiert. Folgende Arten von *Dokumenten* werden unterschieden:

- *Anforderungssammlung*, in der die Wünsche und Rahmenbedingungen aus Sicht einzelner Informanten beschrieben werden,
- *Ist-Analysedokument*, in dem der Ist-Zustand beschrieben wird,
- *Lastenheft*, in dem die Wünsche und Rahmenbedingungen aus Sicht aller befragten Informanten und die projektrelevanten Informationen aller ausgewerteten externen Dokumente dokumentiert werden,
- *Pflichtenheft*, in dem das Zielsystem spezifiziert wird,
- *Glossar*, in dem die projektrelevante Terminologie definiert wird,
- *Quellenkatalog*, in dem Informationen über die identifizierten Informationsquellen abgelegt werden, und
- *Mängelkatalog*, in dem Mängel verwaltet werden.

Ist-Analysedokumente, Anforderungssammlungen, Lastenhefte und Pflichtenhefte werden in Kapitel 5.2 beschrieben, Glossare in Kapitel 5.6, Quellenkataloge in Kapitel 5.8 und Mängelkataloge in Kapitel 5.9.

Da Ist-Analysedokumente, Anforderungssammlungen, Lastenhefte und Pflichtenhefte strukturell große Ähnlichkeiten aufweisen, wird hierfür der Oberbegriff *Anforderungsdokument* eingeführt.

E1: Jede Dokumentation enthält ein Glossar, einen Quellenkatalog und einen Mängelkatalog.

E2: Jede Dokumentation enthält beliebig viele Ist-Analysedokumente, Anforderungssammlungen, Lastenhefte und Pflichtenhefte.

Zur Verdeutlichung der Eigenschaften des Informationsmodells werden in diesem und in den folgenden Kapiteln Auszüge aus dem SESAM-2-Projekt (siehe Kapitel 4.6) gezeigt (Beispiel 1).

In der Requirements-Engineering-Phase des SESAM-2-Projekts wurde ein Ist-Analysedokument angelegt, in dem die wichtigsten Eigenschaften und Probleme des SESAM-1-Systems beschrieben werden. Anschließend fanden 6 Mitarbeiterkolloquien statt, in denen die wichtigen konzeptionellen Änderungen an der Modellbeschreibungssprache diskutiert und beschlossen wurden. Die Protokolle der Mitarbeiterkolloquien wurden zu je einer Anforderungssammlung (»MiKo-1« ... »MiKo-6«) überarbeitet. Zusätzlich wurden Anforderungssammlungen aus internen Gesprächsprotokollen und Notizen erstellt. Die Anforderungssammlungen wurden in ein Lastenheft (»Anforderungen an SESAM-2«) integriert und danach zu einem Pflichtenheft (»Spezifikation für SESAM-2«) überarbeitet.

Beispiel 1: Dokumente im SESAM-2-Projekt

5.2 Anforderungsdokumente

Zu jedem Dokument im herkömmlichen Sinn gehören bestimmte allgemeine Informationen, die auch in einem Anforderungsdokument enthalten sein müssen:

E3: Jedes Anforderungsdokument enthält einen Titel und ein Abstract.

E4: Jeder Titel ist ein beliebiger nicht leerer Freitext.

E5: Die Titel aller Anforderungsdokumente einer Dokumentation müssen paarweise verschieden sein.

E6: Jeder Abstract ist ein beliebiger Freitext.

Freitexte, Sätze, Wörter und TBDs

Der ADMIRE-Ansatz basiert zu einem großen Teil auf informalen Notationen, insbesondere natürlicher Sprache. Viele Teile des Informationsmodells, wie z.B. Titel und Abstract eines Anforderungsdokuments, bestehen aus Freitext.

Ein *Freitext* ist eine Folge von durch Leerzeichen, Satzzeichen oder Sonderzeichen getrennten Wörtern. Die erlaubten Satz- und Sonderzeichen werden in Kapitel 6.1.1 angegeben. Ein *Wort* ist der kleinste sinntragende Teil eines Freitextes. Es besteht aus einer nicht leeren Folge von Buchstaben, Zahlen und den Zeichen Punkt (».«), Bindestrich (»-«) und Unterstrich (»_«). Zwei Wörter sind *gleich*, wenn sie zeichenweise übereinstimmen. Somit sind die Wörter »Anforderung« und »Anforderungen« nicht gleich. Eine *Phrase* ist eine durch Leerzeichen oder Kommata (»,«) getrennte, nicht leere Folge von Wörtern.

Nach diesen Definitionen sind »Attribut«, »SESAM-2« und »evtl.« Wörter, »abgeleitetes Attribut«, »Kapitel 5.2« und »Regel, Instanz einer« Phrasen.

Ein Freitext darf leer sein, solange es nicht ausdrücklich verboten ist. Er darf beliebig lang sein. Es wird nicht streng gefordert, daß er die Grammatik-, Rechtschreibungs- und Zeichensetzungsregeln der deutschen Sprache erfüllt. Die Einschränkung auf die deutsche Sprache ist für das Informationsmodell nicht notwendig. Die Prüfverfahren (Kapitel 8) setzen aber zum Teil die deutsche Sprache voraus.

Wie in Kapitel 5.9 argumentiert wird, können Dokumentationen unvollständig sein. Um Unvollständigkeiten in Freitexten deutlich zu markieren, kann das spezielle Wort »*TBD*« benutzt werden. TBD ist eine Abkürzung für »to be determined«. Jeder Freitext, der dieses Wort enthält, wird als unvollständig betrachtet, unabhängig davon, was sonst noch in dem Freitext steht.

Allgemeine Attribute

E7: Jedes Anforderungsdokument enthält einen eindeutigen Bezeichner, einen Verweis auf den Autor, der das Anforderungsdokument erzeugt oder den Titel oder das Abstract geändert hat, und eine Angabe über das letzte Änderungsdatum.

Der *Bezeichner* wird benötigt, um das Anforderungsdokument innerhalb einer Dokumentation eindeutig referenzieren zu können. Wichtig hierbei ist, daß die Referenzen auch dann noch gültig sind, wenn sich die Dokumentation ändert. Der Bezeichner darf also z.B. nicht von der logischen Position des Eintrags in der Dokumentation abhängen. Er ändert sich nicht, wenn die Dokumentation geändert wird. Er wird auch dann nicht wieder neu vergeben, wenn das Anforderungsdokument gelöscht wird.

E8: Jeder Bezeichner ist eine Phrase.

E9: Die Bezeichner einer Dokumentation müssen paarweise verschieden sein.

Der Bezeichner kann benutzt werden, um von einem Freitext aus auf Teile der Dokumentation zu verweisen. Wenn ein Freitext eine Phrase enthält, die buchstabengetreu mit dem eindeutigen Bezeichner eines Eintrags der Dokumentation übereinstimmt, dann wird die Phrase als ein *textueller Querverweis* aufgefaßt.

Es kann nur ein *Autor* angegeben werden. Gemeint ist die Person, die die hier beschriebenen Attribute des Anforderungsdokuments zuletzt geändert hat. Untergeordnete Teile eines Anforderungsdokuments wie z.B. Kapitel verfügen über eigene Autor-Attribute. Autor-Attribute verweisen auf Einträge des Quellenkatalogs (siehe Kapitel 5.8).

Kapitelhierarchie

Zur Strukturierung der in einem Anforderungsdokument enthaltenen Inhaltselemente (Kapitel 5.3) wird, ähnlich wie in herkömmlichen Dokumenten, eine *Kapitelhierarchie* verwendet. Die Kapitelhierarchie kann beliebig breit und tief sein.

E10: Jedes Anforderungsdokument enthält eine geordnete Menge von direkt untergeordneten Kapiteln.

E11: Jedes Kapitel enthält eine geordnete Menge von untergeordneten Kapiteln.

E12: Jedes Kapitel hat entweder genau ein übergeordnetes Kapitel oder ist direkt dem Anforderungsdokument untergeordnet.

Zu jedem Kapitel im herkömmlichen Sinne gehört eine *Überschrift*, eine *Einleitung* und eine *Kapitelnummer*. Das gilt auch für Anforderungsdokumente. Die Überschrift muß nicht unbedingt eindeutig sein, so können z.B. mehrere Kapitel die Überschrift »Zusammenfassung« haben. Es ist allerdings nicht sinnvoll, wenn die direkten Unterkapitel eines Kapitels oder des Anforderungsdokuments gleiche Überschriften haben. Die Kapitelnummer hingegen muß innerhalb des Anforderungsdokuments eindeutig sein. Sie hängt von der Position des Kapitels in der Kapitelhierarchie ab. Hierbei handelt es sich also um eine abgeleitete Größe. Sie wird im Informationsmodell deswegen nicht aufgeführt.

E13: Jedes Kapitel enthält genau eine Überschrift und eine Einleitung.

E14: Jede Überschrift ist ein beliebiger, nicht leerer Freitext.

E15: Die direkten Unterkapitel eines Kapitels oder des Anforderungsdokuments müssen paarweise verschiedene Überschriften haben.

E16: Jede Einleitung ist ein beliebiger Freitext.

Zu jedem Kapitel gehören einige allgemeine Attribute:

E17: Jedes Kapitel enthält einen eindeutigen Bezeichner, einen Verweis auf den Autor, der das Kapitel erzeugt oder die hier beschriebenen Attribute zuletzt geändert hat, und eine Angabe über das letzte Änderungsdatum.

Vorlagen

In vielen Firmen ist vorgeschrieben, wie ein Anforderungsdokument strukturiert sein soll. Manchmal besteht ein »Pflichtenheft« in Wirklichkeit aus mehreren Einzeldokumenten, in denen jeweils ein wichtiger Aspekt beschrieben wird, z.B. ein Dokument für das Zielsystem und eines für jede Schnittstelle. Die zulässigen Strukturen können durch sog. *Vorlagen* vorgegeben werden.

E18: Für Ist-Analysedokumente, Anforderungssammlungen, Lastenhefte und Pflichtenhefte können jeweils beliebig viele Vorlagen vorgegeben werden.

Jede Vorlage verfügt über eine *Vorlagen-Kapitelhierarchie*, die analog zur Kapitelhierarchie für Anforderungsdokumente aufgebaut ist, mit dem Unterschied, daß ein Vorlagen-Kapitel nur über ein Überschriftsattribut verfügt.

E19: Jedes *Vorlagen-Kapitel* enthält genau eine Überschrift.

Die Eigenschaften E10 bis E12, E14 und E15 gelten ebenfalls für Vorlagen und Vorlagen-Kapitel.

Systemmodelle

In einem *Systemmodell* werden die Zielsysteme, die Personen (bzw. ihre Rollen) und Systeme der *Arbeitsumgebung* und die Schnittstellen der Systeme benannt, und es wird beschrieben, welche Systeme und Personen über welche Schnittstellen direkt miteinander kommunizieren können. Hierdurch wird eine klare Grenze zwischen Zielsystem und Arbeitsumgebung gezogen. Systemmodelle werden in Kapitel 5.5 ausführlich beschrieben.

Weil unterschiedliche Informanten unterschiedliche Vorstellungen von der Arbeitsumgebung, den Schnittstellen und der Grenze zwischen Zielsystem und Arbeitsumgebung haben können, kann eine Dokumentation mehrere Systemmodelle enthalten. Sie sind in Anforderungsdokumenten enthalten. Je nachdem, inwieweit die Informanten über eine Systemvorstellung verfügen, können schon Anforderungssammlungen oder Lastenhefte Systemmodelle enthalten. Wenn verschiedene alternative Systemmodelle diskutiert werden sollen, dann muß für jedes Systemmodell ein eigenes Anforderungsdokument angelegt werden.

E20: Jedes Anforderungsdokument enthält maximal ein Systemmodell.

Inhaltselemente und Annotationen

Der Inhalt eines Kapitels wird durch *Inhaltselemente* und *Annotationen* beschrieben. Ein Inhaltselement enthält z.B. eine gewünschte, zu realisierende Eigenschaft des Zielsystems (Anforderung). Eine Annotation enthält z.B. eine Erklärung oder Begründung. Inhaltselemente und Annotationen werden in den Kapiteln 5.3 und 5.4 ausführlich beschrieben.

Das Anforderungsdokument bildet zusammen mit den Kapiteln und Inhaltselementen eine Hierarchie.

E21: Jedes Inhaltselement gehört zu genau einem Kapitel.

E22: Jedes Kapitel enthält eine geordnete Menge von Inhaltselementen.

Diese Hierarchie-Eigenschaft gilt für Annotationen nicht mehr. Annotationen können über n:m-Beziehungen mit Inhaltselementen, Kapiteln, Systemmodellen und Anforderungsdokumenten verbunden sein.

E23: Eine Annotation bezieht sich (u.a.[1]) auf eine nicht leere Menge von Kapiteln, Systemmodellen, Anforderungsdokumenten und Inhaltselementen.

Ein Anforderungsdokument soll so weit wie möglich in sich abgeschlossen sein. Deswegen dürfen sich Annotationen nicht auf Einträge verschiedener Anforderungsdokumente beziehen. Querbeziehungen zwischen Anforderungsdokumenten können durch Angabe eines Ursprungs (siehe E28, S. 79) oder eines textuellen Querverweises hergestellt werden.

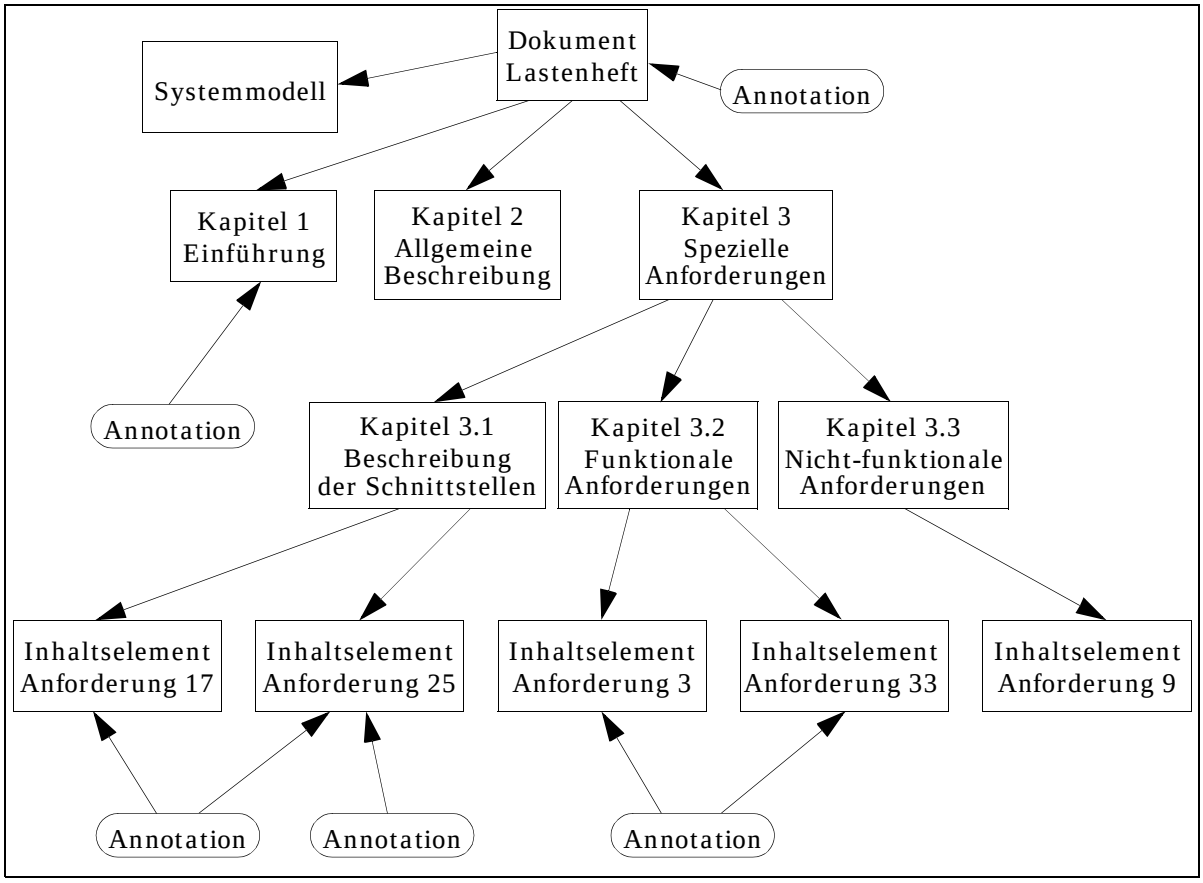
E24: Bezieht sich eine Annotation auf einen Eintrag eines Anforderungsdokuments, dann müssen sich alle anderen Bezüge der gleichen Annotation auf Einträge des gleichen Anforderungsdokuments beziehen.

Beispiel 2 zeigt Auszüge aus dem Lastenheft (»Anforderungen an SESAM-2«) als Beispiel für die Struktur eines Anforderungsdokuments.

5.3 Inhaltselemente

Durch ein Anforderungsdokument wird beschrieben, welche *Probleme* in dem Projekt gelöst werden sollen, was in dem Projekt realisiert werden soll (*Anforderun-*

1. Annotationen können sich auch auf andere Einträge einer Dokumentation beziehen, z.B. auf Glossareinträge oder Mangeleinträge (siehe E61, S. 87). Hier werden nur Annotationen betrachtet, die zu einem Anforderungsdokument gehören.



Beispiel 2: Beispielstruktur für ein Anforderungsdokument, angelehnt an IEEE (1998)

gen), was zum Einsatzzeitpunkt des Zielsystems als gegeben vorausgesetzt werden kann (*Zusicherungen*), unter welchen Randbedingungen die Anforderungen realisiert werden sollen und wie die aktuelle Situation beschaffen ist (*Fakten*).

Zur Beschreibung der Probleme, Anforderungen, Zusicherungen und Fakten werden *Inhaltselemente* verwendet. Der Kern jedes Inhaltselements ist ein beliebiger Freitext, der die eigentliche Information enthält.

E25: Jedes Inhaltselement enthält einen beliebigen Freitext.

Es ist sehr wichtig, die vier Arten der Inhaltselemente streng zu trennen, damit keine Mißverständnisse darüber aufkommen können, was gegeben ist, was als gegeben vorausgesetzt wird und was realisiert werden muß.

E26: Jedes Inhaltselement ist entweder als »Problem«, »Anforderung«, »Zusicherung« oder »Fakt« kategorisiert.

Im weiteren Verlauf dieses Berichts wird ein »Inhaltselement der Kategorie Anforderung« auch einfach »Anforderung« genannt, wenn aus dem Kontext eindeutig ersichtlich ist, ob »Inhaltselement« oder »Anforderung« gemeint ist. Analoges gilt auch für die anderen Arten von Inhaltselementen.

Jedes Inhaltselement sollte entweder genau ein Problem, ein Fakt, eine Anforderung oder eine Zusicherung enthalten, so daß sie einzeln referenziert werden können. Es kann z.B. ausgesagt werden, daß Anforderung 45 erfüllt, Anforderung 46 aber noch nicht erfüllt ist. Diese Bedingung kann wegen des Freitextes strukturell nicht sichergestellt werden und wird in Kapitel 5.10 als Qualitätskriterium »Atomizität« formuliert.

Beispiel 3 zeigt eine Anforderung und ein Fakt:

A21: »Das SESAM-System soll eine Protokolldatei anlegen, in der zumindest folgende Informationen über ein Spiel enthalten sind: der Startzeitpunkt, die Simulationsschrittweite und alle bisher ausgeführten Benutzerkommandos.«

F42: »Der typische Spieler hat keine oder nur sehr geringe Vorkenntnisse im Umgang mit dem SESAM-System.«

Beispiel 3: Eine Anforderung und ein Fakt

5.3.1 Attribute von Inhaltselementen

Zu jedem Inhaltselement gehören bestimmte Attribute, z.B. Angaben über dessen Wichtigkeit oder Stabilität. In vielen rein natürlichsprachlichen Anforderungsdokumenten werden diese Attribute durch oft nicht klar definierte Wörter ausgedrückt. Die Priorität einer Anforderung wird in einigen Fällen durch Modalverben beschrieben: »müssen« für höchste Priorität, »sollen« für zweithöchste Priorität und »können« für optionale Eigenschaften. Um Mißverständnisse wegen ungenauen Sprachgebrauchs zu vermeiden, werden die Attribute explizit notiert.

Allgemeine Attribute

Zu jedem Inhaltselement gehören einige allgemeine Attribute:

E27: Jedes Inhaltselement enthält einen eindeutigen Bezeichner, einen Verweis auf den Autor, der das Inhaltselement geschrieben bzw. zuletzt geändert hat, und eine Angabe über das letzte Änderungsdatum.

Die in den Inhaltselementen enthaltenen Informationen werden von Personen erhoben oder aus Dokumenten abgeleitet. Um z.B. im Falle von Widersprüchen nachvollziehen zu können, woher die Information kam, muß der Ursprung dokumentiert werden. Ein *Ursprung* ist entweder eine Informationsquelle aus dem Quellenkatalog (Kapitel 5.8) oder ein Eintrag (Inhaltselement, Annotation, Kapitel oder Systemmodell) in einem Anforderungsdokument.

E28: Jedes Inhaltselement enthält beliebig viele Ursprungsverweise auf andere Einträge.

Zustand

Wenn Informationen erhoben werden, dann ist es sinnvoll, diese Informationen auch dann aufzubewahren, wenn sie in dem aktuellen Projekt nicht von Bedeutung sind. Deswegen verfügen Inhaltselemente über ein *Zustandsattribut*, das folgende Werte annehmen kann:

- aktuell: Das Inhaltselement ist für das aktuelle Projekt relevant.
- zukünftig: Das Inhaltselement wird erst in einem zukünftigen Projekt relevant.
- verworfen: Das Inhaltselement ist nicht mehr relevant.

Inhaltselemente im Zustand »zukünftig« beschreiben mögliche (wahrscheinliche) Erweiterungen des Zielsystems oder der Arbeitsumgebung. »Verworfene« Inhalts-

elemente können genutzt werden, um festzustellen, ob aktuelle Themen vielleicht früher schon mal diskutiert und verworfen wurden. So kann verhindert werden, daß in zyklischen Abständen immer wieder die gleichen Anforderungen oder Lösungsvorstellungen diskutiert werden.

Darüber hinaus gibt es den speziellen Zustand »TBD«, der immer dann vergeben werden soll, wenn über den Zustand noch Unsicherheit herrscht, und den *Default*-Zustand »aktuell (Default)«, der automatisch voreingestellt wird, wenn das Inhaltselement neu erzeugt wird (siehe Kapitel 5.11). Der Zustand »aktuell (Default)« ist gleichbedeutend mit »aktuell«, weist den Betrachter aber darauf hin, daß der ursprünglich voreingestellte Zustand nicht geändert wurde und es somit möglich ist, daß der Systemanalytiker einfach vergessen hat, den Zustand zu setzen. Wenn der Systemanalytiker den Wert auf »aktuell« setzt, dann ist es eindeutig, daß er den Wert absichtlich so gewählt hat.

E29: Jedes Inhaltselement ist immer genau in einem der Zustände »aktuell (Default)«, »aktuell«, »zukünftig«, »verworfen« oder »TBD«.

Projektstand

Um den aktuellen Stand des Projekts abschätzen zu können, muß dokumentiert werden, welche Anforderungen schon erfüllt sind. Jede Anforderung enthält genau eine der folgenden Angaben über den *Projektstand*:

- nicht erfüllt: Die Anforderung ist noch nicht in dem Zielsystem realisiert.
- teilweise erfüllt: Die Anforderung ist schon teilweise realisiert.
- erfüllt: Die Anforderung wurde inzwischen vollständig realisiert.

Darüber hinaus gibt es einen speziellen »TBD«-Wert, der immer dann verwendet werden soll, wenn über den Projektstand noch Unsicherheit herrscht, und den Default-Wert »nicht erfüllt (Default)«.

E30: Jedes Inhaltselement der Kategorie »Anforderung« enthält immer genau eine der folgenden Angaben über den Projektstand: »nicht erfüllt (Default)«, »nicht erfüllt«, »teilweise erfüllt«, »erfüllt« oder »TBD«.

Priorität

Anforderungen können unterschiedlich wichtig sein. Einige Anforderungen müssen unbedingt umgesetzt werden, weil das Zielsystem sonst von den Kunden oder Benutzern nicht akzeptiert wird. Andere Anforderungen sind eher zweitrangig.

Es ist wichtig, daß die Entwickler genau wissen, wie wichtig eine Anforderung ist, damit sie sich bei der Realisierung des Zielsystems auf die entscheidenden Punkte konzentrieren können und sich nicht von Nebensächlichkeiten, die sie allerdings nicht als Nebensächlichkeiten wahrnehmen, ablenken lassen.

Die möglichen *Prioritäten* können von Projekt zu Projekt variieren. Deswegen bietet das Informationsmodell die Möglichkeit, Werte für das Prioritätsattribut vorzugeben. Prioritäten machen nur für Anforderungen und Probleme Sinn.

E31: Jede Dokumentation enthält eine Menge von möglichen Prioritäten.

E32: Jedes Element der Menge von möglichen Prioritäten ist eine Phrase.

E33: Genau eines der Elemente der Menge von möglichen Prioritäten muß als Default-Wert markiert sein.

Der Default-Wert kann von einem Werkzeug verwendet werden, um einen sinnvollen Wert für die Priorität eines neu erzeugten Inhaltselements voreinzustellen (siehe Kapitel 5.11).

E34: Jedem Inhaltselement der Kategorie »Problem« oder »Anforderung« ist immer genau eine Priorität aus der Menge der möglichen Prioritäten zugeordnet.

Eine typische Menge von Prioritäten enthält folgende Einträge:

- muß (essential): Die Anforderung muß unbedingt erfüllt werden, ansonsten akzeptiert der Kunde das Zielsystem nicht.
- soll (useful): Die Anforderung ist wichtig. In einem Minimal- oder Kernsystem kann aber vorläufig auf sie verzichtet werden.
- kann (desirable): Die Anforderung ist optional. Wenn sie nicht erfüllt wird, wird das Zielsystem dennoch akzeptiert.

Auch für Prioritäten gibt es einen speziellen TBD- und Default-Wert.

E35: Die Menge der möglichen Prioritäten wird initial mit den Werten »muß (Default)«, »muß«, »soll«, »kann« und »TBD« vorbelegt.

E36: Die Menge der möglichen Prioritäten kann erweitert werden.

E37: Ein Wert aus der Menge der möglichen Prioritäten, mit Ausnahme des TBD-Werts, kann entfernt werden, solange er keinem Inhaltselement der Dokumentation zugeordnet ist.

Stabilität

Ein wichtiges Merkmal insbesondere von Anforderungen ist, daß sie sich mit ziemlich großer Wahrscheinlichkeit im weiteren Projektverlauf ändern werden. Wenn der Systemanalytiker oder ein Informant schon in der Problemanalyse- oder Spezifikationsphase Änderungen absehen kann, dann sollte das vermerkt werden. Diese Information kann z.B. von den Entwerfern genutzt werden, indem sie den Entwurf so flexibel gestalten, daß die absehbaren Änderungen mit wenig Aufwand umgesetzt werden können.

Die möglichen *Stabilitäten* können von Projekt zu Projekt variieren. Deswegen bietet das Informationsmodell die Möglichkeit, Werte für das Stabilitätsattribut vorzugeben.

E38: Jede Dokumentation enthält eine Menge von möglichen Stabilitäten.

E39: Jedes Element der Menge von möglichen Stabilitäten ist eine Phrase.

E40: Genau eines der Elemente der Menge von möglichen Stabilitäten muß als Default-Wert markiert sein.

E41: Jedem Inhaltselement einer Dokumentation ist immer genau eine Stabilität aus der Menge der möglichen Stabilitäten zugeordnet.

Eine typische Menge von Stabilitäten enthält folgende Einträge:

- stabil: Es sind keine Änderungen an dem Inhaltselement absehbar.

- nicht stabil: Änderungen an dem Inhaltselement sind absehbar.

Falls ein Inhaltselement »nicht stabil« ist, kann durch Annotationen genauer beschrieben werden, welche Änderungen absehbar sind.

Auch für Stabilitäten gibt es einen speziellen TBD- und Default-Wert.

E42: Die Menge der möglichen Stabilitäten wird initial mit den Werten »stabil (default)«, »stabil«, »nicht stabil« und »TBD« vorbelegt.

E43: Die Menge der möglichen Stabilitäten kann erweitert werden.

E44: Ein Wert aus der Menge der möglichen Stabilitäten, mit Ausnahme des TBD-Werts, kann entfernt werden, solange er keinem Inhaltselement der Dokumentation zugeordnet ist.

Testfälle

Oft tritt das Problem auf, daß sich das Zielsystem nicht so verhält, wie es die Kunden erwarten. Dies führt zu Akzeptanzproblemen und nicht selten zu Streit darüber, ob das Zielsystem die Anforderungen erfüllt. Um diesem Streit vorzubeugen, ist es sinnvoll, zu Anforderungen, Zusicherungen und Fakten *Testfälle* anzugeben, falls sie nicht überprüfbar formuliert werden können. Dieser Fall tritt z.B. bei nicht-funktionalen Anforderungen häufig ein (siehe auch Kapitel 1.2). Eine Anforderung gilt als erfüllt, wenn das Zielsystem alle Testfälle korrekt ausführen kann. Eine Zusicherung oder ein Fakt gelten als erfüllt, wenn die betroffenen Systeme oder Personen alle Testfälle erfüllen.

E45: Jeder Anforderung, jeder Zusicherung und jedem Fakt ist immer eine evtl. leere Menge von Testfällen zugeordnet.

E46: Jeder Testfall ist ein beliebiger Freitext.

5.3.2 Kategorisierung von Inhaltselementen

Inhaltselemente können unterschiedliche Aspekte beschreiben, z.B. Funktionen, Qualitäten, Daten, Ereignisse oder Realisierungsdetails eines Systems, Eigenschaften einer Person oder Vorgaben bzgl. des Projekts. In einem typischen Anforderungsdokument sind sehr viele, oft mehrere hundert Inhaltselemente enthalten. Um Effekte wie z.B. das Vermischen von Informationen unterschiedlicher Arten zu vermeiden und das Auffinden inhaltlich ähnlicher Informationen zu erleichtern, können Inhaltselemente kategorisiert werden. Da sich ein Inhaltselement auch auf mehrere Aspekte beziehen kann, ist die Vergabe mehrerer Kategorien möglich. Aus den gleichen Gründen wie bei einigen Attributen der Inhaltselemente gibt es auch für Kategorien einen speziellen TBD-Wert.

E47: Jedem Inhaltselement ist eine evtl. leere Menge von Kategorien der Kategorisierungshierarchie für Inhaltselemente oder »TBD« zugeordnet.

E48: Keinem Inhaltselement darf die Wurzel der Kategorisierungshierarchie für Inhaltselemente zugeordnet werden.

Kategorisierungshierarchien

Die Menge der möglichen Kategorien wird durch eine *Kategorisierungshierarchie* vorgegeben. In einer Dokumentation gibt es eine Kategorisierungshierarchie für Inhaltselemente und eine für Glossareinträge (siehe E89, S. 91).

Kategorisierungshierarchien müssen folgende Bedingungen erfüllen:

E49: Die Namen der Kategorien einer Kategorisierungshierarchie sind paarweise verschieden.

E50: Jede Kategorisierungshierarchie hat eine Wurzel.

E51: Jede Kategorie mit Ausnahme der Wurzel ist genau einer anderen Kategorie direkt untergeordnet.

E52: Jede Kategorisierungshierarchie kann erweitert werden.

E53: Eine Kategorie darf nur dann aus der Kategorisierungshierarchie entfernt werden, wenn sie keinem Inhaltselement und keinem Glossareintrag zugeordnet ist und wenn sie keine untergeordneten Kategorien hat.

Vorbelegung der Kategorisierungshierarchie für Inhaltselemente

E54: Jede Dokumentation enthält genau eine Kategorisierungshierarchie für Inhaltselemente.

E55: Die Kategorisierungshierarchie für Inhaltselemente wird wie in Abbildung 9 beschrieben vorbelegt.

Im folgenden wird die Vorbelegung der Kategorisierungshierarchie für Inhaltselemente genauer beschrieben. Sie wurde so gewählt, daß möglichst alle Inhalte, die in einem Anforderungsdokument vorkommen sollten, hierdurch abgedeckt werden. Zumindest die erste Ebene der Kategorisierungshierarchie (unterhalb der Wurzel) sollte in den meisten Projekten ungeändert übernommen werden können.

Funktionen

Durch *Funktionen* wird das Verhalten eines Systems oder einer Person beschrieben. Zur vollständigen Beschreibung einer Funktion gehören:

- ein Ereignis, durch das die Funktion ausgelöst wird,
- eine (evtl. leere) Menge von Systemen/Personen und eine (evtl. leere) Menge von Schnittstellen, über die die Eingaben empfangen werden,
- ein System, das die Funktion realisiert,
- eine Beschreibung der Auswirkung der Funktion in Form von Änderungen des internen Zustands des Systems und
- eine (evtl. leere) Menge von Systemen/Personen und eine (evtl. leere) Menge von Schnittstellen, an die die Ausgaben gesendet werden.

Qualitäten

Durch *Qualitäten* werden Einschränkungen und Rahmenbedingungen festgelegt, unter denen die Funktionen ausgeführt werden.

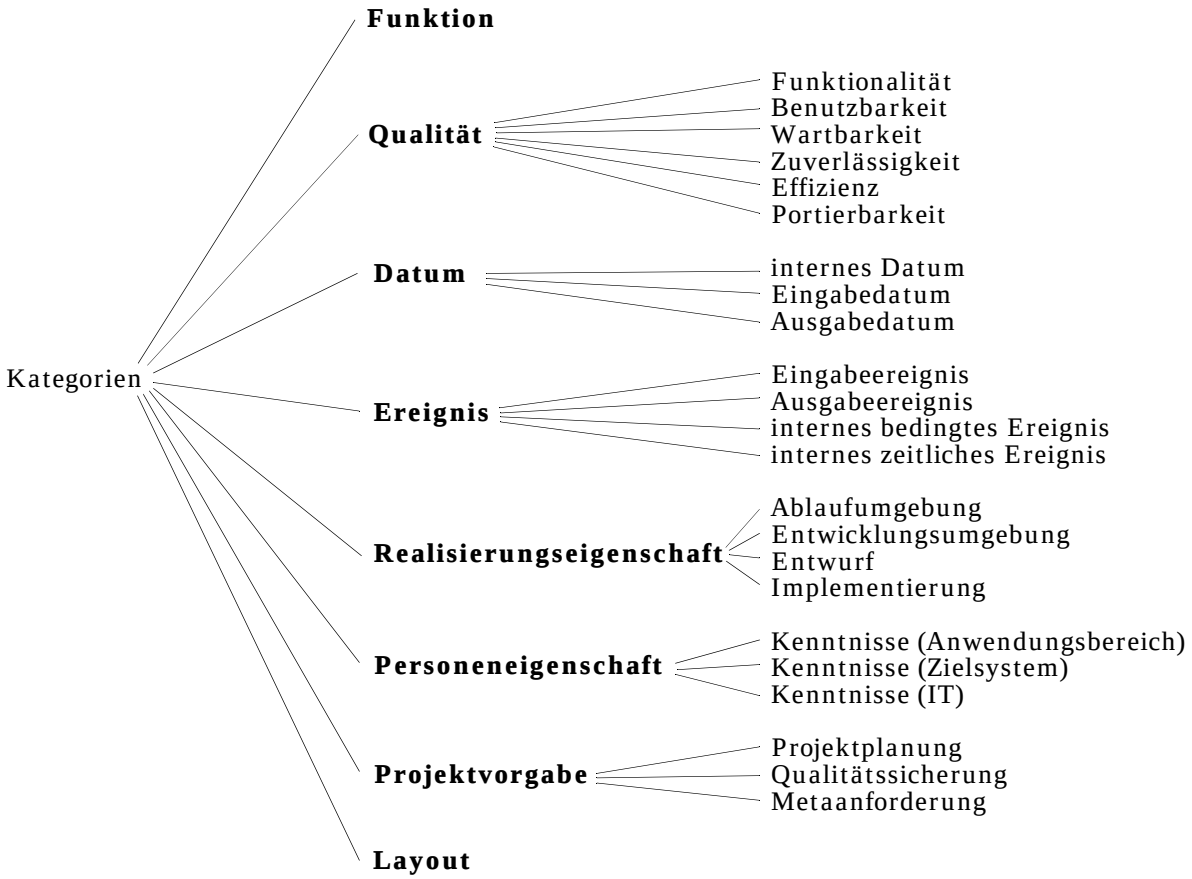


Abbildung 9: Kategorisierungshierarchie für Inhaltselemente

Von einigen Autoren (ISO/IEC, 1991, Ludewig, 1997, S. 6-2) werden *Qualitätenbäume* vorgeschlagen, in denen alle möglichen *Produktqualitäten* für ein Software-System in Form einer Hierarchie angegeben werden. In der ISO-Norm 9126 (ISO/IEC, 1991) wurden 6 Hauptgruppen aufgestellt, die alle Aspekte der Software-Qualität abdecken sollen. In Abbildung 10 werden die normierten Hauptgruppen und eine weitere Hierarchie von Untergruppen abgebildet. In einem vollständigen Anforderungsdokument (siehe Kapitel 5.10) sollten zu allen aufgezählten Qualitäten Aussagen gemacht werden. Falls eine Qualität nicht relevant ist, so sollte zumindest das notiert werden.

Daten

Ein System kann zu verschiedenen Zeitpunkten unterschiedliche *Zustände* annehmen. Zwei Zustände sind verschieden, wenn es mindestens eine Situation gibt, in der sich das System je nach Zustand anders verhält. Die Menge der möglichen Zustände eines Systems kann durch Angabe von Werten (»Initialisierungsphase«, »auf Eingabe wartend«), Angabe von Typen oder Angabe einer Datenbeschreibungssprache beschrieben werden. Neben den internen Zuständen des Zielsystems sind auch *Eingabedaten* und *Ausgabedaten* interessant.

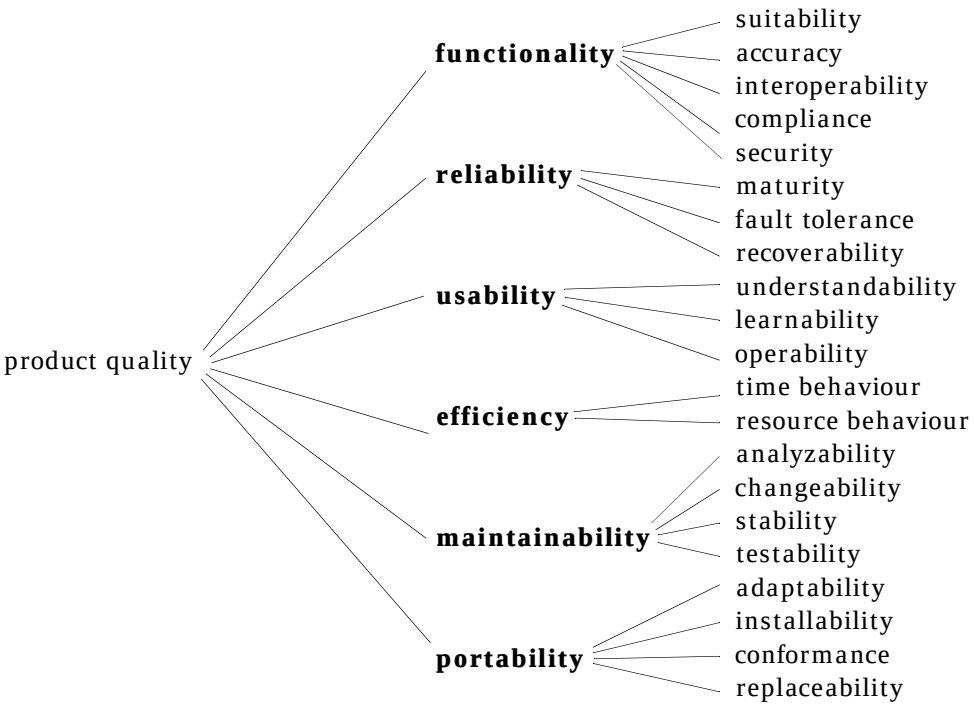


Abbildung 10: Qualitätenbaum nach ISO/IEC 9126 (1991)

Ereignisse

Ein *Ereignis* ist die Beschreibung eines bemerkenswerten Vorkommens zu einem bestimmten Zeitpunkt an einer bestimmten Stelle. Im Zusammenhang mit Software-Systemen sind folgende Arten von Ereignissen von Bedeutung:

- *Eingabeereignis*: Ereignis, das von einem System der Arbeitsumgebung oder einer Person ausgelöst wird, z.B. »Benutzer wählt Menüpunkt aus« oder »Drucker meldet FERTIG«.
- *Ausgabeereignis*: Ereignis, das vom Zielsystem ausgelöst und an Personen oder Systeme der Arbeitsumgebung weitergereicht wird, z.B. »das Zielsystem meldet BERECHNUNG DURCHGEFÜHRT.«
- *internes bedingtes Ereignis*: Ereignis innerhalb des Zielsystems, das immer dann ausgelöst wird, wenn der interne Zustand des Zielsystems bestimmte Bedingungen erfüllt, z.B. »Fehler aufgetreten«.
- *internes zeitliches Ereignis*: Ereignis innerhalb des Zielsystems, das zu einem bestimmten Zeitpunkt ausgelöst wird, z.B. »10 ms nach Ereignis E«

Realisierungseigenschaften

Wie in Kapitel 2.1.2 beschrieben, werden in der Requirements-Engineering-Phase oft »Entwurfsentscheidungen« getroffen, weil Informanten entsprechende Anforderungen formulieren, z.B.

- indem sie die *Ablaufumgebung* vorgeben, z.B. die Hardware und das Betriebssystem, unter denen das Zielsystem ausgeführt werden soll,
- indem sie die *Entwicklungsumgebung* und damit insbesondere die Programmiersprache vorgeben,

- indem sie *Entwurfsstrukturen* vorgeben, z.B.: »die Kundendaten müssen aus Sicherheitsgründen auf einer zweiten Festplatte gespiegelt werden«, oder
- indem sie *Implementierungsaspekte* vorgeben, z.B. »zur Sortierung eines Feldes soll der Quicksort-Sortieralgorithmus verwendet werden«.

Personeneigenschaften

Da zur Umgebung des Zielsystems oft auch Personen gehören, müssen auch personenspezifische Informationen dokumentiert werden können, z.B.

- Vorkenntnisse eines Benutzers im Anwendungsbereich,
- allgemeine Kenntnisse eines Benutzers im Umgang mit Computern und Software und
- spezielle Kenntnisse eines Benutzers im Umgang mit dem Zielsystem und die zu erwartende Häufigkeit, mit der er das Zielsystem benutzen wird.

Projektvorgaben

In einigen Fällen müssen auch *Projektvorgaben*[1] in ein Anforderungsdokument aufgenommen werden. Projektvorgaben sind z.B.:

- Vorgaben bzgl. der *Projektplanung*, z.B. Anforderungen an Tätigkeiten, Personen, Aufwände, Meilensteine oder ganz allgemein an den Projektablauf,

»Wenn der Spieler in das Kommandofeld das Benutzerkommando ›Proceed‹ eingibt und danach die Return-Taste drückt, wird ein Regelauswertungszyklus angestoßen. Die dabei erzeugten Nachrichten des Simulators werden im Nachrichtenfenster dargestellt.«

ID:	A45	*Ursprung*:	Q15: Protokoll Miko-1
Autor:	Melchisedech	*Priorität*:	muß
Datum:	28.08.99	*Stabilität*:	stabil
Projektstand:	erfüllt	*Zustand*:	aktuell
Kategorien:	Funktion		

Testfälle: »Nach Durchführung dieser Funktion muß die Simulationszeit mindestens um die Simulationsschrittweite weitergeschaltet worden sein.«

»Die Durchführung eines Regelauswertungszyklus darf nicht länger als 0,01s pro auszuwertender Regel dauern.«

ID:	A64	*Ursprung*:	Q23: Projektleiter SESAM-2
Autor:	Melchisedech	*Priorität*:	muß
Datum:	28.09.99	*Stabilität*:	nicht stabil
Projektstand:	nicht erfüllt	*Zustand*:	aktuell
Kategorien:	Qualität-Effizienz		

Testfälle: -

Beispiel 4: Inhaltselemente (Anforderungen)

1. Über die Frage, ob Projektanforderungen in ein Anforderungsdokument aufgenommen werden sollen, ist sich die Fachwelt nicht einig (siehe Kapitel 2.1.4).

- Vorgaben bzgl. der *Qualitätssicherung*, z.B. Anforderungen, die die Verifikation und Validierung der Anforderungsdokumente betreffen, und
- *Richtlinien/Metaanforderungen*, d.h. Anforderungen, die die Anforderungsdokumente betreffen, indem sie den Aufbau, das Layout oder den Inhalt vorgeben oder die Einhaltung von Richtlinien und Normen fordern.

Layout

Oft spielt auch das *Layout* einer Schnittstelle eine sehr wichtige Rolle. Besonders deutlich wird das an der Benutzungsoberfläche. Die Akzeptanz eines Software-Systems hängt ganz entscheidend von seiner Benutzungsoberfläche ab. Deswegen enthält das Pflichtenheft oft detaillierte Aussagen z.B. über den Aufbau der Fenster oder die Anordnung der interaktiven Elemente und macht Vorgaben darüber, wie welche Informationen zu präsentieren sind.

Eine Unterteilung der Kategorie »Layout« wird hier nicht vorgeschlagen, weil sie sehr stark von den Schnittstellen abhängt, die das Zielsystem aufweist.

5.4 Annotationen

Wichtige Aspekte eines Anforderungsdokuments sind Erklärungen, Begründungen und Beispiele, also Informationen, die dazu beitragen, daß ein Leser die Dokumentation besser versteht. Oft reicht es nicht aus zu fordern, daß eine Anforderung erfüllt werden soll. Es muß auch gesagt werden, warum sie erfüllt werden soll und ggfs. auch, warum sie einer anderen, ihr widersprechenden Anforderung vorgezogen wird. Solche Einträge werden *Annotationen* genannt.

E56: Eine Dokumentation enthält beliebig viele Annotationen.

E57: Jede Annotation enthält einen beliebigen Freitext.

Zu jeder Annotation gehören einige allgemeine Attribute:

E58: Jede Annotation enthält einen eindeutigen Bezeichner, einen Verweis auf den Autor, der die Annotation geschrieben bzw. zuletzt geändert hat, und eine Angabe über das letzte Änderungsdatum.

E59: Jede Annotation enthält beliebig viele Verweise auf ihre Ursprünge.

E60: Jede Annotation enthält eine nicht leere Menge von Verweisen auf die Inhaltselemente, Systemmodelle, Glossareinträge, Quellenkatalogeinträge, Mangeleinträge, Kapitel oder Dokumente, denen die Annotation zugeordnet wird.

Um komplizierte und sehr schwer verständliche Abhängigkeitsbeziehungen zu vermeiden, dürfen Annotationen keinen Annotationen zugeordnet werden.

Dokumente sollen soweit wie möglich abgeschlossen sein (siehe auch E23 und E24, S. 77). Das heißt:

E61: Alle Einträge, denen die Annotation zugeordnet ist, müssen zum gleichen Dokument gehören.

»Für die maximale Ausführungszeit eines Regelauswertungszyklus kann keine Obergrenze angegeben werden, weil ein SESAM-Modell beliebig viele Regeln enthalten kann und bei Ausführung jede Regel daraufhin geprüft werden muß, ob sie angewandt werden kann.«

ID:	AN46	*Datum*:	28.08.99
Autor:	Melchisedech	*Ursprung*:	Q15: Protokoll Miko-1
Verweise:	A64		

Beispiel 5: Annotation (zu Anforderung A64, siehe Beispiel 4, S. 86)

5.5 Systemmodelle

In einem Software-Projekt wird mindestens ein System realisiert. Systeme kommunizieren über Schnittstellen mit anderen Systemen und mit Personen. Für das Verständnis eines Anforderungsdokuments ist es entscheidend, daß die Systeme, Personen und Schnittstellen benannt und deren Beziehungen dokumentiert werden. Statt Personen werden üblicherweise ihre Rollen angegeben (siehe Kapitel 2.2). Diese Informationen werden in einem *Systemmodell* verwaltet. Ein Systemmodell ist ein flaches Netzwerk aus Knoten, die Systeme, Schnittstellen und Rollen repräsentieren, und Kanten, die Kommunikationsverbindungen repräsentieren.

Es ist sehr wichtig, daß klar zwischen zu realisierenden und vorhandenen Systemen unterschieden wird. *Zielsysteme* sind Systeme, die im Rahmen des Projekts realisiert werden sollen. *Vorgegebene Systeme* sind Systeme, mit denen die Zielsysteme interagieren sollen. Sie sind in der *Arbeitsumgebung* vorhanden oder sollen zu dem Zeitpunkt, zu dem die Zielsysteme eingesetzt werden, vorhanden sein. Personen gehören immer zur Arbeitsumgebung, weil es nicht die Aufgabe eines Software-Projekts sein kann, eine Person zu realisieren.

Jedes System verfügt über Schnittstellen, über die es Eingaben empfangen und Ausgaben erzeugen kann. Es gibt z.B. folgende Arten von Schnittstellen: Dateischnittstellen, graphische Benutzungsoberflächen (GUI), Hardware-Schnittstellen oder Datenbankschnittstellen. Die genaue Charakterisierung einer Schnittstelle erfolgt durch Inhaltselemente.

E62: Jedes Systemmodell enthält beliebig viele *Systemmodelleinträge*.

E63: Jeder Systemmodelleintrag hat einen Namen.

E64: Jeder Systemmodelleintrag ist entweder als »Zielsystem«, »vorgegebenes System«, »Schnittstelle« oder »Person« kategorisiert.

Die Zugehörigkeit von Schnittstellen zu Systemen wird durch *Schnittstellenkanten* dargestellt.

E65: Jedes Systemmodell hat beliebig viele Schnittstellenkanten.

E66: Jede Schnittstellenkante verbindet ein »vorgegebenes System« oder ein »Zielsystem« und eine »Schnittstelle«.

E67: Jede »Schnittstelle« ist über eine Schnittstellenkante mit genau einem »Zielsystem« oder einem »vorgegebenen System« verbunden.

Wenn zwei Systeme miteinander kommunizieren, dann muß eine Schnittstelle des einen Systems mit einer Schnittstelle des anderen Systems verbunden sein. Wenn ein System und eine Person miteinander kommunizieren, dann muß eine Schnittstelle des Systems mit der Person verbunden sein. Personen haben keine Schnittstellen. Solche Verbindungen werden durch *Kommunikationskanten* dargestellt.

Insbesondere in der Problemanalysephase, wenn Vorversionen des endgültigen Systemmodells erstellt und diskutiert werden, kann es sinnvoll sein, Schnittstellen noch nicht zu betrachten. Bevor festgelegt wird, über welche Schnittstellen die Kommunikation erfolgen wird, sollte geklärt werden, welche Systeme und Personen überhaupt direkt miteinander interagieren sollen. Deswegen können in einem Systemmodell auch Systeme ohne Schnittstellen dargestellt werden. Kommunikationskanten können auch direkt Systeme verbinden.

E68: Jedes Systemmodell hat beliebig viele Kommunikationskanten.

E69: Jede Kommunikationskante verbindet genau zwei Systemmodelleinträge.

E70: Eine Kommunikationskante darf nicht ein »Zielsystem« oder ein »vorgegebenes System« mit einer seiner Schnittstellen verbinden.

E71: Zwei Systemmodelleinträge dürfen über maximal eine Kommunikationskante (direkt) miteinander verbunden sein.

In vielen Fällen kann eine Schnittstelle eines Systems von mehreren Personen oder Systemen gleichzeitig benutzt werden. Dieser Umstand wird im Systemmodell durch *Kardinalitäten* reflektiert:

E72: Jede Kommunikationskante hat optional zwei Kardinalitätsangaben.

E73: Eine Kardinalitätsangabe besteht aus der Angabe einer Untergrenze und einer Obergrenze. Die Untergrenze ist eine beliebige natürliche Zahl, die Obergrenze ist entweder eine natürliche Zahl, die größer oder gleich der Untergrenze ist, oder »*« für »beliebig viele«.

E74: Werden zu einer Kommunikationskante keine Kardinalitäten angegeben, so ist das gleichbedeutend mit den Kardinalitäten ((0,1), (0,1)).

Ein Systemmodell ist in sich abgeschlossen, d.h.:

E75: Kommunikationskanten und Schnittstellenkanten dürfen nur solche Systemmodelleinträge verbinden, die zum gleichen Systemmodell gehören.

Zu jedem Systemmodell gehören einige allgemeine Attribute:

E76: Jedes Systemmodell enthält einen eindeutigen Bezeichner, einen Verweis auf den Autor, der das Systemmodell geschrieben bzw. zuletzt geändert hat, und eine Angabe über das letzte Änderungsdatum.

E77: Jedes Systemmodell enthält beliebig viele Verweise auf seine Ursprünge.

Ein Systemmodell eines Ist-Analysedokuments beschreibt den aktuell gegebenen Zustand, d.h.:

E78: Ein Systemmodell eines Ist-Analysedokuments darf keine Systemmodelleinträge der Kategorie »Zielsystem« enthalten.

In der Problemanalysephase werden oft sehr generelle Anforderungen formuliert. Die zugehörigen Inhaltselemente sind nicht direkt an Systeme oder Schnittstellen

bindbar. Spätestens im Pflichtenheft müssen sie präzisiert werden, d.h., es muß angegeben werden, welche Systeme oder Personen die Anforderung realisieren sollen. Wird die Anforderung durch ein Zielsystem realisiert, so bleibt sie weiterhin eine Anforderung. Wird die Anforderung durch ein System oder eine Person der Arbeitsumgebung realisiert, so handelt es sich aus Sicht des Zielsystems um eine Zusicherung.

E79: »Anforderungen« können an beliebig viele »Zielsysteme« und deren »Schnittstellen« gebunden werden.

E80: »Zusicherungen« können an beliebig viele »Personen«, »vorgegebene Systeme« und deren »Schnittstellen« gebunden werden.

E81: »Probleme« oder »Fakten« können an beliebige Systemmodelleinträge gebunden werden.

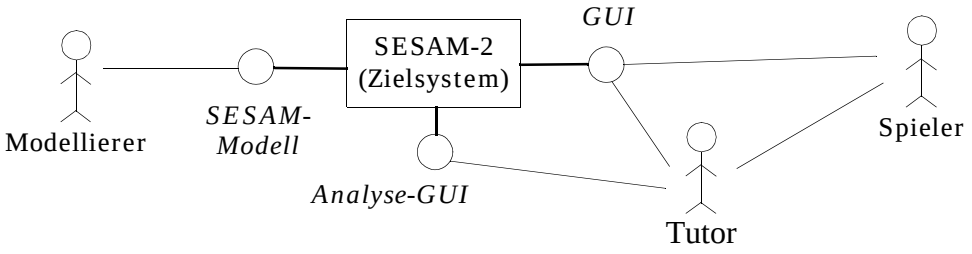
E82: Wird ein Inhaltselement an eine »Schnittstelle« gebunden, so bedeutet das implizit, daß es auch an das zugehörige System gebunden ist.

Da Anforderungsdokumente abgeschlossen sein sollen, gilt folgendes:

E83: Ein Inhaltselement darf nur an solche Systemmodelleinträge gebunden werden, die zum Systemmodell des gleichen Anforderungsdokuments gehören wie das Inhaltselement selbst.

Zur Darstellung eines Systemmodells eignen sich einfache graphische Notationen, wie diejenige aus Beispiel 6.

Das SESAM-2-System wird von Spielern, Tutoren und Modellieren benutzt. Der Spieler kann über die »GUI«-Schnittstelle mit dem SESAM-2-System interagieren. Der Tutor, der detailliertere Informationen über den Spielablauf erhalten soll, kann zusätzlich auf die »Analyse-GUI« zugreifen. Diese zusätzlichen Informationen kann er verwenden, um den Spielverlauf zu interpretieren und dem Spieler Verbesserungsvorschläge zu machen. Deswegen ist auch eine Kommunikationskante zwischen Spieler und Tutor eingezeichnet. Der Modellierer ist für die Erstellung und Pflege der SESAM-Modelle zuständig.



ID: SM18 *Datum*: 28.08.99
Autor: Melchisedech *Ursprung*: Q23: Projektleiter SESAM-2

Legende:
<Name> (Zielsystem) Zielsystem
<Name> Person
<Name> Schnittstelle
Kommunikationskante
Schnittstellenkante

Beispiel 6: Systemmodell für das SESAM-2-System

5.6 Glossar

In jedem Projekt gibt es eine projektspezifische Terminologie. Es werden Wörter benutzt, die für die Personen außerhalb des Projekts keine oder eine andere Bedeutung haben als für die Projektbeteiligten.

An einem Projekt sind Personen mit unterschiedlichem Hintergrundwissen beteiligt, z.B. Experten des Anwendungsbereichs, spätere Benutzer, Finanziers oder Entwickler. Sie alle müssen die »gleiche Sprache sprechen«. Zu jeder Dokumentation gehört deswegen ein *Glossar*, in dem potentiell mehrdeutige oder unbekannte Wörter und Phrasen definiert werden können.

E84: Jedes Glossar enthält beliebig viele *Glossareinträge.*

E85: Jeder Glossareintrag enthält einen Term und eine Definition.

E86: Jeder Term ist eine Phrase.

E87: Jede Definition ist ein beliebiger Freitext.

Die Wörter des *Terms* können von beliebiger Art sein, z.B. Substantive, Verben oder Adjektive. Besteht der Term aus nur einem Wort (»Spieler«), wird er *Einwort-Term* genannt, besteht er aus mehreren Wörtern (»angehender Projektleiter«), wird er *Mehrwort-Term* genannt. Durch die *Definition* wird dem Term eine Bedeutung zugewiesen.

In der Lexikologie bzw. Lexikographie wird typischerweise gefordert, daß zur Angabe eines Terms immer die *Grundform* verwendet wird; d.h. bei Substantiven und Adjektiven der Nominativ Singular (oder Plural, falls die Singularform nicht existiert), bei Verben der Infinitiv. Bei Mehrwort-Termen ist die Bildung der Grundform nicht immer einfach. Da nicht durch strukturelle Vorgaben sichergestellt werden kann, daß ein Term in Grundform vorliegt, wird die Verwendung der Grundform nicht streng gefordert, aber zumindest empfohlen.

Bei Termen kann unterschieden werden, ob sie vorhanden sind, also jeweils von mindestens einem Informanten in den durch die Definitionen vorgegebenen Bedeutungen benutzt werden, oder ob sie im Projektverlauf neu gebildet oder mit einer projektspezifischen Bedeutung belegt werden. Erstere werden *Terme des Anwendungsbereichs* genannt, letztere *projektspezifische Terme.*

E88: Jeder Glossareintrag ist entweder als »projektspezifischer Term« oder »Term des Anwendungsbereichs« kategorisiert.

Durch Terme können unterschiedliche Sachverhalte benannt werden, z.B. Projekte, Systeme, Rollen, Schnittstellen, Eigenschaften, Funktionen, Abläufe, Daten und Ereignisse. Aus ähnlichen Gründen wie bei Inhaltselementen ist es sinnvoll, Glossareinträge zu kategorisieren.

E89: Jede Dokumentation enthält eine Kategorisierungshierarchie für Glossareinträge.

E90: Die Kategorisierungshierarchie für Glossareinträge ist wie in Abbildung 11 dargestellt vorbelegt.

In E49 bis E53 (S. 83) wird beschrieben, wie eine Kategorisierungshierarchie geändert und somit an projektspezifische Bedingungen angepaßt werden kann.

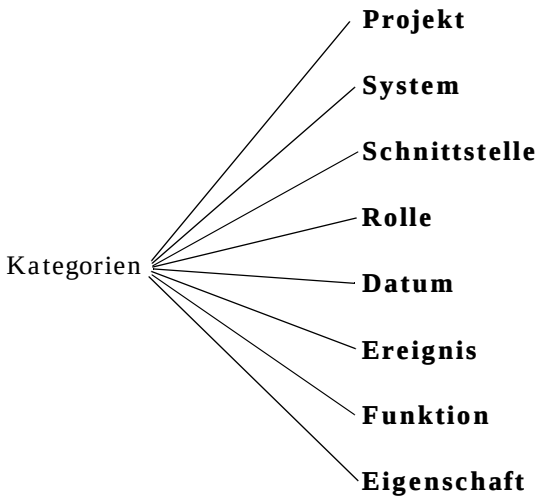


Abbildung 11: Kategorisierungshierarchie für Glossareinträge

E91: Jedem Glossareintrag ist eine evtl. leere Menge von Kategorien der Kategorisierungshierarchie für Glossareinträge oder »TBD« zugeordnet.

E92: Keinem Glossareintrag darf die Wurzel der Kategorisierungshierarchie für Glossareinträge zugeordnet werden.

Es kann vorkommen, daß der gleiche Sachverhalt von verschiedenen Informanten durch unterschiedliche Terme beschrieben wird (*Synonyme*), oder daß ein Term unterschiedliche Sachverhalte beschreibt (*Homonyme*). Im Idealfall sollten solche Effekte vermieden werden, weil das Ziel der Terminologiebildung eine eindeutige Abbildung zwischen Begriff und Term (und umgekehrt) ist. Synonyme und Homonyme lassen sich jedoch nicht immer vermeiden. Homonymie wird dadurch zugelassen, daß mehrere Glossareinträge mit gleichem Term erlaubt sind. Die Synonymiebeziehung kann explizit notiert werden.

E93: Jeder Glossareintrag enthält beliebig viele Synonym-Verweise auf andere Glossareinträge.

E94: Die Synonymiebeziehung ist symmetrisch.

Die letzte Eigenschaft kann durch ein Werkzeug automatisch sichergestellt werden.

In das Glossar können auch *Abkürzungen* eingetragen werden. Zu jeder Abkürzung muß (soweit vorhanden) der abgekürzte Term als Synonym angegeben werden. Die Abkürzungsbeziehung ist somit ein Spezialfall der *Synonymiebeziehung* und wird im Informationsmodell nicht gesondert behandelt.

Zwischen Termen kann eine »Ist-Ein« bzw. eine *Hyponymiebeziehung* vorliegen, z.B. zwischen den Rollen »Benutzer« und »Spieler«. Wenn in einem Anforderungsdokument eine Aussage über Benutzer gemacht wird, dann wird dadurch implizit auch eine Aussage über Spieler als spezielle Benutzer gemacht. Um diese impliziten Zusammenhänge deutlich zu machen, werden diese Informationen in einer Dokumentation verwaltet. Der speziellere Term (»Spieler«) wird *Hyponym* genannt, der allgemeinere (»Benutzer«) *Hyperonym*.

E95: Jeder Glossareintag enthält beliebig viele Hyperonym-Verweise auf andere (allgemeinere) Glossareinträge.

Wenn verschiedene Informanten den gleichen Sachverhalt unterschiedlich bezeichnen (Synonyme), muß im Verlauf der Analyse entschieden werden, welcher Term endgültig benutzt werden soll. Es kann also vorkommen, daß bisher benutzte Terme zukünftig nicht mehr benutzt werden sollen. Deswegen muß zu einem Glossareintrag ein *Zustand* angegeben werden. Für den Zustand gibt es spezielle TBD- und Default-Werte (siehe auch Kapitel 5.3.1).

E96: Jeder Glossareintrag ist immer in einem der folgenden Zustände: »aktuell (Default)«, »aktuell«, »verworfen« oder »TBD«.

Mit dem Glossar wird das Ziel verfolgt, daß die Terme in den Freitexten benutzt werden. Ein Einwort-Term wird in einem Freitext *benutzt*, wenn es eine Flexion des Terms gibt, die in dem Freitext buchstabengetreu vorkommt. So werden z.B. in E84 (S. 91) die Terme »Glossareintrag« und »Glossar« benutzt.

Die Definition der Benutzt-Beziehung kann nicht ohne weiteres auf Mehrwort-Terme ausgedehnt werden. Folgende Freitexte, in denen der Term »Attribut eines Dokuments« benutzt bzw. nicht benutzt wird, sollen das verdeutlichen:

1. »Die Attribute eines SESAM-Modells können frei definiert werden.«
2. »Jedes simulierte Dokument muß mindestens das Attribut *name* enthalten.«
3. »Das Dokumentattribut *name* ist vom Datentyp *string*.«
4. »Alle Attribute eines SESAM-Modells werden in dem Dokument ›Modellübersicht‹ aufgelistet.«

Hierbei fallen folgende Besonderheiten auf:

- Die Reihenfolge der Wörter kann sich ändern (Freitext 2).
- Die Wörter müssen nicht unmittelbar aufeinander folgen (Freitext 2).
- Die Benutzung aller Wörter des Terms in einem Satz bedeutet nicht unbedingt, daß auch der Term benutzt wird (Freitext 4).
- Nicht alle Wörter des Terms müssen benutzt werden, z.B. der unbestimmte Artikel »einer« in Freitext 2.
- Die Wörter können auch zusammengeschrieben werden (Freitext 3).

Das Informationsmodell soll so beschaffen sein, daß ein Werkzeug die Benutzt-Beziehung automatisch erkennen kann. Die Erkennung wird jedoch durch unregelmäßige Deklinationen und Konjugationen, diskontinuierliche Konstituenten (»er hört auf« statt »aufhören«) und die oben beschriebenen Besonderheiten bei Mehrwort-Termen erschwert. Auch wenn es ein Charakteristikum von Fachterminologien ist, daß sie hauptsächlich regelmäßige Wörter enthalten, dürfen die unregelmäßigen Wörter nicht ignoriert werden. Es kann durchaus vorkommen, daß Standardwörter der deutschen Sprache im Projektkontext eine eigene Bedeutung haben. Um auch Sonderfälle richtig zu behandeln, muß der Systemanalytiker manuell eingreifen können, d.h. er muß Flexionen angeben können.

E97: Jeder Glossareintrag enthält beliebig viele Flexionsangaben für den Term.

E98: Jede Flexionsangabe bezieht sich auf ein maximales Wort des Terms oder auf den gesamten Term.

Ein Wort ist ein *maximales Wort* innerhalb eines Freitextes, wenn es buchstabengetreu in der Phrase enthalten ist und wenn es kein längeres Wort gibt, daß an gleicher Stelle ebenfalls in der Phrase enthalten ist. Beispielsweise ist das Wort »Dokuments« ein maximales Wort in der Phrase »Attribut eines Dokuments«, »Doku« aber nicht.

Flexionsangaben für den Einwort-Term »Attribut« sind z.B. »Attribute«, »Attributs«. Flexionsangaben für die einzelnen Wörter des Mehrwort-Terms »Attribut eines Dokuments« sind z.B. »Attribute«, »Attributs«, »Dokumente«, »Dokumenten« und für den gesamten Mehrwort-Term z.B. »Dokumentattribut«, »Dokumentattribute«.

Die Menge der Wörter, die bei der Erkennung eines Mehrwort-Terms ignoriert werden sollen (sog. *SimpleWords*), kann vorgegeben werden (siehe Kapitel 5.7).

Zu jedem Glossareintrag gehören einige allgemeine Attribute:

E99: Jeder Glossareintrag enthält einen eindeutigen Bezeichner, einen Verweis auf den Autor, der den Glossareintrag geschrieben bzw. zuletzt geändert hat, und eine Angabe über das letzte Änderungsdatum.

E100: Jeder Glossareintrag enthält beliebig viele Ursprungsverweise auf Einträge des Quellenkatalogs (siehe Kapitel 5.8).

Term: Modellierer

Definition: »Entwerfer und Realisierer eines SESAM-Modells. Der Modellierer verwendet die Modellierungswerkzeuge zum Zwecke der Modellerstellung oder -modifikation.«

ID:	T75	*Ursprung*:	Q7: Überblick über SESAM-1
Autor:	Melchisedech	*Zustand*:	aktuell
Datum:	16.09.99	*Bereich*:	Anwendungsbereich
Kategorien:	Rolle		
Synonyme:	T76: Modellbauer	*Hyperonyme*:	T27: Benutzer

Beispiel 7: Glossareintrag

5.7 Wortliste

Einige Wörter spielen im ADMIRE-Ansatz eine besondere Rolle, z.B. die in Kapitel 5.6 erwähnten SimpleWords. Einige der Prüfverfahren (Kapitel 8.2) basieren darauf, daß nach bestimmten »verbotenen« Wörtern (z.B. WeakWords) gesucht wird.

Um diese Wörter in Freitexten automatisch erkennen zu können, enthält eine Dokumentation eine *Wortliste*. Jedes Wort in der Wortliste enthält Informationen über seinen Typ. In einigen Fällen kann es vorkommen, daß es sich bei einem »Wort« der Wortliste um eine Phrase handelt (siehe Kapitel 8.2).

E101: Jede Dokumentation enthält genau eine Wortliste.

E102: Jeder Eintrag in der Wortliste enthält eine Phrase.

E103: Jeder Eintrag in der Wortliste ist mindestens einem *Worttyp* zugeordnet.

Es kann durchaus vorkommen, daß ein Wort zu mehreren Worttypen gehört, z.B. »der« gehört sowohl zu dem Worttyp »SimpleWord« als auch zu »Indikator«.

E104: Folgende Worttypen sind im Informationsmodell definiert: »Quantor«, »falsche Konjunktion«, »andere Konjunktion«, »Modalverb«, »Modalverb$_{Anf}$«, »WeakWord«, »SimpleWord« und »Indikator«.

In Kapitel 8 wird beschrieben, für welche Prüfungen welche Worttypen gebraucht werden. Die Wortliste kann jederzeit angepaßt werden. Das heißt:

E105: Es können jederzeit beliebig Wörter in die Wortliste neu aufgenommen und aus ihr entfernt werden.

Weil SimpleWords bei der Erkennung von Mehrwort-Termen automatisch ausgeblendet werden sollen, müssen sie in der Wortliste enthalten sein.

E106: Die Wortliste enthält initial (u.a.[1]) folgende Wörter, denen der Worttyp SimpleWord zugeordnet ist: alle Artikel, Personalpronomen, Reflexivpronomen, Possessivpronomen, Demonstrativpronomen und einige Indefinitpronomen der deutschen Sprache (Tabelle 12).

Wortart	Wörter (ohne Flexionen)
Artikel	der, welcher
Personalpronomen	ich, du, er, sie, es, wir, ihr, mich, dich, ihm, ihn, ihr, ihnen, ihrer, euch
Reflexivpronomen	meiner, mir, mich, deiner, dir, dich, seiner, sich, unser, uns, euer, euch, ihrer
Possessivpronomen	mein, dein, sein, ihr, uns, euer
Demonstrativ-pronomen	dieser, der, die das, dessen, dem, den, deren, denen, jener, derjenige, derselbe, selber, selbst
Indefinitpronomen	jemand, etwas

Tabelle 12: Wichtige SimpleWords der deutschen Sprache

5.8 Quellenkatalog

Die Informationen in einer Dokumentation sind keine Erfindungen der Systemanalytiker, sondern werden durch Befragung von Personen, Lesen von Dokumenten oder Analyse der Arbeitsumgebung gewonnen. Solche Personen und Dokumente werden *Informationsquellen* genannt. Alle relevanten Informationen zu den Informationsquellen werden im *Quellenkatalog* verwaltet.

E107: Jeder Quellenkatalog enthält beliebig viele *Quellenkatalogeinträge.*

E108: Jeder Quellenkatalogeintrag enthält einen Namen und eine Beschreibung.

E109: Der Name eines Quellenkatalogeintrags ist eine Phrase.

E110: Die Beschreibung eines Quellenkatalogeintrags ist ein beliebiger Freitext.

1. Weitere in der initialen Wortliste enthaltene Wörter werden in Kapitel 8.2 angegeben.

Durch den Namen soll eine Informationsquelle innerhalb einer Dokumentation eindeutig identifiziert werden können:

E111: Die Namen der Quellenkatalogeinträge eines Quellenkatalogs müssen paarweise verschieden sein.

Als Quellen kommen Personen, Parteien und externe Dokumente in Frage.

E112: Jeder Quellenkatalogeintrag ist entweder als »Person«, »Partei« oder »externes Dokument« kategorisiert.

Einige Personen nehmen eine besondere Rolle ein: Sie sind gleichzeitig *Autoren*, die Teile der Dokumentation verfassen.

E113: Jeder Quellenkatalogeintrag der Kategorie »Person« kann als Autor markiert werden.

Im Laufe eines Projekts kann es vorkommen, daß eine anfangs für wichtig erachtete Informationsquelle sich als doch nicht relevant erweist. Aus ähnlichen Gründen wie bei Inhaltselementen und Glossareinträgen sollen solche Quellenkatalogeinträge nicht gelöscht werden. Deswegen verfügen Quellenkatalogeinträge über ein Zustandsattribut.

E114: Jeder Quellenkatalogeintrag ist immer in einem der folgenden *Zustände*: »aktuell (Default)«, »aktuell«, »verworfen« oder »TBD«.

Zu jedem Quellenkatalogeintrag gehören einige allgemeine Attribute:

E115: Jeder Quellenkatalogeintrag enthält einen eindeutigen Bezeichner, einen Verweis auf den Autor, der den Quellenkatalogeintrag geschrieben bzw. zuletzt geändert hat, und eine Angabe über das letzte Änderungsdatum.

E116: Jeder Quellenkatalogeintrag enthält beliebig viele Ursprungsverweise auf andere Quellenkatalogeinträge.

Ein externes Dokument wird von Personen geschrieben. Zu einer Partei gehören mehrere Personen. Entsprechende Querverweise können in den Quellenkatalog eingetragen werden.

E117: Jeder Quellenkatalogeintrag der Kategorie »externes Dokument« oder »Partei« enthält beliebig viele Verweise auf Quellenkatalogeinträge der Kategorie »Person«.

Name: Protokoll MiKo-1

Beschreibung: »Protokoll des Mitarbeiterkolloquiums Nr. 1 vom 04.10.1995.«

ID:	Q15	*Zustand*:	aktuell
Autor:	Melchisedech	*Ursprung*:	Q5: Ralf Melchisedech
Datum:	28.08.99	*Kategorie*:	externes Dokument
Verweise:	Q25: Patricia Mandl-Striegnitz, Q24: Anke Drappa, Q23: Marcus Deininger, Q5: Ralf Melchisedech, Q1: Ralf Reißing		

Beispiel 8: Quellenkatalogeintrag

5.9 Mängelkatalog

In der Problemanalyse- und Spezifikationsphase werden über einen längeren Zeitraum von vielen verschiedenen Informanten Informationen erhoben und in die Dokumentation eingetragen. Es kann sein, daß sich die erhobenen Informationen widersprechen oder daß bestimmte Informationen fehlen, weil benötigte Informanten gerade nicht erreichbar sind oder erst noch festgestellt werden muß, welcher Informant überhaupt der Richtige für die gesuchte Information ist.

Dokumentationen können also *Mängel* aufweisen. Auch wenn ein Mangel bekannt ist, ist es nicht immer möglich, ihn sofort zu beheben. Ein einmal erkannter Mangel sollte im Mängelkatalog dokumentiert werden.

E118: Jeder Mängelkatalog enthält beliebig viele *Mangeleinträge.*

E119: Jeder Mangeleintrag enthält eine nicht leere Freitext-Beschreibung.

Mangeleinträge können drei Arten von Verweisen auf Einträge enthalten: Verweise auf Einträge, die den Mangel aufweisen (*Mangelverweise*), Verweise auf Einträge, durch die der Mangel entdeckt oder verursacht wurde, und *Ursprungsverweise.* Wenn z.B. in Anforderung A14 eine projektspezifische Phrase benutzt wird, für die es keinen Glossareintrag gibt, dann ist das Glossar unvollständig. Der zugehörige Mangeleintrag enthält einen Mangelverweis auf das Glossar. A14 selbst ist nicht mangelhaft. In der Beschreibung des Mangeleintrags sollte aber über einen textuellen Querverweis auf A14 verweisen werden.

E120: Jeder Mangeleintrag enthält eine nicht leere Menge von Mangelverweisen auf Inhaltselemente, Annotationen, Systemmodelle, Glossareinträge, Quellenkatalogeinträge, Kapitel, Anforderungsdokumente oder Quellenkataloge.

Um Mängel höherer Ordnung (Mängel von Mängeln) zu vermeiden, dürfen sich Mangeleinträge über Mangelverweise nicht auf Mangeleinträge oder Mängelkataloge beziehen.

E121: Jeder Mangeleintrag enthält beliebig viele Ursprungsverweise auf Quellenkatalogeinträge.

Weil Dokumente abgeschlossen sein sollen (siehe auch E24, S. 77), darf sich ein Mangeleintrag über Mangelverweise nicht auf Einträge verschiedener Dokumente beziehen.

E122: Alle Einträge, auf die sich ein Mangeleintrag über Mangelverweise bezieht, müssen zum gleichen Dokument gehören.

Ein Mangel liegt vor, wenn eines der in Kapitel 7.3 geforderten Qualitätskriterien verletzt ist. Für jedes Qualitätskriterium gibt es eine *Mangelkategorie*:

E123: Jedem Mangeleintrag ist immer genau eine der folgenden Mangelkategorien zugeordnet: »inkorrekt«, »unvollständig«, »nicht adäquat«, »inkonsistent«, »extern inkonsistent«, »überspezifiziert«, »unverständlich«, »nicht realisierbar«, »nicht prüfbar«, »mehrdeutig«, »unpräzise«, »redundant«, »nicht atomar«, »nicht minimal« und »nicht normkonform«.

Im ADMIRE-Ansatz werden folgende *Arten* von Mangeleinträgen unterschieden (siehe auch Kapitel 8):

- *automatisch erzeugter Mangeleintrag*: Ein Werkzeug hat bei einer Prüfung ein Indiz für einen Mangel gefunden und einen Mangeleintrag erzeugt.
- *manuell erzeugter Mangeleintrag*: Der Mangeleintrag wurde von einem Systemanalytiker erzeugt.

E124: Jeder Mangeleintrag enthält eine Angabe über seine Art.

Einige (wenige) Mängel werden durch Zufall oder durch Benutzung der Dokumentation entdeckt. Es ist aber notwendig, Qualitätssicherungsmaßnahmen durchzuführen, wenn eine gute Qualität erreicht werden soll. Um hinterher feststellen zu können, wie effektiv eine Qualitätssicherungsmaßnahme war, kann sie jeweils angegeben werden.

E125: Jeder Mangeleintrag enthält einen Freitext zur Beschreibung der Qualitätssicherungsmaßnahme, in der er gefunden wurde.

Wenn ein Mangel gefunden wird, dann hat der Finder oft schon Lösungsvorstellungen im Kopf. Da der Mangel aber nicht immer sofort behoben werden kann, ist es sinnvoll, diese Vorstellungen zu dokumentieren.

E126: Jeder Mangeleintrag enthält einen Freitext für Lösungsbeschreibungen.

Bevor ein Mangel behoben werden kann, müssen u.U. noch offene Punkte geklärt oder Entscheidungen getroffen werden. Diese können dokumentiert werden.

E127: Jeder Mangeleintrag enthält einen Freitext, in dem offene Punkte und Entscheidungen beschrieben werden können.

E128: Jeder Mangeleintrag enthält einen Marker, durch den das Vorhandensein offener Punkte angezeigt werden kann.

Im Laufe seiner Existenz kann ein Mangeleintrag verschiedene *Zustände* annehmen:

- potentiell: Bei einer Prüfung wurde ein (schwaches) Indiz für einen Mangel gefunden. Es ist nicht sicher, ob der Mangel wirklich vorliegt.
- aktuell: Der Mangel liegt wirklich vor und soll behoben werden.
- behoben: Die Dokumentation wurde so geändert, daß ein ehemals vorhandener Mangel jetzt nicht mehr vorliegt.
- verworfen: Entweder liegt doch kein Mangel vor, oder es wurde entschieden, den Mangel nicht zu beheben.

E129: Jeder Mangeleintrag ist immer in genau einem der folgenden Zustände: »potentiell«, »aktuell«, »behoben« oder »verworfen«.

Zu jedem Mangeleintrag gehören einige allgemeine Attribute:

E130: Jeder Mangeleintrag enthält einen eindeutigen Bezeichner, einen Verweis auf den Autor, der den Mangel geschrieben bzw. zuletzt geändert hat, und eine Angabe über das letzte Änderungsdatum.

Beschreibung: »In der Anforderung wird ein nicht aktueller Term (Modellbauer) benutzt«

ID:	D14	*Version*:	1.0
Autor:	Melchisedech	*Ursprung*:	Q5: Ralf Melchisedech
Datum:	28.08.99	*Art*:	manuell erzeugt
Mangelverweise:	A73	*Mangelkategorie*:	inkonsistent

QS-Maßnahme: »Review des Lastenhefts vom 15.09.99«

Lösungsvorschlag: »›Modellbauer‹ durch ›Modellierer‹ ersetzen.«

Entscheidungen: -

offene Punkte: keine

Beispiel 9: Mangeleintrag

5.10 Qualitätskriterien

In den vorherigen Unterkapiteln 5.1-5.9 wurde beschrieben, welche Informationen in einer Dokumentation enthalten sein können. Daß diese Informationen »sinnvoll« oder »richtig« sein sollen, wurde bisher nicht gefordert. Insofern können Dokumentationen von sehr unterschiedlicher Qualität sein. Ein Ziel des ADMIRE-Ansatzes ist es, qualitativ hochwertige Dokumentationen zu erstellen. Hierfür werden folgende *Qualitätskriterien* definiert: »korrekt«, »vollständig«, »adäquat«, »konsistent«, »extern konsistent«, »nicht redundant«, »nicht überspezifiziert«, »realisierbar«, »überprüfbar«, »atomar«, »verständlich«, »eindeutig«, »präzise«, »minimal« und »normkonform«.

Die Qualitätskriterien sind auf Einträgen des Informationsmodells definiert. Hierbei muß zwischen Einzeleinträgen, NL-Einträgen und Dokumenteinträgen unterschieden werden. Ein *Einzeleintrag* ist entweder ein Inhaltselement, eine Annotation, ein Systemmodell, ein Glossareintrag, ein Quellenkatalogeintrag, ein Kapitel oder der Wurzeleintrag[1] eines Anforderungsdokuments. Ein *NL-Eintrag* ist ein Einzeleintrag, der einen Freitext enthält, also entweder ein Inhaltselement, eine Annotation, ein Glossareintrag, ein Quellenkatalogeintrag, ein Kapitel oder der Wurzeleintrag eines Anforderungsdokuments. Ein *Dokumenteintrag* ist entweder ein Pflichtenheft, ein Lastenheft, eine Anforderungssammlung, ein Ist-Analysedokument, ein Glossar oder ein Quellenkatalog. Ist ein Qualitätskriterium an einer Stelle nicht erfüllt, dann liegt eventuell ein Mangel vor (siehe Kapitel 5.9). Da es keine Mängel höherer Ordnung geben soll, werden auf Mangeleinträgen und Mängelkatalogen keine Qualitätskriterien definiert.

Einige der Qualitätskriterien können nicht sinnvoll binär definiert werden. Eine Anforderung ist nicht entweder verständlich oder unverständlich, sie kann mehr oder weniger verständlich sein. Andererseits sollen die Qualitätskriterien als Zielvorgaben dienen, d.h. die ihnen zugrundeliegenden Bedingungen sollen erfüllt werden. Bei den Definitionen wird deswegen jeweils eine Schwelle angegeben. So

1. Der Wurzeleintrag eines Anforderungsdokuments ist genau der Eintrag, der die in E3 bis E7 (S. 75) beschriebenen Attribute enthält.

ist z.B. das Qualitätskriterium Verständlichkeit erfüllt, wenn der Eintrag hinreichend verständlich ist.

Korrektheit

Ein *Einzeleintrag* ist hinreichend *korrekt*, wenn die Attribute und Verweise des Eintrags objektiv richtig sind, oder, falls die Korrektheit nicht objektiv beurteilbar ist, wenn es eine Person gibt, die diesen Eintrag für richtig hält, oder wenn es ein für das Projekt relevantes externes Dokument gibt, aus dem der Eintrag abgeleitet werden kann.

Eine Anforderung ist somit hinreichend korrekt, wenn sie einen Bedarf mindestens eines Informanten bzgl. des Zielsystems beschreibt. Ein Glossareintrag ist hinreichend korrekt, wenn der Term von mindestens einem Informanten in der durch die Definition angegebenen Bedeutung benutzt wird. Ein Quellenkatalogeintrag ist hinreichend korrekt, wenn hierdurch auf eine Informationsquelle verwiesen wird, die von mindestens einem Informanten als für das Projekt relevant betrachtet wird.

Wichtig bei dieser Definition von Korrektheit ist, daß sie sich auf jeweils einen einzelnen Eintrag bezieht. Zwei sich widersprechende Anforderungen können jeweils einzeln korrekt sein.

Vollständigkeit

Das Qualitätskriterium *Vollständigkeit* ist erfüllt, wenn ein Eintrag alle für das Projekt relevanten und zur Eintragsart passenden Informationen enthält. Vollständigkeit ist sowohl auf Einzeleinträgen als auch auf Dokumenteinträgen definiert.

Ein *Einzeleintrag* ist hinreichend *vollständig*, wenn seine Attribute immer dann mit Werten belegt sind, wenn die Werte auch wirklich benötigt werden, wenn bei mengenwertigen Attributen oder Verweisen alle relevanten Werte bzw. Einträge angegeben sind und wenn kein TBD-Wert verwendet wird.

Damit ein *Dokumenteintrag* hinreichend *vollständig* ist, müssen die in Tabelle 13 genannten Bedingungen erfüllt sein.

Dokument	Bedingungen
Ist-Analyse-dokument	Alle für das Projekt relevanten Fakten über das aktuelle System, seine Arbeitsumgebung und die bisherigen Arbeitsabläufe und alle Probleme, die die Informanten in der bisherigen Lösung sehen, werden beschrieben.
Anforderungs-sammlung	Alle während einer Informationserhebungstätigkeit ermittelten und für das Projekt relevanten Probleme, Fakten, Anforderungen und Zusicherungen werden beschrieben.
Lastenheft	Alle für das Projekt relevanten Probleme, Fakten, Anforderungen und Zusicherungen aus Sicht aller bisher identifizierten aktuellen Informanten werden beschrieben.

Tabelle 13: Vollständigkeitsbedingungen für Dokumenteinträge

Dokument	Bedingungen
Pflichtenheft	• Alle für das Projekt relevanten Probleme, Fakten, Anforderungen und Zusicherungen bzgl. des Zielsystems, seiner Arbeitsumgebung und des Projekts werden beschrieben. • Das Verhalten (inkl. Fehlermeldungen) des Zielsystems zu jeder Klasse von Eingabe- und internen Ereignissen in jeder Klasse von Zuständen wird beschrieben. • Alle für das Zielsystem relevanten Qualitäten werden beschrieben (siehe auch Abbildung 10, S. 85).
Quellenkatalog	Für jede Informationsquelle, die Informationen für das Projekt liefern kann, gibt es einen Eintrag.
Glossar	Für jede potentiell mehrdeutige, unverständliche oder projektspezifische Phrase, die in einem in der Dokumentation enthaltenen Freitext benutzt wird, gibt es einen entsprechenden Glossareintrag.

Tabelle 13: Vollständigkeitsbedingungen für Dokumenteinträge

Adäquatheit

Das Qualitätskriterium *Adäquatheit* ist erfüllt, wenn die Probleme, Wünsche und Rahmenbedingungen aus Sicht der Kunden und sonstigen Informanten angemessen beschrieben werden. Notwendige Voraussetzung ist, daß die Anforderungen im richtigen Detaillierungsgrad beschrieben sind und daß für quantitative Angaben die angemessene Genauigkeit verwendet wird. Adäquatheit ist immer abhängig von der Problemstellung des Projekts. Für eine Echtzeitanwendung kann es z.B. adäquat sein, die Reaktionszeit in Millisekunden zu beschreiben. Für die Spezifikation des SESAM-2-Systems ist eine Genauigkeit im Sekundenbereich eher angemessen. Adäquatheit ist auf Inhaltselementen definiert.

Ein *Inhaltselement* ist hinreichend *adäquat*, wenn die Anforderung, die Zusicherung, das Problem oder das Fakt aus Sicht mindestens eines Informanten und bezogen auf die aktuelle Problemstellung angemessen beschrieben ist.

Konsistenz

Das Qualitätskriterium *Konsistenz* ist erfüllt, wenn eine Dokumentation keine Widersprüche enthält. Wenn sich z.B. zwei Anforderungen widersprechen, kann das Zielsystem nicht realisiert werden, ohne mindestens eine der Anforderungen zu verletzen.

Ein *Einzeleintrag* ist hinreichend *konsistent*, wenn sich seine Attributwerte nicht widersprechen und wenn sie zum Eintragstyp passen. Ein Beispiel für einen nicht konsistenten Eintrag ist ein Inhaltselement, in dessen Freitext ein Begriff definiert wird. Ein *Systemmodell* soll zusätzlich folgende Bedingungen erfüllen:

- Die Systemmodelleinträge sollen paarweise verschiedene Namen haben.
- Das Systemmodell soll zusammenhängend sein.

Eine *Menge von Einzeleinträgen* ist hinreichend *konsistent*, wenn keine Teilmenge einen Widerspruch enthält und wenn folgende Bedingungen erfüllt sind:

- Einige Einträge enthalten *Ursprungsverweise*. Da ein Eintrag nicht (indirekt) Ursprung von sich selbst sein kann, sollen die Ursprungsverweise keine Zyklen aufweisen.
- Einige Einträge können den *Zustand* »verworfen« annehmen. Eine Inkonsistenz liegt vor, wenn auf einen verworfenen Eintrag verwiesen wird. Folgende Bedingungen müssen also erfüllt sein:
  - In den Freitexten sollen nur aktuelle Glossareinträge benutzt werden.
  - Eine Annotation soll nur an aktuelle Einträge gebunden werden.
  - Die Ursprünge eines Eintrags sollen nicht verworfen sein.
  - Ein Quellenkatalogeintrag soll nicht auf verworfene Quellenkatalogeinträge der Kategorie »Person« verweisen.
- Eine Menge von Glossareinträgen sollte folgende Bedingungen erfüllen:
  - Um Homonymie zu vermeiden, sollen die Glossareinträge eines Glossars paarweise verschiedene Terme haben.
  - Das Glossar des Anwendungsbereichs soll in sich abgeschlossen sein, d.h. in der Definition eines Glossareintrags des Anwendungsbereichs sollen keine projektspezifischen Glossareinträge benutzt werden.
  - Jeder Sachverhalt soll durch maximal einen Term beschrieben werden. Deswegen soll in einer Menge synonymer Glossareinträge maximal einer aktuell sein.
  - Da ein Term nicht (indirekt) Oberbegriff von sich selbst sein kann, soll die Hyponymiebeziehung nicht zyklisch sein.
  - In jedem Glossareintrag sollen alle Synonyme und Hyperonyme angegeben sein. Wenn g_1 synonym zu g_2 und g_2 synonym zu g_3 ist, dann ist g_1 auch synonym zu g_3. Die Menge der Synonym-Verweise bzw. Hyperonym-Verweise eines Glossareintrags soll auch Verweise auf alle indirekten Synonyme bzw. Hyperonyme enthalten.
  - Synonyme Glossareinträge sollen gleich kategorisiert sein.

Externe Konsistenz

Das Qualitätskriterium *externe Konsistenz* ist erfüllt, wenn die Einträge in einer Dokumentation keine (inhaltlichen) Widersprüche zu projektrelevanten externen Dokumenten enthalten. Strukturelle Widersprüche z.B. zwischen einem Anforderungsdokument und einer Spezifikationsnorm werden durch das Qualitätskriterium »Normkonformität« abgedeckt.

Eine *Menge von Einzeleinträgen* ist hinreichend *extern konsistent* zu *externen Dokumenten*, wenn keine Teilmenge einen Widerspruch zu den Aussagen der externen Dokumente enthält.

Redundanzfreiheit

Redundanz liegt immer dann vor, wenn die gleiche Aussage mehrfach in einer Dokumentation vorkommt. Redundanz ist gut für die Verständlichkeit eines Textes, weil sie ermöglicht, daß Informationen immer an den Stellen stehen können, wo sie gebraucht werden; auch mehrfach, wenn nötig. Insofern dürfen (und sollen) Annotationen redundant sein. Für Inhaltselemente ist Redundanz aber eher eine schlechte Eigenschaft.

Eine *Menge von Inhaltselementen* ist hinreichend *nicht redundant*, wenn jede Aussage in maximal einem der Inhaltselemente vorkommt.

Keine Überspezifikation

Ein Ziel der Requirements-Engineering-Phase ist es, ein Zielsystem zu spezifizieren, das genau das tut, was die Kunden erwarten. Werden einige Wünsche nicht berücksichtigt, ist die Dokumentation unvollständig. Es ist aber auch schlecht, wenn zuviel vorgegeben wird, z.B. bestimmte, nicht unbedingt notwendige Entwurfsstrukturen oder zu verwendende Algorithmen. Dieser Effekt wird *Überspezifikation* genannt.

Ein *Inhaltselement der Kategorie Anforderung* ist hinreichend *nicht überspezifiziert*, wenn die enthaltene Anforderung nicht einschränkender ist, als es aus Sicht der Informanten wirklich notwendig ist.

Realisierbarkeit

Ein Projekt kann nur dann erfolgreich durchgeführt werden, wenn die Zielsysteme realisierbar sind. Ein Zielsystem ist z.B. dann nicht realisierbar, wenn es ein nicht entscheidbares Problem lösen soll oder wenn die heutige Hardware nicht leistungsstark genug ist, um die Zeit- oder Mengenanforderungen zu erfüllen.

Eine Menge von *Inhaltselementen der Kategorie Anforderung* ist *realisierbar*, wenn es eine Implementierung gibt, die alle diese Anforderungen gleichzeitig erfüllt.

Überprüfbarkeit

Die Eigenschaften der Zielsysteme, der Systeme und Personen der Arbeitsumgebung, der Schnittstellen der Systeme und evtl. auch die Eigenschaften des Projekts werden durch Inhaltselemente beschrieben. Bei der Abnahme wird das Zielsystem daraufhin geprüft, ob es die Anforderungen erfüllt. Die zugehörigen Inhaltselemente müssen so beschaffen sein, daß diese Prüfung auch möglich ist, daß also zweifelsfrei festgestellt werden kann, ob das Zielsystem die Anforderungen erfüllt. Analoge Argumente gelten auch für Zusicherungen und Fakten. Für Probleme wird Überprüfbarkeit nicht gefordert, weil Probleme oft (mit Absicht) sehr abstrakt und vage formuliert werden.

Ein *Inhaltselement der Kategorien Anforderung, Fakt* oder *Zusicherung* ist hinreichend *überprüfbar*, wenn es eine endliche, kosteneffektive Technik gibt, mit der geprüft werden kann, ob die Systeme, Schnittstellen und Personen oder das Projekt die in dem Inhaltselement geforderte Eigenschaft aufweisen.

Atomizität

Oftmals muß auf einzelne Einträge Bezug genommen werden, z.B. in dem Protokoll eines Reviews, in dem ein Mangel in einer Anforderung beschrieben wird. Wenn das zugehörige Inhaltselement aber zwei Anforderungen oder eine Anforderung und eine Definition eines Terms enthält, werden die Verweise sehr kompliziert.

Ein *Inhaltselement* ist hinreichend *atomar*, wenn es nicht sinnvoll in mehrere Einträge aufgespaltet werden kann. Ein *Glossareintrag* ist *atomar*, wenn durch ihn genau einem Term eine Bedeutung zugewiesen wird. Ein *Quellenkatalogeintrag* ist *atomar*, wenn durch ihn auf genau eine Informationsquelle verwiesen wird.

Für Annotationen wird Atomizität nicht gefordert, weil es z.B. sinnvoll sein kann, als Erklärung für einen Sachverhalt mehrere Beispiele anzugeben. Auf alle anderen Eintragstypen läßt sich das Qualitätskriterium nicht sinnvoll anwenden.

Verständlichkeit

Verständlichkeit ist ein wichtiges Kriterium für eine Dokumentation. Nur wenn die Informanten die Einträge verstehen, können sie sie validieren oder das Zielsystem realisieren. Verständlichkeit ist nur auf solchen Einträgen definiert, die einen Freitext enthalten.

Ein *NL-Eintrag* ist hinreichend *verständlich*, wenn alle enthaltenen Freitexte so geschrieben sind, daß jeder erwartete Leser deren Bedeutung erfassen kann, wenn er zusätzlich auf die anderen Einträge des gleichen Dokuments und des Glossars zugreifen kann.

Eindeutigkeit

Eindeutigkeit bezieht sich, analog zu Verständlichkeit, auf beliebige NL-Einträge. Ein *NL-Eintrag* ist hinreichend *eindeutig*, wenn alle darin enthaltenen Freitexte für alle zu erwartenden Leser genau eine Bedeutung haben. Es darf also kein Spielraum für Interpretationen mehr vorhanden sein.

Präzision

Präzision ist eine wichtige Eigenschaft für Inhaltselemente, Glossareinträge und Quellenkatalogeinträge. Ein *Inhaltselement*, ein *Glossareintrag* oder ein *Quellenkatalogeintrag* ist hinreichend *präzise*, wenn alle enthaltenen Freitexte so genau sind, wie es für die Problemstellung erforderlich ist, und wenn in den Inhaltselementen quantitative Angaben soweit wie möglich verwendet werden.

Annotationen müssen, da sie Erklärungen und Beispiele enthalten können, nicht unbedingt präzise sein. Auf alle anderen Eintragstypen läßt sich das Qualitätskriterium nicht sinnvoll anwenden.

Minimalität

Je kürzer eine Dokumentation ist, desto weniger Zeit muß man aufwenden, um sie zu lesen. Oft sind kürzer formulierte Sachverhalte auch prägnanter formuliert. Kürze bzw. *Minimalität* ist also wünschenswert, so lange sie nicht auf Kosten an-

derer Qualitätskriterien erreicht wird. Das Qualitätskriterium »Minimalität« ist auf NL-Einträgen und auf Anforderungsdokumenten definiert.

Ein *NL-Eintrag* ist hinreichend *minimal*, wenn die Freitexte keine unnötigen oder kürzer formulierbaren Teile (z.B. Füllwörter) enthalten. Ein *Anforderungsdokument* ist hinreichend *minimal*, wenn es keine unnötigen Einträge enthält.

Normkonformität

In vielen Firmen (siehe Kapitel 3) gibt es Vorlagen (Templates) oder Normen für den Aufbau eines Anforderungsdokuments. Typischerweise werden dabei die oberen ein oder zwei Ebenen der Kapitelhierarchie vorgegeben. Durch den normierten Aufbau soll der Umgang mit den Dokumenten erleichtert werden. Deswegen sollten alle Anforderungsdokumente die Vorgaben der Norm erfüllen.

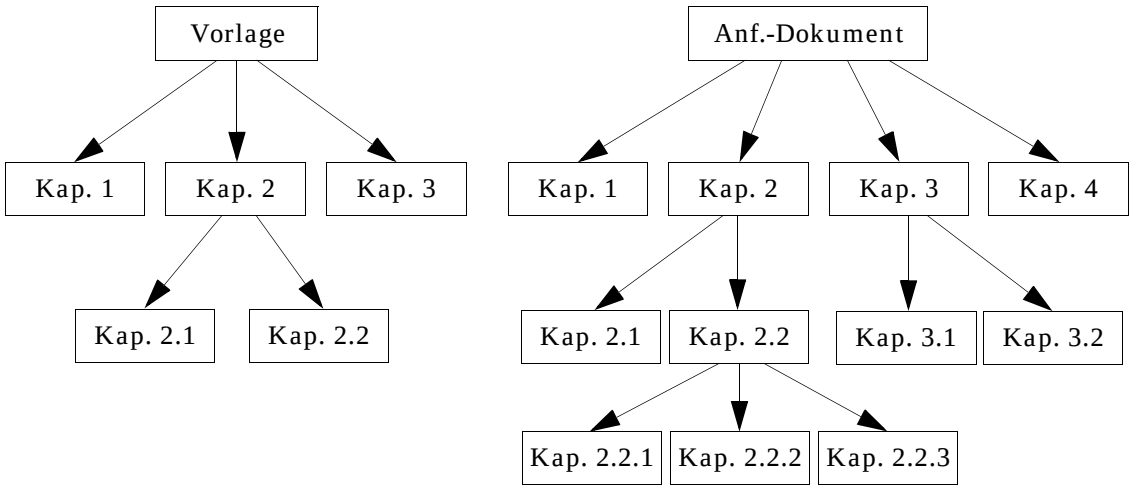


Abbildung 12: Strukturen einer Vorlage (links) und eines dazu konformen Anforderungsdokuments (rechts)

Ein *Anforderungsdokument* ist *konform zu einer gegebenen Norm*, wenn es genauso strukturiert ist, wie durch die Norm festgelegt wird. Es müssen alle Kapitel, die durch die Norm vorgegeben sind, in der gleichen Reihenfolge vorhanden sein. Die Überschriften der Kapitel müssen mit denjenigen der Norm übereinstimmen. Ein Anforderungsdokument darf zusätzliche Kapitel enthalten: Jedes Kapitel darf am Ende zusätzliche Unterkapitel enthalten. Ebenso dürfen die durch die Vorlagen-Kapitelhierarchie vorgegebenen Blattkapitel weiter verfeinert werden (siehe Abbildung 12).

5.11 Sinnvolle Vorbelegungen

Wenn die Dokumentation mit einem Werkzeug erstellt wird, können einige Attribute automatisch berechnet oder sinnvoll vorbelegt werden, um dem Systemanalytiker die Arbeit zu erleichtern.

Einige Attribute und Querverweise können automatisch berechnet werden (Tabelle 14). Falls ein Werkzeug eingesetzt wird, sollte es eine manuelle Änderung der Attribute verhindern.

Eintrag	Attribut	Wert
alle Einträge	eindeutige ID	noch unbenutzte ID
	Änderungsdatum	aktuelles Datum
	Autor	aktuell angemeldeter Benutzer
Glossareintrag	Synonym-Verweis	Zusammen mit einem Synonym-Verweis sollte automatisch der umgekehrte Synonym-Verweis eingetragen werden, um die Symmetrie (E94, S. 92) sicherzustellen.
Mangeleintrag	Art	manuell oder automatisch erzeugt
automatisch erzeugter Mangeleintrag	Beschreibung	(siehe Kapitel 8.2)
	Mangelverweise	(siehe Kapitel 8.2)
	Mangelkategorie	(siehe Kapitel 8.2)
	Zustand	(siehe Kapitel 8.1.1)

Tabelle 14: Automatisch berechenbare Attribute und Querverweise

Für einige Attribute und Querverweise können sinnvolle Vorbelegungen angegeben werden (Tabelle 15). Sie dürfen geändert werden.

Alle anderen hier nicht aufgeführten Attribute und Querverweise sind bei Erzeugung eines neuen Eintrags leer.

Eintrag	Attribut/Verweise	Wert
Quellenkatalogeintrag	Zustand	aktuell (Default)
Quellenkatalogeintrag (»Person«)	Autor-Marker	true
Glossareintrag	Zustand	aktuell (Default)
	Bereich	projektspezifisch
	Flexionen	(siehe unten)
Inhaltselement	Zustand	aktuell (Default)
	Stabilität	(angegebener Default-Wert)
Inhaltselement (»Anforderung«)	Projektstand	nicht erfüllt (Default)
Inhaltselement (»Problem«, »Anforderung«)	Priorität	(angegebener Default-Wert)

Tabelle 15: Sinnvolle Vorbelegungen für Attribute und Querverweise

Eintrag	Attribut/Verweise	Wert
Annotation	Ursprung	Ursprünge der Einträge, auf die die Annotation verweist
	zugeordnete Einträge	zuletzt angezeigter Eintrag
Mangeleintrag	Marker (offene Punkte)	false
manuell gefundener Mangeleintrag	Mangelkategorie	sonstige
	Mangelverweise	zuletzt angezeigter Eintrag
	Zustand	aktuell

Tabelle 15: Sinnvolle Vorbelegungen für Attribute und Querverweise

Interessant ist die Frage, welche Flexionen für einen Glossareintrag voreingestellt werden. Meiner Einschätzung nach wird in naher Zukunft die Forschung in den Bereichen künstliche Intelligenz und natürliche Sprachverarbeitung so weit fortgeschritten sein, daß zu jedem Wort der deutschen Sprache alle Flexionen sicher erkannt werden können. Man kann sich aber auch mit relativ einfachen Heuristiken helfen, die aus den Regeln der deutschen Grammatik (Duden, 1998, Tabelle 16) abgeleitet werden können.

Wortart	Präfixe und Postfixe	Beispiele
regelmäßige Substantive	Endungen: -e, -es, -en, -s, -n, -er, -r, -se, -ens, -ns	Tisch-e, Buch-es, Frau-en, Lehrer-s, Hase-n, Geist-er, Hindernis-se, Herz-ens
Sonderfälle bei Substantiven	is → en, itis → itiden, us → i en → ina - → a, ta y → ies on → a a, o, on, um, us → en	Dosis → Dosen Bronchitis → Bronchitiden Tempus → Tempi Pronomen → Pronomina Abstrakt → Abstrakta Party → Parties Lexikon → Lexika Drama → Dramen
regelmäßige Verben	Infinitivendung: -en Präsens: -e, -st, -t 1. Partizip: -end[a] 2. Partizip: ge-, -t[b] Präterium: -te, -test, tet,- ten	sagen sage, sagt, sagst sagend gesagt sagte, sagten
regelmäßige Adjektive	Endungen: -e, -er, -es, -en, -em Komparativ-Endungen: -er, -ere, -erer, - eres, -eren, -erem Superlativ-Endungen: -sten, -stem, -ster, -ste, -esten, -estem, -ester, -este, -estes	schnell, schnelle schneller, schnellere schnellste, schnellster

Tabelle 16: Präfixe und Postfixe für Flexionen

a. das erste Partizip kann auch dekliniert werden

b. das zweite Partizip kann auch dekliniert werden

Ausgehend von der Annahme, daß der Term eines Glossareintrags in der Grundform vorliegt (siehe auch Kapitel 5.6), können mit Hilfe der in Tabelle 16 angegebenen Präfixe und Postfixe fast alle Flexionen der deutschen Sprache gebildet werden.

Zuerst muß der Term in einzelne maximale Wörter zerlegt werden. Da Simple-Words (Tabelle 12, S. 95) bei der Erkennung von Mehrwort-Termen ignoriert werden sollen, werden sie nicht weiter berücksichtigt. Für alle anderen Wörter werden Flexionen erzeugt:

1. Zuerst müssen Kandidaten für den Wortstamm gefunden werden. Diese entstehen dadurch, daß die in Tabelle 16 beschrieben Postfixe abgeschnitten werden (soweit sie vorhanden sind).
2. Anschließend werden an die Kandidaten für den Wortstamm nacheinander alle in Zeile 1 und 2 beschriebenen Postfixe angehängt.
3. Handelt es sich bei einem Wort evtl. um ein Verb, was an dem Postfix »en« erkannt werden kann, werden auch alle in Zeile 3 genannten Präfixe und Postfixe vor bzw. hinter die Kandidaten für den Wortstamm gehängt.
4. Handelt es sich bei einem Wort evtl. um ein Adjektiv, was daran erkannt werden kann, daß es klein geschrieben wird, werden an die Kandidaten für den Wortstamm auch alle in Zeile 4 beschriebenen Postfixe gehängt.

Die Kandidaten für den Wortstamm und die erzeugten Flexionen werden als Flexionen für die maximalen Wörter des Terms in das Glossar eingetragen.

Auf diese Weise entstehen sehr viele Flexionen, auch solche, die in der deutschen Sprache nicht vorkommen. Unregelmäßige Wörter werden bei diesem Verfahren nicht berücksichtigt. Falls hierdurch Probleme auftreten, muß der Systemanalytiker eingreifen, indem er überflüssige Flexionen entfernt oder weitere Flexionen einträgt.

Kapitel 6

Formales Modell

In diesem Kapitel wird, aufbauend auf der informalen Beschreibung des Informationsmodells in Kapitel 5, ein formales Modell angegeben. In Kapitel 6.1 wird die formale Notation eingeführt und es werden grundlegende Mengen definiert. Anschließend werden in den Kapiteln 6.2 bis 6.9 die Eintragstypen des Informationsmodells in der formalen Notation beschrieben: Eintrag (Kapitel 6.2), Quellenkatalog und Quellenkatalogeintrag (Kapitel 6.3), Glossar und Glossareintrag (Kapitel 6.4), Systemmodell (Kapitel 6.5), Inhaltselement (Kapitel 6.6), Annotation (Kapitel 6.7), Anforderungsdokument (Kapitel 6.8) und Mängelkatalog und Mangeleintrag (Kapitel 6.9). In Kapitel 6.10 wird das Metaspezifikationsformular vorgestellt. Anschließend werden in Kapitel 6.11 alle Einträge zum Informationsmodell zusammengefügt. In Kapitel 6.12 werden einige Hilfsmengen eingeführt. Abschließend werden in Kapitel 6.13 Qualitätsprädikate definiert.

6.1 Formales Modell für informale Spezifikationen

Um das ADMIRE-Informationsmodell möglichst präzise und eindeutig beschreiben zu können, wird eine formale Notation verwendet. Die Notation basiert auf *Mengen*. Mengen können auf zwei Arten definiert werden:

- *extensional*, d.h. durch Angabe der Elemente der Menge. Hierfür wird die übliche Mengenschreibweise verwendet: $M=\{a,b,c,d,e,f,g\}$.
- *intensional*, d.h. durch Angabe einer Entscheidungsvorschrift, anhand der ermittelt werden kann, ob ein Element zu der Menge gehört. Hierfür werden eine UML-ähnliche Notation und Prädikatenlogik 1. Stufe verwendet.

6.1.1 Grundlegende Mengen

Sei S eine gegebene Menge. Die *Potenzmenge* von S wird mit $P(S)$ bezeichnet. Die Menge *Boolean* enthält die Wahrheitswerte *true* und *false*: $Boolean = \{true, false\}$. Die Menge N enthält alle natürlichen Zahlen inkl. 0: $N = \{0, 1, 2, 3, \ldots\}$. Die Menge R enthält alle reellen Zahlen.

Alphabet

Im ADMIRE-Ansatz spielen Freitexte, Wörter und Phrasen eine wichtige Rolle. Diese Mengen basieren alle auf dem *Alphabet* Σ. Das Alphabet wird nicht fest vorgegeben. Es muß aber zumindest alle *Buchstaben* (*Letter*) der deutschen Sprache

(inkl. Umlaute und »ß«), alle *Ziffern* (*Digit*), das *Leerzeichen* (*Blank*) und die bisher noch nicht genannten Zeichen des ASCII-Codes (*ASCII*) enthalten.

$Letter = \{ \text{'a'}, ..., \text{'z'}, \text{'A'}, ..., \text{'Z'}, \text{'ä'}, \text{'ö'}, \text{'ü'}, \text{'Ä'}, \text{'Ö'}, \text{'Ü'}, \text{'ß'} \}$

$Digit = \{ \text{'0'}, ..., \text{'9'} \}$

$Blank = \{ \text{' '} \}$

$\Sigma \supset Letter \cup Digit \cup Blank \cup ASCII$

Ein Element aus Σ wird *Zeichen* genannt.

Das Alphabet wurde so gewählt, daß alle der deutschen Rechtschreibung und Grammatik genügenden Aussagen formuliert werden können. Anpassungen an andere Sprachen können durch Änderung dieser Mengen vorgenommen werden. Der Rest des Informationsmodells enthält keine weiteren Sprachabhängigkeiten.

Freitext, Wort und Phrase

Ein *Freitext* (*Paragraph*) ist eine beliebig lange, evtl. leere Folge von Zeichen des Alphabets Σ. Der *leere Freitext* wird mit ε bezeichnet.

$$Paragraph = \{a_0 \ldots a_n \mid n \in N \wedge \forall i \in \{0, \ldots, n\} \bullet a_i \in \Sigma\} \cup \varepsilon$$

Ein *Wort* (*Word)* ist eine nicht leere Folge von Zeichen, die nur aus Buchstaben, Ziffern und den in der Menge *WordMark* enthaltenen Zeichen bestehen darf.

$$WordMark = Letter \cup Digit \cup \{\text{'-'}, \text{'\_'}, \text{'.'}\}$$

$$Word = \{a_0 \ldots a_n \mid n \in N \wedge \forall i \in \{0, \ldots, n\} \bullet a_i \in WordMark\}$$

Eine *Phrase* (*Phrase*) ist eine nicht leere Folge von Wörtern, die durch Leerzeichen oder Kommata getrennt werden:

$$Phrase = \{w_0 b_1 w_1 \ldots b_n w_n \mid n \in N \wedge w_0 \in Word \wedge \forall i \in N^{\geq 1} \bullet w_i \in Word \wedge b_i \in Blank \cup \{\text{','}\}\}$$

Ein *Datum* (*Date*) ist eine spezielle Phrase, die auf einen gültigen Kalendertag verweist, also z.B. »11.02.1999« oder »17. November 1987«. Der Aufbau der Phrase wird nicht vorgeschrieben.

Im ADMIRE-Informationsmodell gibt es das spezielle Wort »TBD«, das benutzt werden kann, um ein Wort, eine Phrase oder einen Freitext als unvollständig zu kennzeichnen. Der Zusammenhang zwischen Wörtern, Phrasen, Freitext, Datum und »TBD« wird in Abbildung 13 gezeigt.

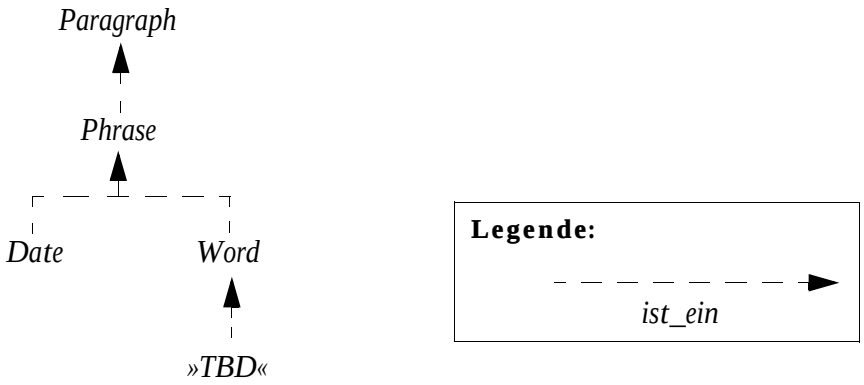


Abbildung 13: Hierarchie der grundlegenden Mengen

6.1.2 UML (Unified Modeling Language)

Die Struktur des ADMIRE-Informationsmodells wird durch eine eingeschränkte Variante der *UML* beschrieben. Auch wenn die UML strenggenommen keine formale Notation ist, kann hier von einem formalen Modell gesprochen werden, weil die verwendete Variante mit formaler Semantik belegt wird. Folgende Teile der UML werden verwendet (siehe Abbildung 15):

- *Klasse*: Die Klassennotation wird verwendet, um *strukturierte Mengen* darzustellen. Dies ist eine durchaus gültige Sichtweise für Klassen, weil Klassen als Typen und Typen wiederum als Mengen aufgefaßt werden können. Zu jeder Klasse gehören Attribute, die einen definierten Typ haben. Klassen haben keine Methoden, weil durch das Informationsmodell kein Verhalten modelliert wird.
- *Assoziation*: Mengen können mit anderen Mengen über binäre, gerichtete, benannte Assoziationen verbunden sein. Zu jeder Assoziation wird eine *Kardinalität* angegeben. Eine Assoziation kann als mengenwertiges Attribut aufgefaßt werden (Abbildung 14).

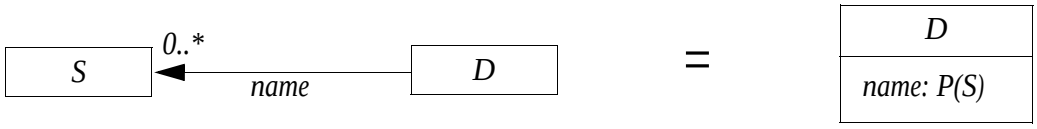


Abbildung 14: Assoziationen und mengenwertige Attribute

- *Vererbung*: Hier wird nur die strukturelle Vererbung verwendet. Die Unterklasse erbt (unverändert) alle Attribute und Assoziationen der Oberklasse. Substituierbarkeit ist gegeben, d.h. die Elemente der Unterklasse können immer dann verwendet werden, wenn Elemente der Oberklasse erwartet werden. Mehrfachvererbung ist nicht möglich.

6.1.3 Prädikatenlogik

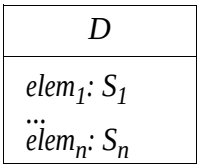
Durch *prädikatenlogische* Formeln können Bedingungen an Mengen formuliert werden, die über die Ausdrucksmächtigkeit der (hier verwendeten Variante der) UML hinausgehen. Folgende vordefinierten Mengen, Funktionen und Assoziationen werden in den prädikatenlogischen Formeln verwendet:

- N-fache Anwendung einer Funktion: Gegeben seien Mengen D und U. D sei eine Oberklasse von U, f eine Funktion $f: D \rightarrow U$ und e_0 ein Element aus D. Dann gilt:
  $e_0 = f^0(e_0)$, $f^n(e_0) = f(f^{n-1}(e_0))$, $f^*(e_0) = \bigcup_{i=N} f^i(e_0)$
- Funktionen auf Mengen: Sei $f: T_1 \times ... \times T_n \rightarrow T$ eine Funktion und U_i jeweils Teilmenge von T_i. Dann ist f auch auf Mengen von Werten definiert:
  $f: P(T_1) \times ... \times P(T_n) \rightarrow P(T)$, mit $f(U_1, \ldots, U_n) = \bigcup_{u_1 \in U_1} \ldots \bigcup_{u_n \in U_n} f(u_1, \ldots, u_n)$
- Zugriff auf einzelne Attribute strukturierter Mengen: Für jedes Attribut einer strukturierten Menge wird eine Funktion definiert. Gehören zur Menge D die Attribute $a_1 \ldots a_n$ mit den Typen $T_1 \ldots T_n$, dann gibt es n Funktionen $a_i: D \rightarrow T_i$.

Definition einer Menge von strukturierten Elementen

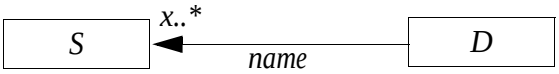
Seien D, D_i zu definierende Mengen, S, S_i, S_{ij} vorgegebene Mengen, i, j, k, n, n_i, m, x, y natürliche Zahlen (inkl. 0), dann gelten folgende Entsprechungen:

Klasse (= strukturierte Menge)



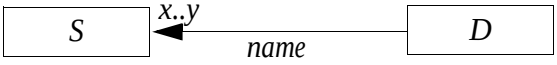
$$D = \{(a_1, \ldots, a_n) \mid \forall i \in \{1, \ldots, n\} \bullet a_i \in S_i\}$$

Assoziation mit angegebener Untergrenze



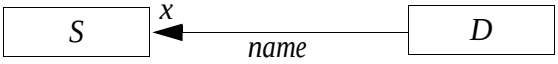
$$D = \{a \mid a \in P(S) \wedge |a| \geq x\}$$

Assoziation mit angegebener Ober- und Untergrenze



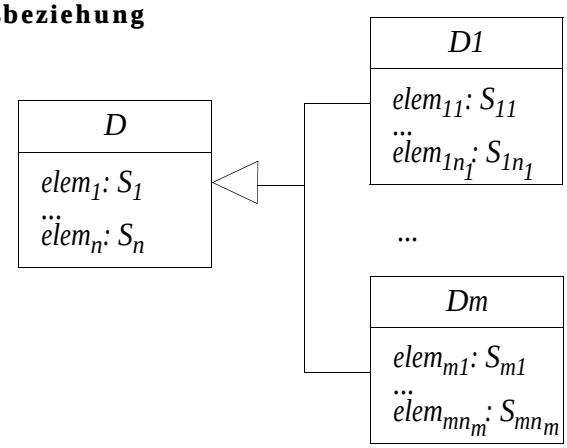
$$D = \{a \mid a \in P(S) \wedge |a| \geq x \wedge |a| \leq y\}$$

Assoziation, bei der Ober- und Untergrenze gleich sind



$$D = \{a \mid a \in P(S) \wedge |a| = x\}$$

Vererbungsbeziehung



$$D_i = \{(a_1, \ldots, a_n, a_{i1}, \ldots, a_{in_i}) \mid \forall k \in \{1, \ldots, n\} \bullet a_k \in S_k \wedge \forall j \in \{1, \ldots, n_i\} \bullet a_{ij} \in s_{ij}\}$$

$$D = \{(a_1, \ldots, a_n) \mid \forall i \in \{1, \ldots, n\} \bullet a_i \in S_i\} \cup D_1 \cup \ldots \cup D_m$$

Abbildung 15: Formale Semantik für eine graphische Notation

- Zugriff über Assoziationen: Für jede Assoziation wird eine mengenwertige Funktion definiert. Sind die Mengen D und S über eine Assoziation mit dem Namen *name* verbunden, dann gibt es auch folgende Funktion: $name: D \rightarrow P(S)$.

- Seien M und K Mengen und k ein beliebiges Element. Dann werden $M|_k$ und $M|_K$ definiert als Mengen, die alle Elemente aus M enthalten, die von k bzw. K aus erreichbar sind:

  $$M|_k = \{m \in M | m \in contains^*(k)\} \quad , \quad M|_K = \{m \in M | m \in contains^*(K)\}$$

  contains: Entry → P(Entry) ist eine im folgenden verwendete Assoziation, die beliebige Einträge (*Entry*, siehe Kapitel 6.2) des Informationsmodells verbinden kann. Durch sie wird eine Enthaltenseins-Hierarchie beschrieben, ähnlich der Aggregation der UML.

- Betrag einer Menge oder eines Freitextes: Sei $p=p_1...p_m$ ein Freitext und $D=\{d_1...d_n\}$ eine Menge. Dann gilt:
  $|p| = m$, $|D| = n$

6.1.4 Prädikate und Funktionen auf Wörtern, Phrasen und Freitexten

Auf den Mengen *Word*, *Phrase* und *Paragraph* werden die Funktion *uppercase* und die Prädikate *substring*, *inflection*, *synonyme*, *hyponyme*, *directuse* und *indirectuse* definiert.

Zwischen zwei Freitexten liegt eine *Substring-Beziehung* vor, wenn ein Freitext den anderen buchstabengetreu enthält. »Aktiv« z.B. ist ein Substring von »Aktivität«, »Aktivitäten« jedoch nicht. Weil Groß-/Kleinschreibung hierbei nicht berücksichtigt werden soll, wird zuerst die *Uppercase-Funktion* definiert:

Definition (*uppercase: Paragraph → Paragraph*): Die Funktion *uppercase* ordnet einem gegebenen Freitext einen Ergebnis-Freitext zu, in dem jeder Kleinbuchstabe *(a...z)* durch den entsprechenden Großbuchstaben (*A..Z*) ersetzt ist und alle anderen Zeichen unverändert sind.

Definition (*substring: Paragraph × Paragraph → Boolean*): Das Prädikat *substring (p, s)* ist *true*, wenn der Freitext *p* in *s* vorkommt, d.h., sei $p=p_1p_2...p_m$, $s=s_1s_2...s_n$, dann gilt:
$substring(p, s) = \exists i > 0 \bullet \forall j \in \{1 \ldots m\} \bullet (uppercase(p_j) = uppercase(s_{i+j-1}))$.

Zwischen zwei Wörtern liegt eine *Flexionsbeziehung* vor, wenn es eine *Grundform* gibt, von der beide Wörter durch Konjugation oder Deklination erzeugt werden können. Zwischen den Wörtern »spiele« und »spielte« z.B. gibt es eine Flexionsbeziehung, weil sie beide aus der Grundform »spielen« durch Konjugation erzeugt werden können.

Definition (*inflection: Word × Word → Boolean*): Das Prädikat *inflection* (w_1, w_2) ist *true*, wenn es ein Wort *w* gibt, so daß die Wörter w_1 und w_2 durch Konjugation oder Deklination aus *w* erzeugt werden können.

Im folgenden wird das *Direkt-Benutzt-Prädikat* definiert. Es ist *true*, wenn in einem Freitext eine Phrase benutzt wird. Wird der Wertebereich auf Freitexte und Wörter eingeschränkt, kann das Direkt-Benutzt-Prädikat auf das Flexions- und das Substring-Prädikat zurückgeführt werden. Die Diskussion in Kapitel 5.6 zeigt aber, daß eine Verallgemeinerung auf Phrasen nicht ohne weiteres möglich ist.

Definition (*directuse: Paragraph × Phrase → Boolean*): Sei *p* eine Phrase und *t* ein Freitext, dann gilt: *directuse (t, p)* ist *true*, wenn die Phrase *p* in dem Freitext *t* benutzt (siehe Kapitel 5.6) wird.

Das Flexionsprädikat und das Direkt-Benutzt-Prädikat können, ohne weitere Hilfsmittel wie z.B. ein vollständiges linguistisches Lexikon, nicht berechnet werden. Ein (vereinfachter) Ansatz zur Berechnung wird in Kapitel 8.1.2 vorgestellt.

Bei dem *Indirekt-Benutzt-Prädikat* werden auch *Synonyme, Hyponyme und Hyperonyme* berücksichtigt.

Definition (*synonyme: Phrase* × *Phrase* → *Boolean*): Seien p_1 und p_2 zwei Phrasen, dann gilt: p_1 ist synonym zu p_2 (*synonyme* (p_1, p_2)), wenn p_1 und p_2 wechselseitig verwendet werden können.

Definition (*hyponyme: Phrase* × *Phrase* → *Boolean*): Seien p_1 und p_2 zwei Phrasen, dann gilt: p_1 ist hyponym zu p_2 (*hyponyme* (p_1, p_2), wenn zwischen p_1 und p_2 eine Ist-Ein-Beziehung vorliegt (p_1 ist ein spezielles p_2).

Das Synonymie-Prädikat und das Hyponymie-Prädikat können ebenfalls nicht ohne weitere Hilfsmittel berechnet werden.

Definition (*indirectuse: Paragraph* × *Phrase* → *Boolean*): Sei p eine Phrase und t ein Freitext, dann gilt: t benutzt p indirekt (*indirectuse*(t,p)), wenn folgende Bedingung erfüllt ist:

$$indircectuse(t, p) = \exists q \in Phrase \bullet ((synonyme(p, q) \vee hyponyme(p, q) \vee hyponyme(q, p)) \wedge directuse(t, q))$$

Hyponyme müssen in »beide Richtungen« berücksichtigt werden. Wenn z.B. in einem Freitext eines SESAM-Dokuments »Benutzer« erwähnt werden, dann werden damit auch implizit Aussagen über »Spieler« (als spezielle Benutzer) gemacht. Aussagen über »Spieler« können aber auch »Benutzer« betreffen, weil einige »Benutzer« auch »Spieler« sind.

Abschließend wird ein Prädikat eingeführt, das immer dann *true* ergibt, wenn ein Wort innerhalb eines Freitextes ein *maximales Wort* (siehe Kapitel 5.6) ist. Ein maximales Wort zeichnet sich dadurch aus, daß es auch dann noch ein Substring der in Leerzeichen eingeschlossenen Phrase ist, wenn vorne und hinten bestimmte, in einem Wort nicht erlaubte Zeichen angehängt werden.

Definition (*maxword: Paragraph* × *Word* → *Boolean*): Sei $p=p_1p_2...p_m$ ein Freitext und $w=w_1w_2...w_n$ ein Wort, dann gilt: *maxword* (p, w) ist *true*, wenn das Wort w in dem Freitext p ein maximales Wort ist.

$$maxword(p, w) = \exists w_0, w_{n+1} \in \Sigma \setminus WordMark \bullet \exists b \in Blank \bullet substring(w_0 w_1 \ldots w_n w_{n+1}, b p_1 \ldots p_m b)$$

6.2 Einträge

Eine Dokumentation enthält folgende *Eintragstypen* (*Entry*): Inhaltselement, Annotation, Systemmodell, Glossareintrag, Quellenkatalogeintrag, Mangeleintrag, Kapitel, Ist-Analysedokument, Anforderungssammlung, Lastenheft, Pflichtenheft, Glossar, Quellenkatalog und Mängelkatalog. Jeder *Eintrag* hat folgende Attribute: eine *eindeutige ID* (*id*), einen Verweis auf einen *Autor* (*author*) und ein *Änderungsdatum* (*changedate*). Ein Autorverweis zeigt immer auf einen Personeneintrag (*PersonSourceEntry*) im Quellenkatalog (*SourceCatalogue*). Es wird die abstrakte Oberklasse *Entry* eingeführt, in der diese Attribute definiert sind.

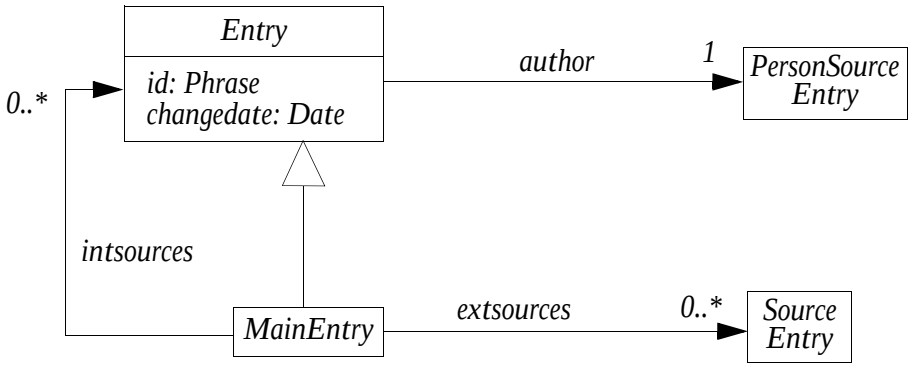


Abbildung 16: Einträge und Haupteinträge

Zu einigen der Einträge können Ursprünge angegeben werden: zu Inhaltselementen, Annotationen, Systemmodellen, Glossareinträgen, Quellenkatalogeinträgen und Mangeleinträgen. Diese Einträge werden *Haupteinträge* (*MainEntry*) genannt. Ein *Ursprung* ist entweder ein Verweis auf einen anderen Eintrag (*intsources*) oder auf eine externe Quelle (*extsources*). Quellenkataloge und Quellenkatalogeinträge (*SourceEntry*, *PersonSourceEntry*) werden in Kapitel 6.3 eingeführt.

Folgende Bedingungen müssen erfüllt sein:

- Das Attribut *id* ist das Schlüsselattribut für jeden Eintrag innerhalb einer Dokumentation (*Model*, siehe Kapitel 6.11):
  $\forall I \in Model \bullet \forall e_1, e_2 \in Entry|_I \bullet (id(e_1) = id(e_2) \Rightarrow e_1 = e_2)$
- Wenn zwei Einträge direkt oder indirekt über eine *contains*-Kante miteinander verbunden sind, dann müssen die Einträge verschieden sein:
  $\forall e_1, e_2 \in Entry \bullet (e_1 \in contains^*(e_2) \Rightarrow e_1 \neq e_2)$
- Ein Haupteintrag darf über Ursprungsverweise nicht auf sich selbst verweisen:
  $\forall m_1, m_2 \in MainEntry \bullet (m_1 \in intsources(m_2) \cup extsources(m_2) \Rightarrow m_1 \neq m_2)$

Weil Dokumentationen in sich abgeschlossen sein sollen, müssen alle Ursprungsverweise und Autorverweise auf Einträge der gleichen Dokumentation verweisen, zu der auch der (Ausgangs-)Eintrag gehört.

$\forall I \in Model \bullet \forall m \in MainEntry|_I \bullet (extsources(m) \subset Entry|_I)$

$\forall I \in Model \bullet \forall m \in MainEntry|_I \bullet (intsources(m) \subset Entry|_I)$

$\forall I \in Model \bullet \forall e \in Entry|_I \bullet (author(e) \subset Entry|_I)$

Im weiteren Verlauf dieses Berichts wird folgende Funktion benötigt:

Definition (*indirectsources: MainEntry* $\rightarrow$ *P(Entry)*): Die Funktion *indirectsources* ordnet einem Haupteintrag die Menge aller direkten und indirekten Ursprünge zu:

$$indirectsources(m) = \{e \in Entry | (\exists i \in N^{\geq 1} \bullet e \in intsources^i(m)) \vee (e \in extsources(intsources^*(m)))\}$$

6.3 Quellenkatalog

Der *Quellenkatalog* (*SourceCatalogue*) enthält alle Informationen über die Informationsquellen. Eine *Quelle* ist entweder eine *Person* (*PersonSourceEntry*), eine *Partei* (*PartySourceEntry*) oder ein *externes Dokument* (*DocSourceEntry*). Jeder *Quellenkatalogeintrag* (*SourceEntry)* hat einen *Namen* (*name*), eine *Beschreibung* (*ref*) und einen *Zustand* (*state*). Personen verfügen über einen Marker (*isauthor*), durch den sie als Autoren gekennzeichnet werden können. Von Dokumenten oder Parteien aus kann auf Personen verwiesen werden (*associatedpersons*). Auf diese Weise entsteht eine n:m-Zuordnung zwischen Personen und Dokumenten bzw. Parteien.

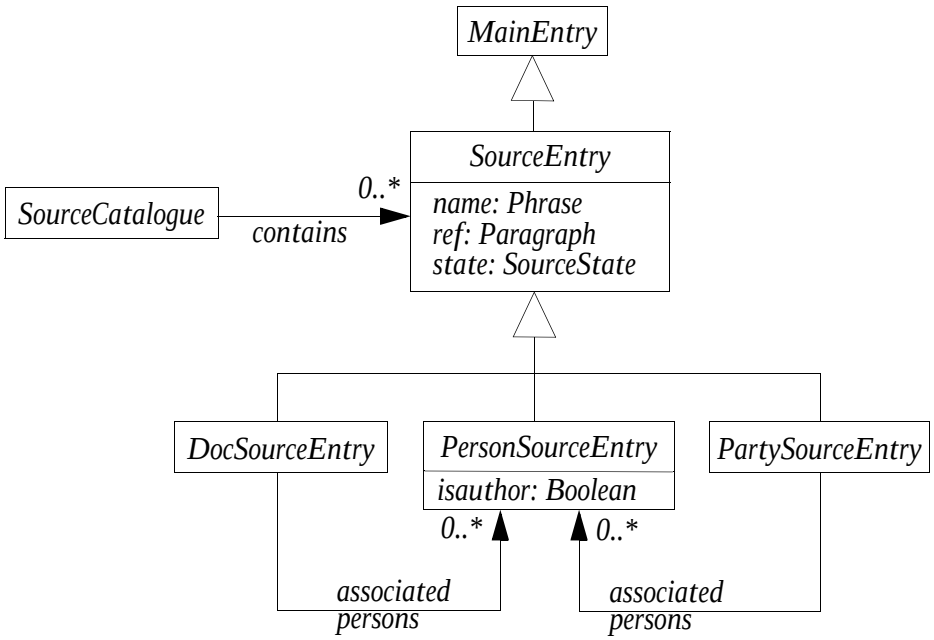


Abbildung 17: Quellenkatalogeinträge

Die Namen der Quellenkatalogeinträge müssen paarweise verschieden sein:

$$\forall I \in Model \bullet \forall q_1, q_2 \in SourceEntry|_I \bullet (name(q_1) = name(q_2) \Rightarrow q_1 = q_2)$$

Der Ursprung einer Quelle darf nur eine externe Quelle (*extsources*) sein:

$$\forall s \in SourceEntry \bullet (|intsources(s)| = 0)$$

Die Menge *SourceState* verfügt über die Elemente »aktuell« (*current*), »verworfen« (*rejected*) und über ein *TBD*- und ein *Default*-Element:

$$SourceState = \{current, rejected, TBD, default\}$$

Weil der Quellenkatalog abgeschlossen sein soll, müssen Quellenkatalogeinträge, die über *associatedpersons*-Verweise verbunden sind, zum gleichen Quellenkatalog gehören:

$$\forall doc \in SourceCatalogue \bullet \forall s \in DocSourceEntry \cup PartySourceEntry|_{doc} \bullet (associatedpersons(s) \subset SourceEntry|_{doc})$$

6.4 Glossar

In einem *Glossar* (*Glossary*) kann die projektrelevante Terminologie verwaltet werden. Ein *Glossareintrag* (*GlEntry*) enthält folgende Informationen: einen *Term* (*term*), eine *Definition* (*definition*), einen *Zustand* (*state*) und eine Angabe, ob es sich um einen projektspezifischen Term oder einen Term des Anwendungsbereichs handelt (*scope*). Glossareinträge können *kategorisiert* werden (*classes*). Sie können *Synonym-Verweise* (*synonymes)* und *Hyperonym-Verweise* (*hyperonymes*) auf andere Glossareinträge enthalten.[1] Zu jedem Glossareintrag können *Flexionen* (*Inflection*) für den gesamten Term oder für einzelne Wörter des Terms angegeben werden (*inflections*).

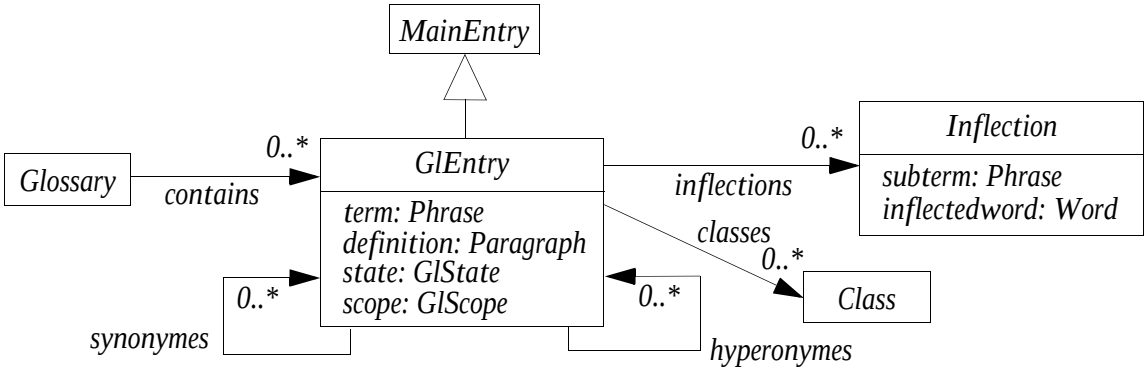


Abbildung 18: Glossareinträge

Der Ursprung eines Glossareintrags darf nur eine externe Quelle (*extsources*) sein:

$$\forall g \in GlEntry \bullet |intsources(g)| = 0$$

Wenn zwei Glossareinträge über Synonym- oder Hyperonym-Verweise verbunden werden, müssen sie verschieden sein:

$$\forall g_1, g_2 \in GlEntry \bullet (g_1 \in synonymes(g_2) \Rightarrow g_2 \neq g_1)$$

$$\forall g_1, g_2 \in GlEntry \bullet (g_1 \in hyperonymes(g_2) \Rightarrow g_2 \neq g_1)$$

Die Synonymiebeziehung muß symmetrisch sein:

$$\forall g_1, g_2 \in GlEntry \bullet (g_1 \in synonymes(g_2) \Rightarrow g_2 \in synonymes(g_1))$$

Weil das Glossar abgeschlossen sein soll, müssen Glossareinträge, die über Synonym- oder Hyperonym-Verweise verbunden sind, zum gleichen Glossar gehören:

$$\forall doc \in Glossary \bullet \forall g \in GlEntry|_{doc} \bullet (synonymes(g) \subset GlEntry|_{doc})$$

$$\forall doc \in Glossary \bullet \forall g \in GlEntry|_{doc} \bullet (hyperonymes(g) \subset GlEntry|_{doc})$$

Die *Flexionsangabe* (*inflectedword*) muß sich entweder auf ein (maximales) Wort des Terms oder auf den gesamten Term (*subterm*) beziehen:

$$\forall g \in GlEntry \bullet \forall i \in inflections(g) \bullet ((subterm(i) = term(g)) \\ \vee (substring(subterm(i), term(g)) \wedge maxword(term(g), subterm(i))))$$

---

1. Beachte den Unterschied zwischen den hier definierten Synonym-Verweisen, Hyperonym-Verweisen und Flexionsangaben und den Synonymie-, Hyponymie- und Flexionsprädikaten aus Kapitel 6.1. Siehe dazu auch Kapitel 8.1.2.

Die Menge *Class* enthält die möglichen Kategorien für Glossareinträge (*classes*):

$Class = TClass \cup \{TBD\}$

Die zugehörige Menge *TClass* wird in Kapitel 6.10 (Metaspezifikation) definiert. Alternativ kann statt einer Kategorie auch ein *TBD*-Element verwendet werden.

In der Metaspezifikation wird eine Kategorisierungshierarchie für Glossareinträge vorgegeben. Glossareinträge dürfen nur diese Kategorien verwenden:

$$\forall I \in Model \bullet \forall g \in GlEntry|_I \bullet (classes(g) \subset glclass(Metaspecification|_I) \cup \{TBD\})$$

Die Menge *GlScope* enthält die Werte »projektspezifischer Term« *(project)* und »Term des Anwendungsbereichs« (*domain*). *GlState* enthält die Zustände »aktuell« (*current*) und »verworfen« (*rejected*) und verfügt über ein *TBD*- und ein *Default*-Element.

$GlScope = \{domain, project\}$

$GlState = \{current, rejected, TBD, default\}$

6.5 Systemmodelle

Durch ein Systemmodell können die für das Projekt relevanten Systeme, Rollen von Personen und Schnittstellen benannt und deren Beziehungen beschrieben werden. Ein *Systemmodell* (*SystemModel*) enthält beliebig viele *Systemmodelleinträge* (*SysModelEntry*). Ein Systemmodelleintrag verweist entweder auf ein *Zielsystem* (*TargetSystem*), ein *vorgegebenes System* (*PredefinedSystem*), eine Person bzw. deren *Rolle* (*Role*) oder eine *Schnittstelle* (*Interface*). Jede Schnittstelle ist über eine *Schnittstellenkante* (*system*) immer genau einem System zugeordnet. Jeder Systemmodelleintrag hat einen *Namen* (*name*).

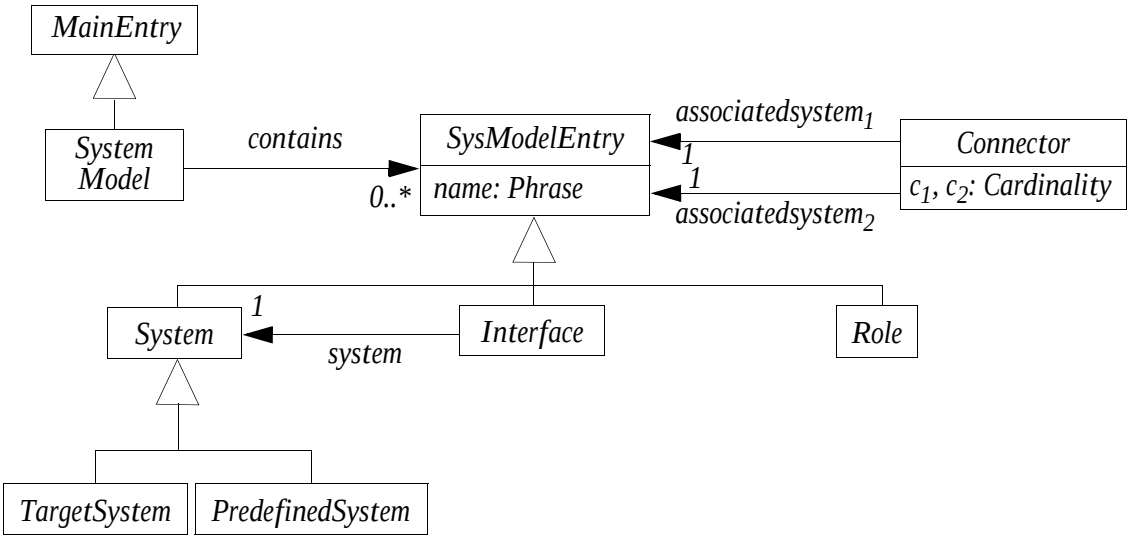


Abbildung 19: Systemmodell

Kommunikationskanten (*Connector*) innerhalb des Systemmodells verbinden genau zwei Systemmodelleinträge. Damit die Kardinalitäten eindeutig zugeordnet werden können, werden hierfür zwei Assoziationen eingeführt ($associatedsystem_1$, $associatedsystem_2$). Weil die Reihenfolge in folgenden Formeln nicht immer von Bedeutung ist, wird folgende Funktion eingeführt:

Definition (*associatedsystems: Connector → P(SysModelEntry)*): Diese Funktion ordnet einem Connector die verbundenen Systemmodelleinträge zu:

$$associatedsystems(c) = associatedsystem_1(c) \cup associatedsystem_2(c)$$

Zwischen zwei Systemmodelleinträgen darf maximal eine Kommunikationskante verlaufen:

$$\forall m \in SystemModel \bullet \forall s_1, s_2 \in SysModelEntry|_m \bullet (|\{c \in Connector | \{s_1, s_2\} = associatedsystems(c)\}| \leq 1)$$

Eine Kommunikationskante darf nicht ein System mit einer seiner Schnittstellen verbinden. D.h. falls zwischen einer Schnittstelle und einem System eine Schnittstellenkante verläuft, darf zwischen ihnen keine Kommunikationskante verlaufen:

$$\forall i \in Interface \bullet \forall s \in System \bullet (s \in system(i) \Rightarrow \forall c \in Connector \bullet (\{i, s\} \neq associatedsystems(c)))$$

Jede Kommunikationskante hat zwei *Kardinalitätsangaben* (c_1, c_2), die jeweils eine Untergrenze und eine Obergrenze vorgeben. »*« steht für »beliebig viele«:

$$Cardinality = N \times (N \cup \{'*'\})$$

Die Untergrenze muß kleiner oder gleich der Obergrenze sein:

$$\forall c \in Connector \bullet ((c_2(c) = '*') \vee (c_1(c) \leq c_2(c)))$$

Ein Systemmodell soll abgeschlossen sein, d.h. die miteinander über Kommunikationskanten oder Schnittstellenkanten verbundenen Systemmodelleinträge müssen alle zum gleichen Systemmodell gehören:

1. $\forall s_1, s_2 \in SysModelEntry \bullet (\exists c \in Connector \bullet (\{s_1, s_2\} = associatedsystems(c))) \Rightarrow (\exists m \in SystemModel \bullet \{s_1, s_2\} \subset contains(m))$
2. $\forall i \in Interface \bullet \exists m \in SystemModel \bullet (\{i\} \cup system(i) \subset contains(m))$

6.6 Inhaltselemente

Die eigentlichen inhaltlichen Informationen eines Anforderungsdokuments werden durch *Inhaltselemente* (*ContElement*) beschrieben. Es gibt vier Arten von Inhaltselementen: *Anforderungen* (*Requirement*), *Fakten* (*Fact*), *Probleme* (*Problem*) und *Zusicherungen* (*Assertion*). Der »Hauptteil« eines Inhaltselements ist ein Freitext (*contents*).

An Anforderungen, Zusicherungen und Fakten können *Testfälle* (*TestCase*) gebunden werden (*tests*). Jeder Testfall enthält eine Freitext-Beschreibung (*descr*).

Jedes Inhaltselement ist in genau einem der folgenden *Zustände* (*state*): »aktuell« (*current*), »verworfen« (*rejected*) »zukünftig« (*future*), »TBD« oder »aktuell (Default)« *(default*). Jede Anforderung enthält eine der folgenden *Projektstandsinformationen* (*pstate*): »erfüllt« (*done*), »teilweise erfüllt (*partly done*), »nicht erfüllt« (*open*), »TBD« oder »nicht erfüllt (Default)« (*default*). Die zugehörigen Mengen *ContState* und *PrjState* werden folgendermaßen definiert:

$$ContState = \{current, rejected, future, TBD, default\}$$

$$PrjState = \{done, partly\ done, open, TBD, default\}$$

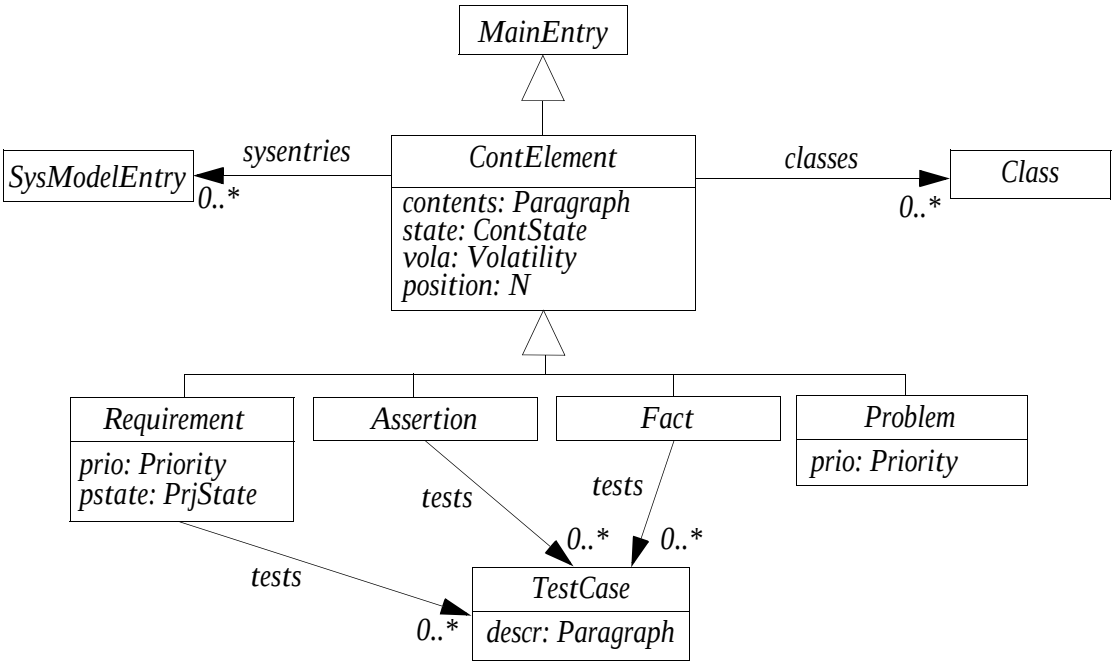


Abbildung 20: Inhaltselemente

Um eine Reihenfolge auf Inhaltselementen definieren zu können, verfügt ein Inhaltselement über eine logische Position (*position*, siehe auch Kapitel 6.8).

Inhaltselemente können an Systemmodelleinträge gebunden werden (*sysentries*). Dabei gibt es folgende Einschränkungen: Anforderungen dürfen nur an Zielsysteme und deren Schnittstellen gebunden werden, Zusicherungen nur an Rollen, vorgegebene Systeme und deren Schnittstellen:

1. $\forall r \in Requirement \bullet \forall i \in Interface \bullet (i \in sysentries(r) \Rightarrow \exists t \in TargetSystem \bullet t \in system(i))$
2. $\forall r \in Requirement \bullet \forall s \in System \cup Role \bullet (s \in sysentries(r) \Rightarrow s \in TargetSystem)$
3. $\forall a \in Assertion \bullet \forall i \in Interface$
   $\bullet (i \in sysentries(a) \Rightarrow \exists p \in PredefinedSystem \bullet p \in system(i))$
4. $\forall a \in Assertion \bullet \forall s \in System \bullet (s \in sysentries(a) \Rightarrow \exists p \in PredefinedSystem \bullet (s = p))$

Ein Inhaltselement darf nur an Systemmodelleinträge gebunden werden, die zu dem gleichen Anforderungsdokument (siehe Kapitel 6.8) gehören wie das Inhaltselement selbst:

$$\forall c \in ContElement \bullet \exists d \in ReqDocument \bullet (c \in ContElement|_d \wedge (\exists m \in SystemModel|_d \bullet \forall s \in sysentries(c) \bullet (s \in contains(m))))$$

Inhaltselemente, die an eine Schnittstelle gebunden werden, müssen automatisch auch an das zugehörige System gebunden sein:

$$\forall c \in ContElement \bullet \forall i \in Interface \bullet (i \in sysentries(c) \Rightarrow system(i) \in sysentries(c))$$

Inhaltselemente enthalten folgende zusätzliche Attribute: eine *Priorität* (*prio*), eine *Stabilität* (*vola*) und eine Menge von *Kategorien* (*classes*). Die zugehörigen Mengen *Priority*, *Volatility* und *Class* werden folgendermaßen definiert:

$$Priority = TPriority \cup \{TBD\}$$

$$Volatility = TVolatility \cup \{TBD\}$$

$$Class = TClass \cup \{TBD\}$$

Die Mengen *TClass*, *TPriority* und *TVolatility* werden in Kapitel 6.10 (Metaspezifikation) definiert. Dort wird auch die Vorbelegung für die möglichen Prioritäten, Stabilitäten und Kategorien festgelegt. Inhaltselemente dürfen nur diese Kategorien, Prioritäten und Stabilitäten verwenden:

$$\forall I \in Model \bullet (classes(ContElement|_I) \subset (contclass(Metaspecification|_I) \cup \{TBD\}))$$

$$\forall I \in Model \bullet (vola(ContElement|_I) \subset (volatility(Metaspecification|_I) \cup \{TBD\}))$$

$$\forall I \in Model \bullet (prio(Requirement \cup Problem|_I) \subset (priority(Metaspecification|_I) \cup \{TBD\}))$$

6.7 Annotationen

Annotationen (*Annotation*) dienen dazu, das Anforderungsdokument oder Teile davon verständlicher zu machen. Der »Hauptteil« einer Annotation ist ein Freitext (*contents*).

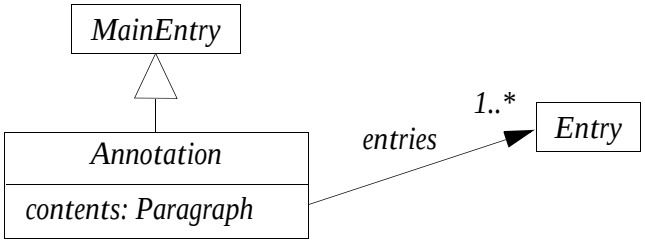


Abbildung 21: Annotationen

Eine Annotation muß sich auf mindestens einen Eintrag beziehen (*entries*). Weil Dokumente in sich abgeschlossen sein sollen, müssen alle Einträge, auf die sich eine Annotation bezieht, zum gleichen Dokument gehören:

$$\forall a \in Annotation \bullet \exists doc \in Document \bullet (entries(a) \subset Entry|_{doc})$$

Eine Annotation darf sich nicht auf andere Annotationen beziehen:

$$\forall a \in Annotation \bullet \forall e \in entries(a) \bullet (e \notin Annotation)$$

6.8 Dokumente und Anforderungsdokumente

Eine Dokumentation enthält folgende Arten von *Dokumenten* (*Document*): *Ist-Analysedokument* (*CurrentState*), *Anforderungssammlung* (*ReqCollection*), *Lastenheft* (*ReqSpecification*), *Pflichtenheft* (*Specification*), *Glossar* (*Glossary*), *Quellenkatalog* (*SourceCatalogue*) und *Mängelkatalog* (*DefectCatalogue*). Ist-Analysedokumente, Anforderungssammlungen, Lastenhefte und Pflichtenhefte werden als *Anforderungsdokumente* (*ReqDocument*) bezeichnet. Jedes Anforderungsdokument hat einen *Titel* (*title*) und einen *Abstract* (*abstract*).

Jedes Anforderungsdokument enthält maximal ein *Systemmodell* (*SystemModel*). Das Systemmodell eines Ist-Analysedokuments darf keine Zielsysteme enthalten:

$$\forall s \in SystemModel|_{CurrentState} \bullet (|TargetSystem|_s| = 0)$$

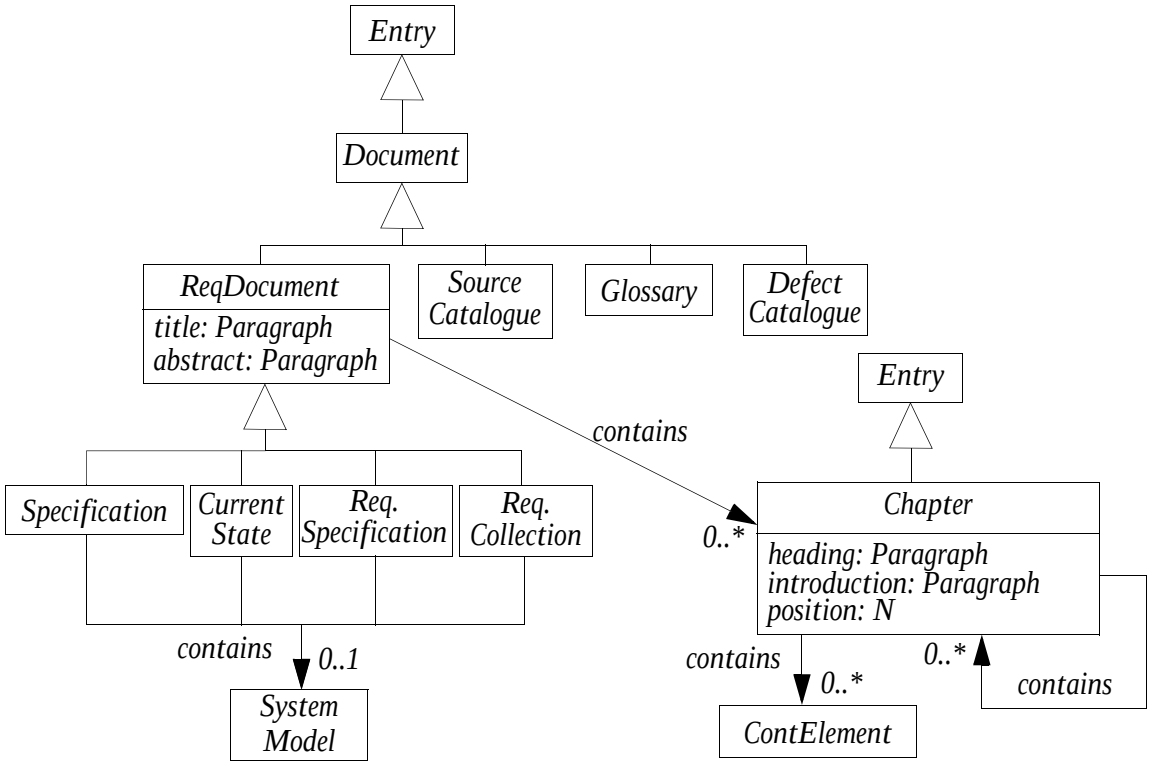


Abbildung 22: Anforderungsdokumente

Durch die Titel sollen die Anforderungsdokumente einer Dokumentation eindeutig identifizierbar sein, d.h. die Titel dürfen nicht leer sein und müssen paarweise verschieden sein:

$$\forall d \in ReqDocument \bullet (|title(d)| > 0)$$

$$\forall I \in Model \bullet \forall d_1, d_2 \in ReqDocument|_I \bullet (title(d_1) = title(d_2) \Rightarrow d_1 = d_2)$$

Jedes Anforderungsdokument enthält *Kapitel* (*Chapter*), die *Inhaltselemente* (*ContElement*, siehe Kapitel 6.6) und weitere Kapitel enthalten können. Jedes Kapitel hat eine *Überschrift* (*heading*) und eine *Einleitung* (*introduction*).

Die Überschrift darf nicht leer sein:

$$\forall c \in Chapter \bullet (|heading(c)| > 0)$$

Die Überschriften der direkten Unterkapitel eines Kapitels bzw. eines Anforderungsdokuments müssen paarweise verschieden sein:

$$\forall c \in ReqDocument \cup Chapter \bullet \forall c_1, c_2 \in \{e \in contains(c) | e \in Chapter\} \bullet (heading(c_1) = heading(c_2) \Rightarrow c_1 = c_2)$$

Die Kapitel werden in einer Hierarchie angeordnet, mit dem Anforderungsdokument als Wurzel. Jedes Kapitel kann beliebig viele Unterkapitel (die wiederum Kapitel sind) enthalten. Um eine Reihenfolge auf den Unterkapiteln eines Oberkapitels/Dokuments definieren zu können, verfügt jedes Kapitel über eine logische Positionsnummer (*position*). Das erste Unterkapitel hat die Nummer 0, das zweite die Nummer 1 usw. Es muß also folgende Bedingung erfüllt sein:

$$\forall c \in ReqDocument \cup Chapter \bullet (position(\{e \in contains(c) | e \in Chapter\}) = \{0, \ldots, |\{e \in contains(c) | e \in Chapter\}| \text{ Ð } 1\})$$

Aus den gleichen Gründen verfügen auch die Inhaltselemente über eine logische Positionsnummer. Die einem Kapitel direkt untergeordneten Inhaltselemente müssen folgende Bedingung erfüllen:

$$\forall c \in Chapter \bullet (position(\{e \in contains(c) \mid e \in ContElement\}) = \{0, \ldots, |\{e \in contains(c) \mid e \in ContElement\}| \dot{-} 1\})$$

Im weiteren Verlauf dieses Berichts wird folgende Funktion benötigt:

Definition (*subchapters: Chapter → P(Chapter)*): Die Funktion *subchapters* ordnet einem gegebenen Kapitel die Menge aller direkten und indirekten Unterkapitel zu:

$$subchapters(c) = \{e \in contains^*(c) \mid e \in Chapter\}$$

Haupteinträge, die in einem Anforderungsdokument enthalten sind, dürfen sich über interne Ursprungsverweise (*intsources*) nur auf solche Einträge beziehen, die ebenfalls in einem Anforderungsdokument enthalten sind:

$$\forall I \in Model \bullet (intsources(MainEntry|_{ReqDocument|_I}) \subset Entry|_{ReqDocument|_I})$$

6.9 Mängelkatalog

Das Informationsmodell muß gewisse *Qualitätskriterien* (siehe Kapitel 6.13) erfüllen, um als »gut« zu gelten. Es ist jedoch recht wahrscheinlich, daß (zeitweise) einige der geforderten Qualitätskriterien nicht erfüllt sind. In diesem Fall liegt ein *Mangel* vor, der durch einen *Mangeleintrag* (*Defect*) dokumentiert werden kann. Die Mangeleinträge werden im *Mängelkatalog* (*DefectCatalogue*) verwaltet.

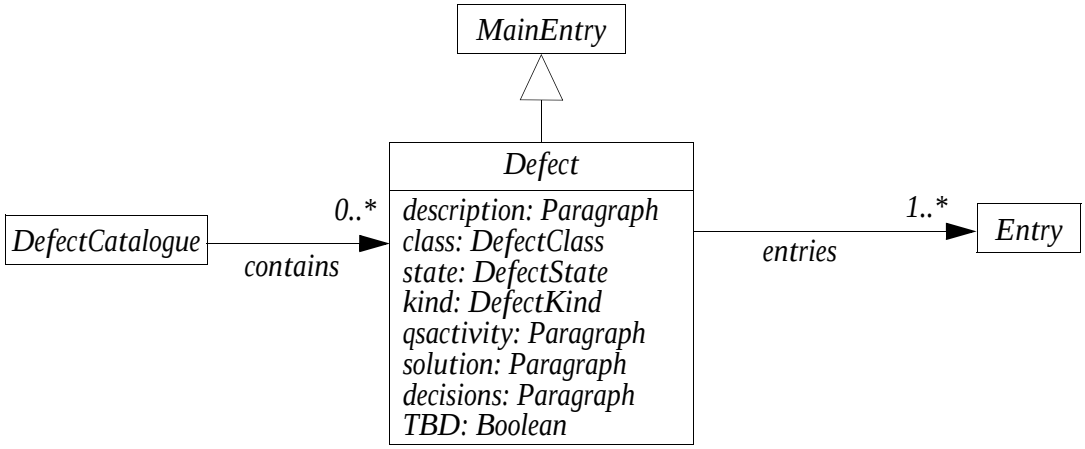


Abbildung 23: Mangeleinträge

Ein *Mangeleintrag* hat folgende Attribute: eine *Beschreibung* des Mangels (*description*), eine *Mangelkategorie* (*class*), einen *Zustand* (*state*), eine *Mangelart* (*kind*), eine Beschreibung der *Qualitätssicherungsmaßnahme,* in der der Mangel gefunden wurde (*qsactivity*), *Lösungsideen* (*solution*), *Entscheidungen* (*decisions*), ein Marker für offene Punkte (*TBD*) und Verweise auf Einträge, die den Mangel aufweisen (*Mangelverweise, entries*).

Jedem Mangel ist immer genau eine der folgenden Kategorien (*class*) zugeordnet: »inkorrekt«, »unvollständig«, »nicht adäquat«, »inkonsistent«, »extern inkonsi-

stent«, »überspezifiziert«, »unverständlich«, »nicht realisierbar«, »nicht prüfbar«, »mehrdeutig«, »unpräzise«, »redundant«, »nicht atomar«, »nicht minimal«, »nicht normkonform« oder »sonstiges«. Die zugehörige Menge *DefectClass* wird folgendermaßen definiert:

$$DefectClass = \{incorrect, incomplete, not\ adequate, inconsistent, external\ inconsistent, overspecified, incomprehensible, unrealizable, not\ verifiable, ambiguous, imprecise, redundant, not\ atomic, not\ minimal, not\ conform, others\}$$

Jeder Mangel ist immer genau in einem der folgenden Zustände: »potentiell« (*potential*), »aktuell« (*open*), »behoben« (*done*) und »verworfen« (*rejected*). Die zugehörige Menge *DefectState* wird folgendermaßen definiert:

$$DefectState = \{potential, open, done, rejected\}$$

Bei Mangeleinträgen wird unterschieden, ob sie »automatisch« (*automatic*) oder »manuell« (*manual*) erzeugt wurden. Die zugehörige Menge *DefectKind* wird folgendermaßen definiert:

$$DefectKind = \{automatic, manual\}$$

Die Beschreibung eines Mangeleintrags darf nicht leer sein:

$$\forall d \in Defect \bullet (|description(d)| > 0)$$

Ein Mangeleintrag darf über Mangelverweise nicht auf Mangeleinträge oder Mängelkataloge verweisen:

$$\forall d \in Defect \bullet \forall e \in entries(d) \bullet (e \notin Defect \wedge e \notin DefectCatalogue)$$

Weil Dokumente in sich abgeschlossen sein sollen, müssen alle Einträge, auf die sich ein Mangel bezieht, zum gleichen Dokument gehören:

$$\forall I \in Model \bullet \forall d \in Defect|_I \bullet \exists doc \in Document|_I \bullet (entries(d) \subset Entry|_{doc})$$

Der Ursprung eines Mangels darf nur eine externe Quelle (*extsources*) sein:

$$\forall d \in Defect \bullet |intsources(d)| = 0$$

6.10 Metaspezifikation

In der *Metaspezifikation* (*Metaspecification*) werden einige globale Mengen definiert:

- *Kategorisierungshierarchien* für Inhaltselemente (*contclass*) und Glossareinträge (*glclass*),
- mögliche *Prioritäten* für Anforderungen und Probleme (*priority*),
- mögliche *Stabilitäten* für Inhaltselemente (*volatility),*
- *Vorlagen* (*Template*) für Ist-Analysedokumente (*cstemplate*), Anforderungssammlungen (*rctemplate*), Lastenhefte (*rstemplate*) und Pflichtenhefte (*sptemplate*) und
- eine *Wortliste*, die *Worteinträge* (*WordEntry*) enthält, die für den ADMIRE-Ansatz eine besondere Bedeutung haben.

Eine Vorlage enthält (*contains*) eine Menge von *Vorlagen-Kapiteln* (*TChapter*). Sie werden in einer Hierarchie angeordnet (und verfügen, analog zu den Kapiteln eines Anforderungsdokuments, über eine logische Positionsnummer), mit der Vorlage als Wurzel. Jedes Vorlagen-Kapitel enthält beliebig viele Unterkapitel.

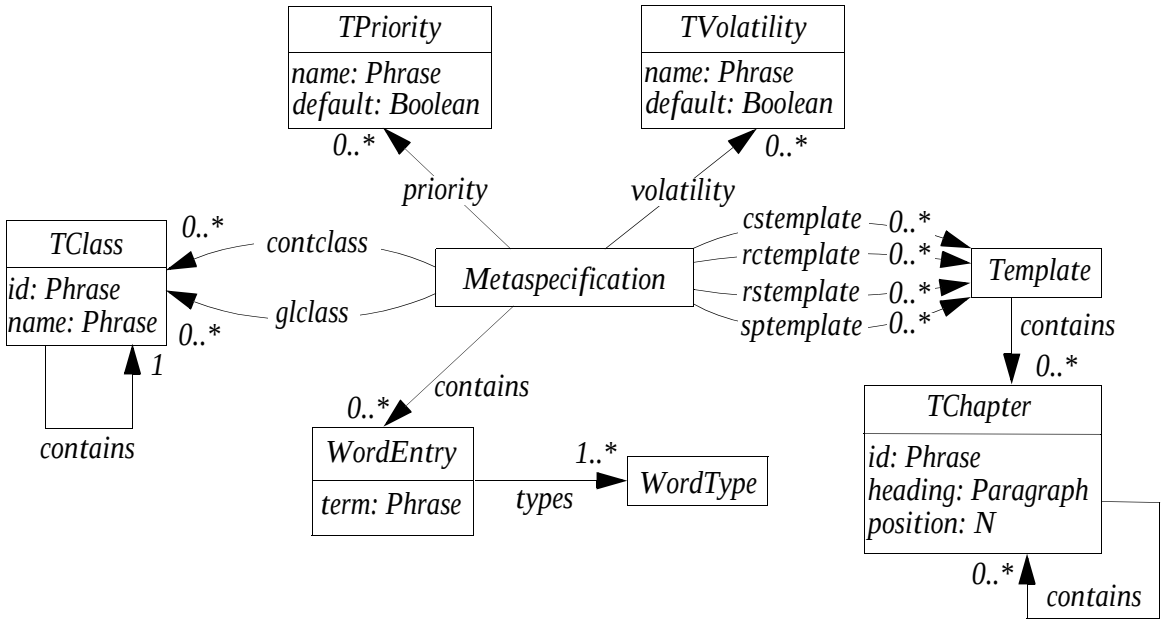


Abbildung 24: Metaspezifikation

Folgende Bedingung muß erfüllt sein (siehe auch Kapitel 6.8):

$\forall tc \in Template \cup TChapter \bullet (position(contains(tc)) = \{0, \ldots, |contains(tc)| \text{ Ð } 1\})$

Die Bezeichner (*id*) aller Vorlagen-Kapitel müssen paarweise verschieden sein:

$\forall I \in Model \bullet \forall t_1, t_2 \in TChapter|_I \bullet (id(t_1) = id(t_2) \Rightarrow t_1 = t_2)$

Die Überschrift eines Vorlagen-Kapitels darf nicht leer sein:

$\forall t \in TChapter \bullet (|heading(t)| > 0)$

Die Überschriften der direkten Unterkapitel eines Vorlagen-Kapitels bzw. einer Vorlage müssen paarweise verschieden sein:

$\forall tc \in Template \cup TChapter \bullet \forall tc_1, tc_2 \in contains(tc)$
$\bullet (heading(tc_1) = heading(tc_2) \Rightarrow tc_1 = tc_2)$

Im weiteren Verlauf dieses Berichts wird folgende Funktion benötigt:

Definition (*tsubchapters: TChapter* $\rightarrow$ *P(TChapter)*): Die Funktion *tsubchapters* ordnet einem Vorlagen-Kapitel die Menge aller direkten und indirekten Unterkapitel zu: $tsubchapters(tc) = contains^*(tc)$

Die Terme (*term*) der Worteinträge müssen paarweise verschieden sein:

$\forall I \in Model \bullet \forall e_1, e_2 \in WordEntry|_I \bullet (term(e_1) = term(e_2) \Rightarrow e_1 = e_2)$

Jeder Worteintrag verweist auf mindestens einen Worttyp (*types*). Die Menge der *Worttypen* (*WordType*) wird folgendermaßen definiert:

$WordType = \{quantor, wrongconjunction, otherconjunction, modalverb, modalverb_{anf}, indicator, weakword, simpleword\}$

Die Bezeichner aller Kategorien müssen paarweise verschieden sein:

$\forall I \in Model \bullet \forall c_1, c_2 \in TClass|_I \bullet (id(c_1) = id(c_2) \Rightarrow c_1 = c_2)$

Im folgenden wird die Funktion *indirectsubclasses* benötigt.

Definition (indirectsubclasses: TClass → P(TClass)): Die Funktion *indirectsubclasses* ordnet einer gegebenen Kategorie alle direkten und indirekten Unterkategorien zu:

$$indirectsubclasses(c) = contains^*(c)$$

Die Kategorisierungshierarchien werden im formalen Modell nicht als Hierarchien, sondern als »Wälder«, d.h. als Menge von Bäumen, modelliert, weil die Wurzel nicht zur Kategorisierung verwendet werden darf (siehe E48 und E91 in Kapitel 6). Die Wurzel-Kategorien des Waldes entsprechen den Kategorien der ersten Ebene der Kategorisierungshierarchie unterhalb der Wurzel.

Die Mengen $contclass(Metaspecification|_I)$ und $glclass(Metaspecification|_I)$ müssen einige Bedingungen erfüllen.

Sei $C(I) \in \{contclass(Metaspecification|_I), glclass(Metaspecification|_I)\}$:

1. Die Unterkategorien-Beziehung muß innerhalb von $C(I)$ abgeschlossen sein:
   $$\forall I \in Model \bullet \forall c \in C(I) \bullet contains(c) \subset C(I)$$

2. Jede Kategorie darf maximal eine Oberkategorie haben:
   $$\forall I \in Model \bullet \forall c \in C(I) \bullet (|\{sc \in C(I) | c \in contains(sc)\}| \leq 1)$$

3. Die Namen aller Kategorien einer Kategorisierungshierarchie müssen paarweise verschieden sein:
   $$\forall I \in Model \bullet \forall c_1, c_2 \in C(I) \bullet (name(c_1) = name(c_2) \Rightarrow c_1 = c_2)$$

4. Eine Kategorie darf nicht echte Unterkategorie von sich selbst sein:
   $$\forall I \in Model \bullet \forall c \in C(I) \bullet \forall i \in N^{\geq 1} \bullet c \notin contains^i(c)$$

Im folgenden werden die Vorbelegungen der Kategorisierungshierarchien angegeben, indem die Elemente der Mengen aufgezählt werden. Jedes Element ist ein Tripel (*id, name, subclasses*), wobei *subclasses* die Menge der IDs der direkt untergeordneten Kategorien enthält. Die Kategorisierungshierarchien für Inhaltselemente und Glossareinträge werden analog zu Abbildung 9 und Abbildung 11 (siehe Kapitel 5.3.2 und 5.6) initialisiert. Zuerst werden folgende Hilfsmengen eingeführt. *FClass, QClass, DClass, EClass, EEClass, PersClass* und *PrjClass*:

FClass = {(*1*, *'Funktion'*, ∅)}

QClass = {(*2*, *'Qualität'*, {*201*, *202*, *203*, *204*, *205*, *206*}), (*201*, *'Funktionalität'*, ∅),
(*202*, *'Zuverlässigkeit'*, ∅), (*203*, *'Benutzbarkeit'*, ∅), (*204*, *'Effizienz'*, ∅),
(*205*, *'Wartbarkeit'*, ∅), (*206*, *'Portierbarkeit'*, ∅)}

DClass = {(*3*, *'Datum'*, {*301*, *302*, *303*}), (*301*, *'internes Datum'*, ∅),
(*302*, *'Eingabedatum'*, ∅), (*303*, *'Ausgabedatum'*, ∅)}

EClass = {(*4*, *'Ereignis'*, {*401*, *402*, *403*, *404*}), (*401*, *'Eingabeereignis'*, ∅),
(*402*, *'Ausgabeereignis'*, ∅), (*403*, *'zeitliches Ereignis'*, ∅),
(*404*, *'bedingtes Ereignis'*, ∅)}

EEClass = {(*5*, *'Realisierungseigenschaft'*, {*501*, *502*, *503*, *504*}), (*501*, *'Ablaufumgebung'*, ∅),
(*502*, *'Entwicklungsumgebung'*, ∅), (*503*, *'Entwurf'*, ∅),
(*504*, *'Implementierung'*, ∅)}

PersClass = {(*6*, *'Personeneigenschaft'*, {*601*, *602*, *603*}), (*601*, *'allgemeine IT-Kenntnisse'*, ∅),
(*602*, *'Kenntnisse im Anwendungsbereich'*, ∅),
(*603*, *'zielsystemspezifische Kenntnisse'*, ∅)}

PrjClass = {(*7*, *'Projektvorgabe'*, {*701*, *702*, *703*}), (*701*, *'Projektplanung'*, ∅),
(*702*, *'Qualitätssicherung'*, ∅), (*703*, *'Metaanforderung'*, ∅)}

LClass = {(*8*, *'Layout'*, ∅)}

Damit kann die Vorbelegung für *contclass* definiert werden:

$$\forall I \in Model \bullet (contclass(Metaspecification|_I) = FClass \cup QClass \cup DClass \cup EClass \cup EEClass \cup PersClass \cup PrjClass \cup LClass)$$

Die Vorbelegung von *glclass* wird folgendermaßen definiert:

$$\forall I \in Model \bullet (glclass(Metaspecification|_I) = \{(1, 'Projekt', \varnothing), (2, 'System', \varnothing), (3, 'Schnittstelle', \varnothing), (4, 'Rolle', \varnothing), (5, 'Datum', \varnothing), (6, 'Ereignis', \varnothing), (7, 'Funktion', \varnothing), (8, 'Eigenschaft', \varnothing)\})$$

Die Mengen der möglichen Prioritäten und Stabilitäten müssen folgende Bedingungen erfüllen. Sei $T(I) \in \{priority(Metaspecification|_I), volatility(Metaspecification|_I)\}$:

1. Die Namen müssen paarweise verschieden sein:
   $$\forall I \in Model \bullet \forall t_1, t_2 \in T(I) \bullet (name(t_1) = name(t_2) \Rightarrow t_1 = t_2)$$

2. Genau einer der Werte muß der Defaultwert sein:
   $$\forall I \in Model \bullet (|\{t \in T(I) | default(t) = true\}| = 1)$$

Die Menge der möglichen Prioritäten ist mit den Werten »muß«, »soll« und »kann« vorbelegt. Die Menge der möglichen Stabilitäten ist mit »stabil« und »nicht stabil« vorbelegt. Defaultwerte sind »muß« und »stabil«.

$$\forall I \in Model \bullet (priority(Metaspecification|_I) = \{('muß', true), ('soll', false), ('kann', false)\})$$

$$\forall I \in Model \bullet (volatility(Metaspecification|_I) = \{('stabil', true), ('nicht stabil', false)\})$$

6.11 Informationsmodell

In den vorangehenden Unterkapiteln (6.3-6.10) wurden die Teile des *ADMIRE-Informationsmodells* definiert. Diese Teile werden in Abbildung 25 zum Gesamtmodell (*Model*) zusammengesetzt. Jede Dokumentation hat einen Namen (*projectname*) und ein Erzeugungsdatum (*creationdate*).

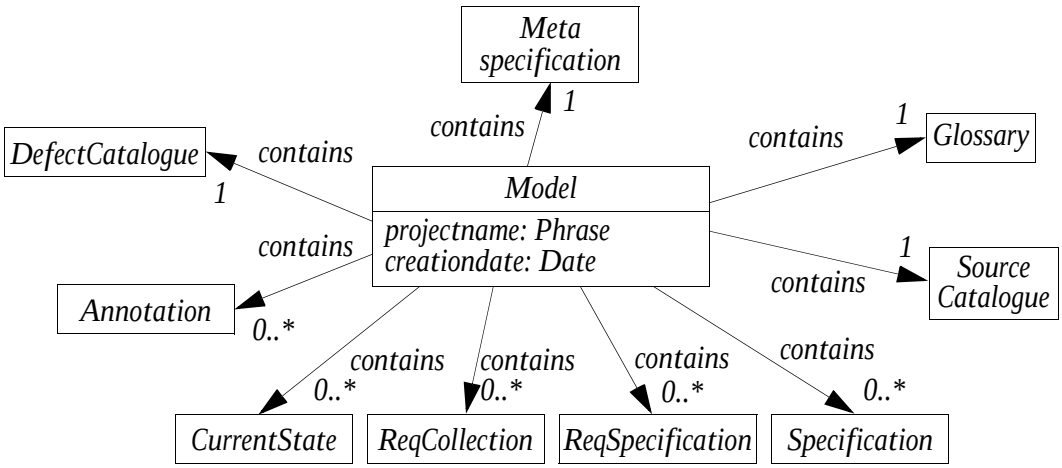


Abbildung 25: ADMIRE-Informationsmodell

In Abbildung 26 wird die zugehörige Eintragshierarchie dargestellt.

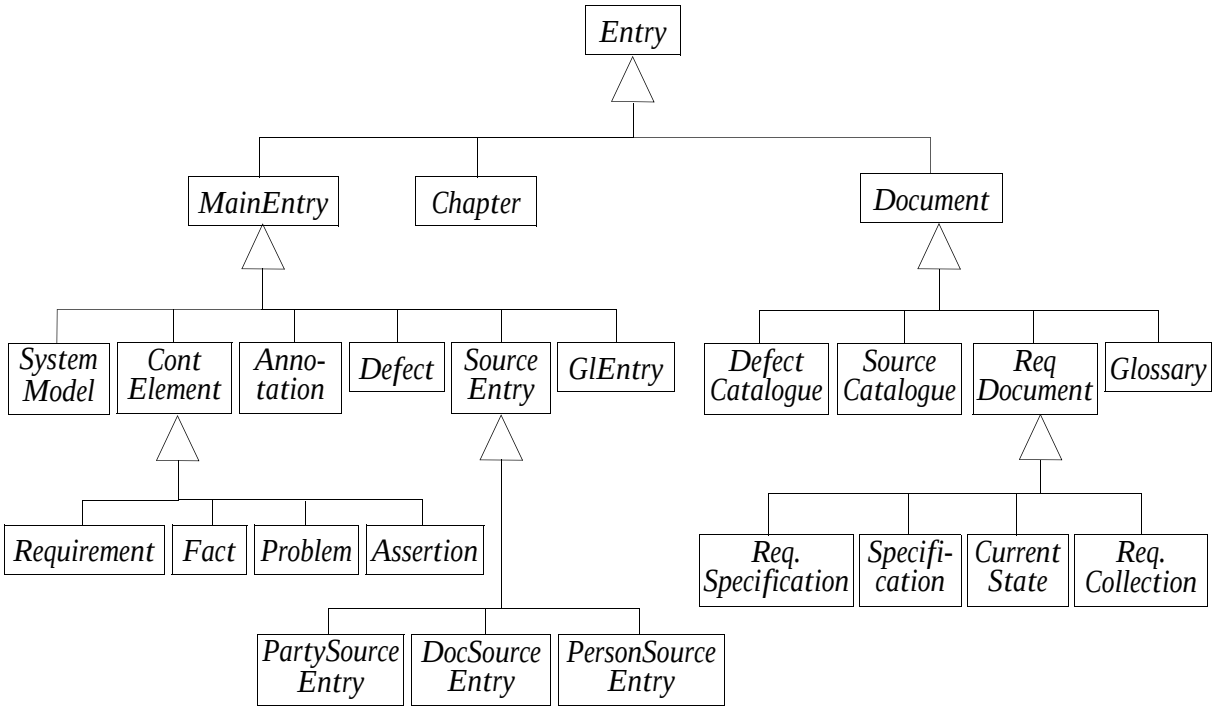


Abbildung 26: Eintragshierarchie

Im weiteren Verlauf dieses Berichts wird folgende Funktion benötigt:

Definition (*model: Entry* $\rightarrow$ *P(Model)*): Die Funktion *model* ordnet einem Eintrag das übergeordnete Informationsmodell zu: $model(e) = \{m \in Model \mid (e \in contains^*(m))\}$

6.12 Hilfsmengen und -funktionen

Folgende Hilfsmengen und -funktionen werden im folgenden benötigt:

Definition (*paragraphs: Entry* $\rightarrow$ *P(Paragraph)*): Die Funktion *paragraphs* ordnet einem Eintrag die Menge aller enthaltenen Freitexte zu:

$$paragraphs(e) = \begin{cases} \{ref(e)\}, & falls\ e \in SourceEntry \\ \{definition(e)\}, & falls\ e \in GlEntry \\ \{title(e), abstract(e)\}, & falls\ e \in ReqDocument \\ \{heading(e), introduction(e)\}, & falls\ e \in Chapter \\ \{contents(e)\} \cup descr(tests(e)), & falls\ e \in Requirement \cup Assertion \cup Fact \\ \{contents(e)\}, & falls\ e \in Problem \cup Annotation \\ \emptyset, & sonst \end{cases}$$

Definition (*CurrentSourceEntry(I)*): Die Menge aller aktuellen Quellenkatalogeinträge einer Dokumentation I:

$$CurrentSourceEntry(I) = \{s \in SourceEntry|_I \mid state(s) \neq rejected\}$$

Definition (*CurrentDocSourceEntry(I)*): Die Menge aller aktuellen externen Dokumente einer Dokumentation I:

$$CurrentDocSourceEntry(I) = \{s \in DocSourceEntry|_I \mid state(s) \neq rejected\}$$

Definition (*MainTerm(I)*): Die Menge aller aktuellen Glossareinträge einer Dokumentation I:

$$MainTerm(I) = \{g \in GlEntry|_I \mid state(g) \neq rejected\}$$

Definition (*ApplMainTerm(I)*): Die Menge aller aktuellen Glossareinträge des Anwendungsbereichs einer Dokumentation I:

$$ApplMainTerm(I) = \{g \in MainTerm(I) \mid scope(g) = domain\}$$

Definition (*CurrentContElement(doc)*): Die Menge aller aktuellen Inhaltselemente eines Anforderungsdokuments:

$$CurrentContElement(doc) = \{c \in ContElement|_{doc} \mid state(c) \neq rejected\}$$

Definition (*CurrentRequirement(doc)*): Die Menge aller aktuellen Anforderungen eines Anforderungsdokuments:

$$CurrentRequirement(doc) = \{r \in Requirement|_{doc} \mid state(r) \neq rejected\}$$

Definition (*EntryAnnotation(e), EntryAnnotation(E)*): Die Menge aller einem Eintrag (e) oder einer Menge von Einträgen (E) zugeordneten Annotationen:

$$EntryAnnotation(e) = \{a \in Annotation \mid e \in entries(a)\}$$

$$EntryAnnotation(E) = \bigcup_{e \in E} EntryAnnotation(e)$$

Definition (*DocAnnotation (doc)*): Die Menge aller (an aktuelle Einträge gebundener) Annotationen eines Anforderungsdokuments:

$$DocAnnotation(doc) = EntryAnnotation(doc \cup Chapter|_{doc} \cup CurrentContElement(doc) \cup SystemModel|_{doc})$$

Definition (*GlossaryAnnotation(I)*): Die Menge aller den aktuellen Glossareinträgen zugeordneten Annotationen:

$$GlossaryAnnotation(I) = EntryAnnotation(MainTerm(I))$$

Definition (*CurrentMainEntry(doc,I), CurrentMainEntry(doc), CurrentMainEntry(I)*): Die Menge aller aktuellen Haupteinträge, die in dem gegebenen Anforderungsdokument bzw. in allen Anforderungsdokumenten der Dokumentation enthalten sind, sowie alle aktuellen Glossareinträge und zugeordneten Annotationen:

$$CurrentMainEntry(doc) = CurrentContElement(doc) \cup SystemModel|_{doc} \cup DocAnnotation(doc)$$

$$CurrentMainEntry(doc, I) = CurrentMainEntry(doc) \cup MainTerm(I) \cup GlossaryAnnotation(I)$$

$$CurrentMainEntry(I) = \bigcup_{doc \in ReqDocument|_I} CurrentMainEntry(doc, I)$$

Definition (*NLDocEntry(doc)*): Die Menge aller aktuellen Einträge in einem Anforderungsdokument, die über Freitexte verfügen. Dazu gehören: das Anforderungsdokument selbst, alle Kapitel, Inhaltselemente und die dem Anforderungsdokument zugeordneten Annotationen:

$$NLDocEntry(doc) = doc \cup Chapter|_{doc} \cup CurrentContElement(doc) \cup DocAnnotation(doc)$$

Definition (*DocEntry(doc)*): Die Menge aller aktuellen Einträge in einem Anforderungsdokument:

$$DocEntry(doc) = NLDocEntry(doc) \cup SystemModel|_{doc}$$

Definition (*DocParagraph(doc)*): Die Menge der Freitexte aller aktuellen Einträge in einem Anforderungsdokument:

$DocParagraph(doc) = paragraphs(DocEntry(doc))$

Definition (*GlossaryEntry(I)*): Die Menge aller aktuellen Glossareinträge inkl. zugeordneter Annotationen:

$GlossaryEntry(I) = MainTerm(I) \cup GlossaryAnnotation(I)$

Definition (*ApplGlossaryEntry(I)*): Die Menge aller aktuellen Glossareinträge des Anwendungsbereichs inkl. zugeordneter Annotationen:

$ApplGlossaryEntry(I) = ApplMainTerm(I) \cup EntryAnnotation(ApplMainTerm(I))$

Definition (*UsedMainTerm(doc)*): Die Menge aller in einem Anforderungsdokument direkt benutzten, aktuellen Glossareinträge:

$UsedMainTerm(doc) = \{g \in MainTerm(I) | \exists p \in DocParagraph(doc) \bullet directuse(p, term(g))\}$

Definition (*UsedGlossaryEntry(doc)*): Die Menge aller in einem Anforderungsdokument direkt benutzten, aktuellen Glossareinträge und zugeordneten Annotationen:

$UsedGlossaryEntry(doc) = UsedMainTerm(doc) \cup EntryAnnotation(UsedMainTerm(doc))$

6.13 Qualitätskriterien

In Kapitel 5 werden neben strukturellen Eigenschaften des ADMIRE-Informationsmodells auch *Qualitätskriterien* beschrieben. In Tabelle 17 wird für jedes Qualitätskriterium ein Qualitätsprädikat definiert. Hierfür werden zuerst Mengen für *Einzeleinträge (SingleEntry)*, *NL-Einträge (NLEntry)* und *Dokumenteinträge (DocumentEntry)* eingeführt:

$SingleEntry = ContElement \cup Annotation \cup GlEntry \cup SystemModel \cup SourceEntry \cup Chapter \cup ReqDocument$

$NLEntry = ContElement \cup Annotation \cup GlEntry \cup SourceEntry \cup Chapter \cup ReqDocument$

$DocumentEntry = ReqDocument \cup Glossary \cup SourceCatalogue$

Qualitätskriterium	Prädikat
korrekt	*correct: SingleEntry* $\rightarrow$ *Boolean*
vollständig	*complete: SingleEntry* $\rightarrow$ *Boolean* *complete: ReqDocument* $\cup$ *SourceCatalogue* $\rightarrow$ *Boolean* *complete*[a]*: Glossary* $\times$ *P(SingleEntry)* $\rightarrow$ *Boolean*
konsistent	*consistent*: *P(SingleEntry)* $\rightarrow$ *Boolean*
extern konsistent	*externalconsistent: P(SingleEntry)* $\times$*P(DocSourceEntry)* $\rightarrow$ *Boolean*
adäquat	*adequate: ContElement* $\rightarrow$ *Boolean*
nicht redundant	*notredundant*: *P(ContElement)* $\rightarrow$ *Boolean*
nicht überspezifiziert	*notoverspecified: Requirement* $\rightarrow$ *Boolean*

Tabelle 17: Qualitätsprädikate

Qualitätskriterium	Prädikat
realisierbar	*achievable*: $P(Requirement) \rightarrow Boolean$
überprüfbar	*verifiable*: $Requirement \cup Fact \cup Assertion \rightarrow Boolean$
atomar	*atomic*: $ContElement \rightarrow Boolean$ *atomic*: $GlEntry \rightarrow Boolean$ *atomic*: $SourceEntry \rightarrow Boolean$
verständlich	*understandable*: $NLEntry \rightarrow Boolean$
eindeutig	*unambiguous*: $NLEntry \rightarrow Boolean$
präzise	*precise*: $ContElement \rightarrow Boolean$ *precise*: $GlEntry \rightarrow Boolean$ *precise*: $SourceEntry \rightarrow Boolean$
minimal	*minimal*: $NLEntry \rightarrow Boolean$ *minimal*: $ReqDocument \rightarrow Boolean$
normkonform	*conform*: $ReqDocument \rightarrow Boolean$

Tabelle 17: Qualitätsprädikate

a. Um die Vollständigkeit eines Glossars definieren zu können, muß angegeben werden, bzgl. welcher Einträge das Glossar vollständig sein soll (siehe auch Tabelle 23, S. 150).

Da Mängel höherer Ordnung vermieden werden sollen, werden die Qualitätskriterien nicht auf Mangeleinträgen oder Mängelkatalogen definiert.

Durch Mangeleinträge können Einträge der Dokumentation als mangelhalft markiert werden. Es sollen aber nur für solche Einträge Mangeleinträge erzeugt werden können, für die ein entsprechendes Qualitätskriterium definiert ist. Folgende Bedingungen müssen also erfüllt werden:

1. Mangeleinträge der Kategorie »inkorrekt« dürfen sich nur auf genau einen Einzeleintrag beziehen:
$$\forall d \in Defect \bullet ((class(d) = incorrect) \Rightarrow ((entries(d) \subset SingleEntry) \wedge (|entries(d)| = 1)))$$

2. Mangeleinträge der Kategorie »unvollständig« dürfen sich nur auf genau einen Einzeleintrag oder einen Dokumenteintrag beziehen:
$$\forall d \in Defect \bullet ((class(d) = incomplete) \Rightarrow ((entries(d) \subset SingleEntry \cup DocumentEntry) \wedge (|entries(d)| = 1)))$$

3. Mangeleinträge der Kategorie »inkonsistent« dürfen sich auf beliebig viele Einzeleinträge beziehen:
$$\forall d \in Defect \bullet ((class(d) \in inconsistent) \Rightarrow ((entries(d) \subset SingleEntry)))$$

4. Mangeleinträge der Kategorie »extern inkonsistent« dürfen sich auf beliebig viele Einzeleinträge beziehen, müssen sich aber auf mindestens ein externes Dokument beziehen:
$$\forall d \in Defect \bullet ((class(d) = external\ inconsistent) \Rightarrow ((entries(d) \subset SingleEntry) \wedge (\exists e \in entries(d) \bullet e \in DocSourceEntry)))$$

5. Mangeleinträge der Kategorie »nicht adäquat« dürfen sich nur auf genau ein Inhaltselement beziehen:

$$\forall d \in Defect \bullet ((class(d) = not\ adequate) \Rightarrow ((entries(d) \subset ContElement) \wedge (|entries(d)| = 1)))$$

6. Mangeleinträge der Kategorie »redundant« dürfen sich nur auf Inhaltselemente beziehen:

$$\forall d \in Defect \bullet ((class(d) = redundant) \Rightarrow (entries(d) \subset ContElement))$$

7. Mangeleinträge der Kategorie »überspezifiziert« dürfen sich nur auf genau eine Anforderung beziehen:

$$\forall d \in Defect \bullet ((class(d) = overspecified) \Rightarrow ((entries(d) \subset Requirement) \wedge (|entries(d)| = 1)))$$

8. Mangeleinträge der Kategorie »nicht realisierbar« dürfen sich nur auf Anforderungen beziehen:

$$\forall d \in Defect \bullet ((class(d) = unrealizable) \Rightarrow (entries(d) \subset Requirement))$$

9. Mangeleinträge der Kategorie »nicht prüfbar« dürfen sich nur auf genau eine Anforderung, ein Fakt oder eine Zusicherung beziehen:

$$\forall d \in Defect \bullet ((class(d) = not\ verifiable) \Rightarrow ((entries(d) \subset Requirement \cup Fact \cup Assertion) \wedge (|entries(d)| = 1)))$$

10. Mangeleinträge der Kategorie »mehrdeutig« oder »unverständlich« dürfen sich nur auf genau einen NL-Eintrag beziehen:

$$\forall d \in Defect \bullet ((class(d) \in \{ambiguous, incomprehensible\}) \Rightarrow ((entries(d) \subset NLEntry) \wedge (|entries(d)| = 1)))$$

11. Mangeleinträge der Kategorie »nicht atomar« und »unpräzise« dürfen sich nur auf genau ein Inhaltselement, einen Glossareintrag oder einen Quellenkatalogeintrag beziehen:

$$\forall d \in Defect|_I \bullet ((class(d) \in \{not\ atomic, imprecise\}) \Rightarrow ((entries(d) \subset ContElement \cup GlEntry \cup SourceEntry) \wedge (|entries(d)| = 1)))$$

12. Mangeleinträge der Kategorie »minimal« dürfen sich nur auf genau einen NL-Eintrag oder ein Anforderungsdokument beziehen:

$$\forall d \in Defect \bullet ((class(d) = minimal) \Rightarrow ((entries(d) \subset NLEntry \cup ReqDocument) \wedge (|entries(d)| = 1)))$$

13. Mangeleinträge der Kategorie »nicht normkonform« dürfen sich nur auf genau ein Anforderungsdokument beziehen:

$$\forall d \in Defect|_I \bullet ((class(d) = not\ conform) \Rightarrow ((entries(d) \subset ReqDocument) \wedge (|entries(d)| = 1)))$$

Mangeleinträge der Kategorie »sonstiges« dürfen sich auf beliebige Einträge (außer Mangeleinträge und Mängelkataloge) beziehen.

Kapitel 7

Prozeßmodell

In diesem Kapitel wird das *ADMIRE-Prozeßmodell* vorgestellt. Das Prozeßmodell macht, abgestimmt auf das Informationsmodell, einen sinnvollen Vorschlag, in welche Phasen der Requirements-Engineering-Prozeß eines typischen Projekts aufgeteilt werden kann, welche Tätigkeiten in den Phasen durchgeführt werden sollen, welche Ergebnisse am Ende der Phasen jeweils vorliegen sollen und welche Qualitätskriterien die Ergebnisse jeweils erfüllen sollen. Im Gegensatz zu den Vorgaben des ADMIRE-Informationsmodells sind die Vorgaben des Prozeßmodells nicht zwingend. In einem konkreten Prozeß kann hiervon abgewichen werden.

Kapitel 7.1 beschreibt die Phasen des Prozeßmodells und Kapitel 7.2 die dabei durchzuführenden Tätigkeiten. Abschließend wird in Kapitel 7.3 angegeben, welche Qualitätskriterien welche Teile der ADMIRE-Dokumentation zu welchen Zeitpunkten erfüllen sollen.

7.1 Phasen

Im Requirements Engineering werden oft zwei Phasen unterschieden: die *Problemanalyse-* und die *Spezifikationsphase* (siehe auch Kapitel 2.3). Da die erstellte Dokumentation im weiteren Projektverlauf verwendet und sehr wahrscheinlich geändert wird (Kapitel 2.3.3), werden im ADMIRE-Prozeßmodell auch die »*späteren Projektphasen*« berücksichtigt (Abbildung 27).

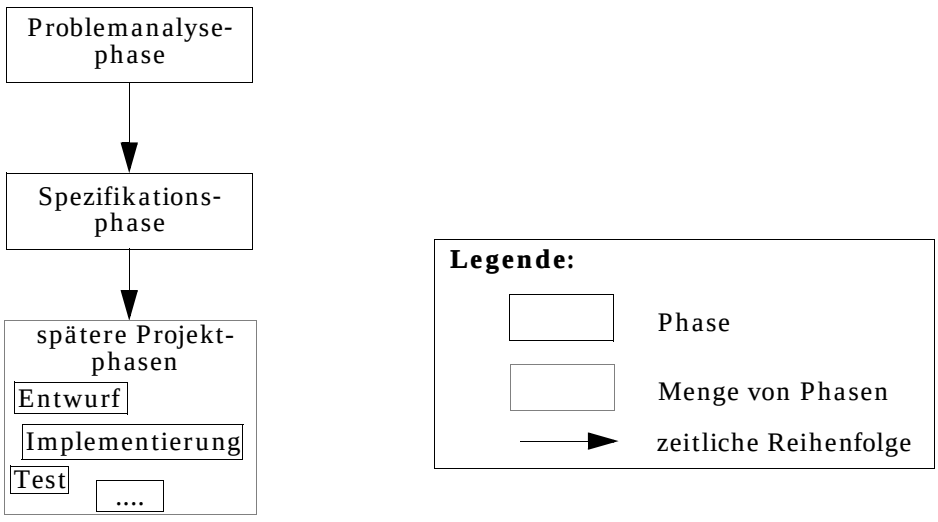


Abbildung 27: Phasen, in denen Requirements-Engineering-Tätigkeiten durchgeführt werden

Die Phasen werden in sequentieller Reihenfolge durchgeführt. Wird ein iterativer Entwicklungsansatz verfolgt oder folgt auf das aktuelle Projekt ein Anschlußprojekt, dann werden die Phasen in jedem (Teil-)Projekt durchgeführt.

Für jede Phase wird angegeben, welche Dokumente vorhanden sein müssen, welche Qualitätskriterien die Dokumente erfüllen sollen und welche Prüfmaßnahmen (Verifikation oder Validierung) durchgeführt werden sollen.

Ziel der Problemanalyse ist es, die »Wünsche« der Kunden zu ermitteln und zu dokumentieren. Die Sicht der Kunden, ihre Probleme und Lösungsvorstellungen stehen im Vordergrund. Darüber hinaus müssen Einschränkungen erkannt und dokumentiert werden, die aus anderen Quellen kommen, z.B. Gesetze und Normen. Hauptergebnisse der Problemanalysephase sind:

- ein *Lastenheft*, in dem die Probleme, Anforderungen und Rahmenbedingungen aus Sicht der Kunden und der sonstigen Informanten beschrieben werden,
- ein *Glossar*, in dem die projektrelevante Terminologie definiert wird, und
- ein *Quellenkatalog*, in dem die für das Projekt relevanten Informationsquellen beschrieben sind.

Ziel der Spezifikationsphase ist es, ein Zielsystem zu spezifizieren, das den ermittelten Wünschen und Rahmenbedingungen genügt. Hauptergebnisse der Spezifikationsphase sind:

- die erste Version des *Pflichtenhefts*[1], in dem das Zielsystem spezifiziert wird,
- das aktualisierte Glossar und
- der aktualisierte Quellenkatalog.

Ziel der Tätigkeiten in den späteren Projektphasen ist es, das Pflichtenheft, das Glossar und den Quellenkatalog immer aktuell zu halten. Wenn neue Informationen oder Wünsche auftauchen, wenn sich vorhandene Einträge ändern oder wenn Mängel erst in späteren Phasen entdeckt werden, werden sie angepaßt. Das Pflichtenheft sollte (im Idealfall) zu jedem Zeitpunkt die aktuellen Anforderungen widerspiegeln.

7.1.1 Problemanalysephase

In der Problemanalysephase werden Anforderungen und Rahmenbedingungen aus Sicht der Kunden und der sonstigen für das Projekt relevanten Informanten erhoben und dokumentiert. Dabei müssen in der Regel folgende Tätigkeiten durchgeführt werden:

- Die Systemanalytiker *erheben Informationen*. Dazu müssen Personen befragt, externe Dokumente gelesen oder die Arbeitsumgebung analysiert werden.

  Kann der Systemanalytiker die Tätigkeit alleine am Arbeitsplatz durchführen, können die Ergebnisse direkt in die Dokumentation eingetragen werden. Anson-

1. Wenn in diesem Kapitel von »einem« Pflichtenheft die Rede ist, ist implizit immer auch der Fall eingeschlossen, daß es mehrere Anforderungsdokumente der Kategorie »Pflichtenheft« gibt, die zusammen das »eigentliche« Pflichtenheft bilden. Analoges gilt auch für Lastenhefte und Ist-Analysedokumente.

sten entsteht zuerst ein Protokoll, das anschließend von den Systemanalytikern überarbeitet werden muß:

- Begriffsdefinitionen werden in das Glossar übernommen.
- Informationen über (neue) Informationsquellen werden in den Quellenkatalog eingetragen.
- Alle sonstigen Informationen werden in eine neue Anforderungssammlung eingetragen.

Die »sonstigen« Informationen können sowohl den Ist- als auch den Soll-Zustand betreffen. Zum Ist-Zustand gehören Informationen über die (aktuelle) Arbeitsumgebung, über die (aktuellen) Benutzer, über deren Arbeitsabläufe, über vorhandene Probleme und über das zu ersetzende oder zu erweiternde System. Zum Soll-Zustand gehören Informationen über zu lösende Probleme, über das Zielsystem, über die zukünftige Arbeitsumgebung und über die zukünftigen Arbeitsabläufe.

- Sobald die Systemanalytiker einen gewissen Einblick in die Problemstellung haben, können sie eigene *Lösungsideen entwickeln*, die als Diskussionsgrundlage verwendet werden können. Diese Lösungsideen müssen ebenfalls als Anforderungssammlung dokumentiert werden.
- Die Anforderungssammlungen selbst sind nur ein Zwischenergebnis. Alle enthaltenen Informationen über den Ist-Zustand müssen in das Ist-Analysedokument *integriert* werden, alle enthaltenen Informationen über den Soll-Zustand in das Lastenheft.
- In der Problemanalysephase dürfen das Lastenheft und die Anforderungssammlungen durchaus noch Widersprüche enthalten. Widersprüche deuten oft auf prinzipiell unterschiedliche Sichtweisen der Informanten hin. Wenn solche grundlegenden Probleme identifiziert werden, müssen sie so früh wie möglich gelöst werden. Hierfür müssen *Konfliktlösungs- oder Entscheidungsverfahren durchgeführt* werden.
- Bevor ein Dokument als »fertig« akzeptiert wird, muß es, u.U. mehrfach, *verifiziert*, *validiert* und *überarbeitet* werden. Bei einer Prüftätigkeit entsteht insbesondere dann, wenn sie nicht von den Systemanalytikern durchgeführt wird, als Zwischenergebnis zuerst ein Protokoll. Die Systemanalytiker verwenden das Protokoll als Vorlage, um Mangeleinträge zu erzeugen.

  Zu folgenden Zeitpunkten müssen folgende Dokumente akzeptiert sein:

  - Eine Anforderungssammlung mit den sie betreffenden Teilen des Glossars muß von den an der entsprechenden Informationserhebungstätigkeit beteiligten Informanten validiert[1] und akzeptiert werden, bevor sie in das Lastenheft oder Ist-Analysedokument integriert werden darf.
  - Das Ist-Analysedokument und das Glossar des Anwendungsbereichs müssen von den Informanten validiert und akzeptiert werden, bevor die Problemanalysephase endet.

---

1. Bevor ein Dokument validiert wird, muß es verifiziert und akzeptiert worden sein. Siehe auch Kapitel 7.1.4.

- Lastenheft, Glossar und Quellenkatalog müssen von den Systemanalytikern verifiziert und akzeptiert werden, bevor die Problemanalysephase endet.
- Der Quellenkatalog muß von den Informanten validiert und akzeptiert werden, bevor die Problemanalysephase endet.

In Tabelle 18 werden die Tätigkeiten aufgelistet, die in der Problemanalysephase durchgeführt werden müssen, sowie die Dokumente, die bei den Tätigkeiten hauptsächlich erstellt oder geändert werden.

Tätigkeit	Hauptergebnisse	Kapitel
Bereite Dokumentation vor	Anforderungssammlung, Quellenkatalog, Glossar	7.2.1
Erhebe Informationen	Anforderungssammlung, Quellenkatalog, Glossar	7.2.2
Entwickle Lösungsideen	Anforderungssammlung	7.2.3
Integriere Anforderungssammlungen	Lastenheft Ist-Analysedokument	7.2.4
Verifiziere Anforderungsdokument	(evtl. Protokoll), Mängelkatalog	7.2.5
Validiere Anforderungsdokument	Protokoll, Mängelkatalog	7.2.6
Führe Entscheidungsverfahren durch	Protokoll, Mängelkatalog	7.2.7
Überarbeite Anforderungsdokument	geändertes Glossar, geändertes Anforderungsdokument, geänderter Quellenkatalog	7.2.8

Tabelle 18: Tätigkeiten in der Problemanalysephase

Als weiterzuverwendende Dokumente liegen vor: ein *Glossar*, ein *Quellenkatalog* und ein *Lastenheft*. Die Anforderungssammlungen und das Ist-Analysedokument werden von jetzt an nicht weiter gepflegt.

7.1.2 Spezifikationsphase

Ziel der *Spezifikationsphase* ist es, ein Zielsystem zu spezifizieren, das den ermittelten Wünschen und Rahmenbedingungen genügt. In dieser Phase müssen zwei Haupttätigkeiten durchgeführt werden:

- Widersprüche im Lastenheft müssen entdeckt und beseitigt werden. Daran müssen in der Regel die betroffenen Informanten beteiligt werden. Ziel ist ein widerspruchsfreies Lastenheft, das die Anforderungen und Rahmenbedingungen aus Sicht der Kunden und sonstigen Informanten beschreibt.
- Die Systemanalytiker müssen das *Pflichtenheft* erstellen. Fehlende Informationen müssen nachgetragen werden, Vagheiten müssen beseitigt werden.

Beide Tätigkeiten können bis zu einem gewissen Grad parallel durchgeführt werden, wobei es in vielen Fällen sicher sinnvoll ist, Widersprüche zu klären, bevor bestimmte Teile des Pflichtenhefts geschrieben werden.

Vor Ende der Spezifikationsphase müssen Lastenheft, Pflichtenheft und Glossar akzeptiert werden:

- Das Lastenheft, das Glossar und der Quellenkatalog müssen von den Systemanalytikern verifiziert und akzeptiert werden.
- Das Pflichtenheft und das Glossar müssen von den Kunden validiert und akzeptiert werden.

In Tabelle 19 werden die Tätigkeiten aufgelistet, die in der Spezifikationsphase durchgeführt werden müssen, sowie die Dokumente, die bei den Tätigkeiten hauptsächlich erstellt oder geändert werden.

Tätigkeit	Hauptergebnisse	Kapitel
Erstelle Pflichtenheft	Pflichtenheft, Glossar	7.2.9
Verifiziere Anforderungsdokument	(evtl. Protokoll), Mängelkatalog	7.2.5
Validiere Anforderungsdokument	Protokoll, Mängelkatalog	7.2.6
Führe Entscheidungsverfahren durch	Protokoll, Mängelkatalog	7.2.7
Überarbeite Anforderungsdokument	geändertes Glossar, geändertes Anforderungsdokument, geänderter Quellenkatalog	7.2.8

Tabelle 19: Tätigkeiten in der Spezifikationsphase

Als weiterzuverwendende Dokumente liegen vor: ein *Glossar*, ein *Quellenkatalog* und ein *Pflichtenheft*. Das Lastenheft wird von jetzt an nicht weiter gepflegt.

7.1.3 Spätere Projektphasen

Ziel der Tätigkeiten in den *späteren Projektphasen* ist es, das Pflichtenheft, das Glossar und den Quellenkatalog immer aktuell zu halten.

Da das Pflichtenheft oft als Vertragsgrundlage dient, darf es nicht unkontrolliert geändert werden. Es muß ein *Änderungsmanagement* eingeführt werden, so daß Änderungen nur auf kontrollierte Weise durchgeführt werden.

Änderungen werden immer dadurch initiiert, daß eine *Änderungsmeldung* eingereicht wird. Urheber kann jede an dem Projekt beteiligte Person sein, z.B. ein Kunde, der eine neue Anforderung einbringen möchte, ein Systemanalytiker, der einen Widerspruch im Pflichtenheft gefunden hat oder ein Entwickler, der festgestellt hat, daß eine Anforderung so nicht realisiert werden kann.

Ein Systemanalytiker muß die Änderungsmeldung entgegennehmen, sie als Mangeleintrag dokumentieren, eine Entscheidung bzgl. der durchzuführenden Änderung unter Beteiligung der betroffenen Personen initiieren und abschließend die Änderung durchführen, wenn die Entscheidung positiv ausgefallen ist.

In Tabelle 20 werden die Tätigkeiten aufgelistet, die in den späteren Projektphasen durchgeführt werden müssen, sowie die Dokumente, die bei den Tätigkeiten hauptsächlich erstellt oder geändert werden.

Tätigkeit	Hauptergebnisse	Kapitel
Bearbeite Änderungsmeldung	Mängelkatalog	7.2.10
Führe Entscheidungsverfahren durch	Protokoll, Mängelkatalog	7.2.7
Überarbeite Anforderungsdokument	geändertes Glossar, geändertes Anforderungsdokument, geänderter Quellenkatalog	7.2.8

Tabelle 20: Requirements-Engineering-Tätigkeiten in den späteren Projektphasen

7.1.4 Lebenszyklen der Anforderungsdokumente

Ein Anforderungsdokument kann im Laufe eines Projekts unterschiedliche Zustände annehmen. Die Menge der möglichen Zustände ist abhängig von der Art des Anforderungsdokuments. Wird der Requirements-Engineering-Prozeß gemäß der Vorgaben aus den Kapiteln 7.1.1 bis 7.1.3 durchgeführt, dann ergeben sich für die Anforderungsdokumente die in Abbildung 28 beschriebenen Zustände und Zustandsübergänge.

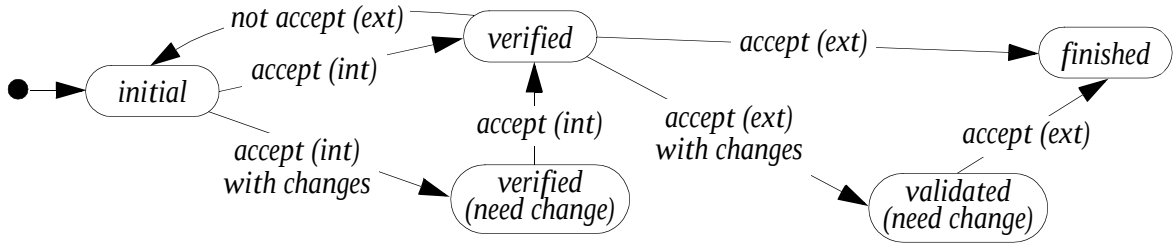
Wenn ein Anforderungsdokument neu angelegt wird, dann ist es im Zustand *initial*. Diesen Zustand behält es bei, bis eine Prüfung auf diesem Anforderungsdokument vorgenommen wurde und es mit oder ohne Änderungen akzeptiert wurde. Mögliche Prüfungen sind Verifikation und Validierung.

Anforderungssammlungen und Ist-Analysedokumente müssen zuerst verifiziert und anschließend validiert werden. Nachdem sie erfolgreich validiert und die notwendigen Änderungen integriert wurden, dürfen sie nicht mehr geändert werden.

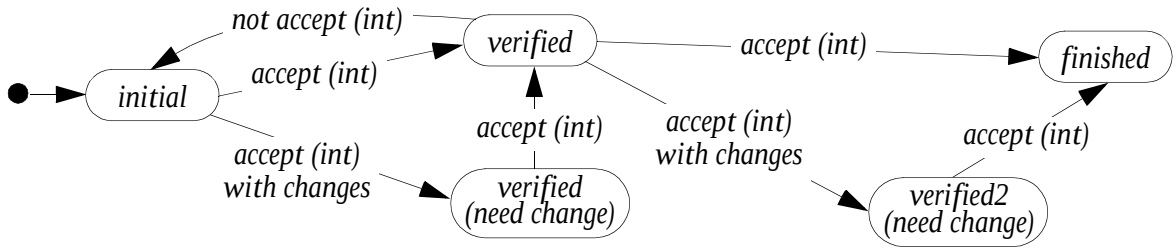
Lastenhefte müssen nur verifiziert werden. Da am Ende der Problemanalysephase und in der Spezifikationsphase unterschiedliche Kriterien angewandt werden sollen (siehe Kapitel 7.3), muß jedes Lastenheft zweimal verifiziert werden. Nach der zweiten Verifikation darf es nicht mehr geändert werden.

Pflichtenhefte müssen verifiziert und validiert werden. Da das Pflichtenheft immer aktuell bleiben soll, darf es auch nach der Validierung geändert werden. Für Pflichtenhefte gibt es somit keinen Endzustand.

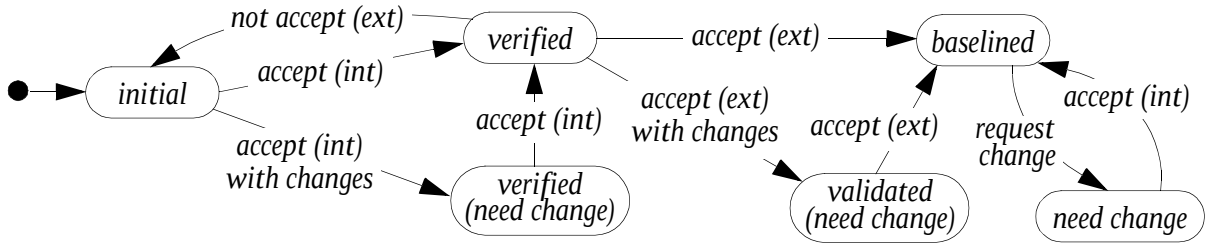
Der Zustand eines Anforderungsdokuments kann immer nach einer Prüfung geändert werden, je nachdem, ob es in der Prüfung akzeptiert (*accept (int)* bzw. *accept (ext)*), mit Änderungen akzeptiert (*accept (int) with changes* bzw. *accept (ext) with changes*) oder nicht akzeptiert (*not accept (int)* bzw. *not accept (ext)*) wird. Wenn ein Anforderungsdokument während einer Prüfung »mit Änderungen akzeptiert« wird, muß es vor endgültiger Freigabe von einer zuständigen Person ohne weitere Änderungen akzeptiert werden. Bei einer Verifikation werden die Dokumente intern, d.h. von den Systemanalytikern akzeptiert (*accept (int)*), bei einer Validierung werden sie extern, d.h. von den Kunden oder Informanten akzeptiert (*accept (ext)*). Wird ein Anforderungsdokument bei einer Prüfung nicht akzeptiert, nimmt es wieder den Zustand



(a) Anforderungssammlung / Ist-Analysedokument



(b) Lastenheft



(c) Pflichtenheft

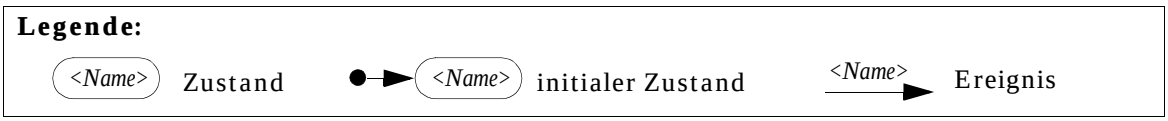


Abbildung 28: Lebenszyklus eines Anforderungsdokuments

initial an. Einzige Ausnahme hiervon ist ein Pflichtenheft im Zustand *need change* (siehe unten).

Beim Pflichtenheft gibt es noch eine Besonderheit zu beachten: Sobald am Ende der Spezifikationsphase der Zustand *baselined* erreicht ist, wird es nur noch auf Grund von Änderungsmeldungen geändert. Wenn solche vorliegen (*request change*), wird der Zustand auf *need change* geändert. Werden die Änderungen akzeptiert, wird wieder der Zustand *baselined* angenommen.

Anforderungsdokumente dürfen nur dann geändert werden, wenn sie in einem der Zustände *initial*, *verified (need change)*, *validated (need change)* oder *need change* sind.

Es kann durchaus vorkommen, daß ein »akzeptiertes« Anforderungsdokument noch aktuelle Mängel enthält. Die Praxis zeigt, daß nicht immer alle Mängel und offenen Punkte beseitigt werden können. Es ist Aufgabe der Prüfer zu entscheiden, ob alle »wichtigen« Mängel beseitigt sind.

In einem konkreten Prozeß kann von diesen Vorgaben abgewichen werden. In Projekten, in denen z.B. Sicherheit eine wichtige Rolle spielt, müssen die Informanten stärker involviert werden. Das könnte bedeuten, daß z.B. auch das Lastenheft und jede Änderung am Pflichtenheft in den späteren Projektphasen extern akzeptiert werden müssen.

7.2 Tätigkeiten

In Kapitel 7.1 wurden folgende Tätigkeiten identifiziert: »bereite Dokumentation vor«, »erhebe Informationen«, »entwickle Lösungsideen«, »integriere Anforderungssammlungen«, »verifiziere Anforderungsdokument«, »validiere Anforderungsdokument«, »führe Entscheidungsverfahren durch«, »überarbeite Anforderungsdokument«, »erstelle Pflichtenheft« und »bearbeite Änderungsmeldung«.

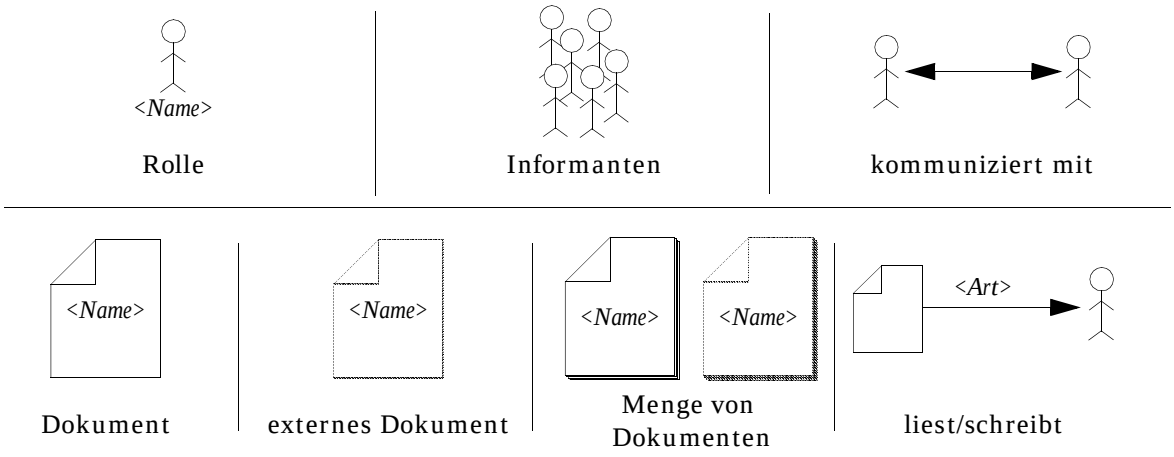


Abbildung 29: Notation für das ADMIRE-Prozeßmodell

In den folgenden Unterkapiteln (7.2.1-7.2.10) werden die einzelnen Tätigkeiten genauer beschrieben. Zu jeder Tätigkeit wird angegeben, welche Personen daran beteiligt sind, welche Dokumente zu Beginn vorliegen müssen und welche Dokumente während der Tätigkeit neu erzeugt oder geändert werden. Hierfür wird eine graphische Notation (siehe Abbildung 29) verwendet, die folgende Entitäten und Beziehungen enthält:

- *Rolle*: eine Person oder Menge von Personen, die bzgl. des Projekts die gleiche Rolle einnehmen. Mögliche Rollen sind: »Systemanalytiker« und »Informant«.
- *Informanten*.
- *Kommuniziert-Mit-Beziehung*: Beziehung, durch die ausgedrückt wird, daß Personen (Vertreter der angegebenen Rollen) miteinander kommunizieren.
- *Dokument*: einzelnes Dokument einer Dokumentation; entweder ein Ist-Analysedokument, eine Anforderungssammlung, ein Lastenheft, ein Pflichtenheft, ein Glossar, ein Quellenkatalog oder ein Mängelkatalog.
- *externes Dokument*: Dokument, das nicht Teil der Dokumentation ist. Mögliche externe Dokumente sind: »Report«, »Protokoll« und »Projektdokument«.
- *Menge von (externen) Dokumenten*.

- *Liest/Schreibt-Beziehung*: Beziehung, durch die ausgedrückt wird, daß ein Vertreter der angegebenen Rolle lesend oder schreibend auf das angegebene Dokument zugreift. Die Art des Zugriffs wird genauer angegeben. Mögliche Angaben sind: »dokumentiert Entscheidungen«, »erzeugt«, »liest«, »überarbeitet«, »validiert« oder »verifiziert«.

7.2.1 Bereite Dokumentation vor

Diese Tätigkeit muß zu Beginn der Requirements-Engineering-Phase durchgeführt werden. Hierbei wird eine neue Dokumentation angelegt. Es werden erste grundlegende Informationen eingetragen.

Wenn ein Projekt in die Problemanalysephase eintritt, dann sind einige für das Requirements Engineering relevante Informationen vorhanden, z.B. eine Beschreibung der Ziele des Projekts und eine initiale Menge von Informationsquellen. Zumindest der Kunde sollte zu Projektbeginn bekannt sein. In dieser Tätigkeit werden ein Quellenkatalog, ein Glossar und eine Anforderungssammlung angelegt und die vorhandenen Informationen eingetragen (Abbildung 30).

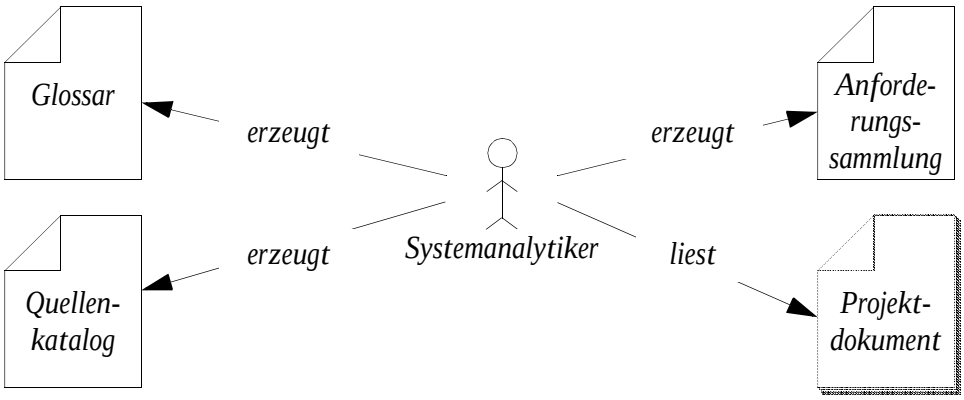


Abbildung 30: Bereite Dokumentation vor

7.2.2 Erhebe Informationen

Ziel dieser Tätigkeit ist es, die Wünsche und Rahmenbedingungen einiger Informanten zu ermitteln. Häufig werden hierbei Brainstorming- oder Interviewtechniken eingesetzt (siehe auch Kapitel 2.3.4). Je nach Projekt variieren die durchzuführenden Informationserhebungstätigkeiten und zu beteiligenden Informanten. Deswegen können diesbezüglich keine genaueren Angaben gemacht werden.

Die Systemanalytiker können die benötigten Informationen auf vielfältige Weisen gewinnen: sie können Informanten befragen, externe Dokumente lesen oder die aktuellen Arbeitsabläufe beobachten. In vielen Fällen ist es nicht möglich, daß die Systemanalytiker die Informationserhebung alleine an ihrem Arbeitsplatz durchführen. Folgende Teilschritte müssen z.B. bei einer Befragung von Informanten durchgeführt werden:

- Durchführung des Interviews: Hierbei entsteht ein *Protokoll*, das von den Systemanalytikern oder den Informanten erstellt werden kann (Abbildung 31).

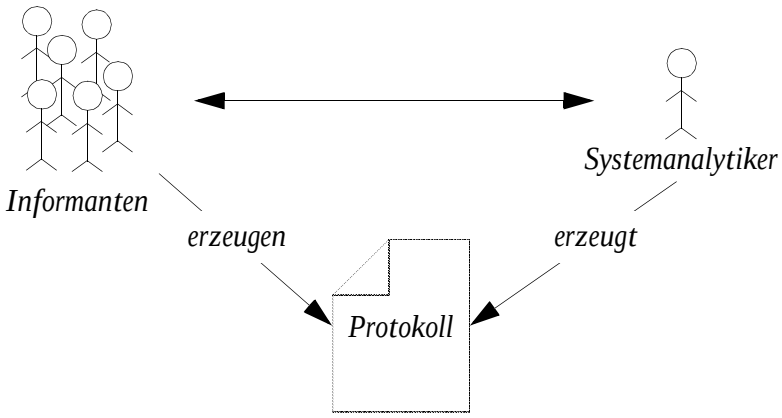


Abbildung 31: Erhebe Informationen von Informanten

- Überarbeitung des Protokolls: Die Systemanalytiker überarbeiten das Protokoll zu einer Anforderungssammlung. Hierbei müssen evtl. auch das Glossar oder der Quellenkatalog geändert werden. Falls Mängel erkannt werden, werden sie in den Mängelkatalog eingetragen und, falls es sich um geringfügige Mängel handelt, behoben. Die Entscheidung, ob ein Mangel geringfügig ist, muß von den Systemanalytikern getroffen werden. Es gehört aber nicht zu dieser Tätigkeit, explizit nach Mängeln zu suchen (Abbildung 32).

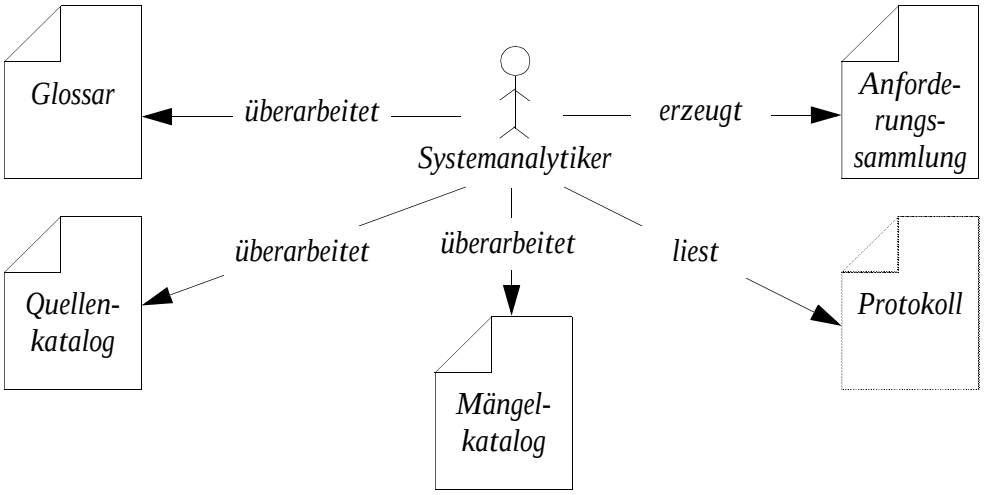


Abbildung 32: Überarbeite Protokoll

Wenn der Systemanalytiker die Informationserhebung alleine am Arbeitsplatz durchführen kann (z.B. wenn er ein externes Dokument auswertet), muß nicht unbedingt ein Protokoll erstellt werden. In diesem Fall kann er die Informationen auch direkt in die Dokumentation eintragen.

7.2.3 Entwickle Lösungsideen

In der Praxis tritt oft folgendes Problem auf: Die Informanten, insbesondere die Kunden, sind Experten in ihren Fachgebieten. Sie haben eine klare Vorstellung von den Problemen, die während des Projekts gelöst werden sollen. Sie haben aber nur geringe Kenntnisse über die Software-Entwicklung und die daraus resultierenden Möglichkeiten (Yu, 1997, S. 226).

Wenn dieser Fall auftritt, müssen die Systemanalytiker Vorschläge unterbreiten oder Alternativen aufzeigen können. Sie müssen sich also schon früh Gedanken über Lösungsmöglichkeiten für die Probleme machen und diese dokumentieren. Die Lösungsideen werden in Anforderungssammlungen eingetragen (Abbildung 33). Diese Anforderungssammlungen können bei weiteren Informationserhebungstätigkeiten als Diskussionsgrundlage verwendet werden.

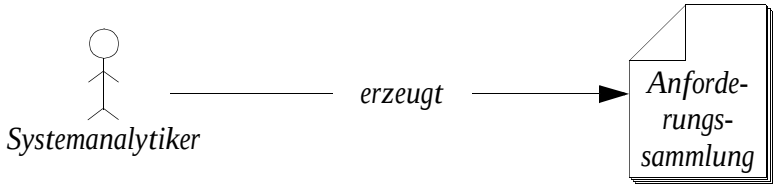


Abbildung 33: Entwickle initiale Lösungsideen

7.2.4 Integriere Anforderungssammlungen

Zu Beginn dieser Tätigkeit liegt den Systemanalytikern eine Menge von akzeptierten, d.h. verifizierten und validierten Anforderungssammlungen vor. Da Anforderungssammlungen die Wünsche und Meinungen einzelner Informanten reflektieren und nur den für sie relevanten Bereich abdecken, können sie Widersprüche enthalten und unvollständig sein.

Die Systemanalytiker integrieren diese Informationen in das Lastenheft und das Ist-Analysedokument. Sind die Dokumente noch nicht vorhanden, werden sie neu erzeugt (Abbildung 34). Hierbei soll der Systemanalytiker insbesondere auf folgende Punkte achten:

- Wenn in mehreren Anforderungssammlungen die *gleichen Einträge* vorkommen, werden sie zusammengefaßt. Zwei Einträge können als gleich betrachtet werden, wenn sie die gleiche Bedeutung (Semantik) haben, sich evtl. aber in der gewählten Formulierung (Syntax) unterscheiden. Liegen semantische Unterschiede vor, z.B. unterschiedliche Prioritäten bei Anforderungen, dürfen sie nicht zusammengefaßt werden.
- Inhaltlich zusammengehörende Einträge, z.B. alle Anforderungen an die Benutzungsschnittstelle, werden nach Möglichkeit in das gleiche Kapitel eingetragen.

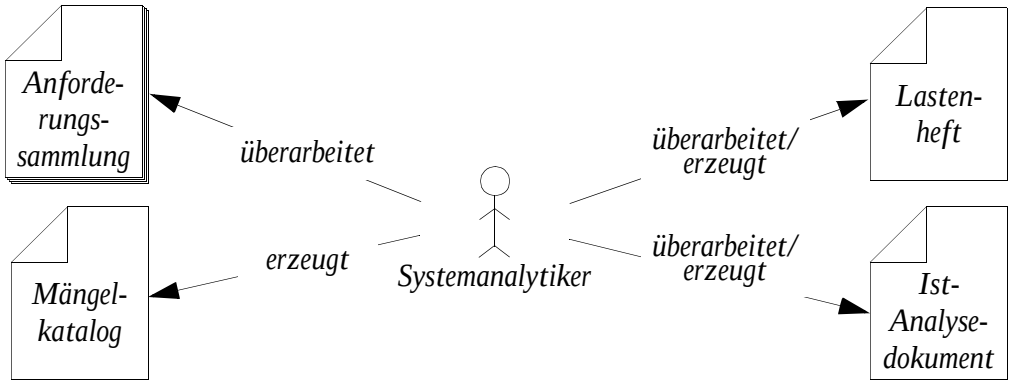


Abbildung 34: Integriere Anforderungssammlungen

Falls bei der Integration Mängel erkannt werden, werden sie in den Mängelkatalog eingetragen und, falls es sich um geringfügige Mängel handelt, behoben. Die

Entscheidung, ob ein Mangel geringfügig ist, muß auch hier von den Systemanalytikern getroffen werden (siehe auch Kapitel 7.2.2). Es gehört aber nicht zu dieser Tätigkeit, explizit nach Mängeln zu suchen.

7.2.5 Verifiziere Dokument

Die *Verifikation* ist eine Prüftätigkeit, die von den Systemanalytikern alleine, d.h. ohne Beteiligung der Informanten, durchgeführt werden kann. Zusammen mit dem Anforderungsdokument werden immer auch Teile des Glossars verifiziert. Je nach Art des Dokuments und Zeitpunkt der Verifikation gelten unterschiedliche Qualitätskriterien (siehe Kapitel 7.3).

Die Verifikation kann am Rechner oder am Schreibtisch durchgeführt werden. Verifikationstechniken werden in Kapitel 2.3.4 beschrieben. Wird die Verifikation am Rechner durchgeführt, können direkt Mangeleinträge erzeugt werden (Abbildung 35), ansonsten entsteht, analog zur Validierung, zuerst ein Protokoll, daß anschließend von den Systemanalytikern überarbeitet werden muß (siehe Kapitel 7.2.6).

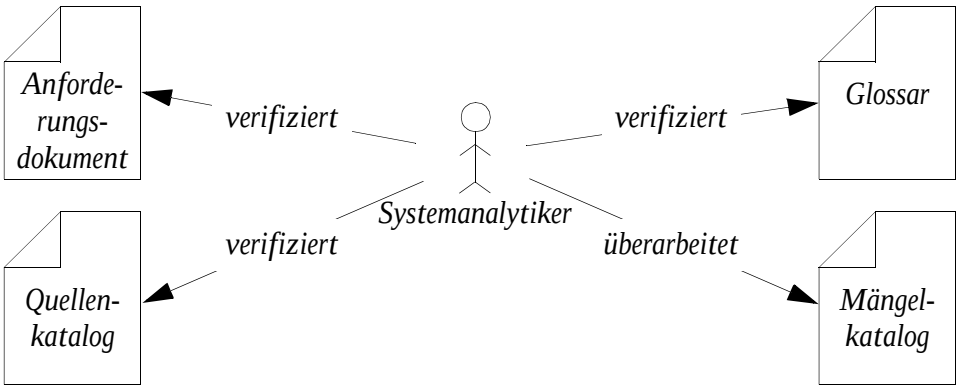


Abbildung 35: Verifiziere Anforderungsdokument

In Tabelle 21 wird aufgelistet, welche Dokumente in welcher Phase verifiziert werden sollen (siehe auch Kapitel 7.1.4).

Phase	zu verifizierende Dokumente
Problemanalysephase	Anforderungssammlung, Glossar (benutzte Terme)
	Ist-Analysedokument, Glossar (Anwendungsbereich)
	Lastenheft, Glossar
	Quellenkatalog
Spezifikationsphase	Lastenheft, Glossar
	Quellenkatalog
	Pflichtenheft, Glossar
spätere Projektphasen	Pflichtenheft, Glossar, Quellenkatalog

Tabelle 21: Zuordnung: Phase → zu verifizierende Dokumente

Am Ende der Verifikation muß von den Systemanalytikern eine Entscheidung darüber getroffen werden, ob oder inwieweit das Dokument akzeptiert wird. Folgende Möglichkeiten gibt es: akzeptiert, akzeptiert, aber Überarbeitung notwendig, und nicht akzeptiert, Überarbeitung und erneute Verifikation notwendig.

7.2.6 Validiere Dokument

Bei der *Validierung* wird, ebenso wie bei der Verifikation, entweder ein Anforderungsdokument zusammen mit Teilen des Glossars oder der Quellenkatalog geprüft. Dabei gibt es aber einige Unterschiede zur Verifikation:

- Voraussetzung für die Validierung ist ein verifiziertes Dokument.
- Die Prüfung muß von den Informanten durchgeführt werden.
- Zur Validierung werden andere Kriterien angewandt (siehe Kapitel 7.3).
- Die Informanten erzeugen ein Protokoll, das von den Systemanalytikern in den Mängelkatalog integriert wird.

Die Validierung wird in zwei wesentlichen Teilschritten durchgeführt:

- Die zu validierenden Dokumente werden den beteiligten Informanten vorgelegt. Diese führen die eigentliche Validierungstätigkeit durch. Sie müssen die Dokumente begutachten und alle Mängel, die sie darin finden, protokollieren. Hierbei entsteht ein Protokoll. Die Systemanalytiker nehmen »nur« eine unterstützende oder leitende Rolle ein. Ihre Aufgabe ist es, die Validierungstätigkeiten zu initiieren und bei Fragen inhaltlicher oder organisatorischer Art unterstützend einzugreifen (Abbildung 36).

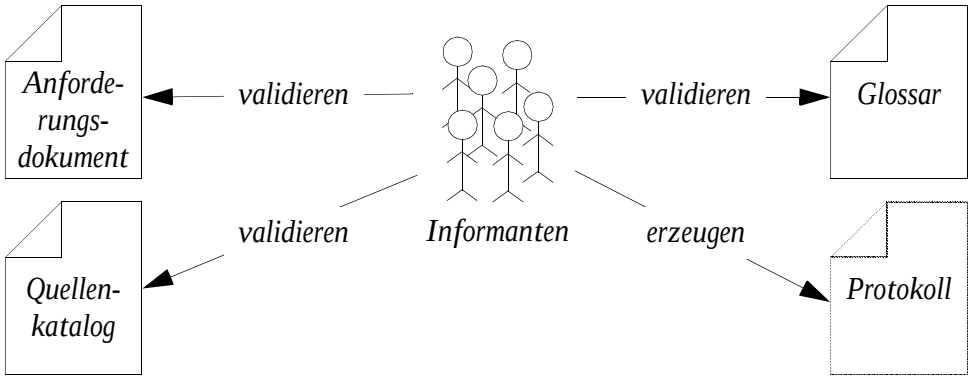


Abbildung 36: Validiere Anforderungsdokument

- Das Protokoll wird von den Systemanalytikern als Vorlage für Mangeleinträge verwendet. Für jeden Protokolleintrag wird ein Mangeleintrag in den Mängelkatalog eingetragen. An der Behebung der Mängel müssen u.U. die Informanten wieder beteiligt werden (Abbildung 37, siehe auch Kapitel 7.2.7).

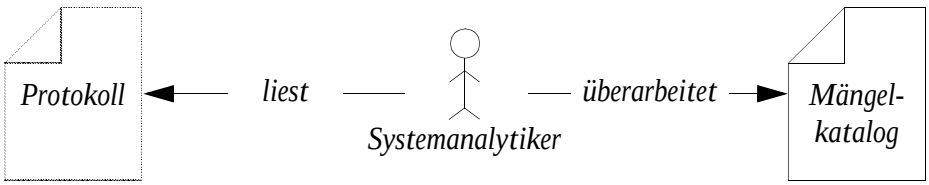


Abbildung 37: Integriere Protokoll

In Tabelle 22 wird aufgelistet, welche Dokumente in welcher Phase validiert werden sollen (siehe auch 7.1.4). Validierungstechniken, insbesondere Reviews, werden in Kapitel 2.3.4 beschrieben.

Phase	zu validierende Dokumente
Problemanalysephase	Anforderungssammlung, Glossar (benutzte Terme)
	Ist-Analysedokument, Glossar (Anwendungsbereich)
	Quellenkatalog
Spezifikationsphase	Pflichtenheft, Glossar, Quellenkatalog

Tabelle 22: Zuordnung: Phase → zu validierende Dokumente

Am Ende der Validierungstätigkeit muß von den Informanten eine Entscheidung darüber getroffen werden, ob oder inwieweit das Dokument akzeptiert wird.

7.2.7 Führe Entscheidungsverfahren durch

Ziel dieser Tätigkeit ist es, aktuelle Mängel in einem Anforderungsdokument, in dem Glossar oder in dem Quellenkatalog zu beheben. In dieser Tätigkeit geht es nur um Mängel, an deren Behebung Informanten beteiligt werden müssen. Unter Umständen müssen Entscheidungen getroffen werden, mit denen einzelne Personen nicht einverstanden sind. Es müssen also Personen beteiligt werden, die befugt sind, solche Entscheidungen zu treffen.

Insbesondere in den späteren Projektphasen dürfen Änderungen am Pflichtenheft nicht unkontrolliert durchgeführt werden. Das Management muß eingeschaltet werden. Eventuell müssen neue Verhandlungen mit den Kunden geführt werden. Von diesen sind die Systemanalytiker zwar nicht unbedingt direkt betroffen, sie müssen aber abwarten, wie die Entscheidung ausfällt. Erst danach wird die Änderung durchgeführt (oder auch nicht).

Folgende Teilschritte müssen durchgeführt werden:

- Aus der Menge der vorhandenen Mängel muß eine Teilmenge ausgewählt werden, die sinnvoll z.B. während einer Sitzung unter Beteiligung der gleichen Personen besprochen werden kann.
- Die Mangeleinträge müssen aufbereitet werden. Die Systemanalytiker müssen Lösungsalternativen und -vorschläge entwickeln und deren Konsequenzen abwägen. Aus diesen Informationen erzeugen sie einen Report, der als Diskussionsgrundlage für das Entscheidungsverfahren verwendet wird (Abbildung 38).

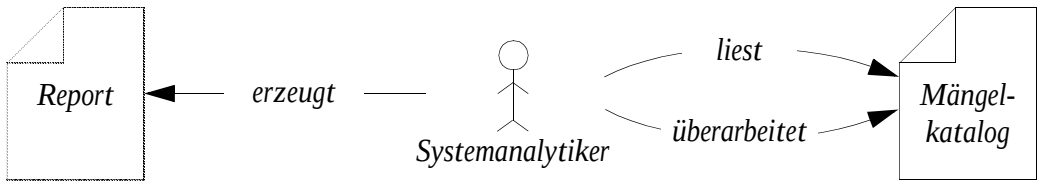


Abbildung 38: Erstelle Report als Entscheidungsgrundlage

- Die Entscheidungen werden dann von den Informanten getroffen. Dabei entsteht ein Protokoll. Es kann auch entschieden werden, daß einzelne Mängel nicht sofort behoben werden und vorerst als offene Mängel weiterbestehen sollen (Abbildung 39).

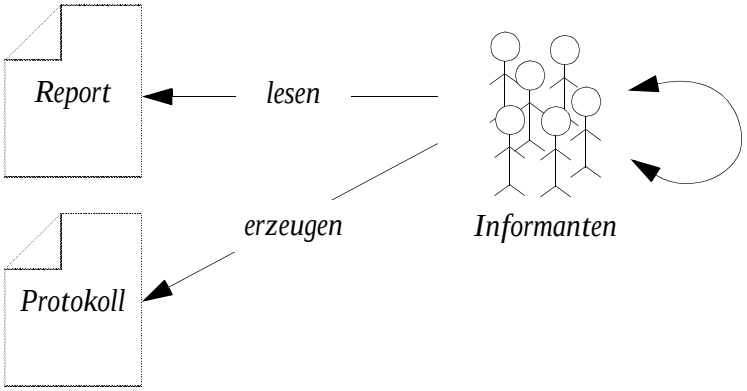


Abbildung 39: Entscheide über Änderung

- Die Systemanalytiker dokumentieren die Entscheidungen im Mängelkatalog (Abbildung 40).

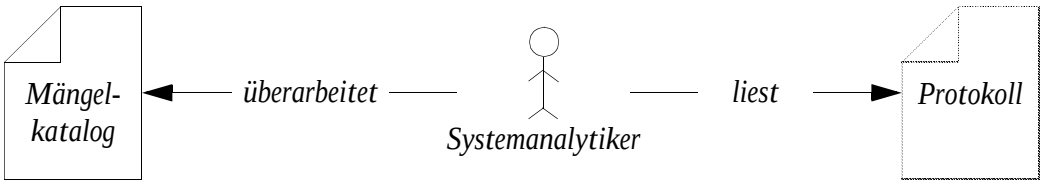


Abbildung 40: Überarbeite Protokoll

7.2.8 Überarbeite Anforderungsdokument

Entsprechend der im Mängelkatalog dokumentierten Entscheidungen bzgl. der Behebung von Mängeln werden das Anforderungsdokument, das Glossar und der Quellenkatalog überarbeitet (Abbildung 41).

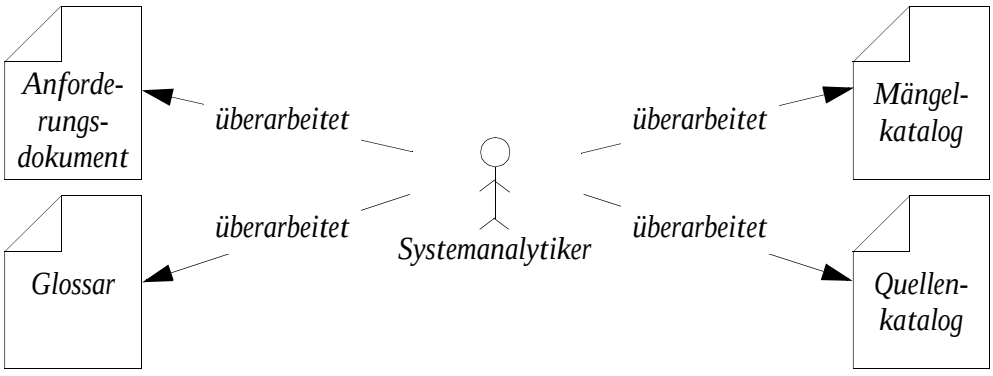


Abbildung 41: Überarbeite Anforderungsdokument

7.2.9 Erstelle Pflichtenheft

Bevor mit der Erstellung des Pflichtenhefts begonnen werden kann, sollte das Lastenheft schon relativ weit fortgeschritten sein. Konflikte sollten (zumindest größ-

tenteils) erkannt und gelöst worden sein. Durch das Lastenheft wird ein Produktraum beschrieben, also eine Menge von möglichen Zielsystemen. Aus dieser Menge wird jetzt ein Zielsystem ausgewählt und spezifiziert.

Einträge werden überarbeitet und Systemen oder Rollen des Systemmodells zugeordnet. Unvollständigkeiten, Widersprüche und Vagheiten, die im Lastenheft noch erlaubt und sinnvoll waren, werden jetzt beseitigt. Wahrscheinlich muß das Glossar um *projektspezifische Terme* erweitert werden. Resultat dieser Tätigkeit ist ein Pflichtenheft und ein überarbeitetes Glossar (Abbildung 42).

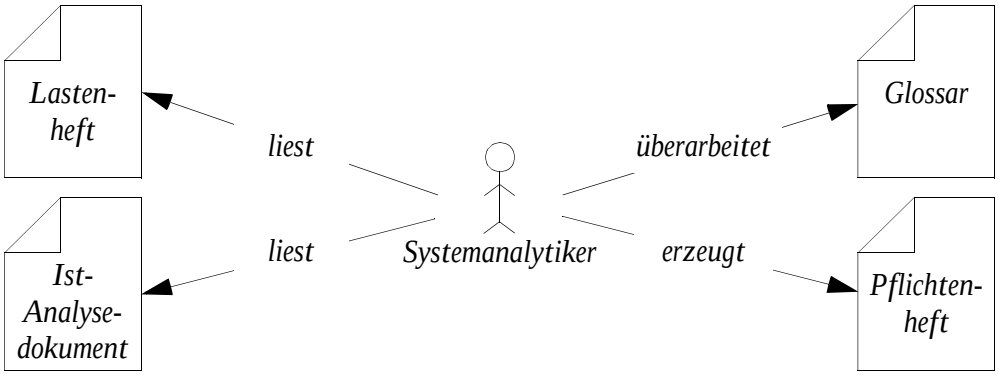


Abbildung 42: Erstelle Pflichtenheft

7.2.10 Bearbeite Änderungsmeldung

Auch nach Ende der Requirements-Engineering-Phase können Pflichtenheft, Glossar und Quellenkatalog geändert werden. Da die Änderungen aber kontrolliert erfolgen sollen, muß für jede Änderung eine *Änderungsmeldung* geschrieben werden. Die Systemanalytiker nehmen sie entgegen und erzeugen einen entsprechenden Mangeleintrag.

Die Systemanalytiker müssen analysieren, ob die Änderung weitere Probleme nach sich ziehen würde, d.h. sie müssen Pflichtenheft und Glossar mit durchgeführter Änderung verifizieren. Falls es sich um geringfügige (und sinnvolle) Änderungen handelt, können sie sofort durchgeführt werden. Die Entscheidung, ob die Änderung geringfügig ist, muß von den Systemanalytikern getroffen werden.

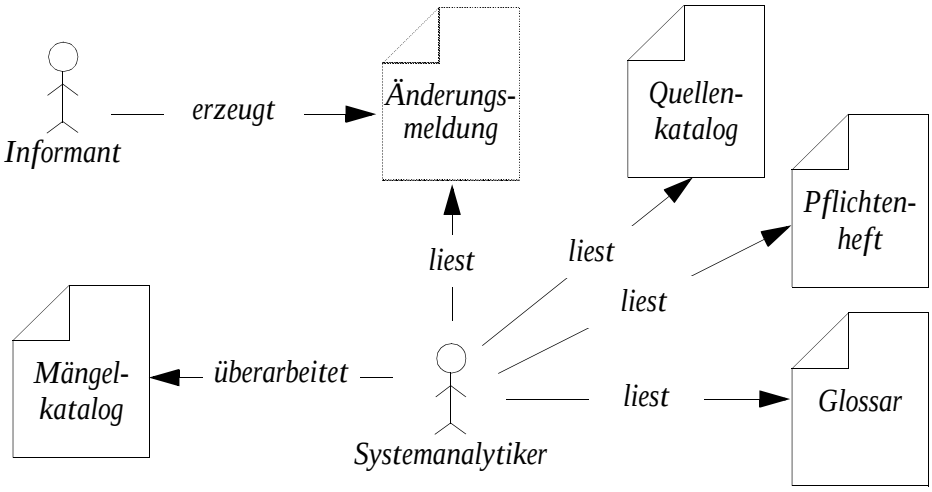


Abbildung 43: Bearbeite Änderungsmeldung

Der Mangeleintrag muß, zusammen mit zugeordneten Annotationen, Informationen enthalten über:

- die Änderungsmeldung,
- aus der Änderung resultierende weitere Mängel und
- Lösungsvorschläge für diese Mängel.

7.3 Validierungs- und Verifikationstabellen

In diesem Unterkapitel wird beschrieben, welche *Qualitätskriterien* ein Quellenkatalog, ein Glossar, eine Anforderungssammlung, ein Ist-Analysedokument, ein Lastenheft oder ein Pflichtenheft in einem dem ADMIRE-Prozeßmodell folgenden Requirements-Engineering-Prozeß erfüllen soll.

Zu jedem Qualitätskriterium wird angegeben, durch welche Prüfungsart es geprüft werden kann:

- Verifikation: Diese Qualitätskriterien können von den Systemanalytikern angewandt werden. Mängel können ohne Rücksprache mit sonstigen Informanten gefunden, nicht aber unbedingt behoben werden. Natürlich können Mängel dieser Art auch in einer Validierungstätigkeit gefunden werden. Solche Kriterien sind z.B. Konsistenz und Redundanzfreiheit.
- Validierung: Diese Qualitätskriterien können nur unter Beteiligung von Informanten angewandt werden. Solche Kriterien sind z.B. Vollständigkeit und Korrektheit.

7.3.1 Anforderungsdokument und Glossar

Gegeben sei ein *Anforderungsdokument doc*, das in einer Dokumentation *I* enthalten ist. Das Anforderungsdokument soll zusammen mit dem *Glossar* oder Teilen des Glossars die in Tabelle 23 geforderten Kriterien erfüllen.

Abhängig von der Art des Anforderungsdokuments und dem Zeitpunkt der Prüfung müssen unterschiedliche Kriterien erfüllt werden.

In den letzten fünf Spalten der Tabelle werden die geforderten Kriterien durch ein Plus (+) markiert. Dabei steht *A* für eine Anforderungssammlung, *I* für ein Ist-Analysedokument, L_1 für ein Lastenheft in der Problemanalysephase, L_2 für ein Lastenheft in der Spezifikationsphase und *P* für ein Pflichtenheft.

Nr.	Qualitätskriterium	zu erfüllende Qualitätsprädikate	A	I	L_1	L_2	P
1.	korrekt (Validierung)	Jeder aktuelle Eintrag des Anforderungsdokuments soll korrekt sein. $\forall m \in DocEntry(doc) \bullet correct(m)$	+	+	+	+	+
2.		Jeder in dem Glossar enthaltene aktuelle Eintrag soll korrekt sein. $\forall e \in GlossaryEntry(I) \bullet correct(e)$			+	+	+
3.		Jeder in dem Anforderungsdokument benutzte aktuelle Glossareintrag soll korrekt sein. $\forall g \in UsedGlossaryEntry(doc) \bullet correct(g)$	+	+			
4.	vollständig (Validierung)	Das Anforderungsdokument soll vollständig sein. $complete(doc)$	+	+	+	+	+
5.		Jeder aktuelle Eintrag des Anforderungsdokuments soll vollständig sein. $\forall e \in DocEntry(doc) \bullet complete(e)$		+		+	+
6.		Jeder aktuelle Eintrag des Glossars soll vollständig sein. $\forall e \in GlossaryEntry(I) \bullet complete(e)$				+	+
7.		Das Glossar soll vollständig sein bzgl. aller aktuellen Einträge des Anforderungsdokuments. $\forall G \in Glossary\|_I \bullet complete(G, DocEntry(doc))$	+	+	+	+	+
8.		Das Glossar soll vollständig sein bzgl. aller aktuellen Glossareinträge und zugeordneten Annotationen. $\forall G \in Glossary\|_I \bullet complete(G, GlossaryEntry(I))$			+	+	+
9.	adäquat (Validierung)	Jedes Inhaltselement des Anforderungsdokuments soll adäquat sein. $\forall c \in CurrentContElement(doc) \bullet adequate(c)$		+		+	+
10.	konsistent (Verifikation)	Die Menge aller aktuellen Einträge des Anforderungsdokuments soll konsistent sein. $consistent(DocEntry(doc))$		+		+	+

Tabelle 23: Qualitätskriterien für Anforderungsdokumente

Nr.	Qualitätskriterium	zu erfüllende Qualitätsprädikate	A	I	L_1	L_2	P
11.	konsistent (Verifikation)	Jeder aktuelle Eintrag soll konsistent sein. $\forall e \in GlossaryEntry(I) \cup DocEntry(doc) \bullet consistent(e)$	+	+	+	+	+
12.		Jeder aktuelle Eintrag soll konsistent bzgl. des aktuellen Glossars sein. $\forall e \in GlossaryEntry(I) \cup DocEntry(doc) \bullet consistent(MainTerm(I) \cup \{e\})$	+		+	+	+
13.		Jeder aktuelle Eintrag soll konsistent bzgl. des Glossars des Anwendungsbereichs sein. $\forall e \in ApplGlossaryEntry(I) \cup DocEntry(doc) \bullet consistent(ApplMainTerm(I) \cup \{e\})$		+			
14.		Jeder aktuelle Eintrag soll konsistent bzgl. der an ihn gebundenen Annotationen sein. $\forall e \in GlossaryEntry(I) \cup DocEntry(doc) \bullet consistent(\{e\} \cup EntryAnnotation(e))$	+	+	+	+	+
15.		Die Menge aller aktuellen Glossareinträge soll konsistent sein. $consistent(MainTerm(I))$				+	+
16.		Die Menge aller aktuellen Glossareinträge des Anwendungsbereichs soll konsistent sein. $consistent(ApplMainTerm(I))$		+			
17.	extern konsistent (Verifikation)	Die Menge aller aktuellen Einträge des Anforderungsdokuments soll extern konsistent sein. $externalconsistent(DocEntry(doc), CurrentDocSourceEntry(I))$		+		+	+
18.		Jeder aktuelle Glossareintrag soll extern konsistent sein. $\forall g \in MainTerm(I) \bullet externalconsistent(\{g\}, CurrentDocSourceEntry(I))$				+	+
19.		Jeder aktuelle Glossareintrag des Anwendungsbereichs soll extern konsistent sein. $\forall g \in ApplMainTerm(I) \bullet externalconsistent(\{g\}, CurrentDocSourceEntry(I))$		+			

Tabelle 23: Qualitätskriterien für Anforderungsdokumente

Nr.	Qualitätskriterium	zu erfüllende Qualitätsprädikate	A	I	L_1	L_2	P
20.	nicht redundant (Verifikation)	Die Menge der Inhaltselemente soll redundanzfrei sein. $notredundant(CurrentContElement(doc))$		+	+	+	+
21.	nicht überspezifiziert (Verifikation)	Jede aktuelle Anforderung soll nicht überspezifiziert sein. $\forall r \in CurrentRequirement(doc) \bullet notoverspecified(r)$					+
22.	realisierbar (Verifikation)	Die Menge aller aktuellen Anforderungen soll realisierbar sein. $achievable(CurrrentRequirement(doc))$					+
23.	überprüfbar (Verifikation)	Jede aktuelle Anforderung, Zusicherung und jeder aktuelle Fakt soll überprüfbar sein. $\forall c \in \{e \in CurrentContElement(doc) \mid e \notin Problem\} \bullet verifiable(c)$		+			+
24.	atomar (Verifikation)	Jedes Inhaltselement soll atomar sein. $\forall c \in CurrentContElement(doc) \bullet atomic(c)$		+	+	+	+
25.		Jeder Glossareintrag soll atomar sein. $\forall g \in MainTerm(I) \bullet atomic(g)$	+	+	+	+	+
26.	verständlich (Validierung)	Jeder aktuelle NL-Eintrag des Anforderungsdokuments soll verständlich sein. $\forall e \in NLDocEntry(doc) \bullet understandable(e)$	+	+	+	+	+
27.		Jeder aktuelle in dem Anforderungsdokument benutzte Glossareintrag und jede zugeordnete Annotation soll verständlich sein. $\forall g \in UsedGlossaryEntry(doc) \bullet understandable(g)$	+	+			
28.		Jeder aktuelle Glossareintrag und jede zugeordnete Annotation soll verständlich sein. $\forall g \in GlossaryEntry(I) \bullet understandable(g)$			+	+	+
29.	eindeutig (Validierung)	Jeder aktuelle NL-Eintrag des Anforderungsdokuments soll eindeutig sein. $\forall e \in NLDocEntry(doc) \bullet unambiguous(e)$	+	+	+	+	+

Tabelle 23: Qualitätskriterien für Anforderungsdokumente

Nr.	Qualitätskriterium	zu erfüllende Qualitätsprädikate	A	I	L_1	L_2	P
30.	eindeutig (Validierung)	Jeder aktuelle in dem Anforderungsdokument benutzte Glossareintrag und jede zugeordnete Annotation soll eindeutig sein. $\forall g \in UsedGlossaryEntry(doc) \bullet unambiguous(g)$	+	+			
31.		Jeder aktuelle Glossareintrag und jede zugeordnete Annotation soll eindeutig sein. $\forall g \in GlossaryEntry(I) \bullet unambiguous(g)$			+	+	+
32.	präzise (Verifikation)	Jedes Inhaltselement soll präzise sein. $\forall c \in CurrentContElement(doc) \bullet precise(c)$		+			+
33.		Jeder aktuelle in dem Anforderungsdokument benutzte Glossareintrag soll präzise sein. $\forall g \in UsedMainTerm(doc) \bullet precise(g)$	+	+			
34.		Jeder aktuelle Glossareintrag soll präzise sein. $\forall g \in MainTerm(I) \bullet precise(g)$			+	+	+
35.	minimal (Verifikation)	Das Anforderungsdokument soll minimal sein. $minimal(doc)$	+	+	+	+	+
36.		Jeder aktuelle NL-Eintrag des Anforderungsdokuments soll minimal sein. $\forall e \in NLDocEntry(doc) \bullet minimal(e)$	+	+	+	+	+
37.		Jeder aktuelle in dem Anforderungsdokument benutzte Glossareintrag und jede zugeordnete Annotation soll minimal sein. $\forall g \in UsedGlossaryEntry(doc) \bullet minimal(g)$	+	+			
38.		Jeder aktuelle Glossareintrag und jede zugeordnete Annotation soll minimal sein. $\forall g \in GlossaryEntry(I) \bullet minimal(g)$			+	+	+
39.	normkonform (Verifikation)	Das Anforderungsdokument soll normkonform sein. $conform(doc)$	+	+	+	+	+

Tabelle 23: Qualitätskriterien für Anforderungsdokumente

7.3.2 Quellenkatalog

Gegeben sei eine Dokumentation I. Der darin enthaltene *Quellenkatalog* soll die in Tabelle 24 beschriebenen Qualitätskriterien erfüllen.

Nr.	Qualitäts-kriterium	zu erfüllende Qualitätsprädikate
1.	korrekt (Validierung)	Jeder aktuelle Quellenkatalogeintrag soll korrekt sein. $\forall s \in CurrentSourceEntry(I) \bullet correct(s)$
2.	vollständig (Validierung)	Der Quellenkatalog soll vollständig sein. $\forall S \in SourceCatalogue\|_I \bullet complete(S)$
3.		Jeder aktuelle Quellenkatalogeintrag soll vollständig sein. $\forall s \in CurrentSourceEntry(I) \bullet complete(s)$
4.	konsistent (Verifikation)	Die Menge der aktuellen Quellenkatalogeinträge soll konsistent sein. $consistent(CurrentSourceEntry(I))$
5.		Jeder aktuelle Quellenkatalogeintrag soll konsistent sein mit den an ihn gebundenen Annotationen. $\forall s \in CurrentSourceEntry(I) \bullet consistent(\{s\} \cup EntryAnnotation(s))$
6.		Jeder aktuelle Quellenkatalogeintrag soll konsistent sein. $\forall s \in CurrentSourceEntry(I) \bullet consistent(s)$
7.	extern konsistent (Verifikation)	Die Menge der aktuellen Quellenkatalogeinträge soll extern konsistent sein. $externalconsistent(CurrentSourceEntry(I), CurrentDocSourceEntry(I))$
8.	atomar (Verifikation)	Jeder aktuelle Quellenkatalogeintrag soll atomar sein. $\forall s \in CurrentSourceEntry(I) \bullet atomic(s)$
9.	verständlich (Validierung)	Jeder aktuelle Quellenkatalogeintrag soll verständlich sein. $\forall s \in CurrentSourceEntry(I) \bullet understandable(s)$
10.	eindeutig (Validierung)	Jeder aktuelle Quellenkatalogeintrag soll eindeutig sein. $\forall s \in CurrentSourceEntry(I) \bullet unambiguous(s)$
11.	präzise (Verifikation)	Jeder aktuelle Quellenkatalogeintrag soll präzise sein. $\forall s \in CurrentSourceEntry(I) \bullet precise(s)$
12.	minimal (Verifikation)	Jeder aktuelle Quellenkatalogeintrag soll minimal sein. $\forall s \in CurrentSourceEntry(I) \bullet minimal(s)$

Tabelle 24: Qualitätskriterien für den Quellenkatalog

Kapitel 8

Validierung und Verifikation

Im Informationsmodell werden folgende Qualitätskriterien auf den Eintragstypen einer Dokumentation definiert (siehe Kapitel 6.13): korrekt, vollständig, adäquat, konsistent, extern konsistent, redundanzfrei, nicht überspezifiziert, realisierbar, überprüfbar, atomar, verständlich, eindeutig, präzise, minimal und normkonform. In diesem Kapitel werden Prüfverfahren vorgestellt, durch die Mängel, d.h. Verletzungen der Qualitätskriterien, aufgespürt werden können.

Ziel einer Prüfung ist es, Mängel zu finden, nicht aber, die Mängel zu beheben. Wenn ein Mangel entdeckt wird, sollte ein entsprechender Mangeleintrag in die Dokumentation eingefügt werden. Wenn ein Mangel ohne Beteiligung der Informanten gefunden werden kann, heißt das nicht, daß er auch ohne deren Beteiligung behoben werden kann. Liegt z.B. ein Widerspruch zwischen zwei Anforderungen vor, müssen die betroffenen Informanten an der Lösung beteiligt werden. Das kann einige Zeit in Anspruch nehmen. Es ist also sehr wahrscheinlich, daß eine Dokumentation bekannte, noch nicht behobene Mängel enthält.

In Kapitel 8.1 werden einige allgemeine *Prüfverfahren* vorgestellt, die die automatische und visuelle Erkennung von Mängeln unterstützen. Anschließend werden in Kapitel 8.2 diese allgemeinen Prüfverfahren konkretisiert. Für jedes Qualitätskriterium werden konkrete Prüfverfahren angegeben.

8.1 Prüfverfahren

Im folgenden werden die Prüfverfahren Inhaltsprüfung, fokussierte Inspektion, Statusprüfung, Inspektion und Ähnlichkeitssuche vorgestellt.

8.1.1 Inhaltsprüfung

Ziel der *Inhaltsprüfung* ist es, automatisch Indizien für Mängel in einzelnen Einträgen oder Teilen einer Dokumentation zu finden. Wenn z.B. in dem Freitext einer Anforderung Wörter wie »manchmal«, »einige« oder »TBD« benutzt werden, deutet das auf eine unpräzise Aussage bzw. eine Unvollständigkeit hin. Die Prüfung erfolgt anhand fest vorgegebener *Prüfkriterien*. In diesem Kapitel wird nur das allgemeine Prüfverfahren beschrieben. Die Prüfkriterien werden in Kapitel 8.2 in Form von prädikatenlogischen Formeln angegeben.

Die Prüfergebnisse einer Inhaltsprüfung sind von unterschiedlicher *Zuverlässigkeit*. Folgende zwei Kategorien werden dabei unterschieden:

- hohe Zuverlässigkeit: Ist das gegebene Prüfkriterium verletzt, dann liegt mit ziemlicher Sicherheit ein Mangel vor.
- niedrige Zuverlässigkeit: Ist das gegebene Prüfkriterium verletzt, dann liegt »nur« ein Indiz für einen Mangel vor.

Die Inhaltsprüfung läuft folgendermaßen ab:

1. Der Systemanalytiker wählt die Prüfkriterien und die zu prüfenden Einträge.
2. Das Werkzeug startet die Inhaltsprüfung. Sie läuft in zwei Schritten ab:
   2.1. *Recheck*: Alle vorhandenen automatisch erzeugten Mangeleinträge, die im Zustand »aktuell« oder »potentiell« sind, werden daraufhin überprüft, ob der beschriebene Mangel immer noch vorliegt. Wenn ja, ändert sich der Zustand nicht, ansonsten wird er in »erledigt« geändert. Auf verworfenen und erledigten Mangeleinträgen wird kein Recheck durchgeführt.
   2.2. *Check*: Auf alle ausgewählten Einträge werden die ausgewählten Prüfkriterien angewandt. Wird dabei ein Mangel oder ein Indiz für einen Mangel gefunden und gibt es für diesen Mangel noch keinen »passenden« (siehe unten) Mangeleintrag, wird automatisch ein neuer Mangeleintrag erzeugt. Je nach Zuverlässigkeit des Prüfkriteriums werden »aktuelle« oder »potentielle« Mangeleinträge erzeugt.
3. Anschließend sollte der Systemanalytiker die neuen Mangeleinträge überprüfen. Liegt wirklich ein Mangel vor, so setzt er den Zustand des Mangeleintrags auf »aktuell« (soweit er nicht schon in diesem Zustand ist). Liegt kein Mangel vor, so ändert er den Zustand in »verworfen«.

Der Zustand eines Mangeleintrags wird durch das Attribut *state* (Kapitel 6.9) beschrieben. Es kann die Werte »potentiell« (*potential*), »aktuell« (*open*), »erledigt« (*done*) oder »verworfen« (*rejected*) annehmen. In den Abbildungen 44 und 45 werden die möglichen Zustandsübergänge eines automatisch erzeugten Mangeleintrags dargestellt.

Wenn die Zuverlässigkeit der Prüfergebnisse der Inhaltsprüfung, durch die der Mangel entdeckt wurde, hoch ist, ist der initiale Zustand des Mangeleintrags »aktuell«. Der Systemanalytiker kann das Ergebnis überschreiben, indem er den aktuellen Mangeleintrag verwirft (*reject*). Ebenso kann er einen verworfenen Mangeleintrag wieder auf »aktuell« setzen (*reopen*). Das Ereignis *not-reconfirm* wird ausgelöst, wenn ein Recheck durchgeführt wird und der durch den Mangeleintrag

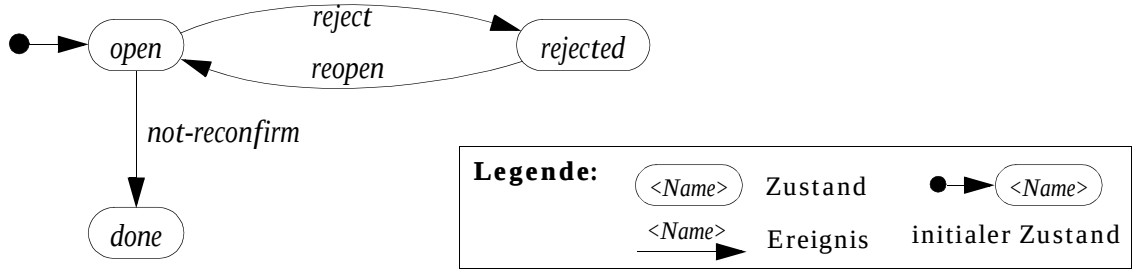


Abbildung 44: Lebenszyklus eines automatisch erzeugten Mangeleintrags bei hoher Zuverlässigkeit des Prüfergebnisses

beschriebene Mangel nicht mehr vorhanden ist. Der Zustand des Mangeleintrags wird dann auf »erledigt« (*done*) gesetzt.

Wenn die Zuverlässigkeit der Prüfergebnisse der Inhaltsprüfung, durch die der Mangel entdeckt wurde, niedrig ist, ist der initiale Zustand des Mangeleintrags »potentiell«. Die Entscheidung, ob wirklich ein Mangel vorliegt, muß ein Systemanalytiker treffen (*reject* oder *confirm*). Erst wenn der Mangeleintrag bestätigt wird (*confirm*), ändert sich der Zustand in »aktuell«. Von da an verhält sich der Mangeleintrag genauso wie ein mit hoher Zuverlässigkeit gefundener Mangel.

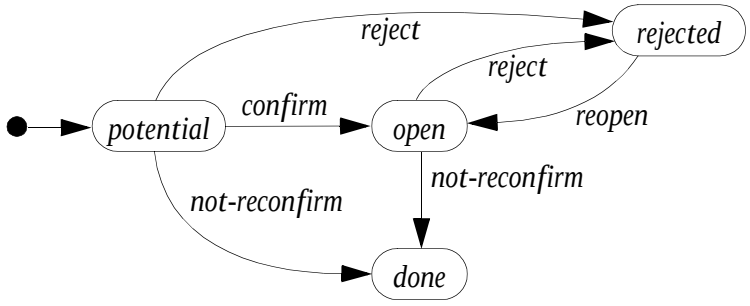


Abbildung 45: Lebenszyklus eines automatisch erzeugten Mangeleintrags bei niedriger Zuverlässigkeit des Prüfergebnisses

Da es für jeden Mangel maximal einen Mangeleintrag geben soll, muß sichergestellt sein, daß noch kein »passender« (äquivalenter) Mangeleintrag existiert, wenn ein neuer Mangeleintrag automatisch erzeugt werden soll. Dafür muß ein Äquivalenz-Prädikat definiert werden.

Definition (*equivalent: Defect* $\times$ *Defect* $\rightarrow$ *Boolean*): Zwei Mangeleinträge d_1 und d_2 sind äquivalent, wenn folgende Bedingungen erfüllt sind:

1. Sie verweisen auf die gleichen Einträge: $entries(d_1) = entries(d_2)$
2. Sie sind nicht im Zustand »erledigt«: $(state(d_1) \neq done) \wedge (state(d_2) \neq done)$
3. Sie gehören zur selben Mangelkategorie: $class(d_1) = class(d_2)$
4. Sie enthalten die gleiche Beschreibung: $description(d_1) = description(d_2)$
5. Sie wurden automatisch erzeugt: $(kind(d_1) = automatic) \wedge (kind(d_2) = automatic)$

Wenn ein Mangeleintrag im Zustand »verworfen« ist und bei der nächsten Prüfung ein äquivalenter Mangel vorhanden ist, wird kein neuer Mangeleintrag erzeugt. Der Systemanalytiker hat dadurch, daß er den Mangeleintrag verworfen hat, entschieden, daß doch kein Mangel vorliegt. Eingriffe eines Systemanalytikers haben »höhere Priorität« als Prüfergebnisse.

Wenn ein Mangeleintrag im Zustand »erledigt« ist und bei der nächsten Prüfung ein entsprechender Mangel vorhanden ist, wird ein neuer Mangeleintrag erzeugt. Die Situation kann folgendermaßen interpretiert werden: Ein vorhandener Mangel wurde behoben. Der zugehörige Mangeleintrag ist deswegen im Zustand »erledigt«. Anschließend wurde die Dokumentation so geändert, daß die gleichen Einträge wieder das gleiche Indiz für einen Mangel aufweisen. Deswegen muß dann auch ein neuer Mangeleintrag erzeugt werden.

Die in den folgenden Abschnitten (8.1.2 ... 8.1.5) vorgestellten Prüfverfahren erzeugen keine Mangeleinträge, sondern unterstützen den Prüfer dabei, Mängel aufzuspüren oder zu vermeiden. Die Mangeleinträge müssen manuell erzeugt werden.

Ein manuell erzeugter Mangeleintrag ist initial im Zustand »aktuell«. Dieser Zustand kann nur manuell geändert werden. Wie jeden anderen Mangeleintrag auch kann der Systemanalytiker den Mangeleintrag verwerfen (*reject*) und wieder auf »aktuell« setzen (*reopen*). Um anzuzeigen, daß der Mangel behoben wurde, kann er den Zustand auf »erledigt« setzen (*solve*) (Abbildung 46).

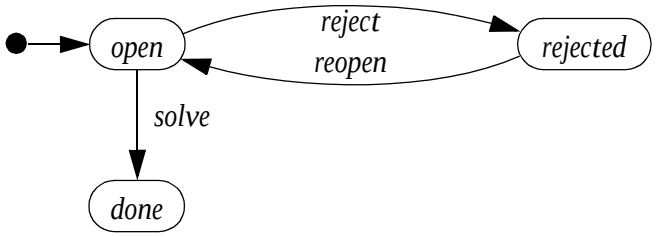


Abbildung 46: Lebenszyklus eines manuell erzeugten Mangeleintrags

Das ADMIRE-Werkzeug berücksichtigt manuell erzeugte Mangeleinträge bei der Inhaltsprüfung nicht, d.h. Check und Recheck werden nicht durchgeführt. Es kann nicht feststellen, ob ein manuell erzeugter und ein automatisch erzeugter Mangeleintrag beide auf den gleichen Mangel verweisen.

Das ADMIRE-Werkzeug stellt sicher, daß die Zustände eines Mangeleintrags nur gemäß der in den Abbildungen 44 bis 46 beschriebenen Zustandsübergänge geändert werden. Hierdurch werden also die Bedingungen des Informationsmodells verschärft.

8.1.2 Fokussierte Inspektion

Bestimmte Wörter oder Phrasen der Freitexte haben eine besondere Bedeutung, z.B. die Flexionen eines Terms aus einem Glossareintrag oder die Wörter der Wortliste. Bei einigen dieser Wörter oder Phrasen ist es ein gutes Zeichen, wenn sie benutzt werden, bei anderen ein schlechtes. So sollten die aktuellen Glossareinträge benutzt werden, im Gegensatz zu den sog. WeakWords (Kapitel 8.2.13).

Die Idee der *fokussierten Inspektion* ist, daß der Systemanalytiker eine visuelle Rückkopplung erhält, wenn eine dieser besonderen Phrasen erkannt wird. Die erkannte Phrase wird markiert. Der Systemanalytiker kann aus einer Markierung oder dem Ausbleiben einer Markierung Rückschlüsse ziehen, um evtl. schon während der Formulierung des Freitextes Korrekturen vorzunehmen. Die fokussierte Inspektion findet (automatisch) keine Mängel. Sie hilft dem Systemanalytiker, Mängel zu vermeiden.

In folgender Beispiel-Anforderung aus den Anforderungsdokumenten des SESAM-2-Projekts werden alle erkannten Terme des Glossars fett und alle erkannten »verbotenen« Wörter durchgestrichen dargestellt:

»Immer, wenn der ~~angehende~~ ~~Projektleiter~~ ein gültiges Kommando eingibt und mit RETURN bestätigt, wird es ausgeführt. Dabei wird die **Simulationszeit** entsprechend des durch das **SESAM-Modell** vorgegebenen **Zeitverbrauchs** weitergeschaltet. Anschließend wird ~~evtl.~~ ein **Regelauswertungszyklus** gestartet.«

Die Phrase »angehender Projektleiter« ist durchgestrichen, weil sie im Glossar als nicht aktuell markiert ist, also nicht benutzt werden soll. Das richtige Wort wäre »Spieler«. Das Wort »evtl.« ist ein unpräzises Wort (»WeakWord«) und sollte in Anforderungen nicht benutzt werden. Die Phrasen »Simulationszeit«, »SESAM-Modell«, »Zeitverbrauchs« und »Regelauswertungszyklus« sind fett, weil es entsprechende aktuelle Glossareinträge gibt. Das Wort »Kommando« wirkt auf den ersten Blick ebenfalls wie ein projektspezifischer Term. Es wird aber nicht markiert, weil es keinen Glossareintrag mit entsprechendem Term gibt. Hieran kann man deutlich sehen, daß auch aus dem Ausbleiben einer Markierung sinnvolle Schlüsse gezogen werden können. Der Systemanalytiker sollte jetzt entweder den richtigen Term benutzen oder das Glossar erweitern.

Es werden zwei Arten von fokussierten Inspektionen unterschieden:

- *positive fokussierte Inspektion*: Hierbei werden Phrasen markiert, deren Benutzung explizit erwünscht ist. Zur Markierung werden Schriftformate verwendet, die die Phrasen betonen sollen, z.B. Fettschrift oder Unterstreichung.
- *negative fokussierte Inspektion*: Hierbei werden Phrasen markiert, die auf ein eventuelles Problem hindeuten. Zur Markierung werden Schriftformate verwendet, die warnen oder auf einen möglichen Fehler aufmerksam machen sollen, z.B. Durchstreichung oder eine andere Farbe, insbesondere rot.

Zu jeder konkreten fokussierten Inspektion (Kapitel 8.2) wird eine Markierungsfunktion angegeben, die einem Eintrag alle zu markierenden Wörter zuordnet.

In Kapitel 6.1.4 werden die zur Erkennung der benutzten Terme des Glossars und der Wörter der Wortliste notwendigen Prädikate definiert. Von diesen Prädikaten sind folgende nicht berechenbar: *inflection*, *directuse* und *indirectuse*. Hierfür würde ein linguistisches Lexikon benötigt, in dem alle relevanten Wörter und Phrasen der deutschen Sprache verzeichnet sind, auch die projektspezifischen, inklusive deren Bedeutungen. Dieses Lexikon steht im ADMIRE-Ansatz nicht zur Verfügung. Es steht aber ein Glossar zur Verfügung, in dem die für das Projekt relevante Terminologie definiert ist. Daraus können die Grundformen der Terme, die Flexionsangaben für die Terme, die Synonym-Verweise und die Hyperonym-Verweise entnommen werden.

Für die o.g. nicht berechenbaren Prädikate werden jetzt die Ersatzprädikate $inflection_G$, $directuse_G$, $directuse_W$ und $indirectuse_G$ eingeführt, die mit den vorhandenen Mitteln des ADMIRE-Ansatzes berechnet werden können.

Das Prädikat $inflection_G$ kann direkt auf die Flexionsverweise des Glossars zurückgeführt werden.

Definition ($inflection_G$: $Word \times Word \times GlEntry \rightarrow Boolean$): Das Prädikat $inflection_G$ ist *true*, wenn es zu dem Glossareintrag g Flexionsverweise mit gleichem Subterm gibt, deren Flexionsangaben mit den Wörtern w_1 und w_2 übereinstimmen:

$$inflection_G(w_1, w_2, g) = (\exists i_1, i_2 \in inflections(g) \bullet ((subterm(i_1) = subterm(i_2)) \wedge (inflectedword(i_1) = w_1) \wedge (inflectedword(i_2) = w_2)))$$

Bei der Definition des Direkt-Benutzt-Prädikats $directuse_G$ wird die Eigenschaft des Glossars ausgenutzt, daß sich Flexionen entweder auf ein maximales Wort des Terms oder auf den gesamten Term beziehen.

Bei der Erkennung von benutzten Termen müssen SimpleWords ignoriert werden (siehe E106, S. 95 und Kapitel 6.10):

Definition (*SimpleWord*): Die Menge *SimpleWord* enthält alle als SimpleWord typisierten Phrasen der Wortliste. Gegeben sei eine Dokumentation I. Dann gilt:

$$SimpleWord = \{p \in Phrase \mid \exists w \in WordEntry|_I \bullet ((term(w) = p) \wedge (simpleword \in types(w)))\}$$

Jetzt kann das Direkt-Benutzt-Prädikat definiert werden:

Definition ($directuse_G$: $Paragraph \times GlEntry \rightarrow Boolean$): Das Prädikat $directuse_G$ ist *true*, wenn der Glossareintrag g in dem Freitext s direkt benutzt wird:

$$directuse_G(s, g) = directuse_1(s, g) \vee directuse_2(s, g)$$

mit:

1. Der Freitext enthält Flexionen aller maximalen Wörter des Terms, wobei SimpleWords ignoriert werden.

$$directuse_1(s, g) = (\forall w \in Word \bullet ((maxword(term(g), w) \wedge w \notin SimpleWord) \Rightarrow (\exists v \in Word \bullet inflection_G(v, w, g) \wedge maxword(s, v))))$$

2. Der Freitext enthält ein Wort, das als Flexion für den gesamten Term eingetragen wurde.

$$directuse_2(s, g) = \exists v \in Word \bullet \exists i \in inflections(g) \bullet (subterm(i) = term(g)) \wedge (inflectedword(i) = v) \wedge maxword(s, v)$$

Dieses Prädikat kann automatisch berechnet werden. An dem Beispiel aus Kapitel 5.6 (S. 93) wird jetzt gezeigt, in welchen Fällen das Prädikat richtige oder falsche Ergebnisse liefert. Das Prädikat wird auf den Glossareintrag mit dem Term »Attribut eines Dokuments« und folgende Freitexte angewandt:

1. »Die Attribute eines SESAM-Modells können frei definiert werden.«
2. »Jedes simulierte Dokument muß mindestens das Attribut *name* enthalten.«
3. »Das Dokumentattribut *name* ist vom Datentyp *string*.«
4. »Alle Attribute eines SESAM-Modells werden in dem Dokument ›Modellübersicht‹ aufgelistet.«

Der Glossareintrag enthält Flexionen für die maximalen Wörter des Terms (»Attribut«, »Dokuments«) und für den gesamten Term (z.B. »Dokumentattribute«). Das Wort »einer« ist ein SimpleWord und wird bei der Berechnung ignoriert.

Für Freitext 1 ergibt das Prädikat *false*, weil er nur eine Flexion von »Attribut« enthält, aber keine Flexion von »Dokument« oder »Dokumentattribut«. Für Freitext 2 ergibt das Prädikat *true*, weil er eine Flexion von »Attribut« und eine Flexion von »Dokument« enthält. Für Freitext 3 ergibt das Prädikat *true*, weil er eine Flexion von »Dokumentattribut« enthält. Für Freitext 4 ergibt das Prädikat ebenfalls *true*, aus den gleichen Gründen wie für Freitext 2. Nur ist hier das Ergebnis falsch, weil der Term »Attribut eines Dokuments« nicht benutzt wird. Flexionen der Wörter »Attribut« und »Dokument« sind in dem Freitext zwar enthalten, sie gehören aber nicht zusammen.

Mit Hilfe des Prädikats $directuse_G$ kann eine allgemeine Markierungsfunktion definiert werden.

Definition (*$mark_G$: Paragraph × P(GlEntry) → P(Word)*): Die Funktion *$mark_G$* ordnet einem Freitext und einer Menge von Glossareinträgen die zu markierenden Wörter zu:

$$mark_G(p, G) = mark_1(p, G) \cup mark_2(p, G)$$

mit:

1. Es werden alle Wörter in dem Freitext markiert, die Flexionen der maximalen Wörter des Terms sind.

$$mark_1(p, G) = \bigcup_{g \in G} \{w \in Word \mid maxword(p, w) \wedge directuse_1(p, g) \wedge (\exists i \in inflections(g) \bullet (subterm(i) \neq term(g)) \wedge (inflectedword(i) = w))\}$$

2. Es werden alle Wörter in dem Freitext markiert, die Flexionen des gesamten Terms sind.

$$mark_2(p, G) = \bigcup_{g \in G} \{w \in Word \mid maxword(p, w) \wedge directuse_2(p, g) \wedge (\exists i \in inflections(g) \bullet (subterm(i) = term(g)) \wedge (inflectedword(i) = w))\}$$

Unter Zuhilfenahme des Direkt-Benutzt-Prädikats (*$directuse_G$*) und der Synonym-Verweise und Hyperonym-Verweise des Glossars kann auch die Indirekt-Benutzt-Beziehung definiert werden:

Definition (*$indirectuse_G$: Paragraph × GlEntry → Boolean*): Das Prädikat *$indirectuse_G$* ist *true,* wenn der Glossareintrag *g* in dem Freitext *s* indirekt benutzt wird:

$$indircectuse(s, g) = (\exists h \in GlEntry \bullet (directuse(s, h) \wedge ((h \in synonymes^*(g)) \vee (h \in hyperonymes^*(g)) \vee (g \in hyperonymes^*(h)))))$$

Die Markierungsfunktion *$mark_G$* funktioniert nur, wenn für die zu markierenden Wörter Glossareinträge mit Flexionsangaben vorhanden sind. Bei einigen Prüfungen sollen aber auch Wörter und Phrasen markiert werden, die in der Wortliste enthalten sind, z.B. WeakWords. Hierfür muß ein eigenes *directuse*-Prädikat und eine eigene Markierungsfunktion definiert werden.

Definition (*$directuse_W$: Paragraph × Phrase → Boolean*): Das Prädikat *$directuse_W$* ist *true,* wenn die Phrase *p* in dem Freitext *s* direkt benutzt wird:

$$directuse_W(s, p) = (\forall w \in Word \bullet ((maxword(p, w) \wedge w \notin SimpleWord) \Rightarrow maxword(s, w)))$$

Definition (*$mark_W$: Paragraph × P(Phrase) → P(Word)*): Die Funktion *$mark_W$* ordnet einem Freitext und einer Menge von Phrasen die zu markierenden Wörter zu:

$$mark_W(p, P) = \{w \in Word \mid \exists q \in P \bullet (directuse_W(p, q) \wedge maxword(p, w) \wedge maxword(q, w))\}$$

8.1.3 Statusprüfung

Eines der am schwersten zu prüfenden Qualitätskriterien ist die Vollständigkeit. Ein Werkzeug kann mit Ausnahme weniger Aspekte nicht automatisch entscheiden, ob alle »notwendigen« Informationen in einem Anforderungsdokument enthalten sind. Es gibt aber einige Anhaltspunkte, die auf Unvollständigkeiten hindeuten. Wird z.B. eine Kategorie der Kategorisierungshierarchie für Inhaltselemente in den Inhaltselementen eines Anforderungsdokuments nicht oder nur sehr selten verwendet, liegt ein Indiz für eine Unvollständigkeit vor.

Die Idee der *Statusprüfung* ist, daß das Werkzeug dem Systemanalytiker auf übersichtliche Weise mitteilt, welche Informationen in einem Anforderungsdokument mit welcher Häufigkeit enthalten sind. In einer Statusprüfung z.B. (siehe Kapitel 8.2.2) werden alle Terme des Glossars aufgelistet. Zu jedem Term wird angegeben, in wievielen Einträgen des Anforderungsdokuments er direkt oder indirekt benutzt wird. In einer anderen Statusprüfung werden die Kategorien der Kategorisierungshierarchie aufgelistet. Zu jeder Kategorie wird angegeben, wieviele Inhaltselemente des Anforderungsdokuments entsprechend kategorisiert sind. Der Systemanalytiker kann daraus Indizien für Unvollständigkeiten ableiten.

In Kapitel 8.2.2 wird für jede konkrete Statusprüfung eine Funktion angegeben, die einem Anforderungsdokument die anzuzeigenden Informationen zuordnet.

Die Prüfung läuft folgendermaßen ab:

1. Der Systemanalytiker wählt die durchzuführenden Statusprüfungen aus.
2. Der Systemanalytiker wählt die Anforderungsdokumente aus, auf denen die Statusprüfungen durchgeführt werden sollen.
3. Das Werkzeug wendet die Funktionen der Statusprüfungen an und präsentiert dem Systemanalytiker die Ergebnisse in aufbereiteter Form.
4. Der Systemanalytiker versucht, aus diesen Informationen Indizien für Mängel herauszulesen und erzeugt ggfs. Mangeleinträge.

Durch eine Statusprüfung können automatisch keine Mängel entdeckt werden. Der Systemanalytiker erhält jedoch einen Überblick über den Entwicklungsstand des Anforderungsdokuments.

8.1.4 Inspektion

Einige Mängel in einer Dokumentation lassen sich nur durch Inspektionen finden. Ein Werkzeug kann z.B. unmöglich feststellen, ob eine Anforderung wirklich von einem Informanten gewünscht wird.

Für eine *Inspektion* gibt es folgende prinzipiellen Möglichkeiten:

- Prüfung am Rechner (nur Verifikation durch Systemanalytiker).
- Prüfung ohne Rechner (Validierung oder Verifikation).

Bei der Prüfung am Rechner setzt sich der Systemanalytiker an das Werkzeug und liest den zu prüfenden Teil der Dokumentation. Immer, wenn ihm dabei ein Mangel auffällt, erzeugt er einen Mangeleintrag.

Bei der Prüfung ohne Rechner muß ein Ausdruck eines Teils der Dokumentation (der sog. Prüfling) vorliegen. Während der Prüfung erstellen die Prüfer ein *Protokoll*, in das sie alle gefundenen Mängel eintragen. Dieses Protokoll wird nach der Prüfung von den Systemanalytikern entgegengenommen. Sie überarbeiten es und erzeugen Mangeleinträge (siehe auch Kapitel 7.2.6).

Ein Werkzeug kann die Inspektion auf folgende Weisen unterstützen:

- durch die Verwaltung der Mangeleinträge und
- durch Filtermechanismen, um nur die für die Prüfung relevanten Teile einer Dokumentation auszuwählen.

Mögliche *Filter* sind z.B.: »alle Anforderungen, die Aussagen über das Zeitverhalten des SESAM-2-Systems enthalten« oder »alle Inhaltselemente, in denen der Term ›Tutor‹ benutzt wird«.

Voraussetzung dafür, daß mit dem Filtermechanismus sinnvolle Ergebnisse erzielt werden können, ist, daß die Systemanalytiker die Dokumentation sorgfältig angelegt und die o.g. Informationen sinnvoll eingetragen haben.

Folgende Filter sind im ADMIRE-Ansatz vordefiniert:

- Für jede Inhaltsprüfung ist ein Filter definiert, der alle Einträge ermittelt, die das Prüfkriterium der Inhaltsprüfung verletzen.
- Für jede fokussierte Inspektion ist ein Filter definiert, der alle Einträge ermittelt, in denen durch die Markierungsfunktion mindestens ein Wort markiert werden würde.
- Die in Kapitel 8.2 definierten Filter.

Ein Indiz für einen Mangel ist z.B. die Benutzung eines WeakWords in dem Freitext eines Inhaltselements. Hierfür ist sowohl eine Inhaltsprüfung als auch eine fokussierte Inspektion definiert (Kapitel 8.2.13) und somit auch ein Filter, der alle Inhaltselemente ermittelt, in denen WeakWords benutzt werden.

Durch solche Filter werden Inspektionen unterstützt, in denen ein Anforderungsdokument auf das Vorhandensein von Mängeln ganz bestimmter Arten hin untersucht werden soll.

Darüber hinaus kann der Systemanalytiker neue Filter definieren. Folgende Eigenschaften des Informationsmodells können hierfür verwendet werden:

- die in den Freitexten direkt und indirekt benutzten Glossareinträge, und
- die Attributwerte der Einträge, insbesondere die Ursprungsverweise der Haupteinträge, die Prioritäts- und Stabilitätsangaben der Inhaltselemente, die den Inhaltselementen und Glossareinträgen zugeordneten Kategorien und die Zustandsangaben der Inhaltselemente, Glossareinträge und Quellenkatalogeinträge.

8.1.5 Ähnlichkeitssuche

Während der Erstellung, Änderung oder Prüfung einer Dokumentation stellt sich immer wieder die Frage, inwieweit ein gegebener Eintrag Ähnlichkeiten zu anderen Einträgen aufweist. Inkonsistenzen treten häufig zwischen Einträgen auf, die Aussagen über ähnliche Aspekte machen. Eine Anforderung, die das Layout der Oberfläche von System X und eine, die die Wartbarkeit von System Y betrifft, sind hingegen wahrscheinlich völlig unabhängig voneinander.

Ein Werkzeug kann hierbei unterstützend eingreifen, indem es einen Mechanismus zur *Ähnlichkeitssuche* anbietet. Die Grundidee ist folgende: Das Werkzeug sucht alle Inhaltselemente und Annotationen heraus, die zu einem gegebenen Inhaltselement oder einer gegebenen Annotation die größte Ähnlichkeit aufweisen. Bei der »Ähnlichkeit« handelt es sich nicht um ein absolutes Prädikat. Es interessiert also nicht, ob zwei Einträge »ähnlich« sind, sondern, ob ein Eintrag zu einem gegebenen Eintrag »ähnlicher« ist als ein anderer.

Für die Ähnlichkeitssuche kann die Semantik der Freitexte nicht ausgewertet werden. Eine Dokumentation enthält aber folgende »semantischen« Informationen:

1. die in den Freitexten benutzten Terme des Glossars, und
2. die den Inhaltselementen zugeordneten Kategorien.

Die Ähnlichkeitssuche wird auf Inhaltselemente und zugeordnete Annotationen eingeschränkt. Hierfür wird die Menge *SimEntry* eingeführt:

$$SimEntry = ContElement \cup Annotation$$

Zuerst werden *Ähnlichkeitsmaße* definiert. Ein Ähnlichkeitsmaß ordnet einem Paar von Einträgen eine reelle Zahl zu. Ausgehend von einem gegebenen *SimEntry* e soll das Ähnlichkeitsmaß von (e, e_1) größer sein als das von (e, e_2), wenn e_1 zu e bzgl. der o.g. »semantischen Informationen« mehr Gemeinsamkeiten aufweist als e_2.

Für das Ähnlichkeitsmaß AK_{Term} werden die in den Freitexten benutzten Terme ausgewertet. Je mehr Terme in beiden Einträgen gemeinsam benutzt werden, desto größer wird das Ergebnis. Bei der Berechnung von AK_{Term} werden nur aktuelle Glossareinträge berücksichtigt. Die Synonymiebeziehung wird nicht zur Berechnung herangezogen, weil in einer konsistenten Dokumentation eine Menge von synonymen Glossareinträgen maximal einen aktuellen Glossareintrag enthält. Die Hyponymiebeziehung muß berücksichtigt werden, weil Aussagen, die einen Glossareintrag betreffen, implizit auch dessen Hyponyme und Hyperonyme betreffen (siehe Kapitel 5.6).

Ein Glossareintrag wird in zwei *SimEntry direkt gemeinsam benutzt*, wenn der Term des Glossareintrags in den Freitexten jedes der *SimEntry* jeweils direkt benutzt wird. Zwei unterschiedliche Glossareinträge werden in zwei *SimEntry indirekt gemeinsam benutzt*, wenn der eine Term in den Freitexten des ersten *SimEntry* direkt benutzt wird, wenn der andere Term in den Freitexten des zweiten *SimEntry* direkt benutzt wird und wenn beide Terme ein gemeinsames Hyperonym haben.

In den folgenden Beispiel-Freitexten wird der Glossareintrag »SESAM-2-System« direkt gemeinsam benutzt. Die beiden Glossareinträge »Spieler« und »Tutor« werden indirekt gemeinsam benutzt. »Spieler« und »Tutor« haben als gemeinsames Hyperonym »Benutzer«.

> »Der Tutor kann den internen Zustand des SESAM-2-Systems inspizieren.«
>
> »Der Spieler kann ein neues SESAM-Modell laden, indem er den Menüpunkt ›Lade-Modell‹ im Hauptmenü des SESAM-2-Systems auswählt.«

Jetzt kann das Ähnlichkeitsmaß AK_{Term} definiert werden:

Definition (AK_{Term}: $SimEntry \times SimEntry \times [0..1] \rightarrow R$): Das Ähnlichkeitsmaß AK_{Term} ergibt sich aus der Summe der Anzahl der direkt gemeinsam benutzten Glossareinträge und der Anzahl der indirekt gemeinsam benutzten Glossareinträge. Durch den Parameter t kann eingestellt werden, inwieweit die indirekt gemeinsam benutzten Glossareinträge berücksichtigt werden.

$$AK_{TDirect}(e, e_1) = \left|\{g \in MainTerm(model(e)) \mid (directuse_G(contents(e), g) \wedge directuse_G(contents(e_1), g))\}\right|$$

$$AK_{TIndirect}(e, e_1) = \left|\{(g_1, g_2) \mid g_1, g_2 \in MainTerm(model(e)) \wedge g_1 \neq g_2 \wedge directuse_G(contents(e), g_1) \wedge directuse_G(contents(e_1), g_2) \wedge (\exists g \in MainTerm(model(e)) \bullet (g \in hyperonymes^*(g_1) \wedge g \in hyperonymes^*(g_2)))\}\right|$$

$$AK_{Term}(e, e_1, t) = AK_{TDirect}(e, e_1) + t \cdot AK_{TIndirect}(e, e_1)$$

Für das Ähnlichkeitsmaß AK_{Class} werden die Kategorien ausgewertet, die den Einträgen zugeordnet wurden. Hierfür wird die Hilfsfunktion c eingeführt.

Definition (c: *SimEntry* → *P(Class)*): Die Funktion c ordnet einem gegebenen *SimEntry* seine Kategorien zu. Die Funktion ist abschnittsweise definiert:

$$c(e) = \begin{cases} classes(e), \text{ falls } e \in ContElement \\ classes(\{f \in ContElement \mid f \in entries(e)\}), \text{ falls } e \in Annotation \end{cases}$$

Als Kategorien für eine Annotation werden die Kategorien aller Inhaltselemente verwendet, auf die sich die Annotation bezieht.

Analog zur Definition der direkt und indirekt gemeinsamen Benutzt-Beziehung zwischen Glossareinträgen und *SimEntry* wird eine direkt und eine indirekt gemeinsame Benutzt-Beziehung zwischen Kategorien und *SimEntry* definiert. Eine Kategorie wird in zwei *SimEntry direkt gemeinsam benutzt*, wenn beiden *SimEntry* die Kategorie zugeordnet wurde. Zwei unterschiedliche Kategorien werden in zwei *SimEntry indirekt gemeinsam benutzt*, wenn die eine Kategorie dem ersten und die andere Kategorie dem zweiten *SimEntry* zugeordnet wurde und wenn die Kategorien eine gemeinsame Oberkategorie haben.

Jetzt kann das Ähnlichkeitsmaß AK_{Class} definiert werden:

Definition (AK_{Class}: *SimEntry* × *SimEntry* × *[0..1]* → *R*): Das Ähnlichkeitsmaß AK_{Class} ergibt sich aus der Summe der Anzahl der direkt gemeinsam benutzten Kategorien und der Anzahl der indirekt gemeinsam benutzten Kategorien. Durch den Parameter k kann eingestellt werden, inwieweit die indirekt gemeinsam benutzten Kategorien berücksichtigt werden.

$$AK_{CDirect}(e, e_1) = \left|\{c \in contclass(Metaspecification|_{model(e)}) \mid (c \in c(e) \wedge c \in c(e_1))\}\right|$$

$$AK_{CIndirect}(e, e_1) = \left|\{(c_1, c_2) \mid c_1, c_2 \in contclass(Metaspecification|_{model(e)}) \wedge c_1 \neq c_2 \wedge c_1 \in c(e) \wedge c_2 \in c(e_1) \wedge (\exists c \in contclass(Metaspecification|_{model(e)}) \bullet (c_1 \in indirectsubclasses(c) \wedge c_2 \in indirectsubclasses(c)))\}\right|$$

$$AK_{Class}(e, e_1, k) = AK_{CDirect}(e, e_1) + k \cdot AK_{CIndirect}(e, e_1)$$

Das Gesamtähnlichkeitsmaß wird aus den Einzelmaßen berechnet, wobei auf die Wichtung Einfluß genommen werden kann:

Definition (AK_G: *SimEntry* × *SimEntry* × *[0..1]*^{4} → *R*). Das Gesamtähnlichkeitsmaß berechnet sich aus der gewichteten Summe der Einzelmaße:

$$AK_G(e, e_1, w_1, w_2, t, k) = w_1 \cdot AK_{Term}(e, e_1, t) + w_2 \cdot AK_{Class}(e, e_1, k)$$

Folgendes Beispiel, bestehend aus zwei Anforderungen, einer Annotation und einer Zusicherung, soll verdeutlichen, wie die Ähnlichkeitssuche eingesetzt werden

kann, um Inkonsistenzen zu finden. Gegeben seien folgende in dem Pflichtenheft weit verstreute Einträge[1]:

Eintrag 1: »Jedesmal, wenn der **Spieler** in das **Kommandoeingabefeld** ein gültiges **Benutzerkommando** eingibt, wird zuerst das **Benutzerkommando** ausgeführt und anschließend ein **Regelauswertungszyklus** gestartet. Nach Beendigung des **Regelauswertungszyklus** wartet das **SESAM-2-System** auf die nächste Eingabe des **Spielers**.« (Anforderung, kategorisiert als »Funktion« und als »Eingabeereignis«)

Eintrag 2: »Der **Regelauswertungszyklus** wird genau dann ausgeführt, wenn die **Simulationszeit** um mindestens eine **Simulationsschrittweite** seit der letzten Ausführung des **Regelauswertungszyklus** fortgeschritten ist.« (Anforderung, kategorisiert als »Funktion« und als »bedingtes internes Ereignis«)

Eintrag 3: »Das **SESAM-2-System** ist so konzipiert, daß der **Spieler**, ähnlich wie in realen Projekten, zeitlich unter Druck gesetzt wird. Das bedeutet u.a., daß die **Simulationszeit** nach einer gewissen Zeitspanne automatisch weiterschaltet und somit einen weiteren **Regelauswertungszyklus** auslösen kann.« (Annotation, gebunden an eine als »Funktion« kategorisierte Anforderung)

Eintrag 4: »Der **Benutzer** hat grundlegende Kenntnisse im Umgang mit fensterbasierten Programmen.« (Zusicherung, kategorisiert als »Personeneigenschaft«).

Im Glossar ist ein Hyperonym-Verweis von »Spieler« auf »Benutzer« eingetragen. Die Kategorien »bedingtes internes Ereignis« und »Eingabeereignis« haben als gemeinsame Oberkategorie »Ereignis«.

Die Einträge 1 bis 3 enthalten zwei Widersprüche. In Eintrag 1 wird gefordert, daß der Regelauswertungszyklus jedesmal gestartet wird, nachdem ein Benutzerkommando ausgeführt wurde. Dagegen wird in Eintrag 2 gefordert, daß der Regelauswertungszyklus nur abhängig von der Simulationszeit gestartet wird. Da die Simulationszeit durch das Ausführen von Benutzerkommandos nicht immer geändert wird, besteht hier ein Widerspruch. Der Widerspruch zwischen den Einträgen 1 und 3 besteht darin, daß zum einen gefordert wird, daß das SESAM-2-System auf die nächste Eingabe des Spielers wartet, zum anderen wird aber festgestellt, daß die Simulationszeit nach einer gewissen Zeitspanne automatisch weiterschaltet.

Für die Ähnlichkeitsmaße ergeben sich folgende Werte, wenn die Gewichte w_1 und w_2 jeweils auf 1 und die Parameter t und k jeweils auf 0,5 gesetzt werden:

zu vergleichende Einträge	AK_{Term}	AK_{Class}	AK_G
Einträge 1 und 2	1	1,5	2,5
Einträge 1 und 3	3	1	4
Einträge 1 und 4	0	0	0

Tabelle 25: Werte der Ähnlichkeitsmaße

1. Benutzte Terme des Glossars werden fett dargestellt.

Tabelle 25 zeigt, daß die Einträge 2 und 3 gewisse Ähnlichkeiten zu Eintrag 1 aufweisen. Eintrag 4 beschreibt ganz andere Aspekte als Eintrag 1. Somit hat das Gesamtähnlichkeitsmaß den Wert 0.

Die Ähnlichkeitssuche kann immer dann gut eingesetzt werden, wenn ein neues Inhaltselement oder eine neue Annotation in ein bestehendes Anforderungsdokument eingefügt werden soll. Der Systemanalytiker gibt den Eintrag und eine Wichtung der Ähnlichkeitsmaße vor. Das Werkzeug ermittelt anhand des oben beschriebenen Ähnlichkeitsmaßes, welche Einträge dem gegebenen Eintrag am ähnlichsten sind und listet sie in absteigender Reihenfolge auf.

Da ein Anforderungsdokument durchaus mehrere hundert Inhaltselemente und Annotationen enthalten kann, ist es sinnvoll, eine *Schwelle* für das Gesamtähnlichkeitsmaß anzugeben, unterhalb derer keine Einträge mehr angezeigt werden.

Bei diesem Verfahren handelt es sich nicht einfach um eine Filtertechnik, durch die Einträge anhand gegebener Kriterien ausgewählt werden. Vielmehr werden die Kriterien durch den gegebenen Eintrag vorgegeben. Das Verfahren findet (automatisch) keine Mängel oder Indizien für Mängel. Für sinnvolle Ergebnisse wird der Systemanalytiker mit den einstellbaren Werten, also den Wichtungen w_1, w_2, den Parametern t und k und der Schwelle, experimentieren müssen.

8.2 Prüfung der Qualitätskriterien

In diesem Unterkapitel wird zu jedem Qualitätsprädikat (siehe Kapitel 6.13) angegeben, durch welche Prüfverfahren dessen Erfüllung oder Nicht-Erfüllung festgestellt werden kann. Die konkreten Prüfkriterien für die Inhaltsprüfungen, die Markierungsfunktionen für die fokussierten Inspektionen, die Anzeigefunktionen für die Statusprüfungen und die Filter für die Inspektionen werden durch Prädikate und Funktionen formal beschrieben.

8.2.1 Korrektheit

Das Prädikat *korrekt* ist erfüllt, wenn die Einzeleinträge oder Attribute der Einzeleinträge »richtig« sind oder für »richtig« gehalten werden. Diese Eigenschaften können nicht automatisch geprüft werden.

In den folgenden Unterkapiteln werden drei Möglichkeiten aufgezeigt, Indizien für inkorrekte Einträge zu finden:

- Prüfung auf Nachvollzogenheit
- Prüfung auf die Benutzung von All-Quantoren
- Prüfung, ob Default-Werte vorhanden sind

Nachvollzogenheit

Ein Haupteintrag kann als korrekt angesehen werden, wenn es einen Informanten gibt, der die Korrektheit bestätigt, oder wenn er aus einem externen Dokument abgeleitet werden kann. Korrektheit kann somit auf *Nachvollzogenheit* zurückgeführt werden.

Jeder Haupteintrag enthält Verweise auf seine Ursprünge. Wenn eine externe Quelle (*extsources*) angegeben ist, dann ist der Haupteintrag nachvollziehbar. Wenn aber nur interne Quellen (*intsources*) angegeben sind, dann müssen deren Ursprünge (usw.) durchsucht werden, bis mindestens eine externe Quelle gefunden wird. Wird keine externe Quelle gefunden, dann liegt ein Mangel vor.

Definition (*traced*: *MainEntry* → *Boolean*): Ein Haupteintrag *e* ist nachvollzogen, wenn er direkt oder indirekt auf mindestens einen Quellenkatalogeintrag verweist:
$traced(e) = \exists s \in SourceEntry \bullet s \in indirectsources(e)$

Die Nachvollzogenheit kann durch eine Inhaltsprüfung überprüft werden. Die Zuverlässigkeit der Prüfergebnisse ist hoch.

Generalisierungen/All-Quantoren

Generalisierungen kommen im normalen Sprachgebrauch häufig vor. Sie helfen, der Realität eine gewisse Ordnung aufzuzwingen, auch wenn diese Ordnung die Realität nicht immer präzise widerspiegelt. Im normalen Sprachgebrauch machen Aussagen wie »Immer, wenn ich gerade spazieren gehen möchte, regnet es.« keine Schwierigkeiten. Jeder Zuhörer weiß, daß diese Aussage nicht ausnahmslos zu jedem Zeitpunkt und an jedem Ort gültig ist, daß der benutzte All-Quantor in Wirklichkeit keiner ist.

Anders ist es in Anforderungsdokumenten. Dort sind Verallgemeinerungen kritisch. Oft spielen die Ausnahmen vom allgemeinen Verhalten eine sehr wichtige Rolle. Bei jeder Anforderung, die einen *All-Quantor* enthält, sollte sich der Systemanalytiker die Frage stellen, ob es zu dieser Aussage nicht vielleicht doch eine Ausnahme geben kann.

Solche All-Quantoren können in der Anforderung gut versteckt sein. Zum Beispiel steckt auch in jeder Bedingung ein All-Quantor. Der Systemanalytiker sollte (sich) in solchen Fällen die Frage stellen, ob es nicht doch Situationen geben kann, in der die Bedingung zwar erfüllt ist, sich das Zielsystem aber anders als beschrieben verhält.

Ein Werkzeug kann nicht feststellen, ob die Benutzung eines All-Quantors an der jeweiligen Stelle im Freitext sinnvoll oder richtig ist. Es kann aber den Prüfer insoweit unterstützen, daß es benutzte All-Quantoren markiert (fokussierte Inspektion) oder findet (Inhaltsprüfung, Filter). Zu jedem so gefundenen All-Quantor soll sich der Prüfer dann die Frage stellen: »Wirklich alle? Ist unter keinen Umständen eine Ausnahme denkbar?« Falls doch, ist die Anforderung nicht korrekt.

Diese Prüfung wird auf Freitexte (*contents*) von Inhaltselementen beschränkt, weil ein falsch benutzter All-Quantor gerade bei Inhaltselementen große Auswirkungen haben kann.

Um diese Prüfungen durchführen zu können, müssen alle in Tabelle 26 aufgezählten Phrasen (inklusive Flexionen) in die Wortliste (siehe Kapitel 6.10) eingetragen und dem Worttyp *quantor* zugeordnet werden.

Definition (*Quantor*): Die Menge *Quantor* enthält alle All-Quantoren der Wortliste. Gegeben sei eine Dokumentation *I*. Dann gilt:
$Quantor = \{p \in Phrase \mid \exists w \in WordEntry|_I \bullet ((term(w) = p) \wedge (quantor \in types(w)))\}$

Worttyp	Phrasen
All-Quantor	all, allemal, allenthalben, allseits, ganz, gesamt, jeder, jedermann, jedesmal, jederzeit, jedweder, jeglicher, jeweils, immer, immerzu, kein, nichts, nie, niemals, niemand, nimmermehr, nirgends, nirgendwo, nirgendwohin, sämtlich, stets, überall, überallher, überallhin, zeitlebens wenn, falls, im Falle, sofern, soweit, unter der Voraussetzung, vorausgesetzt

Tabelle 26: Wichtige All-Quantoren der deutschen Sprache

Definition (*quantorfree*: *ContElement* → *Boolean*): In dem Freitext eines Inhaltselements wird kein All-Quantor benutzt, wenn folgende Bedingung erfüllt ist:
$quantorfree(c) = \neg(\exists q \in Quantor \bullet directuse_W(contents(c), q))$

Die Erfüllung dieses Prädikats kann durch eine Inhaltsprüfung überprüft werden. Die Zuverlässigkeit der Prüfergebnisse ist niedrig.

Die Benutzung eines All-Quantors kann auch durch eine negative fokussierte Inspektion festgestellt werden. Hierfür wird folgende Markierungsfunktion definiert:

Definition ($mark_{quantor}$: *ContElement* → *P(Word)*): Die Markierungsfunktion $mark_w$ (siehe S. 161) wird auf Inhaltselemente angewandt:
$mark_{quantor}(c) = mark_W(contents(c), Quantor)$

Verwendete Default-Werte

Ein Indiz für einen nicht korrekten Eintrag liegt vor, wenn für ein Attribut der vom Werkzeug voreingestellte Default-Wert nicht verändert wurde. Das könnte darauf hindeuten, daß der Systemanalytiker einfach vergessen hat, den Wert anzupassen. Da das Indiz aber sehr schwach ist, wird hierfür »nur« ein Filter definiert.

Definition ($filter_{defaultused}$: *Document* → *P(ContElement* ∪ *GlEntry* ∪ *SourceEntry)*): Dieser Filter ordnet einem Dokument alle enthaltenen Quellenkatalogeinträge, Glossareinträge oder Inhaltselemente zu, in denen mindestens ein Attribut noch seinen Default-Wert hat:

$$\begin{aligned} filter_{defaultused}(doc) = {} & \{g \in GlEntry|_{doc} \mid state(g) = default\} \\ & \cup \{c \in ContElement|_{doc} \mid default(vola(c)) = true \vee state(c) = default\} \\ & \cup \{r \in Requirement|_{doc} \mid pstate(r) = default \vee default(prio(r)) = true\} \\ & \cup \{p \in Problem|_{doc} \mid default(prio(p)) = true\} \\ & \cup \{s \in SourceEntry|_{doc} \mid state(s) = default\} \end{aligned}$$

Prädikat/Funktion	Prüfverfahren
traced: *MainEntry* → *Boolean*	Inhaltsprüfung (hoch)
quantorfree: *ContElement* → *Boolean*	Inhaltsprüfung (niedrig)
$mark_{quantor}$: *ContElement* → *P(Word)*	negative fokussierte Inspektion
$filter_{defaultused}$: *Document* → *P(ContElement* ∪ *GlEntry* ∪ *SourceEntry)*	Inspektion

Tabelle 27: Prüfverfahren für das Qualitätskriterium Korrektheit

8.2.2 Vollständigkeit

Die Erfüllung des Vollständigkeitsprädikats kann nur teilweise automatisch überprüft werden, weil hierbei die Anforderungen und Wünsche der Informanten eine wichtige Rolle spielen. Ein Werkzeug kann unmöglich feststellen, ob in einem Anforderungsdokument alle Wünsche der betroffenen Informanten berücksichtigt worden sind.

Es gibt aber die Möglichkeit, einige Aspekte automatisch zu prüfen. Folgende Prüfungen werden in diesem Kapitel beschrieben:

- Prüfung auf formale Vollständigkeit
- Prüfung auf Vollständigkeit eines Anforderungsdokuments
- Prüfung auf Vollständigkeit des Glossars bzgl. eines Freitextes

Prüfung auf formale Vollständigkeit

Das Informationsmodell läßt gewisse Unvollständigkeiten zu. So darf z.B. ein Glossareintrag mit leerer Definition eingetragen werden. Die Definition kann später nachgetragen werden. Das kann sinnvoll sein, weil die Definition u.U. mit den Informanten abgesprochen werden muß. An vielen Stellen im Informationsmodell kann der Systemanalytiker TBD-Einträge vornehmen, um Unvollständigkeiten explizit zu markieren. Solche und ähnliche Vollständigkeitskriterien werden in dem Prädikat *formalcomplete* zusammengefaßt.

Definition (*formalcomplete*: *SingleEntry* → *Boolean*): Ein Einzeleintrag e ist formal vollständig, wenn folgende Bedingung erfüllt ist:

$$formalcomplete(e) = \forall i \in \{1, \ldots, 31\} \bullet f_i(e) ,$$

mit:

1. Zu jedem Haupteintrag ist mindestens ein Ursprung angegeben.
   $f_1(e) = (e \in MainEntry \Rightarrow (|intsources(e)| + |extsources(e)| \geq 1))$
2. Die Beschreibung eines Quellenkatalogeintrags ist nicht leer.
   $f_2(e) = (e \in SourceEntry \Rightarrow (|ref(e)| > 0))$
3. Die Beschreibung eines Quellenkatalogeintrags enthält kein TBD.
   $f_3(e) = (e \in SourceEntry \Rightarrow (directuse_W(ref(e), 'TBD') = false))$
4. Der Zustand eines Quellenkatalogeintrags ist nicht gleich TBD.
   $f_4(e) = (e \in SourceEntry \Rightarrow (state(e) \neq TBD))$
5. Jede Informationsquelle der Kategorie »Partei« oder »externes Dokument« verweist auf mindestens eine Informationsquelle der Kategorie »Person«.
   $f_5(e) = (e \in DocSourceEntry \cup PartySourceEntry \Rightarrow (|associatedpersons(e)| \geq 1))$
6. Die Definition eines Glossareintrags ist nicht leer.
   $f_6(e) = (e \in GlEntry \Rightarrow (|definition(e)| > 0))$
7. Die Definition eines Glossareintrags enthält kein TBD.
   $f_7(e) = (e \in GlEntry \Rightarrow (directuse_W(definition(e), 'TBD') = false))$

8. Der Zustand eines Glossareintrags ist nicht gleich TBD.
   $f_8(e) = (e \in GlEntry \Rightarrow (state(e) \neq TBD))$

9. Zu jedem Glossareintrag ist mindestens eine Kategorie angegeben.
   $f_9(e) = (e \in GlEntry \Rightarrow (|classes(e)| \geq 1))$

10. Zu jedem Glossareintrag ist mindestens eine Flexion angegeben.
    $f_{10}(e) = (e \in GlEntry \Rightarrow (|inflections(e)| \geq 1))$

11. Die Kategorien eines Glossareintrags sind ungleich TBD.
    $f_{11}(e) = (e \in GlEntry \Rightarrow (TBD \notin classes(e)))$

12. Die Überschrift eines Kapitels enthält kein TBD.
    $f_{12}(e) = (e \in Chapter \Rightarrow (directuse_W(heading(e), 'TBD') = false))$

13. Die Einleitung eines Kapitels ist nicht leer.
    $f_{13}(e) = (e \in Chapter \Rightarrow (|introduction(e)| > 0))$

14. Die Einleitung eines Kapitels enthält kein TBD.
    $f_{14}(e) = (e \in Chapter \Rightarrow (directuse_W(introduction(e), 'TBD') = false))$

15. Der Abstract eines Anforderungsdokuments ist nicht leer.
    $f_{15}(e) = (e \in ReqDocument \Rightarrow (|abstract(e)| > 0))$

16. Der Abstract eines Anforderungsdokuments enthält kein TBD.
    $f_{16}(e) = (e \in ReqDocument \Rightarrow (directuse_W(abstract(e), 'TBD') = false))$

17. Der Titel eines Anforderungsdokuments enthält kein TBD.
    $f_{17}(e) = (e \in ReqDocument \Rightarrow (directuse_W(title(e), 'TBD') = false))$

18. Jedes Ist-Analysedokument und Pflichtenheft enthält ein Systemmodell.
    $f_{18}(e) = (e \in CurrentState \cup Specification \Rightarrow (\exists s \in SystemModel|_e))$

19. Ein Systemmodell einer Anforderungssammlung, eines Lastenhefts oder eines Pflichtenhefts enthält mindestens ein Zielsystem.
    $f_{19}(e) = ((e \in SystemModel \wedge (\exists d \in ReqCollection \cup ReqSpecification \cup Specification \bullet e \in contains(d))) \Rightarrow (|\{t \in TargetSystem | t \in contains(e)\}| > 0))$

20. Der Freitext eines Inhaltselements ist nicht leer.
    $f_{20}(e) = (e \in ContElement \Rightarrow (|contents(e)| > 0))$

21. Der Freitext eines Inhaltselements enthält kein TBD.
    $f_{21}(e) = (e \in ContElement \Rightarrow (directuse_W(contents(e), 'TBD') = false))$

22. Zu jedem Inhaltselement ist mindestens eine Kategorie angegeben.
    $f_{22}(e) = (e \in ContElement \Rightarrow (|classes(e)| \geq 1))$

23. Die Kategorien eines Inhaltselements sind ungleich TBD.
    $f_{23}(e) = (e \in ContElement \Rightarrow (TBD \notin classes(e)))$

24. Der Zustand eines Inhaltselements ist ungleich TBD.
$f_{24}(e) = (e \in ContElement \Rightarrow (state(e) \neq TBD))$

25. Die Priorität einer Anforderung oder eines Problems ist ungleich TBD.
$f_{25}(e) = (e \in Requirement \cup Problem \Rightarrow (prio(e) \neq TBD))$

26. Die Stabilität eines Inhaltselements ist ungleich TBD.
$f_{26}(e) = (e \in ContElement \Rightarrow (vola(e) \neq TBD))$

27. Der Projektstand einer Anforderung ist ungleich TBD.
$f_{27}(e) = (e \in Requirement \Rightarrow (pstate(e) \neq TBD))$

28. Die Beschreibung eines Testfalls ist nicht leer.
$f_{28}(e) = (e \in Requirement \cup Assertion \cup Fact \Rightarrow (\forall t \in tests(e) \bullet (|descr(t)| > 0)))$

29. Die Beschreibung eines Testfalls enthält kein TBD.
$f_{29}(e) = (e \in Requirement \cup Assertion \cup Fact \Rightarrow \forall t \in tests(e) \bullet (directuse_W(descr(t), 'TBD') = false))$

30. Der Inhalt einer Annotation ist nicht leer.
$f_{30}(e) = (e \in Annotation \Rightarrow (|contents(e)| > 0))$

31. Der Inhalt einer Annotation enthält kein TBD.
$f_{31}(e) = (e \in Annotation \Rightarrow (directuse_W(contents(e), 'TBD') = false))$

Die Erfüllung dieses Prädikats kann durch eine Inhaltsprüfung überprüft werden. Die Zuverlässigkeit der Prüfergebnisse ist hoch.

Prüfung auf Vollständigkeit eines Anforderungsdokuments

Einige Indizien für Unvollständigkeiten können aus dem Abgleich des Anforderungsdokuments mit anderen Einträgen der Dokumentation gewonnen werden:

- Im Glossar wird die projektrelevante Terminologie definiert. Werden aktuelle Glossareinträge in den Freitexten eines Anforderungsdokuments nicht benutzt, könnte das darauf hindeuten, daß in dem Anforderungsdokument bestimmte Aspekte nicht behandelt worden sind.
- Jedes Anforderungsdokument enthält eine Kapitelhierarchie. Wenn ein Kapitel nur wenige oder gar keine Inhaltselemente enthält, sei es direkt oder auch unter Berücksichtigung der Unterkapitel, liegt ein Indiz für einen Mangel vor.
- In der Metaspezifikation wird festgelegt, welche Kategorien den Inhaltselementen zugeordnet werden können. Wird eine Kategorie keinem Inhaltselement oder nur sehr wenigen Inhaltselementen in einem Anforderungsdokument zugeordnet, liegt ein Indiz für eine Unvollständigkeit vor.
- Jedes Anforderungsdokument kann ein Systemmodell enthalten. An die Systemmodelleinträge können Inhaltselemente gebunden werden. Ein Indiz für eine Unvollständigkeit liegt vor, wenn an einen Systemmodelleintrag keine oder nur wenige Inhaltselemente gebunden sind.
- Wenn im Quellenkatalog aktuelle Quellenkatalogeinträge enthalten sind, auf die keine oder nur wenige Haupteinträge zurückgeführt (siehe *traced* in Kapitel

8.2.1) werden können, liegt ein Indiz dafür vor, daß Informationsquellen nicht ausreichend berücksichtigt worden sind.

- Ein ähnlich geartetes Indiz für eine Unvollständigkeit liegt vor, wenn nicht auf alle Einträge eines als Vorlage für andere Anforderungsdokumente dienenden Anforderungsdokuments verwiesen wird. So sollte z.B. auf alle aktuellen Einträge einer Anforderungssammlung, die in ein Lastenheft oder Ist-Analysedokument integriert wurde, verwiesen werden.

Problematisch hierbei ist, daß es keine klaren Zielvorgaben gibt. Ist es in Ordnung, wenn nur zwei Effizienzanforderungen in einem Anforderungsdokument enthalten sind, wenn ein Term nur ein einziges Mal benutzt wird, wenn in einem Einleitungskapitel überhaupt keine Inhaltselemente enthalten sind, wenn an ein System der Arbeitsumgebung nur drei Zusicherungen gebunden sind oder wenn ein Informant nur für eine einzige Anforderung im Pflichtenheft verantwortlich ist? Es ist nicht klar, ab welcher Häufigkeit jeweils ein Mangel vorliegt. Deswegen können Inhaltsprüfungen hierbei auch nicht sinnvoll eingesetzt werden, wohl aber *Statusprüfungen*. Für die Statusprüfungen werden folgende Funktionen definiert:

Definition (*termcount*: $GlEntry \times ReqDocument \rightarrow N \times N$): Die Funktion *termcount* ordnet einem Glossareintrag und einem Anforderungsdokument die Anzahl der in den aktuellen Einträgen des Anforderungsdokuments direkt und nur indirekt benutzten Glossareinträge zu:

$$termcount(g, d) = (\left|\{e \in DocEntry(d) \mid \exists p \in paragraphs(e) \bullet directuse_G(p, g)\}\right|, \left|\{e \in DocEntry(d) \mid \exists p \in paragraphs(e) \bullet indirectuse_G(p, g) \wedge (directuse_G(p, g) = false)\}\right|)$$

Definition (*chaptercount*: $Chapter \rightarrow N \times N \times N \times N$): Die Funktion *chaptercount* ordnet einem Kapitel die Anzahl der in dem Kapitel oder seinen Unterkapiteln enthaltenen Anforderungen, Fakten, Probleme und Zusicherungen zu:

$$chaptercount(c) = (\left|\{r \in Requirement|_c \mid state(r) \neq rejected\}\right|, \left|\{f \in Fact|_c \mid state(f) \neq rejected\}\right|, \left|\{p \in Problem|_c \mid state(p) \neq rejected\}\right|, \left|\{a \in Assertion|_c \mid state(a) \neq rejected\}\right|)$$

Definition (*classcount*: $Class \times ReqDocument \rightarrow N$): Die Funktion *classcount* ordnet einer Kategorie und einem Anforderungsdokument die Anzahl der entsprechend kategorisierten aktuellen Inhaltselemente des Dokuments zu:

$$classcount(c, d) = (\left|\{r \in CurrentContElement(d) \mid \exists sc \in indirectsubclasses(c) \bullet sc \in classes(r)\}\right|)$$

Definition (*sysentrycount*: $SysModelEntry \times ReqDocument \rightarrow N$): Die Funktion *sysentrycount* ordnet einem Systemmodelleintrag und einem Anforderungsdokument die Anzahl der Inhaltselemente des Anforderungsdokuments zu, die auf den gegebenen Systemmodelleintrag verweisen:

$$sysentrycount(s, d) = \left|\{c \in CurrentContElement(d) \mid s \in sysentries(c)\}\right|$$

Definition (*extsourcecount*: $SourceEntry \times ReqDocument \rightarrow N$): Die Funktion *extsourcecount* ordnet einem Quellenkatalogeintrag die Anzahl der aktuellen Haupteinträge des Anforderungsdokuments zu, die auf den gegebenen Quellenkatalogeintrag direkt oder indirekt verweisen:

$$extsourcecount(s, d) = \left|\{m \in CurrentMainEntry(d) \mid s \in indirectsources(m)\}\right|$$

Definition (*intsourcecount*: *Entry* → *N*): Die Funktion *intsourcecount* ordnet einem Eintrag die Anzahl der aktuellen Haupteinträge der Dokumentation zu, die auf den gegebenen Eintrag direkt oder indirekt verweisen:

$$intsourcecount(e) = \left|\{ m \in CurrentMainEntry(model(e)) \mid e \in indirectsources(m) \}\right|$$

Wenn das Anforderungsdokument *doc* der Dokumentation *I* durch Statusprüfungen geprüft werden soll, müssen die oben definierten Funktionen auf Teile der Dokumentation angewandt und die Ergebnisse angezeigt werden. Dafür werden folgende Anzeigefunktionen definiert:

- $A_{TermCount}$: *ReqDocument* → *P(GlEntry* × *N* × *N)*. Die Funktion *termcount* wird auf alle aktuellen Glossareinträge angewandt:

$$A_{TermCount}(doc) = \bigcup_{g \in MainTerm(model(doc))} (g, termcount(g, doc))$$

- $A_{ChapterCount}$: *ReqDocument* → *P(Chapter* × *N* × *N* × *N* × *N)*. Die Funktion *chaptercount* wird auf alle Kapitel des Anforderungsdokuments angewandt:

$$A_{ChapterCount}(doc) = \bigcup_{c \in Chapter} (c, chaptercount(c))$$

- $A_{ClassCount}$: *ReqDocument* → *P(Class* × *N)*. Die Funktion *classcount* wird auf alle Kategorien der Kategorisierungshierarchie für Inhaltselemente angewandt:

$$A_{ClassCount}(doc) = \bigcup_{c \in contclass(Metaspecification)} (c, classcount(c, doc))$$

- $A_{SysEntryCount}$: *ReqDocument* → *P(SysModelEntry* × *N)*. Die Funktion *sysentrycount* wird auf alle Systemmodelleinträge des Anforderungsdokuments angewandt:

$$A_{SysEntryCount}(doc) = \bigcup_{e \in SysModelEntry} (e, sysentrycount(e, doc))$$

- $A_{ExtSourceCount}$: *ReqDocument* → *P(SourceEntry* × *N)*. Die Funktion *extsourcecount* wird auf alle aktuellen Quellenkatalogeinträge angewandt:

$$A_{ExtSourceCount}(doc) = \bigcup_{s \in CurrentSourceEntry(model(doc))} (s, extsourcecount(s, doc))$$

- $A_{IntSourceCount}$: *ReqDocument* → *P(Entry* × *N)*. Die Funktion *intsourcecount* wird auf alle aktuellen Einträge des Anforderungsdokuments angewandt:

$$A_{IntSourceCount}(doc) = \bigcup_{e \in DocEntry(doc)} (e, intsourcecount(e))$$

Die Statusprüfung kann auch auf mehrere Anforderungsdokumente gleichzeitig angewandt werden. Sinnvoll ist das z.B., wenn eine Dokumentation mehrere Pflichtenhefte enthält, die zusammen das »eigentliche« Pflichtenheft bilden.

Prüfung auf Vollständigkeit des Glossars bzgl. eines Freitextes

Das Glossar ist vollständig, wenn es für alle potentiell mehrdeutigen Phrasen in den Freitexten einer Dokumentation und für jeden in einem Systemmodell vorkommenden Systemmodelleintrag einen entsprechenden Glossareintrag gibt.

Die Benutzung von aktuellen Termen des Glossars in einem Freitext kann durch eine positive fokussierte Inspektion festgestellt werden. Hierfür wird folgende Markierungsfunktion definiert.

Definition ($mark_{termused}$: $Glossary \times NLEntry \rightarrow P(Paragraph \times P(Word))$[1]): Die Markierungsfunktion $mark_G$ (siehe S. 161) wird auf alle Freitexte des NL-Eintrags angewandt:

$$mark_{termused}(G, e) = \bigcup_{p \in paragraph(e)} (p, mark_G(p, \{g \in GlEntry|_G \mid state(g) \neq rejected\}))$$

Der Abgleich zwischen den Systemmodelleinträgen und dem Glossar kann durch eine Inhaltsprüfung erfolgen. Hierfür wird folgendes Prädikat definiert:

Definition ($sysentrydefined$: $Glossary \times SystemModel \rightarrow Boolean$): Die Systemmodelleinträge sind definiert, wenn folgende Bedingung erfüllt ist:

$$sysentrydefined(G, s) = (\forall e \in SysModelEntry|_s \bullet name(e) \in term(\{g \in G \mid state(g) \neq rejected\}))$$

Die Zuverlässigkeit der Prüfergebnisse der Inhaltsprüfung ist hoch.

Prädikat	Prüfverfahren
$formalcomplete: SingleEntry \cup ReqDocument \rightarrow Boolean$	Inhaltsprüfung (hoch)
$A_{TermCount}: ReqDocument \rightarrow P(GlEntry \times N \times N)$	Statusprüfung
$A_{ChapterCount}: ReqDocument \rightarrow P(Chapter \times N \times N \times N \times N)$	Statusprüfung
$A_{ClassCount}: ReqDocument \rightarrow P(Class \times N)$	Statusprüfung
$A_{SysEntryCount}: ReqDocument \rightarrow P(SysModelEntry \times N)$	Statusprüfung
$A_{ExtSourceCount}: ReqDocument \rightarrow P(SourceEntry \times N)$	Statusprüfung
$A_{IntSourceCount}: ReqDocument \rightarrow P(Entry \times N)$	Statusprüfung
$mark_{termused}: Glossary \times NLEntry \rightarrow P(Paragraph \times P(Word))$	positive fokussierte Inspektion
$sysentrydefined: Glossary \times SystemModel \rightarrow Boolean$	Inhaltsprüfung (hoch)

Tabelle 28: Prüfverfahren für das Qualitätskriterium Vollständigkeit

8.2.3 Adäquatheit

Das Prädikat *adäquat* ist erfüllt, wenn die Inhaltselemente aus Sicht mindestens eines Informanten und bezogen auf die aktuelle Problemstellung angemessen sind. Weil die Beurteilung der Adäquatheit sehr subjektiv ist und weil Dokumentatio-

1. Diese Funktion hat als Rückgabewert eine Menge von Paaren (p, W), wobei p ein Freitext und W eine Menge von Wörtern ist. Zu jedem Freitext, auf den die Funktion $mark_G$ angewandt wird, wird die Menge der zu markierenden Wörter zurückgeliefert. Es würde nicht ausreichen, nur eine Menge von Wörtern als Ergebnis zu liefern, weil klar zugeordnet werden muß, in welchem Freitext welche Wörter markiert werden sollen.

   Wenn z.B. in dem Titel eines Anforderungsdokuments der Term »Attribut eines Dokuments« benutzt wird, dann werden in dem Titel alle Flexionen von »Attribut« und »Dokument« markiert. Es muß aber verhindert werden, daß z.B. das Wort »Attribut« auch in dem Abstract des gleichen Anforderungsdokuments markiert wird, wenn der Abstract zwar eine Flexion von »Attribut« enthält, der Term »Attribut eines Dokuments« aber nicht benutzt wird.

nen einen sehr hohen Anteil natürlicher Sprache enthalten, kann die Erfüllung dieses Qualitätskriteriums nur durch Inspektion geprüft werden.

8.2.4 Konsistenz

Auch für das Prädikat *konsistent* gilt, daß seine Erfüllung nur bis zu einem gewissen Grad automatisch überprüft werden kann, weil z.B. Widersprüche zwischen Anforderungen im wesentlichen von der Semantik der Freitexte abhängen.

Es gibt aber die Möglichkeit, einige Aspekte automatisch zu prüfen. Folgende Prüfungen werden in diesem Kapitel beschrieben:

- Prüfung der Terminologie
- Prüfung der Zustände
- Prüfung der Systemmodelle
- Prüfung der Ursprungsverweise
- Prüfung der textuellen Querverweise
- Prüfung auf Widersprüche in Inhaltselementen und Annotationen

Prüfung der Terminologie

Das Informationsmodell läßt gewisse Inkonsistenzen in einer Menge von Glossareinträgen zu. Einige dieser Inkonsistenzen können automatisch gefunden werden.

Definition (formal konsistent, $formalconsistent: P(GlEntry) \rightarrow Boolean$): Eine Menge von Glossareinträgen ist formal konsistent, wenn folgende Bedingung erfüllt ist:

$$formalconsistent(G) = \forall i \in \{1, \ldots, 7\} \bullet f_i(G)$$

mit:

1. Die Terme sind paarweise verschieden.
   $$f_1(G) = (\forall g_1, g_2 \in G \bullet (term(g_1) = term(g_2) \Rightarrow g_1 = g_2))$$
2. Das Glossar des Anwendungsbereichs ist in sich abgeschlossen, d.h. in der Definition eines Terms des Anwendungsbereichs werden nur Terme des Anwendungsbereichs benutzt.
   $$f_2(G) = (\forall g_1, g_2 \in G \bullet ((directuse_G(definition(g_1), g_2) \wedge scope(g_1) = domain) \Rightarrow (scope(g_2) = domain)))$$
3. Jede Menge synonymer Terme enthält maximal einen aktuellen Term.
   $$f_3(G) = (\forall g \in G \bullet (state(g) = current \Rightarrow (\forall s \in synonymes^*(g)\big|_G \bullet state(s) \neq current)))$$
4. Die Hyponymiebeziehung ist zyklusfrei.
   $$f_4(G) = (\forall g \in G \bullet \forall i \in N^{\geq 1} \bullet (g \notin hyperonymes^i(g)))$$
5. Die Menge der Synonym-Verweise eines Glossareintrags enthält auch alle indirekten Synonyme.
   $$f_5(G) = (\forall g \in G \bullet \forall i \in N^{\geq 1} \bullet (synonymes(g) \supset synonymes^i(g)))$$

6. Die Menge der Hyperonym-Verweise eines Glossareintrags enthält auch alle indirekten Hyperonyme.

$$f_6(G) = (\forall g \in G \bullet \forall i \in N^{\geq 1} \bullet (hyperonymes(g) \supset hyperonymes^i(g)))$$

7. Synonyme Glossareinträge sind gleich kategorisiert.

$$f_7(G) = (\forall g_1, g_2 \in G \bullet (g_1 \in synonymes(g_2) \Rightarrow (classes(g_1) = classes(g_2))))$$

Diese Bedingungen können jeweils durch eine Inhaltsprüfung überprüft werden. Die Zuverlässigkeit der Prüfergebnisse der Inhaltsprüfung ist hoch.

Prüfung der Zustände

Folgende Einträge einer Dokumentation verfügen über Zustandsangaben: Glossareinträge, Inhaltselemente, Quellenkatalogeinträge und Mangeleinträge. Problematisch wird es, wenn sich aktuelle Einträge auf verworfene Einträge beziehen.

Aktuelle Einträge sind:

- alle Inhaltselemente, Glossareinträge, Quellenkatalogeinträge und Mangeleinträge, deren Zustand nicht »verworfen« ist. Hiebei gilt also, daß Einträge im Zustand »TBD« oder »zukünftig« auch als aktuell aufgefaßt werden.
- alle Annotationen, die an mindestens einen nicht verworfenen Eintrag gebunden sind.
- alle sonstigen Einträge.

Hierfür wird folgendes Hilfsprädikat definiert:

Definition (*current*: *Entry* → *Boolean*): Das Prädikat *current* ist *true*, wenn der gegebene Eintrag aktuell ist:

$$current(e) = \begin{cases} false, falls\ (e \in GlEntry \cup ContElement \cup SourceEntry) \wedge (state(e) = rejected) \\ false, falls\ (e \in Defect) \wedge (state(e) = rejected) \\ false, falls\ (e \in Annotation) \wedge (\forall f \in entries(e) \bullet (current(f) = false)) \\ true, sonst \end{cases}$$

Definition (*formalconsistent*: *SingleEntry* → *Boolean*): Ein Einzeleintrag ist formal konsistent bzgl. seines Zustands, wenn folgende Bedingung erfüllt ist:

$$formalconsistent(e) = \forall i \in \{1, \ldots, 4\} \bullet f_i(e)$$

mit:

1. In den Freitexten werden nur aktuelle Glossareinträge benutzt.

$$f_1(e) = (\forall p \in paragraphs(e) \bullet \forall g \in MainTerm(model(e)) \bullet (directuse_G(p, g) \Rightarrow current(g)))$$

2. Eine Annotation ist nur an aktuelle Einträge gebunden.

$$f_2(e) = (e \in Annotation \Rightarrow (\forall f \in entries(e) \bullet current(f)))$$

3. Die Ursprünge eines Haupteintrags sind aktuell.

$$f_3(e) = (e \in MainEntry \Rightarrow (\forall s \in indirectsources(e) \bullet current(s)))$$

4. Ein Quellenkatalogeintrag verweist nicht auf verworfene Personen-Quellenkatalogeinträge.

$$f_4(e) = (e \in DocSourceEntry \cup PartySourceEntry \Rightarrow (\forall s \in associatedpersons(e) \bullet current(s)))$$

Die Erfüllung dieses Prädikats kann durch eine Inhaltsprüfung überprüft werden. Die Zuverlässigkeit der Prüfergebnisse ist hoch.

Die Benutzung von verworfenen Termen des Glossars in einem Freitext kann auch durch eine negative fokussierte Inspektion festgestellt werden. Hierfür wird folgende Markierungsfunktion definiert.

Definition ($mark_{rejectedtermused}$: $Glossary \times NLEntry \rightarrow P(Paragraph \times P(Word))$): Die Markierungsfunktion $mark_G$ (siehe S. 161) wird auf alle Freitexte des NL-Eintrags angewandt (siehe auch Fußnote auf S. 175):

$$mark_{rejectedtermused}(G, e) = \bigcup_{p \in paragraphs(e)} (p, mark_G(p, \{g \in GlEntry|_G \mid state(g) = rejected\}))$$

Prüfung der Systemmodelle

Ein Systemmodell soll einen zusammenhängenden Graphen enthalten. Wenn Teile des Systemmodells von anderen Teilen nicht erreichbar sind, bedeutet das, daß sie nicht miteinander kommunizieren können. Somit ist einer der Teile sinnlos. Desweiteren sollen die Namen der Systemmodelleinträge eines Systemmodells paarweise verschiedenen sein.

Hierfür wird folgendes Hilfsprädikat definiert:

Definition (*connected*: $SysModelEntry \times SysModelEntry \rightarrow Boolean$): Das Prädikat *connected* ist *true*, wenn die beiden Systemmodelleinträge direkt miteinander über Kommunikationskanten oder Schnittstellenkanten verbunden sind:

$$connected(e_1, e_2) = \begin{cases} true, falls\ (e_1 \in System) \wedge (e_2 \in Interface) \wedge (e_1 \in system(e_2)) \\ true, falls\ (e_1 \in Interface) \wedge (e_2 \in System) \wedge (e_2 \in system(e_1)) \\ true, falls\ \exists c \in Connector \bullet (\{e_1, e_2\} = associatedsystems(c)) \\ false, sonst \end{cases}$$

Definition (*formalconsistent*: $SystemModel \rightarrow Boolean$): Ein Systemmodell ist formal konsistent, wenn folgende Bedingung erfüllt ist:

$$formalconsistent(s) = f_1(s) \wedge f_2(s)\ ,$$

mit:

1. Die Namen der Systemmodelleinträge sind paarweise verschieden.

$$f_1(s) = (\forall e_1, e_2 \in SysModelEntry|_s \bullet (name(e_1) = name(e_2) \Rightarrow e_1 = e_2))$$

2. Das Systemmodell ist zusammenhängend.

$$f_2(s) = (\forall e_1, e_2 \in SysModelEntry|_s \bullet connected(e_1, e_2) \vee (\exists i \in N \bullet \exists f_0, \ldots, f_i \in SysModelEntry \bullet (connected(e_1, f_0) \wedge connected(f_i, e_2)) \wedge \forall j \in \{1, \ldots, i\} \bullet connected(f_{j-1}, f_j)))$$

Die Erfüllung dieses Prädikats kann durch eine Inhaltsprüfung überprüft werden. Die Zuverlässigkeit der Prüfergebnisse ist hoch.

Prüfung auf Benutzung von textuellen Querverweisen

Ein Freitext kann textuelle Querverweise enthalten (siehe Kapitel 5.2). Für den Systemanalytiker ist es eine Hilfe, wenn textuelle Querverweise (auf wirklich vorhandene Einträge) markiert werden.

Der Aufbau der eindeutigen Bezeichner wird durch das Informationsmodell nicht vorgegeben. Insofern kann prinzipiell jede Phrase ein eindeutiger Bezeichner eines Eintrags sein. Deswegen können »ungültige« textuelle Querverweise nicht automatisch erkannt werden. Das Vorhandensein eines textuellen Querverweises in dem Freitext eines NL-Eintrags kann durch eine positive fokussierte Inspektion festgestellt werden. Hierfür wird folgende Markierungsfunktion definiert:

Definition ($mark_{crossreference}$: $NLEntry \rightarrow P(Paragraph \times P(Word))$): Die Markierungsfunktion $mark_W$ (siehe S. 161) wird auf alle Freitexte des NL-Eintrags angewandt (siehe auch Fußnote auf S. 175):

$$mark_{crossreference}(e) = \bigcup_{p \in paragraphs(e)} (p, mark_W(p, id(Entry|_{model(e)})))$$

Prüfung der Ursprungsverweise

Die internen Ursprungsverweise der Haupteinträge (*intsources*) dürfen keine Zyklen aufweisen. Ein Zyklus würde schließlich bedeuten, daß ein Haupteintrag indirekt von sich selbst abgeleitet worden wäre.

Definition ($formalconsistent$: $MainEntry \rightarrow Boolean$): Ein Haupteintrag ist formal konsistent, wenn er über Ursprungsverweise nicht indirekt auf sich selbst verweist:

$$formalconsistent(m) = \left(m \notin \bigcup_{i \in \mathbb{N}^{\geq 1}} intsources^i(m) \right)$$

Die Erfüllung dieses Prädikats kann durch eine Inhaltsprüfung überprüft werden. Die Zuverlässigkeit der Prüfergebnisse ist hoch.

Prüfung auf Widersprüche in Inhaltselementen und Annotationen

Für eine werkzeugunterstützte Prüfung auf Widersprüche in Inhaltselementen und Annotationen kann die Ähnlichkeitssuche eingesetzt werden (Kapitel 8.1.5).

Prädikat	Prüfverfahren
$formalconsistent$: $P(GlEntry) \rightarrow Boolean$	Inhaltsprüfung (hoch)
$formalconsistent$: $SingleEntry \rightarrow Boolean$	Inhaltsprüfung (hoch)
$mark_{rejectedtermused}$: $Glossary \times NLEntry \rightarrow P(Paragraph \times P(Word))$	negative fokussierte Inspektion
$formalconsistent$: $SystemModel \rightarrow Boolean$	Inhaltsprüfung (hoch)
$mark_{crossreference}$: $NLEntry \rightarrow P(Paragraph \times P(Word))$	positive fokussierte Inspektion
$formalconsistent$: $MainEntry \rightarrow Boolean$	Inhaltsprüfung (hoch)
$consistent$: $P(ContElement \cup Annotation) \rightarrow Boolean$	Ähnlichkeitssuche

Tabelle 29: Prüfverfahren für das Qualitätskriterium Konsistenz

8.2.5 Externe Konsistenz

Das Prädikat *extern konsistent* ist erfüllt, wenn die Einträge einer Dokumentation und die vorhandenen externen Dokumente keine Widersprüche aufweisen. Ein Werkzeug kann nicht überprüfen, ob die Einträge extern konsistent sind, weil externe Dokumente nicht Teil der Dokumentation sind.

Es kann den Systemanalytiker durch folgenden Filter dabei unterstützen, alle Haupteinträge, die sich auf ein externes Dokument beziehen, ausfindig zu machen.

Definition ($filter_{sources}$: $ReqDocument \times DocSourceEntry \rightarrow P(MainEntry)$): Dieser Filter ordnet einem Anforderungsdokument und einem externen Dokument alle Haupteinträge zu, die direkt oder indirekt auf das externe Dokument zurückgeführt werden können:

$$filter_{sources}(d, s) = \{m \in MainEntry|_d \mid s \in indirectsources(m)\}$$

Prädikat	Prüfverfahren
$filter_{sources}$: $ReqDocument \times DocSourceEntry \rightarrow P(MainEntry)$	Inspektion

Tabelle 30: Prüfverfahren für das Qualitätskriterium externe Konsistenz

8.2.6 Redundanzfreiheit

Das Prädikat *nicht redundant* ist erfüllt, wenn in den Inhaltselementen eines Anforderungsdokuments keine Informationen mehrfach enthalten sind. Für die Suche nach redundanten Inhaltselementen bietet sich eine Ähnlichkeitssuche (Kapitel 8.1.5) an.

Prädikat	Prüfverfahren
$notredundant$: $P(ContElement) \rightarrow Boolean$	Ähnlichkeitssuche

Tabelle 31: Prüfverfahren für das Qualitätskriterium Redundanzfreiheit

8.2.7 Keine Überspezifikation

Das Prädikat *nicht überspezifiziert* ist erfüllt, wenn durch eine Anforderung keine unnötigen Realisierungsvorgaben gemacht werden. Die Betonung liegt auf »unnötig«, weil eine Anforderung durchaus Realisierungsvorgaben enthalten darf. Bei der Prüfung geht es also nicht um die Frage, ob eine Anforderung Realisierungsvorgaben enthält, sondern, ob es notwendige Vorgaben sind. Oft ist es für Informanten einfacher, ihre Anforderungen in Form eines Beispielsystems zu beschreiben. Aufgabe des Systemanalytikers ist es, festzustellen, wo das »Beispiel« aufhört und die »wahre« Anforderung beginnt.

Eine Möglichkeit besteht darin, nach Gründen für die Realisierungsdetails zu fahnden (siehe z.B. Yu (1997, S. 226), Harvell (1997, S. 27)). Wenn es gute Gründe gibt, dann liegt eine Anforderung vor, wenn nicht, dann liegt eine unnötige Realisierungsvorgabe vor. Da die Begründung wichtig ist, muß sie auf jeden Fall in Form einer Annotation festgehalten werden. Ein sehr schwaches Indiz für eine Überspezifikation liegt vor, wenn ein Inhaltselement existiert, an das keine Anno-

tation gebunden wurde. Wegen der Schwäche des Indizes wird keine Inhaltsprüfung definiert, sondern nur ein Filter.

Definition ($filter_{notannotated}$: $ReqDocument \rightarrow P(Requirement)$): Dieser Filter ordnet einem Anforderungsdokument alle enthaltenen Anforderungen zu, denen keine Annotationen zugeordnet wurden:

$$filter_{notannotated}(d) = \{r \in Requirement|_d \mid r \notin entries(DocAnnotation(d))\}$$

Falls die Kategorisierungshierarchie für Inhaltselemente eine Kategorie für »Realisierungsvorgaben« enthält, kann folgender Filter angewandt werden.

Definition (Filter, $filter_{classes}$: $ReqDocument \times Class \rightarrow P(Requirement)$): Dieser Filter ordnet einem Anforderungsdokument und einer Kategorie der Kategorisierungshierarchie alle Anforderungen zu, denen die Kategorie oder eine Unterkategorie zugeordnet wurde:

$$filter_{classes}(d, c) = \{r \in Requirement|_d \mid \exists sc \in indirectsubclasses(c) \bullet sc \in classes(c)\}$$

Prädikat	Prüfverfahren
$filter_{notannotated}$: $ReqDocument \rightarrow P(Requirement)$	Inspektion
$filter_{classes}$: $ReqDocument \times Class \rightarrow P(Requirement)$	Inspektion

Tabelle 32: Prüfverfahren für das Qualitätskriterium nicht überspezifiziert

8.2.8 Realisierbarkeit

Das Prädikat *realisierbar* ist erfüllt, wenn es eine Implementierung gibt, die eine gegebene Menge von Anforderungen gleichzeitig erfüllt. Eine Möglichkeit, die Realisierbarkeit abzuschätzen, besteht darin, daß die Systemanalytiker oder Informanten Lösungsvorstellungen entwickeln. Bei den Lösungsvorstellungen handelt es sich nicht um Realisierungsvorgaben. Es soll nur die Existenz einer Lösung festgestellt werden.

Ein Werkzeug ist nicht in der Lage, die Realisierbarkeit informaler Anforderungen zu bewerten. Hier sind *formale*, insbesondere *operationale Notationen* im Vorteil. Liegt ein Pflichtenheft in einer ausführbaren Notation vor, kann es, zumindest theoretisch, *animiert* werden (siehe Kapitel 2.3.4). Unter Umständen kann die Realisierbarkeit sogar bewiesen werden. Informale Notationen bieten das nicht.

In besonders kniffligen Fällen, in denen die Realisierbarkeit auch von einem Experten nicht abgeschätzt werden kann, bietet sich *Prototyping* (siehe Kapitel 2.3.4) an. Prototyping wird vom ADMIRE-Ansatz nicht explizit unterstützt.

Dieses Kriterium kann also nur durch Inspektion geprüft werden. Hierbei sollten aber nicht die Kunden gefragt werden, sondern Personen aus der Entwicklungsabteilung. Die Inspektion kann durch folgenden Filter unterstützt werden:

Definition ($filter_{target}$: $TargetSystem \rightarrow P(Requirement)$): Dieser Filter ordnet einem Zielsystem alle Anforderungen zu, die an das Zielsystem oder an eine seiner Schnittstellen gebunden sind:

$$filter_{target}(t) = \{r \in Requirement \mid t \in sysentries(r) \vee (\exists i \in Interface \bullet (t = system(i)) \wedge (i \in sysentries(r)))\}$$

Prädikat	Prüfverfahren
$filter_{target}$: $TargetSystem \rightarrow P(Requirement)$	Inspektion

Tabelle 33: Prüfverfahren für das Qualitätskriterium Realisierbarkeit

8.2.9 Überprüfbarkeit

Das Prädikat *überprüfbar* ist erfüllt, wenn es klare Kriterien gibt, anhand derer festgestellt werden kann, ob das Zielsystem und die Systeme oder Personen der Arbeitsumgebung die Anforderungen, Fakten und Zusicherungen erfüllen.

Im Informationsmodell wird vorgeschrieben, daß zu einer Anforderung, einer Zusicherung oder einem Fakt Testfälle angegeben werden können. Wird kein Testfall angegeben, liegt ein schwaches Indiz für einen Mangel vor.

Hierfür wird folgender Filter definiert:

Definition ($filter_{notestcase}$: $ReqDocument \rightarrow P(Requirement \cup Fact \cup Assertion)$): Dieser Filter ordnet einem Anforderungsdokument alle Anforderungen, Fakten und Zusicherungen zu, an die keine Testfälle gebunden sind:

$$filter_{notestcase}(doc) = \{c \in Requirement \cup Fact \cup Assertion|_{doc} \mid |tests(c)| = 0\}$$

Prädikat	Prüfverfahren
$filter_{notestcase}$: $ReqDocument \rightarrow P(Requirement \cup Assertion \cup Fact)$	Inspektion

Tabelle 34: Prüfverfahren für das Qualitätskriterium Überprüfbarkeit

8.2.10 Atomizität

Das Prädikat *atomar* ist erfüllt, wenn die Inhaltselemente, Glossareinträge und Quellenkatalogeinträge nicht sinnvoll in mehrere Einträge aufgespaltet werden können. Die Atomizitätsforderung kann auf viele verschiedene Weisen verletzt werden: Ein Inhaltselement könnte z.B. zwei Anforderungen oder eine Anforderung und eine Begründung enthalten. In dem Freitext einer Anforderung könnte ein neuer Begriff eingeführt und erklärt werden.

Ein Indiz dafür, daß ein Inhaltselement nicht atomar ist, liegt vor, wenn der Freitext Konjunktionen enthält. *Konjunktionen* sind Wörter (Phrasen), die Phrasen miteinander verknüpfen, z.B. »und«, »oder«, »entweder oder«, »wenn«, »aber«, »obwohl«. Konjunktionen sollten in Inhaltselementen sehr sparsam eingesetzt werden, weil durch sie Phrasen auf oftmals nicht triviale Weise zusammengefügt werden.

Wenn ein Inhaltselement *anreihende Konjunktionen* (»und«) enthält, so kann das bedeuten, daß eigentlich mehrere atomare Anforderungen, Fakten, Zusicherungen oder Probleme in ein Inhaltselement eingetragen wurden. Es sollte dann gesplittet werden. Natürlich kann es sinnvoll sein, in Bedingungen oder Aufzählungen anreihende Konjunktionen zu benutzen, ohne daß das Inhaltselement deswegen gesplittet werden sollte.

Durch *restriktive* (»aber«) oder *konzessive Konjunktionen* (»obwohl«) werden (scheinbare) Gegensätze ausgedrückt. Hierbei spielt die Meinung des Systemanalytikers eine wichtige Rolle. Inhaltselemente, die solche Konjunktionen enthalten, sollten in ein Inhaltselement und eine Annotation oder einen Mangeleintrag gesplittet werden.

Ein Beispiel: »Obwohl ein SESAM-Modell mehrere hundert Regeln enthalten kann, soll ein Regelauswertungszyklus nicht länger als 0,5s dauern«. Durch diesen Satz wird zweierlei zum Ausdruck gebracht:

1. Es müssen zwei Anforderungen erfüllt werden (Datenmenge und Antwortzeit).
2. Der Autor ist so erstaunt darüber, daß beide Anforderungen gleichzeitig erfüllt werden sollen, daß er sie als sich scheinbar widersprechend ansieht.

Der erste Aspekt kann klarer durch Benutzung der Konjunktion »und« formuliert werden oder besser durch Angabe zweier Anforderungen. Der zweite Aspekt gehört zwar nicht in den Anforderungstext, er enthält aber implizit eine wichtige Aussage: Dem Systemanalytiker sind Zweifel gekommen, ob beide Anforderungen gleichzeitig erfüllt werden können, also ob das Zielsystem »realisierbar« ist. Diese Zweifel können als Mangeleintrag dokumentiert werden.

Kausale Konjunktionen (»weil«) leiten eine Begründung ein. Sie sollten in den Freitexten (*contents*) von Inhaltselementen nicht auftauchen. Die Begründung sollte als Annotation zu dem Inhaltselement angegeben werden.

Konjunktionen	Phrasen
anreihend	und, wie, sowie, sowohl als/wie auch
disjunktiv	oder, entweder oder
restriktiv	aber, allein, nur, sondern, jedoch
kausal	da, denn, weil
konzessiv	gleichwohl, ob, obgleich, obschon, obwohl, obzwar, wenn auch, wenngleich, wennschon, wiewohl, trotzdem, ungeachtet

Tabelle 35: Konjunktionen der deutschen Sprache

Wenn also im Freitext eines Inhaltselements eine der o.g. Konjunktionen benutzt wird, dann liegt ein Indiz für einen Mangel vor. Damit die Benutzung von Konjunktionen in einem Freitext automatisch festgestellt werden kann, müssen die in Tabelle 35 genannten Konjunktionen (inklusive Flexionen) in die *Wortliste* eingetragen und dem *Worttyp wrongconjunction* oder *otherconjunction* zugeordnet werden (siehe Kapitel 6.10).

Definition (*WrongConjunction, OtherConjunction*): Die Menge *WrongConjunction* enthält alle restriktiven, kausalen und konzessiven Konjunktionen der Wortliste. Die Menge *OtherConjunction* enthält alle anreihenden und disjunktiven Konjunktionen der Wortliste. Gegeben sei eine Dokumentation I. Dann gilt:

$$WrongConjunction = \{p \in Phrase \mid \exists w \in WordEntry \mid_I \bullet ((term(w) = p) \wedge (wrongconjunction \in types(w)))\}$$

$$OtherConjunction = \{p \in Phrase \mid \exists w \in WordEntry \mid_I \bullet ((term(w) = p) \wedge (otherconjunction \in types(w)))\}$$

Die Benutzung von Konjunktionen kann durch eine Inhaltsprüfung oder eine negative fokussierte Inspektion festgestellt werden. Für die Inhaltsprüfung werden die Prädikate $conjunctionfree_{wrong}$ und $conjunctionfree_{other}$ definiert. Für die negative fokussierte Inspektion werden die Markierungsfunktionen $mark_{wrongconjunction}$ und $mark_{otherconjunction}$ definiert.

Definition ($conjunctionfree_{wrong}$: $ContElement \rightarrow Boolean$): Ein Inhaltselement ist frei von restriktiven, kausalen und konzessiven Konjunktionen, wenn folgende Bedingung erfüllt ist:

$$conjunctionfree_{wrong}(e) = \neg(\exists c \in WrongConjunction \bullet directuse_W(contents(e), c))$$

Die Zuverlässigkeit der Prüfergebnisse der Inhaltsprüfung ist hoch.

Definition ($conjunctionfree_{other}$: $ContElement \rightarrow Boolean$): Ein Inhaltselement ist frei von anreihenden und disjunktiven Konjunktionen, wenn folgende Bedingung erfüllt ist:

$$conjunctionfree_{other}(e) = \neg(\exists c \in OtherConjunction \bullet directuse_W(contents(e), c))$$

Die Zuverlässigkeit der Prüfergebnisse der Inhaltsprüfung ist niedrig, weil die Benutzung von anreihenden und disjunktiven Konjunktionen sinnvoll sein kann.

Definition ($mark_{wrongconjunction}$: $ContElement \rightarrow P(Word)$): Die Markierungsfunktion $mark_w$ (siehe S. 161) wird auf Inhaltselemente angewandt:

$$mark_{wrongconjunction}(c) = mark_W(contents(c), WrongConjunction)$$

Definition ($mark_{otherconjunction}$: $ContElement \rightarrow P(Word)$): Die Markierungsfunktion $mark_w$ wird auf Inhaltselemente angewandt:

$$mark_{otherconjunction}(c) = mark_W(contents(c), OtherConjunction)$$

Prädikat	Prüfverfahren
$conjunctionfree_{wrong}$: $ContElement \rightarrow Boolean$ $conjunctionfree_{other}$: $ContElement \rightarrow Boolean$	Inhaltsprüfung (hoch) Inhaltsprüfung (niedrig)
$mark_{wrongconjunction}$: $ContElement \rightarrow P(Word)$ $mark_{otherconjunction}$: $ContElement \rightarrow P(Word)$	negative fokussierte Inspektion negative fokussierte Inspektion

Tabelle 36: Prüfverfahren für das Qualitätskriterium Atomizität

8.2.11 Verständlichkeit

Das Prädikat *verständlich* ist auf allen Eintragstypen definiert, die Freitexte enthalten. Seine Erfüllung kann nicht automatisch überprüft werden. Im ADMIRE-Ansatz ist für die Prüfung auf Verständlichkeit deswegen nur die Inspektion vorgesehen. Sinnvolle Filter können nicht angegeben werden.

In Kapitel 10.2.4 werden einige Metriken zur Messung der Verständlichkeit vorgestellt und deren Integration in den ADMIRE-Ansatz diskutiert.

8.2.12 Eindeutigkeit

Das Prädikat *eindeutig* ist erfüllt, wenn es für einen Freitext unabhängig vom Leser genau eine Interpretation gibt. Mehrdeutigkeit kann auf verschiedene Weisen entstehen:

- durch Wörter und Phrasen, deren Bedeutung nicht klar definiert ist[1],
- durch syntaktisch mehrdeutige Satzkonstruktionen und
- durch semantisch mehrdeutige Satzkonstruktionen.

Im folgenden werden Prüfverfahren angegeben, um die ersten beiden Arten von Mehrdeutigkeiten aufzuspüren. Semantisch mehrdeutige Satzkonstruktionen können im ADMIRE-Ansatz nicht automatisch erkannt werden.

Benutzung von Modalverben

Probleme bereiten oft Modalverben. Modalverben werden benutzt, um zwischen den möglichen Modi einer Aussage zu unterscheiden. Es gibt u.a. folgende Modi:

- Indikativ zur Beschreibung von tatsächlichen Gegebenheiten, also Fakten.
- Optativ zur Beschreibung von Wünschen und Möglichkeiten, also Anforderungen und Zusicherungen.

Andere Modi, wie z.B. Imperativ und Konjunktiv, sollten in Anforderungsdokumenten nicht verwendet werden (Jackson, 1995, S. 125ff).

Im Deutschen gibt es die *Modalverben* »müssen«, »sollen«, »können«, »dürfen«, »wollen« und »mögen«. Im normalen Sprachgebrauch werden die Modalverben oft sehr ungenau verwendet. So wird z.B. zwischen »müssen« und »sollen« nicht klar unterschieden. Bei der Formulierung des Freitextes eines Inhaltselements sollte eine gewisse Disziplin eingehalten werden, um Mehrdeutigkeiten zu vermeiden:

- Anforderungen, Zusicherungen und Probleme sollen durch die Modalverben »sollen«, »nicht dürfen« oder den Indikativ formuliert werden. Die Unterscheidung zwischen »müssen« als harte Forderung und »sollen« als nicht ganz so harte Forderung eignet sich nicht, um daraus Prioritäten für Anforderungen abzuleiten. Der Unterschied ist zu fein. Das Informationsmodell bietet für diese Unterscheidung andere Möglichkeiten (siehe Kapitel 6.6).
- Fakten sollen durch den Indikativ formuliert werden. Es sollen also keine Modalverben benutzt werden.

Definition (*ModalVerb*, $ModalVerb_{Anf}$): Die Menge *ModalVerb* enthält alle Modalverben (inklusive Flexionen) der Wortliste. Die Menge $ModalVerb_{Anf}$ enthält die Modalverben »können«, »müssen«, »wollen« und »mögen« (inklusive Flexionen). Gegeben sei eine Dokumentation I. Dann gilt:

$$ModalVerb = \{p \in Phrase \mid \exists w \in WordEntry|_I \bullet ((term(w) = p) \wedge (modalverb \in types(w)))\}$$

$$ModalVerb_{Anf} = \{p \in Phrase \mid \exists w \in WordEntry|_I \bullet ((term(w) = p) \wedge (modalverb_{anf} \in types(w)))\}$$

---

1. Prüfungen, die den Abgleich zwischen Freitexten und Glossar betreffen, werden in Kapitel 8.2.4 (Konsistenz) beschrieben.

Damit die Benutzung von Modalverben durch Inhaltsprüfung oder negative fokussierte Inspektion überprüft werden kann, müssen die Modalverben inklusive ihrer Flexionen in die *Wortliste* eingetragen und dem *Worttyp* $modalverb_{anf}$ oder *modalverb* zugeordnet werden (siehe Kapitel 6.10).

Für die Inhaltsprüfung wird das Prädikat *rightmode* definiert und für die negative fokussierte Inspektion die Markierungsfunktion $mark_{modalverb}$.

Definition (*rightmode: ContElement* → *Boolean*): Der Freitext eines Inhaltselements ist im richtigen Modus geschrieben, wenn folgende Bedingung erfüllt ist:

$$rightmode(c) = f_1(c) \wedge f_2(c)$$

mit:

1. In dem Freitext eines Inhaltselements der Kategorie »Anforderung«, »Zusicherung« oder »Problem« dürfen die Modalverben »können«, »müssen«, »wollen« und »mögen« nicht benutzt werden.
$$f_1(c) = (c \in Requirement \cup Problem \cup Assertion \Rightarrow (\forall m \in ModalVerb_{Anf} \bullet (directuse_W(contents(c), m) = false)))$$

2. In dem Freitext eines Fakts dürfen keine Modalverben benutzt werden.
$$f_2(c) = (c \in Fact \Rightarrow (\forall m \in ModalVerb \bullet (directuse_W(contents(c), m) = false)))$$

Definition ($mark_{modalverb}$: *Requirement* ∪ *Assertion* ∪ *Problem* → *P(Word)*): Die Markierungsfunktion $mark_w$ (siehe S. 161) wird auf Anforderungen, Zusicherungen und Probleme angewandt:

$$mark_{modalverb}(e) = mark_W(contents(e), ModalVerb_{Anf})$$

Definition ($mark_{modalverb}$: *Fact* → *P(Word)*): Die Markierungsfunktion $mark_w$ wird auf Fakten angewandt:

$$mark_{modalverb}(e) = mark_W(contents(e), ModalVerb)$$

Die Zuverlässigkeit der Prüfergebnisse der Inhaltsprüfung ist niedrig.

Benutzung von Indikatoren

Auch wenn jedes einzelne Wort eindeutig ist, so können Sätze oder Abschnitte trotzdem mehrdeutig sein, z.B. wenn Indikatoren benutzt werden. Bei *Indikatoren* handelt es sich um Wörter, die nur in einem bestimmten Kontext Sinn ergeben. Beispiele für Indikatoren sind Personal-, Possessiv- und Demonstrativpronomen sowie einige Adverben des Ortes und der Zeit (siehe Tabelle 37). Wenn der Kontext, in dem das Wort benutzt wird, nicht eindeutig ist, können Mißverständnisse entstehen. Oft ist es besser, statt Indikatoren absolute Aussagen zu benutzen, z.B. statt »heute« sollte »09.03.1998« angegeben werden.

Definition (*Indicator*): Die Menge *Indicator* enthält alle Indikatoren der Wortliste. Gegeben sei eine Dokumentation *I*. Dann gilt:

$$Indicator = \{p \in Phrase \mid \exists w \in WordEntry|_I \bullet ((term(w) = p) \wedge (indicator \in types(w)))\}$$

Wortart	Wörter (ohne Flexionen)
Personalpronomen	ich, du, er, sie, es, wir, ihr, mich, dich, ihm, ihn, ihr, ihnen, ihrer, euch
Possessivpronomen	mein, dein, sein, ihr, uns, euer
Demonstrativ-pronomen	dieser, der, die das, dessen, dem, den, deren, denen, jener, derjenige, derselbe, selber, selbst
Relativpronomen	der, welcher
Adverben des Ortes	da, dort, dorthin, dorther, hier, hierher, hierhin
Adverben der Zeit	bald, damals, danach, dann, darauf, demnächst, dereinst, derzeit, eben, ehedem, ehemals, einmal, einst, einstmals, endlich, erst, fortan, gerade, gestern, gleich, hernach, heutigentags, heutzutage, hierauf, hinterher, indes, indessen, inzwischen, jetzt, kürzlich, längst, letztens, morgen, nachher, neulich, nun, nächstens, schließlich, seinerzeit, soeben, sofort, sogleich, übermorgen, unlängst, unterdessen, vorher, vorhin, zuerst, zuletzt, zurzeit, zuvor, zwischendurch

Tabelle 37: Wichtige Indikatoren der deutschen Sprache

Wenn in einem Satz Indikatoren benutzt werden, so kann das als Indiz für Mehrdeutigkeiten gewertet werden. Das Indiz ist sehr schwach, weil in vielen Fällen die Benutzung von Indikatoren sinnvoll ist. Werden Indikatoren um jeden Preis vermieden, so geht das u.U. sehr stark zu Lasten der Verständlichkeit.

Damit die Benutzung von Indikatoren automatisch durch eine Inhaltsprüfung oder eine negative fokussierte Inspektion überprüft werden kann, müssen die in Tabelle 37 genannten Indikatoren inklusive ihrer Flexionen in die *Wortliste* eingetragen und dem *Worttyp indicator* zugeordnet werden (siehe Kapitel 6.10). Für die Inhaltsprüfung wird das Prädikat *indicatorfree* und für die negative fokussierte Inspektion die Markierungsfunktion $mark_{indicator}$ definiert.

Definition (*indicatorfree: NLEntry* → *Boolean*): Ein NL-Eintrag ist frei von Indikatoren, wenn folgende Bedingung erfüllt ist:

$$indicatorfree(e) = \neg(\exists i \in Indicator \bullet \exists p \in paragraphs(e) \bullet directuse_W(p, i))$$

Definition ($mark_{indicator}$: *NLEntry* → *P(Paragraph* × *P(Word))*): Die Markierungsfunktion $mark_W$ (siehe S. 161) wird auf alle Freitexte des NL-Eintrags angewandt (siehe auch Fußnote auf S. 175):

$$mark_{indicator}(e) = \bigcup_{p \in paragraphs(e)} (p, mark_W(p, Indicator))$$

Die Zuverlässigkeit der Prüfergebnisse der Inhaltsprüfung ist niedrig.

Prädikat	Prüfverfahren
rightmode: ContElement → *Boolean*	Inhaltsprüfung (niedrig)
$mark_{modalverb}$: *Fact* → *P(Word)* $mark_{modalverb}$: *Requirement*∪*Assertion*∪*Problem* → *P(Word)*	neg. fokussierte Inspektion
indicatorfree: NLEntry → *Boolean*	Inhaltsprüfung (niedrig)
$mark_{indicator}$: *NLEntry* → *P(Paragraph* × *P(Word))*	neg. fokussierte Inspektion

Tabelle 38: Prüfverfahren für das Qualitätskriterium Eindeutigkeit

8.2.13 Präzision

Das Prädikat *präzise* ist erfüllt, wenn die Freitexte keine vagen Aussagen enthalten. In vielen Fällen lassen sich unpräzise Aussagen in einem Inhaltselement an der Benutzung bestimmter Wörter, sog. »WeakWords«, festmachen. Tabelle 39 enthält eine Aufzählung wichtiger, in der deutschen Sprache vorkommender WeakWords (entnommen aus Duden (1998)).

Unpräzise Wörter und Phrasen
aber, allzu, ab und zu, absolut, andere, äußerst, auch, außerordentlich, bald, bei weitem, beinahe, besonders, besser, beste, bestimmt, bisweilen, ca., damals, dann und wann, demnächst, denkbar, denn, dereinst, doch, durchaus, eben, ehedem, ehemals, eher, ein bißchen, ein paar, ein wenig, eilends, einfach, einige, einigermaßen, einmal, einst, einstmals, einzeln, enorm, erstaunlich, etliche, etwa, etwas, etwelche, fabelhaft, fast, furchtbar, ganz, gar, genau, genug, gering, gesamt, gewaltig, gewiß, größtenteils, groß, gut, häufig, halbwegs, halt, hin und wieder, höchst, hoffentlich, inzwischen, irgend, irgendetwas, irgendwelche, irgendwer, irgendwas, irgendwo, irgendwoher, irgendwohin, irgendwie, ja, kaum, klein, kolossal, kürzlich, längst, langsam, letztens, leicht, mäßig, mal, man, manchmal, manche, mehr, mehrere, mehrfach, mehrmals, meist, meistens, minder, mitunter, möglich, möglichst, möglicherweise, nach Möglichkeit, nahezu, neulich, nur, oft, öfter, öfters, phantastisch, riesig, rund, schätzungsweise, schlecht, schnell, schon, schrecklich, schwer, schwerlich, sehr, sicher, sicherlich, solche, sonstige, teils, teilweise, u.a., überaus, überhaupt, übrige, ungemein, ungewöhnlich, ungezählt, unlängst, unterdessen, usw., verblüffend, vereinzelt, verschieden, viel, vielfach, vielleicht, vielmal, vollendet, vollkommen, vorhin, vorerst, wahnsinnig, wenig, weitaus, weitere, winzig, wirklich, wohl, womöglich, wunders wie, zahlreich, zahllos, ziemlich, zirka, zunächst, zeitweise, zu meist, zuweilen

Tabelle 39: Wichtige WeakWords der deutschen Sprache

Definition (*WeakWord*): Die Menge *WeakWord* enthält alle unpräzisen Wörter und Phrasen der Wortliste. Gegeben sei eine Dokumentation I. Dann gilt:

$$WeakWord = \{p \in Phrase \mid \exists w \in WordEntry|_I \bullet ((term(w) = p) \wedge (weakword \in types(w)))\}$$

Wenn ein WeakWord in dem Freitext eines Inhaltselements benutzt wird, ist der Freitext mit hoher Wahrscheinlichkeit unpräzise. Damit die Benutzung von WeakWords automatisch durch eine Inhaltsprüfung oder eine negative fokussierte Inspektion überprüft werden kann, müssen die in Tabelle 39 genannten WeakWords (inklusive Flexionen) in die *Wortliste* eingetragen und dem *Worttyp weakword* zugeordnet werden (siehe Kapitel 6.10).

Für die Inhaltsprüfung wird das Prädikat *weakwordfree* und für die negative fokussierte Inspektion die Markierungsfunktion $mark_{weakword}$ definiert.

Definition (*weakwordfree: ContElement* $\rightarrow$ *Boolean*): Ein Inhaltselement ist frei von unpräzisen Wörtern, wenn folgende Bedingung erfüllt ist:

$$weakwordfree(c) = \neg(\exists w \in WeakWord \bullet directuse_W(contents(c), w))$$

Die Zuverlässigkeit der Prüfergebnisse der Inhaltsprüfung ist hoch.

Definition ($mark_{weakword}$: *ContElement* $\rightarrow$ *P(Word)*): Die Markierungsfunktion $mark_w$ wird auf Inhaltselemente angewandt:

$$mark_{weakword}(c) = mark_W(contents(c), WeakWord)$$

Prädikat	Prüfverfahren
weakwordfree: ContElement → Boolean	Inhaltsprüfung (hoch)
$mark_{weakword}$: *ContElement → P(Word)*	negative fokussierte Inspektion

Tabelle 40: Prüfverfahren für das Qualitätskriterium Präzision

8.2.14 Minimalität

Das Prädikat *minimal* ist erfüllt, wenn die Freitexte eines NL-Eintrags keine unnötigen oder kürzer formulierbaren Teile (z.B. Füllwörter) enthalten und wenn die Anforderungsdokumente keine unnötigen Einträge enthalten.

Typische Füllwörter sind z.B.: »aber«, »auch«, »denn«, »doch«, »eben«, »etwa«, »halt«, »ja«, »mal«, »nur«, »schon«, »vielleicht« oder »wohl«. Die Benutzung von Füllwörtern kann durch eine negative fokussierte Inspektion oder durch eine Inhaltsprüfung überprüft werden. Da die Füllwörter aber auch zu den WeakWords (Tabelle 39) gehören, werden hier keine zusätzlichen Prüfungen eingeführt.

Alle anderen Minimalitätseigenschaften können nicht automatisch überprüft werden. Die Erfüllung des Minimalitätsprädikats kann also nur durch Inspektion geprüft werden.

8.2.15 Normkonformität

Das Prädikat *normkonform* ist erfüllt, wenn es eine Vorlage gibt, dessen Vorlagen-Kapitelhierarchie mit der Kapitelhierarchie des Anforderungsdokuments übereinstimmt. Dabei muß beachtet werden, daß ein Anforderungsdokument zusätzliche Kapitel enthalten darf: Jedes Kapitel darf am Ende weitere Unterkapitel enthalten. Ebenso dürfen die durch die Vorlagen-Kapitelhierarchie vorgegebenen Blattkapitel weiter verfeinert werden. Folgende Aspekte müssen geprüft werden:

- Zu jedem Kapitel der Vorlagen-Kapitelhierarchie muß es ein Kapitel der Kapitelhierarchie des Anforderungsdokuments geben, das die gleiche Überschrift hat und an der gleichen Stelle eingeordnet ist.
- Ein Kapitel der einen Hierarchie und ein Kapitel der anderen Hierarchie sind an der gleichen Stelle eingeordnet, wenn die logischen Positionen (*position*, siehe Kapitel 6.8 und 6.10) der Kapitel und aller Oberkapitel jeweils übereinstimmen.

Hierfür werden folgende Funktionen definiert:

Definition (*template: ReqDocument → Template*): Die Funktion *template* ordnet einem Anforderungsdokument die zu seiner Art passende Vorlage zu:

$$template(doc) = \begin{cases} cstemplate(Metaspecification|_{model(doc)}), \text{ falls } doc \in CurrentState \\ rctemplate(Metaspecification|_{model(doc)}), \text{ falls } doc \in ReqCollection \\ rstemplate(Metaspecification|_{model(doc)}), \text{ falls } doc \in ReqSpecification \\ sptemplate(Metaspecification|_{model(doc)}), \text{ falls } doc \in Specification \end{cases}$$

Definition (*superchapters: Chapter → P(Chapter), superchapters: TChapter → P(TChapter)*): Die Funktion *superchapters* ordnet einem (Vorlagen-)Kapitel seine Ober-(Vorlagen-) Kapitel zu:

$$superchapters(c) = \begin{cases} \{sc \in Chapter \mid c \in subchapters(sc)\}, \text{ falls } c \in Chapter \\ \{tsc \in TChapter \mid c \in tsubchapters(tsc)\}, \text{ falls } c \in TChapter \end{cases}$$

Definition (*level: Chapter ∪ TChapter → N*): Die Funktion *level* ordnet einem Kapitel bzw. Vorlagen-Kapitel seine Tiefe in der (Vorlagen-) Kapitelhierarchie zu:

$$level(c) = |superchapters(c)|$$

Definition (*match: Chapter × TChapter → Boolean*): Das Prädikat *match* prüft, ob ein Kapitel und ein Vorlagen-Kapitel zusammenpassen. Dafür müssen die Überschriften des Kapitels und des Vorlagen-Kapitels übereinstimmen sowie die Paare aus logischer Position und Tiefe in der (Vorlagen-)Kapitelhierarchie für alle Ober-(Vorlagen-)Kapitel:

$$match(c, tc) = (heading(c) = heading(tc)) \wedge (\{(position(sc), level(sc)) \mid sc \in superchapters(c)\} = \{(position(tsc), level(tsc)) \mid tsc \in superchapters(tc)\})$$

Jetzt kann das Prädikat *structuralconform* definiert werden:

Definition (*structuralconform: ReqDocument → Boolean*): Ein Anforderungsdokument ist strukturell konform zu seiner Vorlage, wenn folgende Bedingung erfüllt ist:

$$structuralconform(doc) = (\exists t \in template(doc) \bullet (\forall tc \in TChapter|_t \bullet \exists c \in Chapter|_{doc} \bullet match(c, tc)))$$

Die Erfüllung dieses Prädikats kann durch eine Inhaltsprüfung überprüft werden. Die Zuverlässigkeit der Prüfergebnisse ist hoch.

Prädikat	Prüfverfahren
structuralconform: *ReqDocument* → *Boolean*	Inhaltsprüfung (hoch)

Tabelle 41: Prüfverfahren für das Qualitätskriterium Normkonformität

Kapitel 9

ADMIRE - Werkzeug

In diesem Kapitel wird das ADMIRE-Werkzeug vorgestellt, das Dokumentationen verwalten kann, die Tätigkeiten des ADMIRE-Prozeßmodells unterstützt und insbesondere die beschriebenen Prüfverfahren realisiert. Neben einem allgemeinen Überblick über die wichtigsten Eigenschaften und Funktionen (Kapitel 9.1) und einigen Informationen zur Realisierung (Kapitel 9.2) werden fünf typische Szenarien beschrieben, die zeigen, wie das Werkzeug in einem Requirements-Engineering-Prozeß eingesetzt werden kann (Kapitel 9.3).

9.1 Überblick

9.1.1 Arbeitsumgebung

An einem dem ADMIRE-Prozeßmodell folgenden Requirements-Engineering-Prozeß sind Personen beteiligt, die typischerweise viele verschiedene Rollen einnehmen. Aus Sicht des ADMIRE-Werkzeugs werden drei Rollen unterschieden (Abbildung 47):

- die Systemanalytiker, die hauptverantwortlich die Requirements-Engineering-Tätigkeiten durchführen. Sie benutzen das Werkzeug, um Dokumentationen zu erstellen, zu bearbeiten und zu verwalten. Sie können über die »*GUI*«-Schnittstelle auf Teile der Dokumentation lesend und schreibend zugreifen.

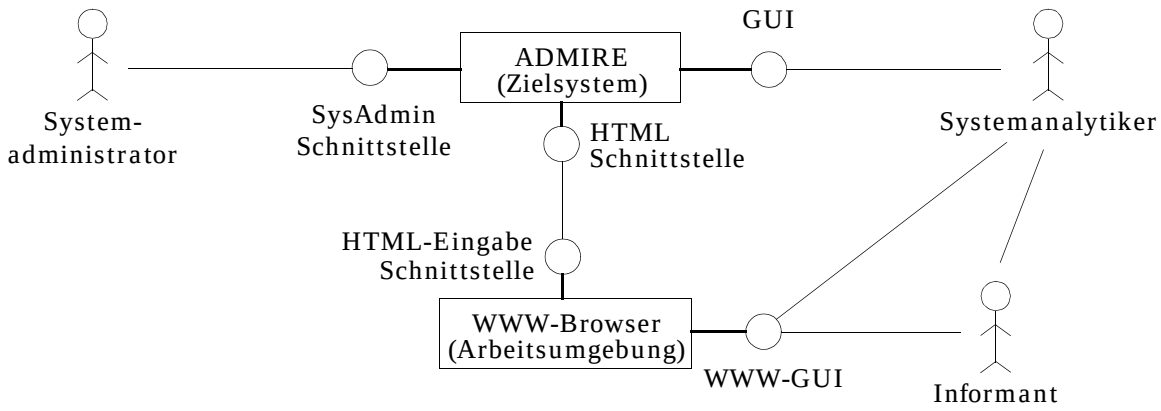


Abbildung 47: Systemmodell des ADMIRE-Werkzeugs
(Legende: siehe Beispiel 6, S. 90)

- die Informanten, von denen Informationen erhoben werden. Sie sollen Teile der Dokumentation validieren, müssen in bestimmten Situationen an der Lösung von Konflikten beteiligt werden oder sollen das Zielsystem realisieren. Sie haben keinen direkten Zugriff auf das ADMIRE-Werkzeug. Die Interaktion erfolgt immer indirekt über die Systemanalytiker. Sollen sie die Dokumentation oder Teile der Dokumentation prüfen, generieren die Systemanalytiker Reporte (HTML-Dateien), die die Informanten mit einem WWW-Browser anschauen oder ausdrucken können. Die bei einer Informationserhebung oder Prüfung erstellten Protokolle werden den Systemanalytikern ausgehändigt, die dann die notwendigen Änderungen an der Dokumentation durchführen. Dadurch soll sichergestellt werden, daß Änderungen nur kontrolliert erfolgen.
- die Systemadministratoren, die über die »*SysAdmin-Schnittstelle*« direkt auf den internen Zustand des ADMIRE-Werkzeugs zugreifen und ihn beliebig ändern können.[1] Sie treten in einem »normalen« Prozeß nicht in Erscheinung. Sie sollen nur dann eingreifen, wenn unerwartete Schwierigkeiten mit dem ADMIRE-Werkzeug auftreten.

9.1.2 Wichtige Funktionen

Im ADMIRE-Werkzeug sind Funktionen zur Eingabe, Ausgabe, Änderung, Verwaltung und Prüfung von Dokumentationen realisiert. Auf einzelnen Einträgen können folgende Aktionen durchgeführt werden:

- Eingabe eines neuen Eintrags,
- Änderung eines Eintrags und
- Löschen eines Eintrags.

Für jede Eintragsart stellt das GUI ein Fenster zur Verfügung, in das alle durch das Informationsmodell vorgegebenen Informationen eingegeben werden können. Bevor ein Eintrag aber tatsächlich in die Dokumentation aufgenommen wird, muß der Systemanalytiker die Eingabe bestätigen. Daraufhin prüft das Werkzeug, ob der neue Eintrag den formalen Bedingungen des Informationsmodells genügt. Wenn nicht, wird die Eingabe nicht akzeptiert. Auf diese Weise wird sichergestellt, daß eine Dokumentation zu jedem Zeitpunkt alle Bedingungen des Informationsmodells erfüllt. Die Qualitätskriterien (Kapitel 5.10) werden hierbei nicht berücksichtigt. Analog können Einträge geändert werden: Der zu ändernde Eintrag wird in dem zur Eintragsart passenden Fenster dargestellt. Der Systemanalytiker kann beliebig Änderungen vornehmen. Die Dokumentation selbst wird erst geändert, wenn die Änderung bestätigt wird und wenn alle Bedingungen des Informationsmodells erfüllt sind.

Bei der Eingabe oder Änderung eines Eintrags werden die in Kapitel 5.11 beschriebenen Attribute automatisch vorbelegt.

Das ADMIRE-Werkzeug verhindert, daß Einträge gelöscht werden, wenn von anderen Einträgen auf sie verwiesen wird. Der Systemanalytiker muß die Querverweise vor dem Löschen entfernen. Eine Ausnahme hierbei bildet die *contains*-Kante.

---

1. Die Bedingungen des Informationsmodells werden hierbei nicht überprüft. Deswegen sollten solche Änderungen nur von Personen durchgeführt werden, die sich gut mit dem Informationsmodell und dem ADMIRE-Werkzeug auskennen.

Wenn ein Eintrag *e* über die *contains* Kante mit einem Eintrag *c* verbunden ist ($e \in contains(c)$) und keinerlei sonstige Beziehungen zu anderen Einträgen hat, dann darf *e* gelöscht werden. Ansonsten könnte praktisch kein Eintrag jemals wieder gelöscht werden, weil fast alle Einträge direkt oder indirekt über *contains*-Kanten mit der Wurzel der Dokumentation (*model*) verbunden sind. Textuelle Querverweise werden beim Löschen ebenfalls nicht berücksichtigt.

Das ADMIRE-Werkzeug unterstützt die Validierung und Verifikation von Dokumentationen durch die in den Kapiteln 8.2.1 - 8.2.15 beschriebenen Prüfverfahren. Der Systemanalytiker kann einstellen, auf welche Einträge die Prüfkriterien der Inhaltsprüfungen, die Anzeigefunktionen der Statusprüfungen, die Markierungsfunktionen der fokussierten Inspektionen und die Ähnlichkeitssuche angewandt werden sollen. Das ADMIRE-Werkzeug bietet darüber hinaus die Möglichkeit, *Sichten* zu generieren. Sichten entstehen dadurch, daß auf eine Dokumentation Filter angewandt werden.

Desweiteren verfügt das ADMIRE-Werkzeug über einen Reportgenerator. Dokumente oder Sichten können als Dateien im HTML-Format abgespeichert und mit einem WWW-Browser angezeigt und ausgedruckt werden. Bei der Generierung werden vielfältige Arten von Hyperlinks in den HTML-Code eingetragen, z.B. von einem Freitext zu den benutzten Glossareinträgen oder von einem Kapitel zu seinem Oberkapitel.

9.1.3 Benutzungsschnittstelle

Das ADMIRE-Werkzeug verfügt über drei Schnittstellen: das »GUI« für die Systemanalytiker, die »HTML-Schnittstelle« zur Generierung von Reporten und die »SysAdmin-Schnittstelle« für die Systemadministratoren.

Die HTML-Schnittstelle ist eine Dateischnittstelle. Alle über diese Schnittstelle erzeugten Dateien liegen im HTML-Format (Ragget, 1997) vor.

Bei der »SysAdmin-Schnittstelle« handelt es sich um eine textuelle Schnittstelle, über die der Systemadministrator direkt auf die zugrundeliegende Datenbank zugreifen kann. Die Schnittstelle wird in (Jepson, Hughes, 1998) beschrieben.

Die Fenster des »GUI« werden im folgenden genauer beschrieben.

Alle Aktionen, die der Systemanalytiker durchführen kann, werden von einem Fenster des »GUI« aus gestartet. Folgende Fenster bietet das GUI an:

Abbildung 48: Das Hauptfenster des ADMIRE-Werkzeugs

- das *Hauptfenster* (Abbildung 48, S. 193). Es enthält das *Hauptmenü* (Tabelle 42, S. 194), von dem aus folgende Aktionen angestoßen werden können: Der Systemanalytiker kann eine neue Dokumentation anlegen, eine zuvor gespeicherte Dokumentation laden, die aktuelle Dokumentation speichern, Dokumente und Sichten als HTML-Dokumente exportieren, Einträge suchen, Filter definieren und anwenden, das Werkzeug verlassen, Anforderungsdokumente, Glossareinträge, Mangeleinträge, Annotationen oder Quellenkatalogeinträge bearbeiten, die Prüfungen konfigurieren und starten, die Metaspezifikation anzeigen oder eine geänderte Metaspezifikation neu laden[1].

Dokumentation	Bearbeiten	Prüfung	Meta-spezifikation
Neu	Anforderungs-dokumente	Einstellungen	Anzeigen
Laden	Glossar	Prüfung starten	Laden
Speichern	Mängelkatalog		
Exportieren	Annotationen		
Suche	Quellenkatalog		
Filter			
Ende			

Tabelle 42: Struktur des Hauptmenüs

- das *Suchfenster* (Abbildung 61, S. 206), in dem Einträge gesucht werden können, deren eindeutiger Bezeichner bekannt ist.
- das *Sichtenfenster*, in dem Filter definiert und angewandt werden können.
- das *Dokumentfenster* (Abbildung 49, S. 195), in dem Anforderungsdokumente, Kapitel und Inhaltselemente verwaltet werden können. Der Auswahlbaum auf der linken Seite zeigt die Anforderungsdokumente und ihre Kapitelhierarchien. Auf der rechten Seite wird entweder das aktuell im Baum ausgewählte Anforderungsdokument, das ausgewählte Kapitel oder ein in dem Kapitel enthaltenes Inhaltselement angezeigt. Vom Dokumentfenster aus kann das *Systemmodellfenster* (Abbildung 50, S. 195) aufgerufen werden, das einen graphischen Editor für Systemmodelle enthält.
- das *Glossarfenster* (Abbildung 51, S. 196), in dem Glossareinträge verwaltet werden können. In der Auswahlliste auf der linken Seite werden die Terme der vorhandenen Glossareinträge dargestellt. Es werden nur solche Terme angezeigt, die den Bedingungen des Anzeigefilters (unten links) genügen. Rechts wird der aktuell ausgewählte Glossareintrag dargestellt. Von diesem Fenster

1. Zum Zeitpunkt der Abgabe dieser Dissertation gibt es keine Fenster zur Änderung der Metaspezifikation. Die Daten liegen in Form von Konfigurationsdateien vor, die mit einem ASCII-Editor bearbeitet und anschließend neu geladen werden können. Weil die Konfigurationsdateien (zukünftig) durch GUI-Fenster ersetzt werden sollen, werden sie im Systemmodell (Abbildung 47, S. 191) nicht als Schnittstelle dargestellt.

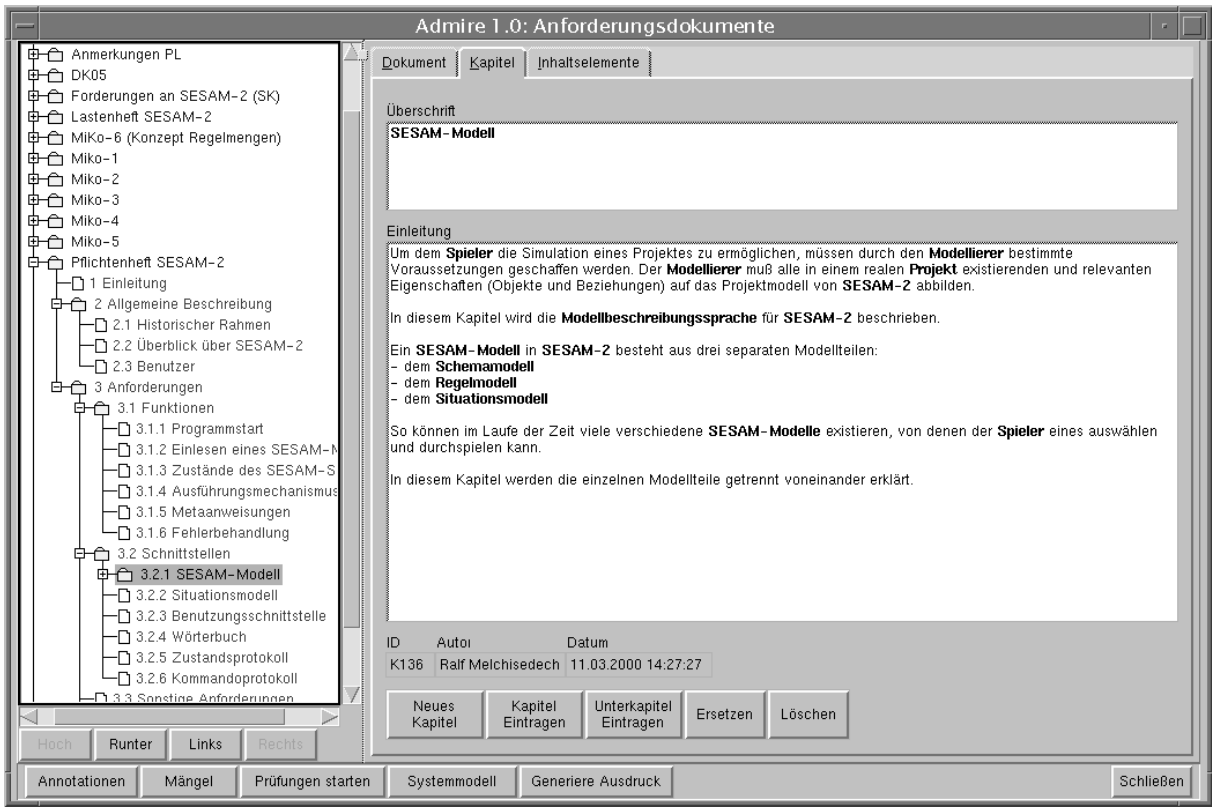


Abbildung 49: Dokumentfenster

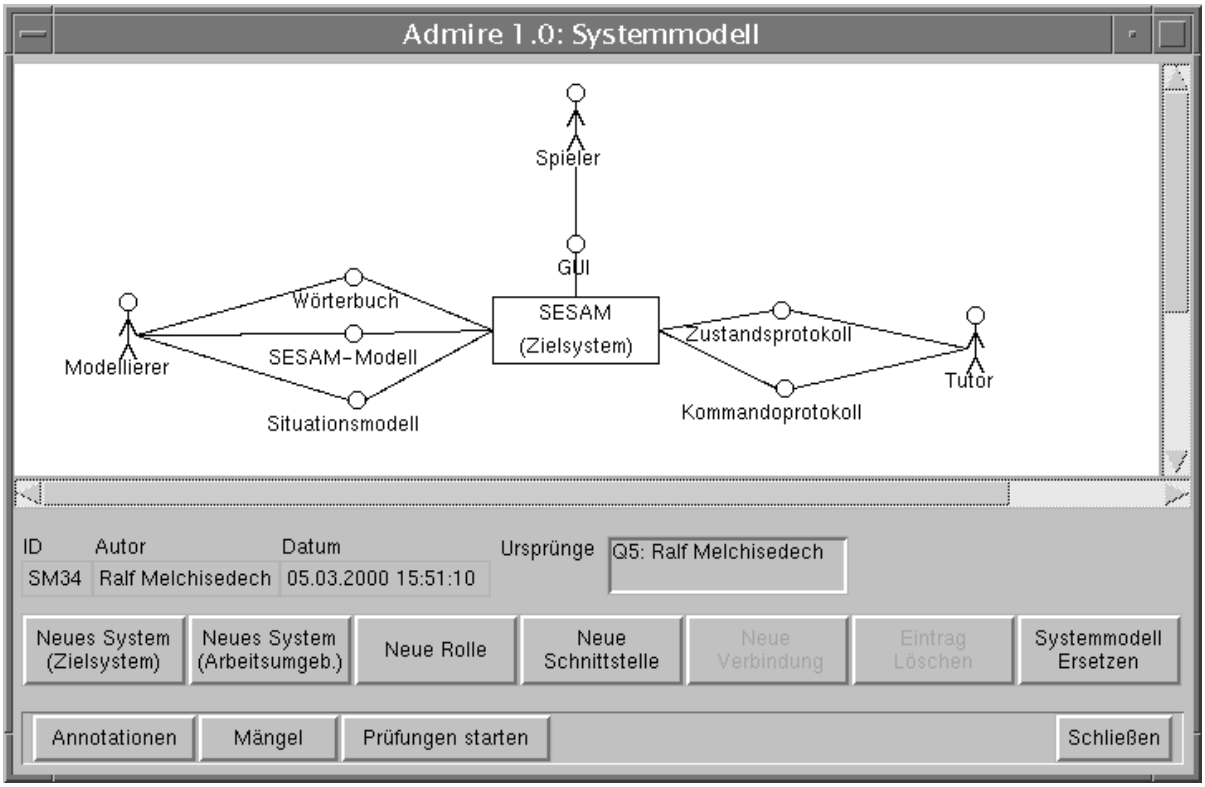


Abbildung 50: Systemmodellfenster

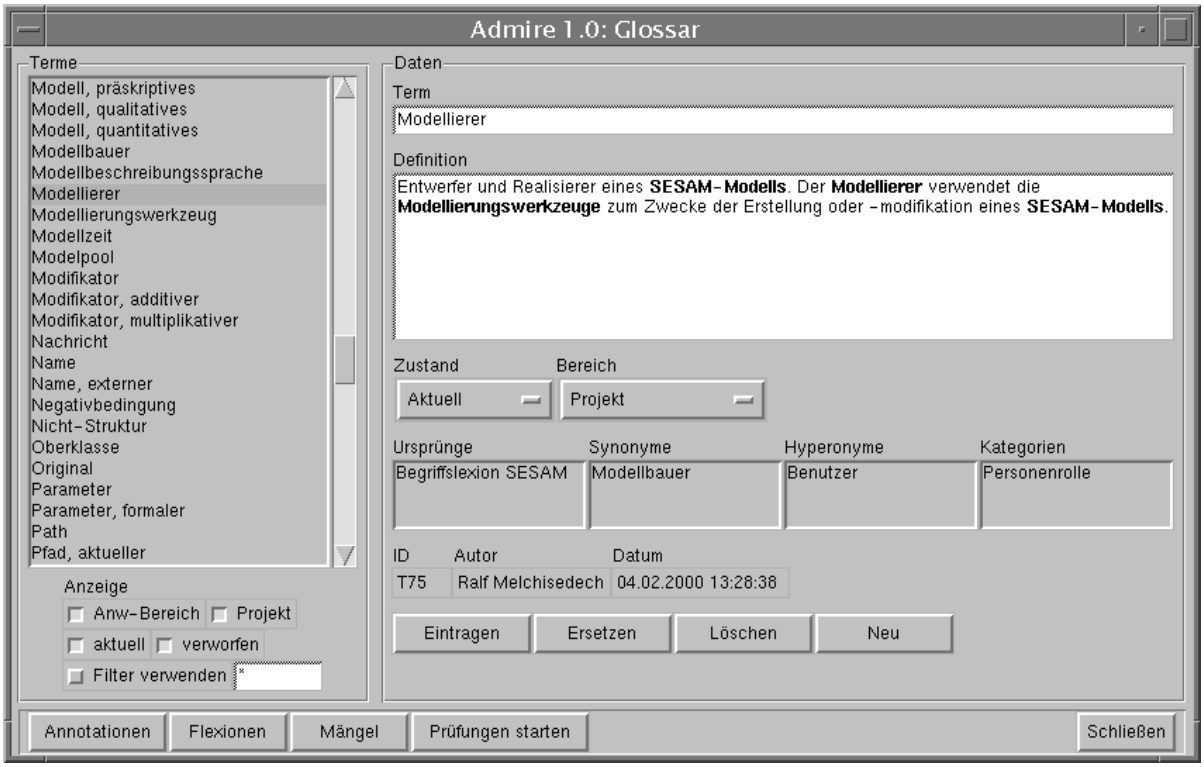


Abbildung 51: Glossarfenster

aus kann der *Flexionseditor* aufgerufen werden. In dem Flexionseditor können die Flexionen für einen Term bzw. für die Wörter des Terms verwaltet werden.

- das *Mängelkatalogfenster* (Abbildung 57, S. 202), in dem der Mängelkatalog verwaltet werden kann.
- das *Annotationsfenster*, in dem Annotationen verwaltet werden können.
- das *Quellenkatalogfenster*, in dem der Quellenkatalog verwaltet werden kann.
- das *Konfigurationsfenster* (Abbildung 58, S. 203), in dem eingestellt werden kann, welche Prüfungen auf welchen Einträgen durchgeführt werden sollen.
- das *Prüfungsfenster* (Abbildung 56a, S. 201), von dem aus die Inhaltsprüfungen, die Statusprüfungen und die Ähnlichkeitssuche gestartet werden können.
- das *Metaspezifikationsfenster*, in dem die Einträge der Metaspezifikation angezeigt werden.

9.2 Realisierungsaspekte

Diplomarbeiten

Eine frühe Version des ADMIRE-Werkzeugs wurde von Uwe Mindrup (1998) und Martin Richter (1998) implementiert. In der Diplomarbeit von Uwe Mindrup entstanden das Grundsystem zur Eingabe und Verwaltung der Dokumentationen und ein einfacher Sichtendialog. In der Diplomarbeit von Martin Richter wurden die ersten Prüfverfahren realisiert.

Implementierung

Die Implementierung erfolgte in Tcl/Tk (Ousterhout, 1994), einer frei verfügbaren Skriptsprache, die sich besonders zur Verarbeitung von Zeichenketten und zur Definition graphischer Oberflächen eignet. Ursprünglich war geplant, nur die Dialoge in Tcl/Tk zu realisieren, die restliche Funktionalität in Ada 95. Da Tcl/Tk aber ein tragfähiges Modulkonzept bietet, sich hervorragend zur Manipulation von Zeichenketten (d.h. Wörtern, Phrasen und Freitexten) eignet und für das ADMIRE-Werkzeug ausreichend effizient ist, wurde die Entscheidung, eine zusätzliche »konventionelle« Programmiersprache einzusetzen, fallengelassen.

Zur permanenten Datenhaltung wurde die ebenfalls frei verfügbare, (eingeschränkt) relationale Datenbank MSQL 2.0 (Jepson, Hughes, 1998) verwendet.

Das ADMIRE-Werkzeug läuft momentan auf Sun-Workstations unter dem Betriebssystem Solaris 2.6 und auf PCs unter Linux. Da Tcl/Tk für viele gängige Plattformen (Unix, Windows, Mac) verfügbar ist und MSQL im Quellcode vorliegt, sind Portierungen auf andere Unix-Plattformen (in der Regel) problemlos möglich. Portierungen nach Windows oder Mac-OS sind aufwendiger, weil für diese Plattformen eine andere Datenbank verwendet werden müßte. Da MSQL jedoch über eine SQL-ähnliche Schnittstelle verfügt, müßte die Anbindung an eine andere Datenbank mit relativ geringem Aufwand möglich sein.

In der folgenden Tabelle werden einige wichtige Kennzahlen für die Implementierung angegeben:

LOC (Tcl/Tk-Code, inkl. Kommentaren und Leerzeilen)	35741
reine Kommentarzeilen	10057
Leerzeilen	5956
LOC (Tcl/Tk-Code, ohne Kommentare und Leerzeilen)	19728

Tabelle 43: Kennzahlen für die Implementierung

9.3 Typische Benutzungsszenarien

Im folgenden wird am Beispiel von fünf Szenarien (oder Use Cases) gezeigt, wie das ADMIRE-Werkzeug den Systemanalytiker bei seinen Tätigkeiten unterstützen kann. Hierbei steht die direkte Interaktion zwischen Systemanalytiker und ADMIRE-Werkzeug im Vordergrund.

In den Beispielen werden Auszüge aus der SESAM-2-Dokumentation (Kapitel 4.6) verwendet.

9.3.1 Szenario 1: »Eingabe und Prüfung einer Anforderung«

In diesem Szenario wird gezeigt, wie der Systemanalytiker eine Anforderung eingibt und dabei durch die automatischen Markierungen der fokussierten Inspektionen und durch die Inhaltsprüfungen unterstützt wird. Das ADMIRE-Werkzeug muß gestartet sein. Das Hauptfenster muß angezeigt werden.

1. Der Systemanalytiker wählt im Hauptmenü »Bearbeiten - Anforderungsdokumente«.
2. Das ADMIRE-Werkzeug öffnet das Dokumentfenster.
3. Der Systemanalytiker wählt im Auswahlbaum das Anforderungsdokument »Lastenheft SESAM-2«, darin das Kapitel »3.1 Funktionen« und klickt auf die Notebook-Lasche »Inhaltselemente«.
4. Das ADMIRE-Werkzeug zeigt auf der rechten Seite des Dokumentfensters eine leere Anforderung.
5. Der Systemanalytiker trägt eine neue Anforderung ein (Abbildung 52 und 53) und wählt »Eintragen«.

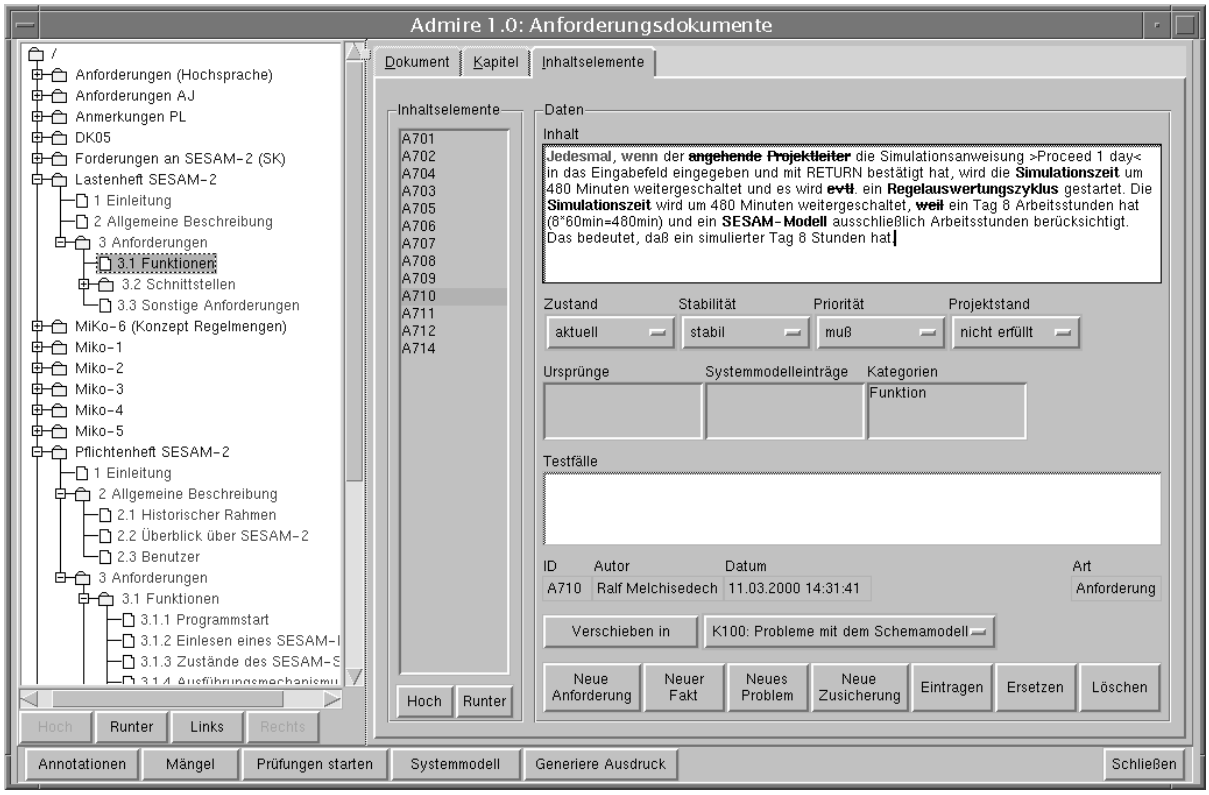


Abbildung 52: Dokumentfenster mit neuer Anforderung

6. Das ADMIRE-Werkzeug führt eine fokussierte Inspektion durch und markiert Wörter und Phrasen des Freitextes[1].

Der Systemanalytiker wird durch die Markierungen auf einige wichtige Punkte aufmerksam gemacht:

- Die Wörter »jedesmal« und »wenn« werden kursiv dargestellt, weil es sich um All-Quantoren handelt. Der Systemanalytiker soll durch die warnende Markierung dazu angeregt werden, darüber nachzudenken, ob die Benutzung der All-

1. Da die Texte in den Bildschirmauszügen nur schwer lesbar sind und das ADMIRE-Werkzeug zur Markierung Farben einsetzt, wird im folgenden für die Ergebnisse der fokussierten Inspektionen folgendes Markierungsschema verwendet: Wörter, die nicht benutzt werden sollen, werden durchgestrichen. Wörter, die benutzt werden sollen, werden in Fettschrift dargestellt. Wörter, deren Benutzung nicht mit Sicherheit auf einen Mangel schließen läßt, werden kursiv dargestellt. Im ADMIRE-Werkzeug werden letztere fett und rot dargestellt.

Anforderung:

»*Jedesmal, wenn* der ~~angehende Projektleiter~~ die Simulationsanweisung ›Proceed 1 day‹ in das Eingabefeld eingegeben und mit RETURN bestätigt hat, wird die **Simulationszeit** um 480 Minuten weitergeschaltet und es wird ~~evtl~~. ein **Regelauswertungszyklus** gestartet. Die **Simulationszeit** wird um 480 Minuten weitergeschaltet, ~~weil~~ ein Tag 8 Arbeitsstunden hat (8*60min=480min) und ein **SESAM-Modell** ausschließlich Arbeitsstunden berücksichtigt. Das bedeutet, daß ein simulierter Tag 8 Stunden hat.«

Attribute: Zustand »aktuell«; Stabilität: »stabil«; Priorität: »muß«; Projektstand: »nicht erfüllt«; Systemmodelleinträge: »-«; Ursprünge: »-«; Kategorie: »Funktion«; Testfälle: »-«

Abbildung 53: Die ursprüngliche Eingabe

Quantoren hier wirklich korrekt ist. Da das in diesem Beispiel der Fall ist, beschließt der Systemanalytiker, daran nichts zu ändern.

- Die Phrase »angehender Projektleiter« wird durchgestrichen dargestellt, weil es sich hierbei um den Term eines verworfenen Glossareintrags handelt. Der zugehörige aktuelle Glossareintrag hat den Term »Spieler«. In der Analysephase wurden beide Terme solange benutzt, bis festgestellt wurde, daß das Glossar zwei aktuelle Terme mit gleicher Bedeutung enthält. Um die Begriffskonsistenz möglichst hoch zu halten, wurde beschlossen, einen der beiden Terme zu verwerfen. Um zu verhindern, daß er versehentlich weiter benutzt wird, wurde er nicht gelöscht, sondern als »verworfen« markiert.
- Die Phrasen »Simulationszeit«, »Regelauswertungszyklus« und »SESAM-Modell« werden durch Fettschrift hervorgehoben, weil es sich um benutzte, aktuelle Glossareinträge handelt.
- Das Wort »evtl.« wird durchgestrichen dargestellt, weil es sich hierbei um ein WeakWord handelt. Der Systemanalytiker beschließt, diese Ungenauigkeit zu beseitigen. Weil er aber noch nicht genau abschätzen kann, unter welchen Bedingungen ein neuer Regelauswertungszyklus gestartet werden soll, verwendet er den TBD-Marker.
- Das Wort »weil« wird durchgestrichen dargestellt, weil es eine kausale Konjunktion ist und eine Begründung einleitet. Da Inhaltselemente keine Begründungen enthalten sollen, beschließt der Systemanalytiker, das Inhaltselement in ein Inhaltselement und eine Annotation aufzuspalten.
- Auffallend ist auch, daß das wie ein Fachwort wirkende Wort »Simulationsanweisung« nicht markiert wird. Das Wort ist nicht im Glossar definiert. Dafür kann es zwei Gründe geben: (1) Das Glossar ist unvollständig. (2) Es gibt einen anderen Term für den eigentlich gemeinten Begriff. In diesem Fall trifft (2) zu. Der richtige Term ist »Benutzerkommando«.

7. Der Systemanalytiker ändert die Anforderung (Abbildung 54, S. 200) und wählt »Ersetzen«.
8. Das ADMIRE-Werkzeug speichert das Inhaltselement. Der Eintrag erhält den eindeutigen Bezeichner »A710«.
9. Der Systemanalytiker wählt im Dokumentfenster »Annotationen«.

»*Jedesmal, wenn* der **Spieler** das **Benutzerkommando** ›Proceed 1 day‹ in das Eingabefeld eingegeben und mit RETURN bestätigt hat, wird die **Simulationszeit** um 480 Minuten weitergeschaltet. *Wenn* folgende Bedingung (TBD) erfüllt ist, wird anschließend ein **Regelauswertungszyklus** gestartet.«

Abbildung 54: Der überarbeitete Anforderungstext

10. Das ADMIRE-Werkzeug öffnet das Annotationsfenster.
11. Der Systemanalytiker gibt eine Annotation ein (Abbildung 55).

Annotation:

»Die **Simulationszeit** wird um 480 Minuten weitergeschaltet, weil ein Tag 8 Arbeitsstunden hat (8*60min=480min) und ein **SESAM-Modell** ausschließlich Arbeitsstunden berücksichtigt. Das bedeutet, daß ein simulierter Tag 8 Stunden hat.«

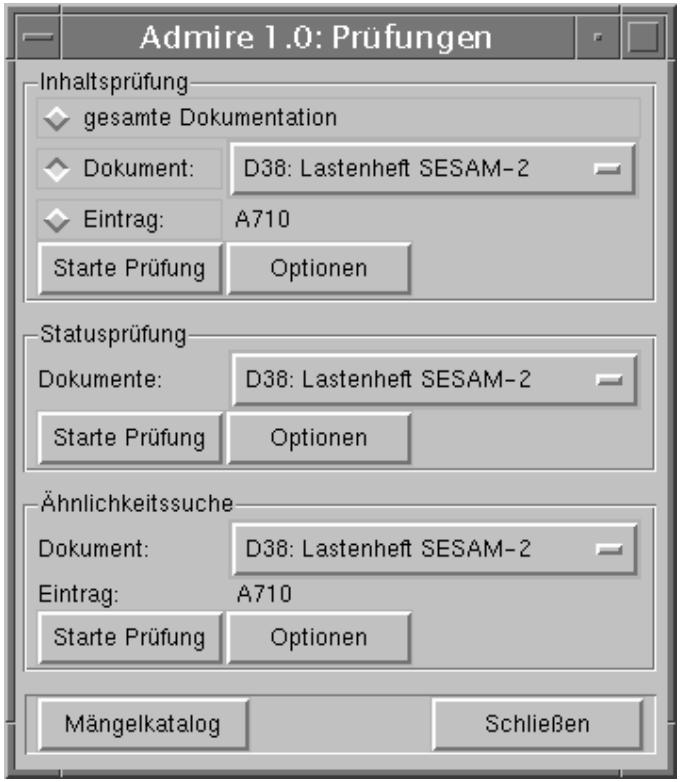
Verweise: »A710«

Abbildung 55: Eine neue Annotation

12. Der Systemanalytiker wählt »Eintragen« und anschließend »Schließen«.
13. Das ADMIRE-Werkzeug speichert die Annotation, schließt das Annotationsfenster und zeigt wieder das Dokumentfenster an.
14. Der Systemanalytiker wählt im Dokumentfenster »Prüfungen starten«.
15. Das ADMIRE-Werkzeug öffnet das Prüfungsfenster. Der zuletzt im Dokumentfenster bearbeitete Eintrag (A710) wird als zu prüfender Eintrag voreingestellt (Abbildung 56a, S. 201).
16. Der Systemanalytiker startet die Inhaltsprüfung.
17. Das ADMIRE-Werkzeug führt die Inhaltsprüfungen durch und zeigt die Ergebnisse an (Abbildung 56b, S. 201).
18. Der Systemanalytiker wählt »Speichern&Schließen«.
19. Das ADMIRE-Werkzeug erzeugt Mangeleinträge und schließt das Ergebnisfenster.

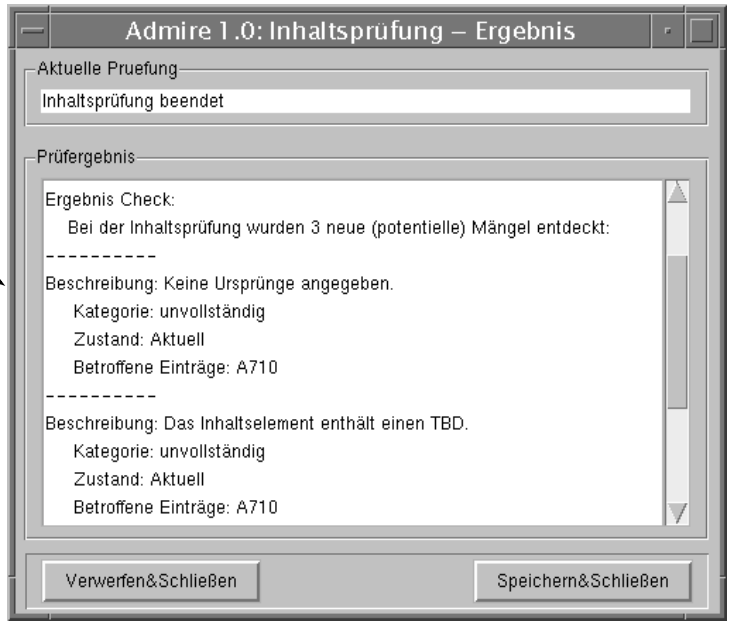
Da es sehr wichtig ist, daß Inhaltselemente immer auf eine Informationsquelle zurückgeführt werden können, beschließt der Systemanalytiker, den Mangel zu beheben und die Anforderung A710 entsprechend zu ändern.

20. Der Systemanalytiker ändert im Dokumentfenster die immer noch angezeigte Anforderung A710, indem er zusätzlich die externe Quelle »Q25: Protokoll des Miko-2« einträgt und anschließend »Ersetzen« wählt.
21. Das ADMIRE-Werkzeug speichert das geänderte Inhaltselement.
22. Der Systemanalytiker klickt auf das Prüfungsfenster und startet die Inhaltsprüfungen erneut.
23. Das ADMIRE-Werkzeug führt die Inhaltsprüfungen durch und zeigt die Ergebnisse an (Abbildung 56c, S. 201).



Prüfungsfenster (Abbildung 56a)

Ergebnisse der 1. Inhaltsprüfung (Abbildung 56b)



Ergebnisse der 2. Inhaltsprüfung (Abbildung 56c)

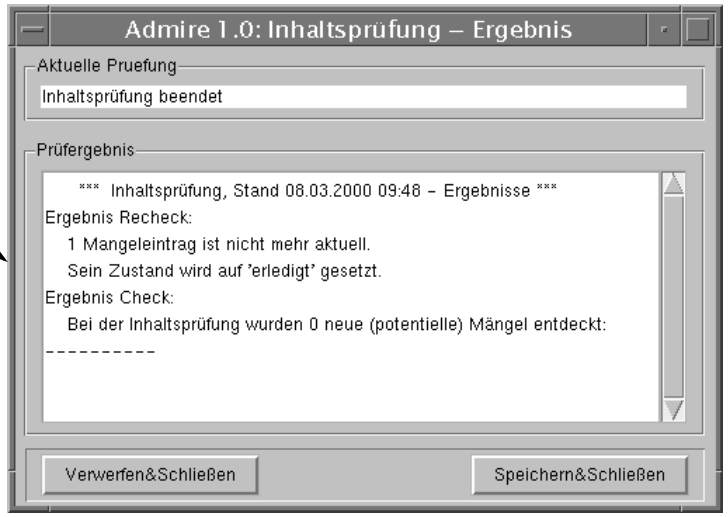


Abbildung 56: Inhaltsprüfung

24. Der Systemanalytiker wählt »Speichern&Schließen«.
25. Das ADMIRE-Werkzeug ändert das Zustandsattribut des nicht mehr aktuellen Mangeleintrags auf »erledigt« und schließt das Ergebnisfenster der Inhaltsprüfung.
26. Der Systemanalytiker wählt nacheinander »Schließen« im Prüfungsfenster und im Dokumentfenster.
27. Das ADMIRE-Werkzeug schließt das Prüfungsfenster und das Dokumentfenster und zeigt wieder das Hauptfenster an.
28. Der Systemanalytiker wählt im Hauptmenü »Bearbeiten - Mängelkatalog«.
29. Das ADMIRE-Werkzeug öffnet das Mängelkatalogfenster und zeigt die soeben erzeugten Mangeleinträge an:

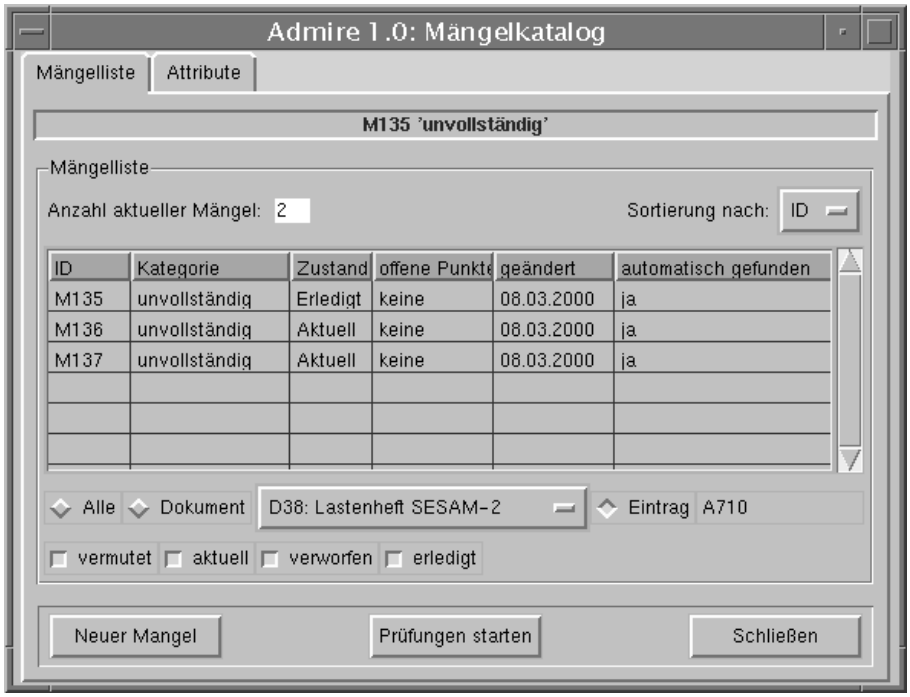


Abbildung 57: Mängelkatalogfenster

30. Der Systemanalytiker wählt »Schließen« im Mängelkatalogfenster.
31. Das ADMIRE-Werkzeug schließt das Mängelkatalogfenster und zeigt wieder das Hauptfenster an.

9.3.2 Szenario 2: »Prüfung des Glossars auf Konsistenz«

In diesem Szenario wird gezeigt, wie das ADMIRE-Werkzeug einen Systemanalytiker dabei unterstützen kann, das Glossar auf Konsistenz zu prüfen. Voraussetzung ist, daß das ADMIRE-Werkzeug gestartet wurde und das Hauptfenster angezeigt wird.

1. Der Systemanalytiker wählt im Hauptmenü »Prüfung - Prüfung starten«.
2. Das ADMIRE-Werkzeug öffnet das Prüfungsfenster (Abbildung 56a, S. 201).

Da der Systemanalytiker momentan nur an der Konsistenz des Glossars interessiert ist, sollen auch nur solche Inhaltsprüfungen durchgeführt werden, die für dieses Qualitätskriterium relevante Ergebnisse liefern.

3. Der Systemanalytiker wählt »Inhaltsprüfung - Optionen«.
4. Das ADMIRE-Werkzeug öffnet das Konfigurationsfenster für Inhaltsprüfungen (Abbildung 58).

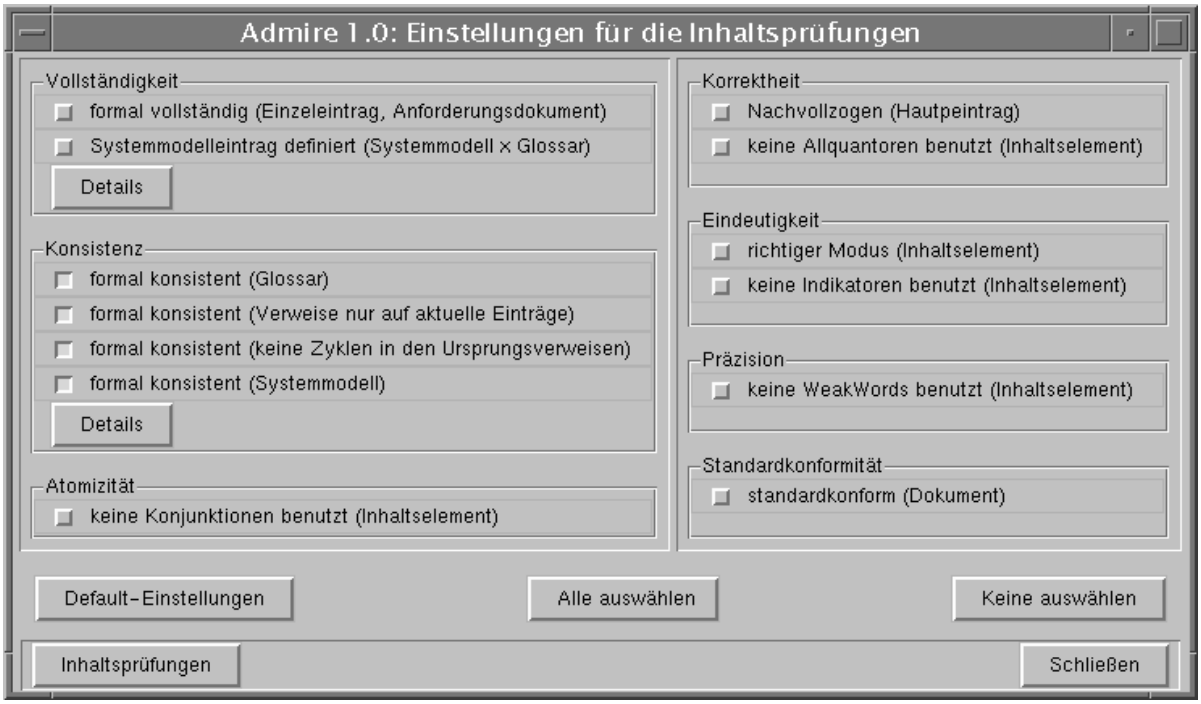


Abbildung 58: Konfigurationsfenster für Inhaltsprüfungen

5. Der Systemanalytiker selektiert alle »Konsistenz«-Prüfungen, deselektiert alle anderen Prüfungen und wählt »Schließen«.
6. Das ADMIRE-Werkzeug schließt das Konfigurationsfenster und zeigt wieder das Prüfungsfenster an.
7. Der Systemanalytiker wählt als zu prüfendes Dokument »Glossar« und startet die Inhaltsprüfungen.
8. Das ADMIRE-Werkzeug führt die gewählten Inhaltsprüfungen auf den Einträgen des Glossars durch und zeigt die Ergebnisse an (Abbildung 59, S. 204).

Der Systemanalytiker stellt fest, daß in dem Glossar noch einige Inkonsistenzen enthalten sind. Er beschließt, die Mängel zu speichern. Die Inkonsistenzen bestehen (u.a.) darin, daß es mehrere synonyme Glossareinträge gibt, die aktuell sind. Ein wichtiges Ziel der Terminologiebildung ist es, daß es für jeden Begriff genau (maximal) einen Term gibt. Es sollten also jeweils alle synonymen Glossareinträge bis auf einen verworfen werden. Der Systemanalytiker kann aber ohne weitere Rücksprache nicht entscheiden, welcher Term jeweils benutzt werden soll. Deswegen behebt er die Mängel vorerst nicht.

9. Der Systemanalytiker wählt nacheinander »Speichern&Schließen« im Ergebnisfenster der Inhaltsprüfung und »Schließen« im Prüfungsfenster.

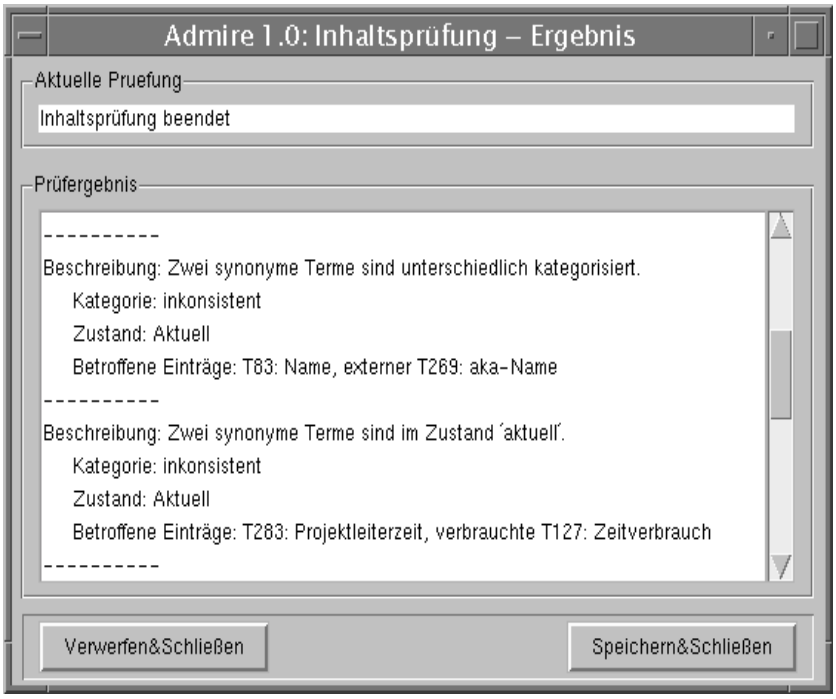


Abbildung 59: Ergebnisse der Inhaltsprüfung

10. Das ADMIRE-Werkzeug schließt das Ergebnisfenster der Inhaltsprüfung und das Prüfungsfenster und zeigt wieder das Hauptfenster an.

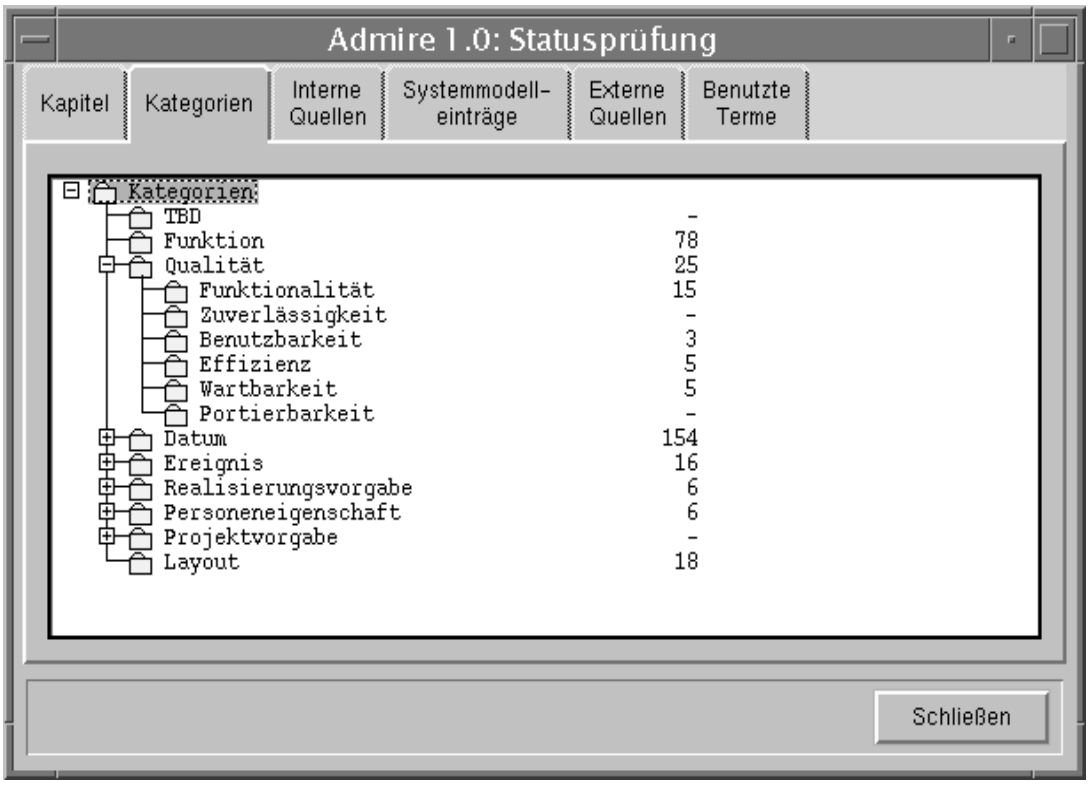
9.3.3 Szenario 3: »Prüfung eines Anforderungsdokuments auf Vollständigkeit«

In diesem Szenario wird gezeigt, wie das ADMIRE-Werkzeug einen Systemanalytiker durch Statusprüfungen dabei unterstützen kann, sich einen Überblick über den Inhalt des Pflichtenhefts zu verschaffen. Voraussetzung ist, daß das ADMIRE-Werkzeug gestartet wurde und das Hauptfenster angezeigt wird.

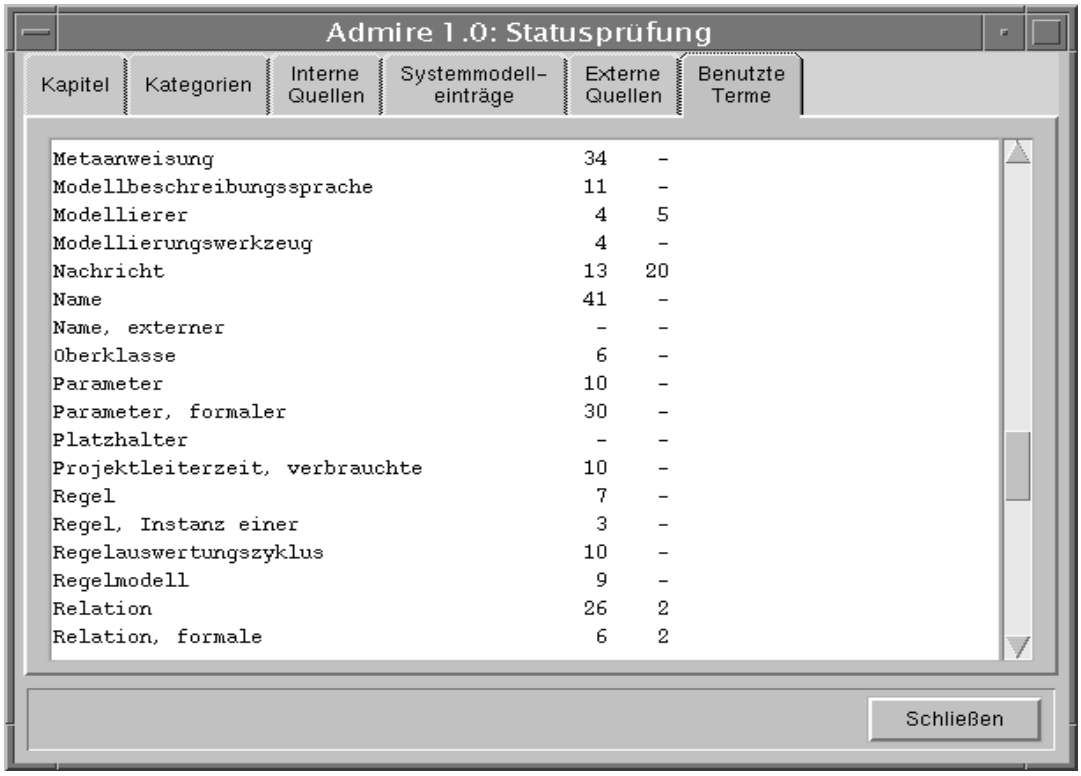
1. Der Systemanalytiker wählt im Hauptmenü »Prüfung - Prüfung starten«.
2. Das ADMIRE-Werkzeug öffnet das Prüfungsfenster (Abbildung 56a, S. 201).
3. Der Systemanalytiker wählt »Pflichtenheft SESAM-2« als zu prüfendes Dokument und startet die Statusprüfungen.
4. Das ADMIRE-Werkzeug führt die Statusprüfungen durch und zeigt die Ergebnisse an (Abbildung 60, S. 205).

In Abbildung 60a werden die Ergebnisse der Statusprüfung *classcount* dargestellt. Zu jeder Kategorie der Kategorisierungshierarchie für Inhaltselemente wird angegeben, wieviele Inhaltselemente des Pflichtenhefts entsprechend kategorisiert sind. In Abbildung 60b werden die Ergebnisse der Statusprüfung *termcount* dargestellt. Zu jedem aktuellen Glossareintrag wird angegeben, in wievielen Einträgen des Pflichtenhefts der Glossareintrag direkt (linke Spalte) oder nur indirekt (rechte Spalte) benutzt wird.

Der Systemanalytiker schaut sich die präsentierten Informationen in Ruhe an und versucht, Hinweise auf Unvollständigkeiten in dem Pflichtenheft zu finden. Einige Punkte fallen ihm dabei sofort auf:



a) Häufigkeit, mit der die Kategorien in den Inhaltselementen benutzt werden



b) Häufigkeit, mit der die aktuellen Glossareinträge in den Freitexten benutzt werden

Abbildung 60: Ergebnisse der Statusprüfung

- Daten werden sehr ausführlich beschrieben. Das ist nicht weiter verwunderlich, weil der größte Teil des Pflichtenhefts dazu dient, die Datenbeschreibungssprache für SESAM-Modelle zu definieren.
- Qualitäten dagegen werden nur in sehr geringem Umfang oder, wie z.B. »Zuverlässigkeit«, überhaupt nicht beschrieben.
- Projektvorgaben werden nicht beschrieben. Das liegt daran, daß diese Informationen in anderen (externen) Projektdokumenten enthalten sind.
- Einige Terme, z.B. »Platzhalter«, werden in dem Pflichtenheft weder direkt noch indirekt benutzt. Da sie in das Glossar eingetragen wurden, muß es sich um wichtige Konzepte für SESAM handeln oder gehandelt haben. Wenn sie immer noch relevant sind, müßten sie benutzt werden. Wenn nicht, sollten sie im Glossar als »verworfen« markiert werden.
- Auffallend ist auch, daß insgesamt 25 Inhaltselementen die Kategorie »Qualität« oder eine Unterkategorie zugeordnet wurde. Zählt man aber die Inhaltselemente, denen Unterkategorien von Qualität zugeordnet wurden, so erhält man 28. Das liegt daran, daß einigen Inhaltselementen mehrere Unterkategorien zugeordnet wurden. Ein Inhaltselement, das z.B. als »Qualität - Benutzbarkeit« und »Qualität- Effizienz« kategorisiert ist, wird für jede Unterkategorie gezählt, aber nur einmal für die übergeordnete Kategorie »Qualität«.

Da es dem Systemanalytiker bei dieser Prüfung nur darum ging, einen Überblick über den Fortschritt des Pflichtenhefts zu erhalten, erzeugt er keine Mangeleinträge. Er merkt sich aber vor, daß noch bestimmte Informationen fehlen.

5. Der Systemanalytiker wählt nacheinander »Schließen« im Ergebnisfenster der Statusprüfung und im Prüfungsfenster.
6. Das ADMIRE-Werkzeug schließt das Ergebnisfenster der Statusprüfung und das Prüfungsfenster und zeigt das Hauptfenster an.

9.3.4 Szenario 4: »Prüfung eines Anforderungsdokuments auf Konsistenz«

In diesem Szenario wird gezeigt, wie das ADMIRE-Werkzeug den Systemanalytiker dabei unterstützen kann, Inkonsistenzen und Redundanzen in einem Anforderungsdokument zu finden.

Ausgangslage ist folgende: Der Systemanalytiker hat das Pflichtenheft ausgedruckt und liest es. Als er Anforderung A660 liest, meint er, daß er ähnliche Aussagen auch schon mal an früherer Stelle des Pflichtenhefts gelesen hat. Deswegen möchte er eine Ähnlichkeitssuche durchführen.

Das ADMIRE-Werkzeug muß gestartet sein. Das Hauptfenster muß angezeigt werden.

Abbildung 61: Suchfenster

1. Der Systemanalytiker wählt im Hauptmenü »Dokumentation - Suche«.
2. Das ADMIRE-Werkzeug öffnet das Suchfenster (Abbildung 61, S. 206).
3. Der Systemanalytiker gibt den eindeutigen Bezeichner »A660« ein und wählt »Suche«.

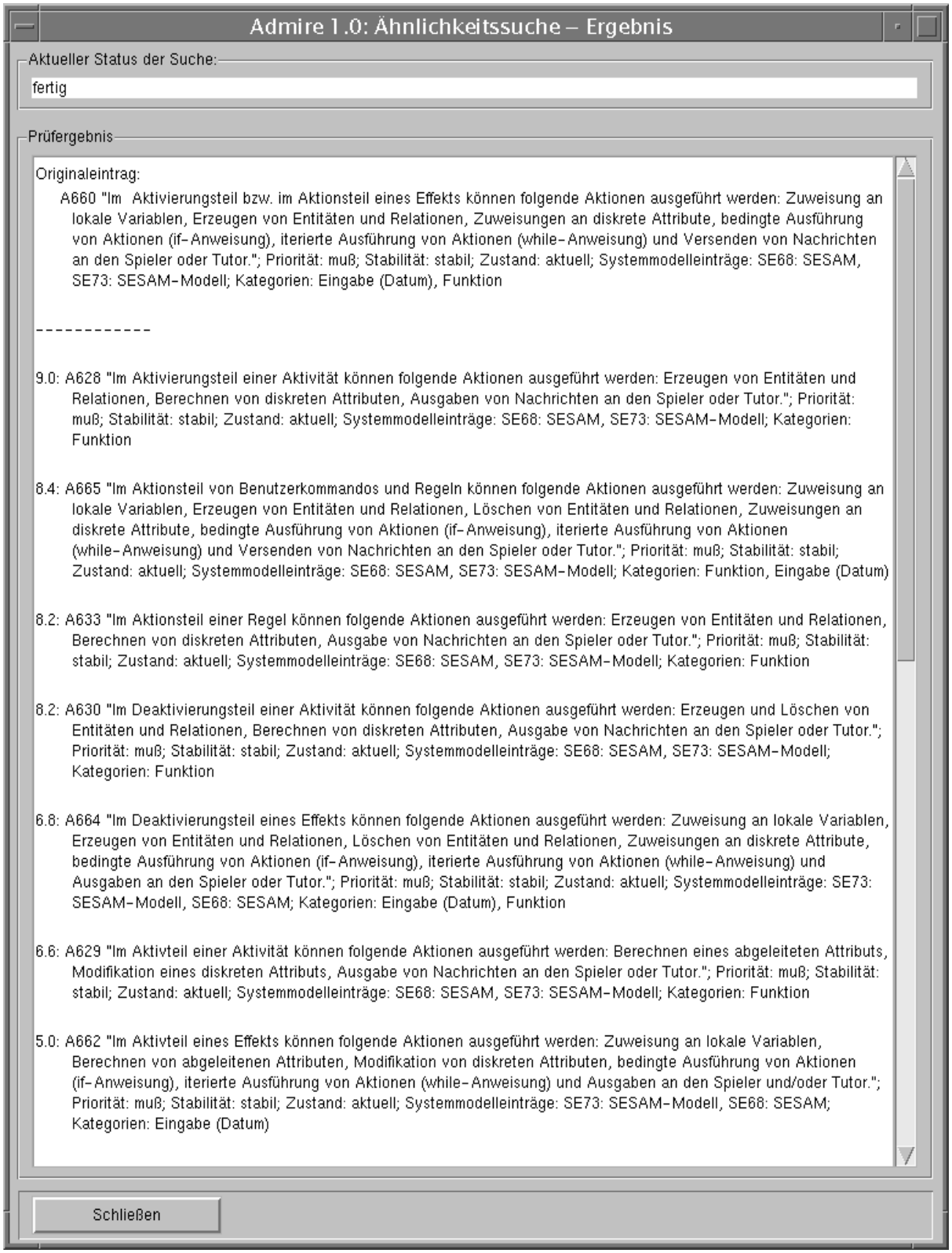


Abbildung 62: Ergebnis der Ähnlichkeitssuche

4. Das ADMIRE-Werkzeug öffnet das Dokumentfenster (Abbildung 52, S. 198) und zeigt die Anforderung »A660« an.

5. Der Systemanalytiker wählt »Prüfungen starten«.

6. Das ADMIRE-Werkzeug öffnet das Prüfungsfenster (Abbildung 56a, S. 201) und zeigt den Bezeichner der Anforderung »A660« als aktuellen Eintrag an.

7. Der Systemanalytiker wählt als aktuelles Dokument für die Ähnlichkeitssuche »Pflichtenheft SESAM-2« und startet die Ähnlichkeitssuche.

8. Das ADMIRE-Werkzeug führt die Ähnlichkeitssuche durch und gibt eine Liste von Inhaltselementen als Ergebnis aus, absteigend sortiert nach dem Wert des Ähnlichkeitsmaßes (Abbildung 62, S. 207).

Der Systemanalytiker kann erkennen, daß es anscheinend zwei Kapitel im Pflichtenheft gibt, in denen beschrieben wird, welche Aktionen in welchen Teilen eines Effekts ausgeführt werden sollen. »Effekt« ist ein Oberbegriff für »Aktivität«, »Regel« und »Benutzerkommando«. Das eine Kapitel enthält (u.a.) die Anforderungen A628, A629, A630 und A633, das andere die Anforderungen A660, A662, A664 und A665. Es liegt also zum einen Redundanz vor. Zum anderen kann man aber auch feststellen, daß die Anforderungen im Detail Widersprüche aufweisen. Hier ist noch eine Überarbeitung des Pflichtenhefts notwendig. Der Systemanalytiker beschließt, einen entsprechenden Mangeleintrag in den Mängelkatalog einzutragen.

9. Der Systemanalytiker wählt nacheinander »Schließen« im Ergebnisfenster der Ähnlichkeitssuche und im Prüfungsfenster.

10. Das ADMIRE-Werkzeug schließt das Ergebnisfenster der Ähnlichkeitssuche und das Prüfungsfenster und zeigt wieder das Hauptfenster an.

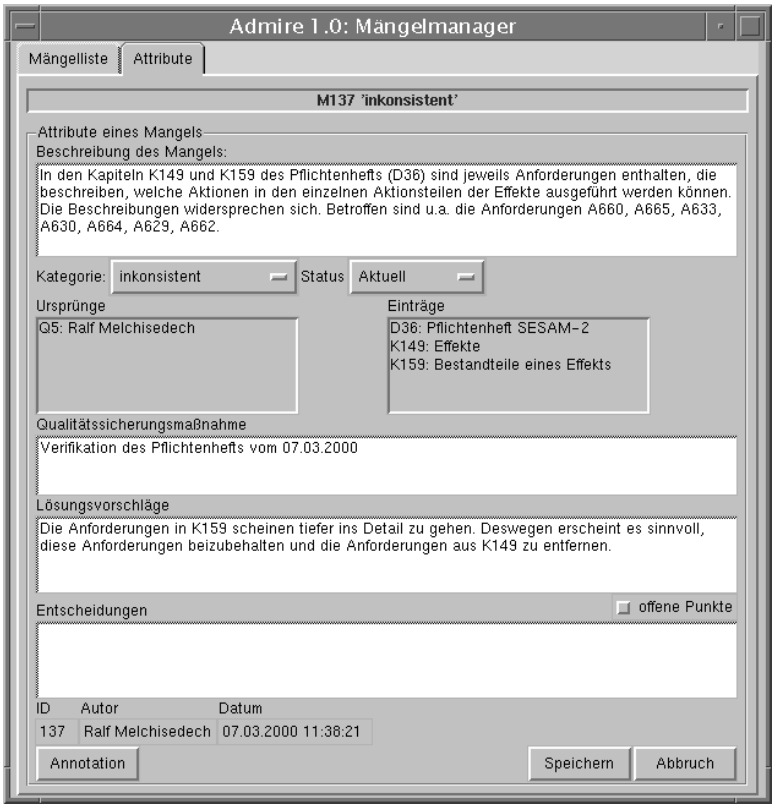


Abbildung 63: Ein manuell erzeugter Mangeleintrag

11. Der Systemanalytiker wählt im Hauptmenü »Bearbeiten - Mängelkatalog«.
12. Das ADMIRE-Werkzeug öffnet das Mängelkatalogfenster (Abbildung 57, S. 202) und wählt »Neuer Mangel«.
13. Der Systemanalytiker erzeugt einen Mangeleintrag (Abbildung 63, S. 208).
14. Der Systemanalytiker wählt nacheinander »Speichern« und »Schließen«.
15. Das ADMIRE-Werkzeug speichert den Mangeleintrag, schließt das Mängelkatalogfenster und zeigt wieder das Hauptfenster an.

9.3.5 Szenario 5: »Entscheidung über ein wichtiges Konzept«

In diesem Szenario wird gezeigt, wie das ADMIRE-Werkzeug die Systemanalytiker und Informanten bei der Entscheidungsfindung unterstützen kann.

In dem Vorgängerprojekt SESAM-1 gab es ein wichtiges Konzept in der Datenbeschreibungssprache für SESAM-Modelle: die Attributmodi. Bisheriger Konsens war, daß das Konzept in abgeänderter Form auch in SESAM-2 übernommen werden soll. In der Spezifikationsphase wurde dieser Konsens aber immer wieder angezweifelt, so daß sich die Systemanalytiker entschlossen haben, das Problem endgültig zu klären.

In einer Sitzung soll nun entschieden werden, ob das Konzept der Attributmodi in SESAM-2 übernommen werden soll. Das ADMIRE-Werkzeug ist an der Sitzung nicht direkt beteiligt. Es kann vom Systemanalytiker »nur« zur Vor- und Nachbereitung eingesetzt werden.

Vor der Sitzung

Das ADMIRE-Werkzeug muß gestartet sein. Das Hauptfenster muß angezeigt werden.

1. Der Systemanalytiker wählt im Hauptmenü »Bearbeiten - Filter«.
2. Das ADMIRE-Werkzeug öffnet das Sichtenfenster.
3. Der Systemanalytiker stellt nacheinander folgende Kriterien für den Filtermechanismus ein: »alle Einträge des Pflichtenhefts, in denen der Term ›Attributmodus‹ benutzt wird« und »alle Einträge der sonstigen Anforderungsdokumente und des Glossars, in denen der Term ›Attributmodus‹ benutzt wird« und wählt »Erstellen«.
4. Das ADMIRE-Werkzeug generiert zwei Reporte im HTML-Format, in denen alle Einträge enthalten sind, die den o.g. Bedingungen genügen. Auszüge aus den Reporten werden in den Abbildungen 64 (S. 210) und 65 (S. 211) gezeigt.

Nach der Sitzung

Inzwischen wurde die Sitzung durchgeführt. In der Sitzung wurde die Entscheidung getroffen, daß das Konzept der Attributmodi doch nicht in SESAM-2 umgesetzt werden soll. Das Pflichtenheft soll entsprechend geändert werden. Diese Entscheidung wird als Mangeleintrag in den Mängelkatalog eingetragen.

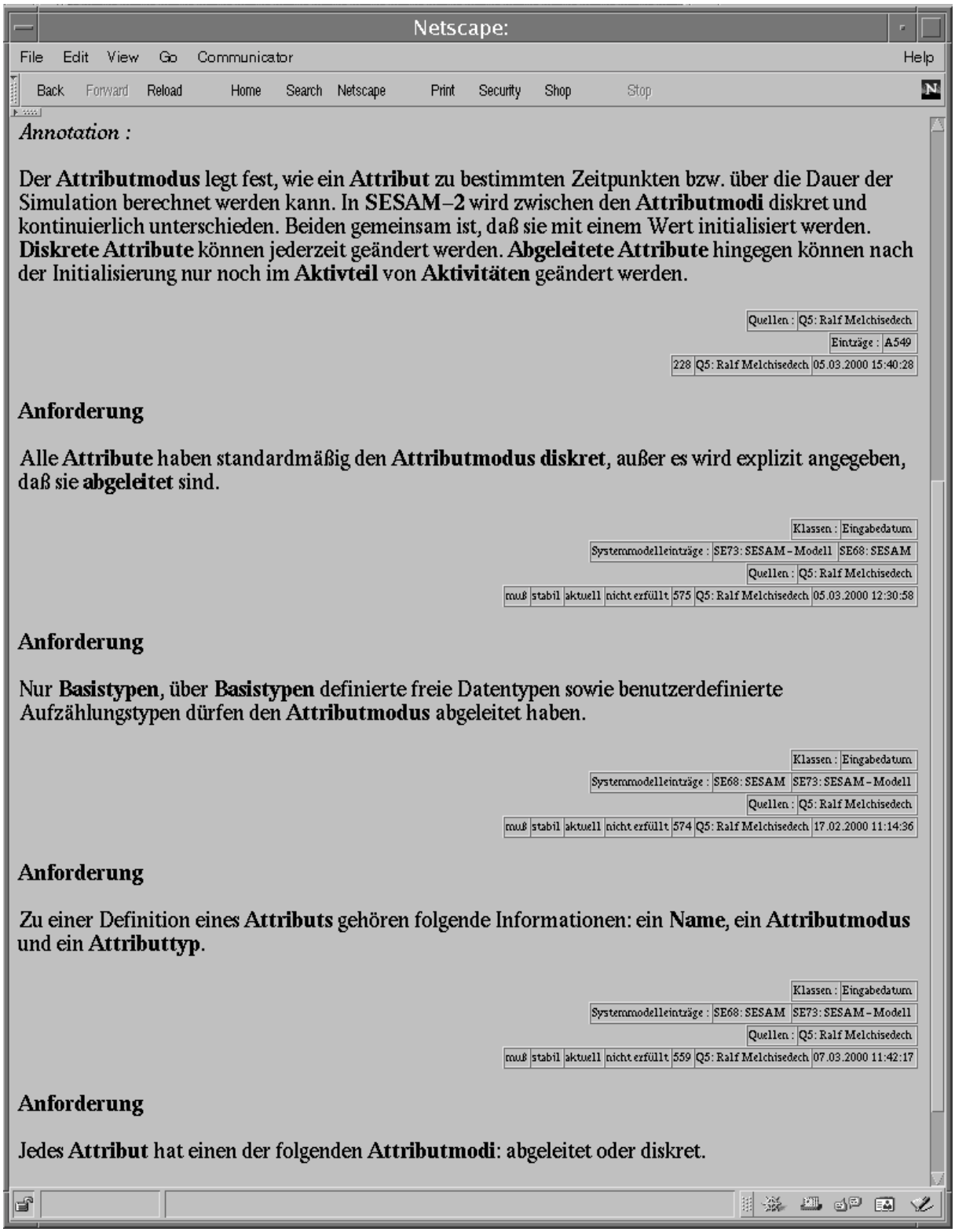


Abbildung 64: Auszug aus dem generierten Report (Pflichtenheft)

1. Der Systemanalytiker wählt im Hauptmenü »Bearbeiten - Mängelkatalog«.
2. Das ADMIRE-Werkzeug öffnet das Mängelkatalogfenster.
3. Der Systemanalytiker wählt »Neuer Mangel« und trägt Informationen über den Mangel ein (siehe Abbildung 66, S. 212).
4. Der Systemanalytiker wählt nacheinander »Speichern« und »Schließen«.

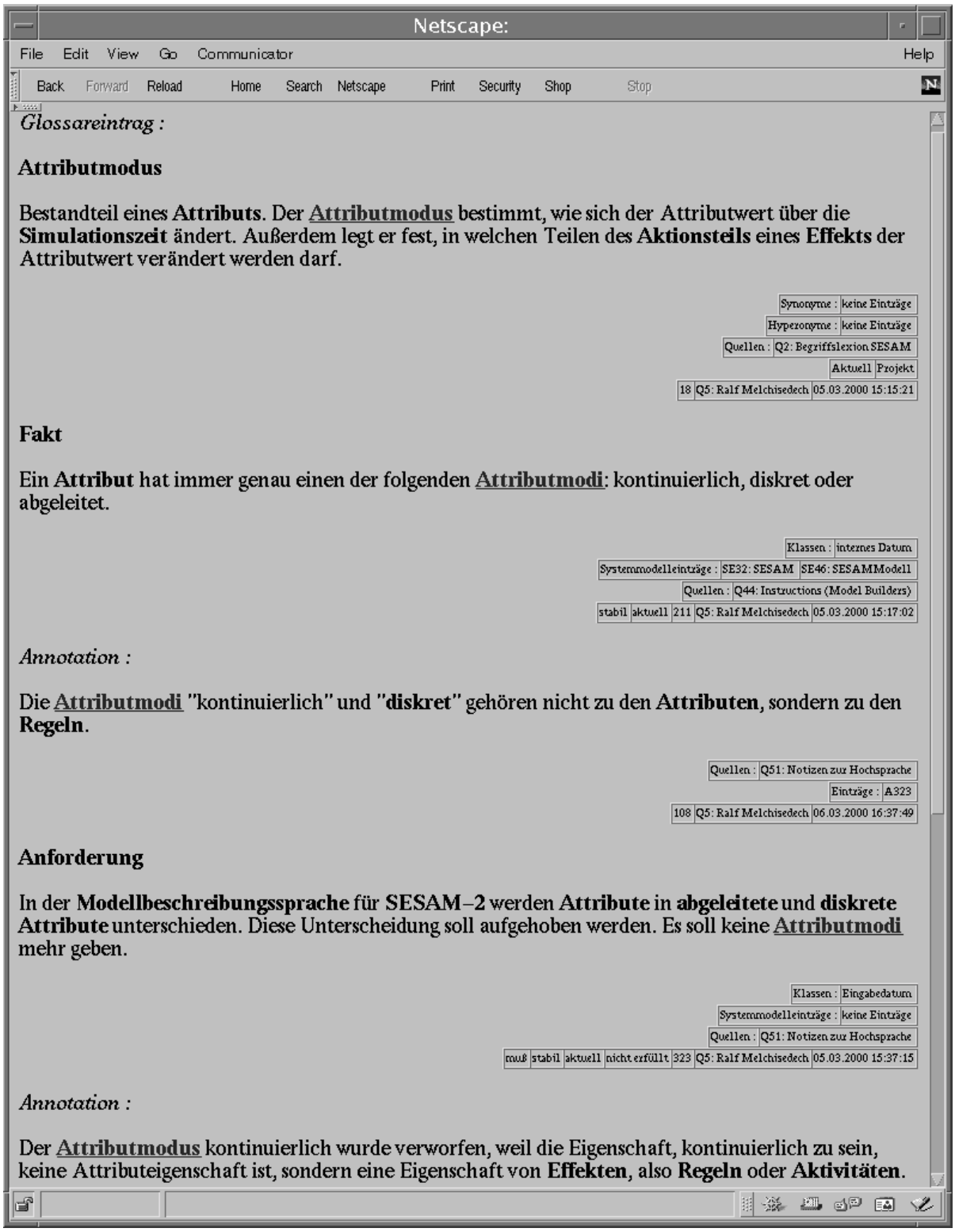


Abbildung 65: Auszug aus dem generierten Report (sonstige Dokumente)

5. Das ADMIRE-Werkzeug speichert den Mangeleintrag, schließt das Mängelkatalogfenster und zeigt wieder das Hauptfenster an.

In den vorangegangenen fünf Szenarien wurde gezeigt, wie das ADMIRE-Werkzeug den Systemanalytiker bei seinen Tätigkeiten unterstützen kann. Die Beispiele sollen deutlich machen, daß das ADMIRE-Werkzeug sowohl »im Kleinen« hilfreich sein kann, wenn der Systemanalytiker z.B. eine einzelne Anforderung

Abbildung 66: Ein manuell erzeugter Mangeleintrag
(Fenster nur ausschnittsweise dargestellt)

eingeben möchte, als auch »im Großen«, wenn der Systemanalytiker versucht, festzustellen, ob ein umfangreiches Anforderungsdokument Inkonsistenzen oder Redundanzen enthält oder ob es vollständig ist.

Die in diesem Kapitel verwendeten Beispiele stammen alle aus vorhandenen Dokumenten des SESAM-2-Projekts. Die Inhalte der SESAM-2-Dokumente wurden nur insoweit geändert, daß sie sinnvoll in eine Dokumentation eingetragen werden konnten.

Weitere Informationen über das ADMIRE-Werkzeug finden Sie auf den WWW-Seiten der Abteilung Software Engineering (http://www.informatik.uni-stuttgart.de/ifi/se/se.html) unter »Forschung«.

Kapitel 10

Vergleich und Bewertung

In diesem Kapitel wird der ADMIRE-Ansatz mit anderen Ansätzen aus der Literatur verglichen und bewertet. Zuerst werden in Kapitel 10.1 verwandte Ansätze vorgestellt und die wesentlichen Unterschiede zu ADMIRE herausgearbeitet. In Kapitel 10.2 werden Ansätze vorgestellt, die den ADMIRE-Ansatz gut ergänzen können. Abschließend werden in Kapitel 10.3 die Stärken und Schwächen des ADMIRE-Ansatzes diskutiert und einige Erweiterungsmöglichkeiten vorgeschlagen.

10.1 Verwandte Ansätze

10.1.1 Werkzeuge

Kommerzielle Requirements-Engineering-Werkzeuge können in drei Kategorien eingeteilt werden: Textverarbeitungsprogramme, notationszentrierte Werkzeuge und Requirements-Management-Werkzeuge. In notationszentrierten Werkzeugen spielt natürliche Sprache eine untergeordnete Rolle. Sie können in vielen Fällen als Ergänzung zur natürlichsprachlichen Spezifikation verwendet werden und werden deswegen in Kapitel 10.2 (»Ergänzende Ansätze«) beschrieben.

Textverarbeitung

Textverarbeitungsprogramme gehören zu den am häufigsten eingesetzten Requirements-Engineering-Werkzeugen. Sie wurden zwar nicht speziell zur Unterstützung von Requirements-Engineering-Tätigkeiten entwickelt, sie werden aber in sehr vielen Projekten als einziges Werkzeug in der Requirements-Engineering-Phase eingesetzt. Die folgende Diskussion bezieht sich auf das Textverarbeitungsprogramm FrameMaker Version 5.5 (Adobe). Die meisten Aussagen gelten auch für andere Programme wie z.B. Word (Microsoft), Star Office (Star Division) oder Latex (Lamport, 1985).

Eine wichtige Eigenschaft von FrameMaker ist, daß beliebige Informationen in fast beliebigen textuellen und graphischen Notationen eingegeben werden können. FrameMaker verfügt über ein Zeichenwerkzeug und einen Formeleditor und bietet die Möglichkeit, Bilder zu importieren.

Im folgenden werden einige Eigenschaften von FrameMaker aufgezeigt, die bei der Erstellung eines Anforderungsdokuments hilfreich sein können:

- Es können Tabellen verwendet werden, um einzelne Einträge deutlich voneinander zu trennen und zusätzliche Attribute anzugeben.
- Zwischen Einträgen können Querverweise manuell angelegt werden. Im Falle einer Änderung oder Verschiebung werden sie automatisch aktualisiert. Sie sind nicht typisiert. Querverweise können zwischen beliebigen Textstellen hergestellt werden.
- Einige Kapitel, z.B. ein Index oder ein Inhaltsverzeichnis, können automatisch generiert werden. Das setzt voraus, daß der Systemanalytiker sehr diszipliniert mit Absatzformaten umgeht und an allen relevanten Stellen Marker setzt.
- Es können nur einige einfache (halb-)automatische Prüfungen durchgeführt werden, z.B. eine Rechtschreibprüfung. Prüfungen auf Erfüllung der in Kapitel 5.10 geforderten Qualitätskriterien werden rudimentär unterstützt. Für den Abgleich mit dem Glossar z.B. stehen folgende Hilfsmittel zur Verfügung:
  - Der generierte Index kann mit dem Glossar abgeglichen werden. Der Abgleich muß manuell erfolgen. Inkonsistenzen in der Benutzung von Termen können so recht gut erkannt werden.
  - Die Rechtschreibprüfung kann verwendet werden, um potentielle Terme zu erkennen. Voraussetzung dafür ist, daß der Systemanalytiker das »persönliche« Wörterbuch pflegt und dafür Sorge trägt, daß nur aktuelle Terme des Glossars dort eingetragen sind. Alle unbekannten Phrasen, die dann von der Rechtschreibprüfung gefunden werden, sind Kandidaten für das Glossar.

Einige Textverarbeitungsprogramme (z.B. Word) bieten darüberhinaus Makromechanismen an, mit denen z.B. eindeutige Bezeichner generiert oder Masken zur Bearbeitung der Einträge realisiert werden können. Zur Programmierung der Makros wird eine Programmiersprache bereitgestellt, mit deren Hilfe das Textverarbeitungsprogramm um Funktionen eines Requirements-Management-Werkzeugs (siehe unten) erweitert werden kann.

Die vorhergehende Diskussion zeigt, daß Textverarbeitungsprogramme einige recht interessante Funktionen zur Unterstützung der Requirements-Engineering-Tätigkeiten aufweisen. Es ist aber viel Disziplin und »Handarbeit« der Systemanalytiker notwendig.

Requirements-Management-Werkzeuge

Bei *Requirements-Management-Werkzeugen* steht als Notation die natürliche Sprache im Vordergrund. Es werden zwei Ansätze unterschieden:

- Werkzeuge, die primär auf einer Datenbank basieren: Anforderungen werden als einzelne Datensätze behandelt. Aus diesen Datensätzen kann dann ein Anforderungsdokument generiert werden. In diese Kategorie fallen die Werkzeuge DOORS (QSS), RDT (GEC-Marconi Systems), XTie-RT (Teledyne Brown Engineering) und RTM (Integrated Chipware Inc.).
- Werkzeuge, die auf vorhandenen Textverarbeitungssystemen basieren und diese um bestimmte Funktionalitäten erweitern, z.B. um eine Datenbank, die zu einzelnen Abschnitten gehörende Attribute verwaltet. In diese Kategorie gehören Vital Link (Compliance Automation Inc.) und Requisite Pro (Rational Software Corporation).

Gemeinsam ist allen Werkzeugen, daß sie eine Menge von Anforderungen verwalten können. Zu jeder Anforderung können neben dem eigentlichen Text auch Attribute gespeichert werden (z.B. Priorität der Anforderung, Urheber der Anforderung, Erklärungen usw.). Um zwischen verschiedenen Arten von Anforderungen unterscheiden zu können, besteht die Möglichkeit, neue Anforderungstypen und Attribute zu definieren. Die Anforderungen werden in einer objektorientierten oder relationalen Datenbank verwaltet.

Anforderungen können zueinander in Beziehung stehen. Diese Beziehungen werden durch Links verwaltet. Die Links müssen vom Systemanalytiker von Hand gesetzt werden. Im Vordergrund stehen dabei hierarchische Parent/Child-Beziehungen, also z.B. Beziehungen zwischen abstrakten Anforderungen und sie verfeinernden Anforderungen. In RDT ist es zusätzlich möglich, Link-Attribute anzugeben. Einige Werkzeuge (z.B. Vital Link) bieten dem Benutzer die Möglichkeit, neue Beziehungsklassen zu definieren.

Basierend auf dem Konzept der Links bieten einige Werkzeuge einfache Konsistenzprüfungen an:

- DOORS überprüft, ob es zu jeder verfeinerten Anforderung einen Parent gibt bzw. ob jede Parent-Anforderung verfeinert wurde.
- Requisite Pro unterstützt das Konzept der »Suspect Links«. Jedesmal, wenn eine Anforderung geändert wird, werden alle Anforderungen, die über Links mit der geänderten Anforderung verbunden sind, als »evtl. zu ändern« markiert.

Darüber hinaus gibt es in den Werkzeugen eine Reihe von Funktionen, die für die praktische Anwendung unbedingt notwendig sind:

- Verwaltung mehrerer Projekte/Dokumente. Links können auch zwischen Einträgen verschiedener Dokumente eingetragen werden.
- Import vorhandener Word- oder FrameMaker-Dokumente. Der Systemanalytiker kann bestimmte Teile der Dokumente als Anforderungen oder Erklärungen markieren und muß zusätzliche Informationen (Attribute) hinzufügen. XTie-RT erlaubt es, Anforderungen anhand von benutzerdefinierten Schlüsselwörtern halbautomatisch zu erkennen.
- Erzeugung und Verwaltung von Sichten.
- Generierung von Reporten und Anforderungsdokumenten.
- Unterstützung des Mehrbenutzerbetriebs; Vergabe individueller Rechte.
- Configuration Management.
- Erweiterbarkeit. DOORS verfügt über eine Skriptsprache (DXL, DOORS Extension Language), mit der zusätzliche Funktionen realisiert werden können.

Im Gegensatz zu ADMIRE ist das zugrundeliegende Informationsmodell recht einfach: eine Menge von Anforderungstypen, die sich durch ihre Attribute unterscheiden und über Beziehungen untereinander verbunden sein können. Ein Glossar und Systemmodelle (bzw. Kontextdiagramme) können nicht angelegt werden. Es wird nicht klar zwischen System und Arbeitsumgebung bzw. zwischen Ist- und Soll-Zustand unterschieden. Mängel können nicht verwaltet werden. Die möglichen Prüfungen beschränken sich auf einfache Konsistenzprüfungen. Die Inhalte der Freitexte werden dabei nicht berücksichtigt.

EPOS

Bei EPOS (*E*ntwicklungs- und *P*rojektmanagement-*o*rientiertes *S*pezifikationssystem, siehe Lauber (1982, 1985 und 1985a) und Partsch (1991, S. 193ff)) handelt es sich um ein Software-Entwicklungssystem, das ursprünglich zur Realisierung von Echtzeitsystemen konzipiert wurde, dann aber so erweitert wurde, daß es auch in anderen Anwendungsbereichen angewandt werden kann.

EPOS deckt die Tätigkeiten Anforderungsdefinition, Entwurf, Implementierung, Projektmanagement, Configuration Management und Qualitätssicherung ab. Alle anfallenden Informationen werden in einer gemeinsamen Datenbank verwaltet. Folgende Notationen werden unterstützt: EPOS-R für die Anforderungsdefinition, EPOS-S für den Entwurf und EPOS-P für das Projektmanagement und die Produktverwaltung. Die folgende Diskussion bezieht sich nur auf EPOS-R.

Auch wenn EPOS inzwischen leicht veraltet ist, konnten einige Grundideen in den ADMIRE-Ansatz integriert werden. EPOS-R unterscheidet zwischen Freitexten (Annotationen in ADMIRE), Anforderungen und Voraussetzungen (Fakten, Zusicherungen), die in einer Kapitelhierarchie angeordnet werden können. Anforderungen und Voraussetzungen können verfeinert (»ersetzt«) werden. Anforderungen können quittiert werden, d.h. ihr Zustand kann auf »erledigt« gesetzt werden. Es gibt ein Glossar. Termen, Anforderungen, Voraussetzungen und Kapiteln können frei wählbaren Kategorien zugeordnet werden. Als Notationen werden natürliche Sprache, Entscheidungstabellen und in neueren Versionen Zustandsautomaten verwendet.

Das Glossar in EPOS besteht aus einer flachen Liste von Termen, die nur über ihre Kategorienzugehörigkeit miteinander in Beziehung gesetzt werden können. Synonymie- und Hyponymiebeziehungen werden nicht unterstützt. Beziehungen zwischen Freitext und Glossar müssen von Hand eingetragen werden. In EPOS wird nicht klar zwischen Zielsystem und Arbeitsumgebung unterschieden. Es kann kein Systemmodell (Kontextdiagramm) erstellt werden. Die Informationsquellen werden nicht explizit erfaßt. Attribute wie Priorität und Stabilität von Anforderungen können nicht eingetragen werden.

Mit Hilfe des Analysewerkzeugs EPOS-A können einige einfache Prüfungen durchgeführt werden: Rechtschreibprüfung der Freitexte, Integritätsprüfung der Datenbank, d.h. Prüfung auf Gültigkeit der manuell eingegebenen Verweise, und Prüfungen auf Redundanz, Eindeutigkeit und Vollständigkeit der Entscheidungstabellen und Zustandsautomaten.

Weiterführende Prüfungen auf Erfüllung der in Kapitel 5 beschriebenen Qualitätskriterien können nur manuell durchgeführt werden. EPOS-R verwaltet keine Mängel. Mit Ausnahme der Rechtschreibprüfung und der Konsistenzprüfung der in den Freitexten eingetragenen Querverweise werden die Freitexte bei den Prüfungen nicht berücksichtigt.

10.1.2 NLP-Ansätze

In der Literatur werden einige Ansätze beschrieben, in denen Techniken der Forschungsbereiche »Künstliche Intelligenz« (KI) und »Verarbeitung natürlicher Sprache« (natural language processing, NLP) in den frühen Phasen der Software-Entwicklung eingesetzt werden. Diese können grob unterteilt werden in »parsende

Ansätze«, »generierende Ansätze«, »parsende und generierende Ansätze« und »eingeschränkte Sprachen« als wichtige Spezialfälle der »parsenden Ansätze«.

Parsende Ansätze

Ausgehend von der Tatsache, daß in einem Software-Projekt viele Dokumente anfallen und ein nicht unwesentlicher Teil dieser Dokumente in natürlicher Sprache geschrieben ist, ist die Kernidee dieser Ansätze, die natürlichsprachlichen Texte mittels eines Parsers syntaktisch und semantisch zu interpretieren und in eine Wissensdatenbank (»knowledge base«) einzutragen. Die Datenbank kann dann automatisch verifiziert werden. Zur Validierung müssen die Daten wieder in natürliche Sprache paraphrasiert werden.

In (Borillo et al., 1992, Bras, Toussaint, 1993, Castell et al., 1994) wird das LESD-Projekt (*L*inguistic *E*ngineering for *S*oftware *D*evelopment) vorgestellt. Anwendungsbereich ist die Weltraum-Industrie, in der besonders umfangreiche Dokumentationen entstehen. Für einen Satelliten z.B. werden ca. 50000 natürlichsprachliche Entwicklungsdokumente erzeugt. Ziel dieses Projekts ist es, Werkzeuge zu entwickeln, mit deren Hilfe die Qualität von natürlichsprachlichen Anforderungsdokumenten verbessert werden kann. Hierbei stehen die Qualitätskriterien Konsistenz, Vollständigkeit, Nachvollziehbarkeit, Änderbarkeit und Überprüfbarkeit im Vordergrund.

Das LESD-Werkzeug besteht aus folgenden Komponenten:

- einem Sprachmodell, das aus einem Wörterbuch (dictionary), einer Grammatik und einer Wissensdatenbank (knowledge base) besteht. Das Wörterbuch enthält für jeden Eintrag morphologische, syntaktische und semantische Informationen. Die Grammatik enthält Regeln, wie die Einträge des Wörterbuchs verknüpft werden dürfen. Die Wissensdatenbank enthält zu Projektbeginn Informationen über den Anwendungsbereich in Form eines semantischen Netzwerks und wird mit jeder neuen Eingabe erweitert.
- einer NLP-Komponente, die die natürlichsprachlichen Aussagen morphologisch und syntaktisch parst.
- einer Domain-Interpretations-Komponente, die die geparsten Aussagen in die vorhandene Wissensdatenbank integriert.
- einer Reasoning-Komponente, die die Wissensdatenbank durchsucht und versucht, automatisch Mängel aufzudecken.

In (Biebow, Szulman, 1992 und 1994) wird das Werkzeug DISERT (*D*etection of *I*ncoherences in *S*oftware *E*ngineering *R*equirements *T*exts), später auch DASERT (*D*etection of *A*nomalies in *S*oftware *E*ngineering *R*equirements *T*exts) genannt, beschrieben. Ziel ist es, eine funktionale Spezifikation in einer graphischen Notation (SADT, siehe z.B. Ross, Schomann, 1977) zu erstellen und mit Freitext-Kommentaren anzureichern. SADT-Diagramme und Freitexte werden in ein semantisches Netzwerk überführt und von einer Wissensdatenbank verwaltet. Der Systemanalytiker kann die Wissensdatenbank auf Inkonsistenzen und Mehrdeutigkeiten untersuchen. DA(I)SERT wurde für den französischen Sprachraum entwickelt.

In (Cordes, Carver, 1988) wird das Werkzeug KAPS (*K*nowledge-Based *A*ssistent for *P*rogram *S*pecification) beschrieben. Ziel dieses Ansatzes ist es, die Qualität von

Anforderungsdokumenten zu verbessern. Das System erhält als Eingabe ein informales Anforderungsdokument. Die natürlichsprachlichen Sätze werden geparst, in eine Wissensdatenbank überführt und anhand vorgegebener Regeln verifiziert. Dann werden zwei Ausgaben generiert: (1) das Eingabedokument, angereichert mit Warnungen, Fehlermeldungen und einer umfangreichen Liste von Querverweisen; (2) eine neu generierte, strukturierte Version des ursprünglichen Dokuments, das zur Validierung an die Kunden gegeben werden kann. Treten beim Parsen Probleme auf, z.B. nicht eindeutige Parse-Bäume, fragt das System interaktiv nach. Ein Schwachpunkt dieses Ansatzes ist, daß aus syntaktischen Eigenschaften eines Satzes auf den Inhalt geschlossen wird. So gibt es z.B. folgende Regel (Cordes, Carver, 1988, S. 270): »Sätze, die das Wort ›have‹ enthalten, beschreiben physikalische Eigenschaften eines Objekts.« Der Systemanalytiker muß sich beim Formulieren an diese Regeln halten, weil das Werkzeug ansonsten falsche Einträge für die Wissensdatenbank produziert.

In (Edwards et al., 1995) wird das Werkzeug RECAP (*R*equirements *E*licitation, *C*apture and *A*nalysis *P*rocess Prototype Tool) vorgestellt, das den Systemanalytiker dabei unterstützen soll, gegebene natürlichsprachliche Anforderungsdokumente nach Anforderungen zu durchsuchen und an gegebene Schablonen anzupassen. Kommentare und ähnliches werden dabei ausgeblendet. Voraussetzung dafür ist, daß die Anforderungen bestimmte syntaktische Merkmale aufweisen. Die Anforderungen können dann anhand gegebener, domänenabhängiger Regeln untersucht werden. Im Gegensatz zu den vorher vorgestellten Ansätzen werden die informalen Anforderungen strukturiert, nicht aber in eine formale Notation überführt.

Ein wichtiger Vorteil der parsenden Ansätze ist, daß sie die Inhalte der Freitexte explizit berücksichtigen. Es können viele Inkonsistenzen und Mehrdeutigkeiten automatisch entdeckt werden. Wenn z.B. zu einem Indikator nicht eindeutig angegeben werden kann, worauf sich der Indikator bezieht, dann gibt es mehrere mögliche Parse-Bäume. Das wiederum ist ein klares Indiz für eine Mehrdeutigkeit. Beispiel: »Das Zielsystem kommuniziert mit dem Datenbanksystem. Es wartet auf einen Event.« Bezieht sich der Indikator »es« jetzt auf »Zielsystem« oder »Datenbanksystem«?

Auch Unvollständigkeiten können teilweise erkannt werden, z.B. der fehlende Aktor in folgender Anforderung: »Jedes Datum soll mit vierstelligen Jahreszahlen gespeichert werden«. Wer soll die Daten speichern? Das Zielsystem? Wenn es mehrere gibt: welche der Zielsysteme? Alle? Unvollständigkeiten, die sich auf die Wünsche der Informanten beziehen, können jedoch nicht erkannt werden.

Den Vorteilen stehen aber auch einige Nachteile gegenüber:

- Es gibt heutzutage keinen Parser für natürliche Sprache, der beliebige Texte »verstehen« kann. Der Systemanalytiker muß sich bei der Formulierung des Textes den Einschränkungen des Parsers unterwerfen. Es kann störend sein, wenn er eine richtige und gut formulierte Anforderung eingeben möchte, sie aber umformulieren muß, nur weil der Parser sie nicht akzeptiert.
- Jeder Parser benutzt ein linguistisches Lexikon. Das Lexikon muß umfassend sein. Jedes Wort, das in den Texten benutzt wird, muß im Lexikon vorhanden sein, oder aber es müssen im Lexikon Flexionsregeln angegeben sein, wie aus einem vorhandenen Wort das gesuchte Wort gebildet werden kann. Fehlt das Wort, liefert der Parser eine Fehlermeldung.

- Selbst wenn für den Anwendungsbereich ein sorgfältig erstelltes linguistisches Lexikon vorliegt, muß es im Projektverlauf angepaßt werden, weil viele Terme erst während des Projekts gebildet werden. Das linguistische Lexikon muß also während des Projekts erweitert werden können. Da zu jedem Eintrag morphologische, syntaktische und semantische Informationen angegeben werden müssen, kann der Eintrag nur von einem versierten Linguisten vorgenommen werden.
- Ziel der meisten der hier vorgestellten Ansätze ist es, aus den natürlichsprachlichen Vorgaben ein formales Modell zu erstellen. Fehlerhafte Eingaben werden dabei nicht toleriert. Entdeckt der Parser einen Fehler, kann das formale Modell nicht erzeugt werden. Mängel können nicht verwaltet werden. Sie müssen sofort behoben werden, auch wenn das, wie in Kapitel 4.1 argumentiert wurde, oft nicht möglich ist.

Generierende Ansätze

Die Grundidee in den vorhergehenden Ansätzen war, natürliche Sprache zu parsen und in eine mehr oder weniger formale Notation zu überführen. Im folgenden werden Ansätze beschrieben, in denen die umgekehrte Vorgehensweise vorgeschlagen wird: Ausgehend von einem Anforderungsdokument in einer formalen Notation wird natürliche Sprache generiert, damit es von den Kunden validiert werden kann. Man sagt auch: »Das formale Modell wird paraphrasiert«.

In (Dalianis, 1992) wird ein System beschrieben, das ER-Modelle verwalten kann. Der Benutzer kann in sehr eingeschränkter natürlicher Sprache Anfragen stellen, z.B. »What is a person?«. Das System durchsucht das ER-Diagramm nach einer entsprechenden Entität und generiert natürlichsprachliche Aussagen, die die Einbettung der »Person« in das ER-Diagramm beschreiben. Wenn z.B. eine Ist-Ein-Beziehung zwischen den Entitäten *person* und *owner* vorliegt, dann wird folgende Ausgabe generiert: »Some persons are owners.« Wenn die Entität *person* das Attribut *address* hat, wird die Ausgabe »A person can have exactly one address.« generiert (Dalianis, 1992, S. 431).

In (Salek et al., 1994) werden das Metaview- und das REVIEW-System beschrieben. Bei dem Metaview-System handelt es sich um einen Generator für Spezifikationswerkzeuge. Das REVIEW-System erweitert das Metaview-System um Textgenerierungsfunktionalität. Als Beispiel wird in dem Papier ein instanziertes SA-Werkzeug behandelt. Zur Validierung wird aus SA-Diagrammen ein natürlichsprachlicher Text generiert. Einige Auszüge: »Library holdings are data stores whose form is a disk file, whose access method is random, and whose cardinality is 4. [...] Database access is a process which changes and reads holdings info. [...]« (Salek et al., 1994, S. 228).

Der Vorteil der generierenden Ansätze ist, daß auch solche Personen die Anforderungsdokumente validieren können, die die formale Notation nicht verstehen. Insbesondere im zweiten Fall (REVIEW-System) müssen sie jedoch mit den der formalen Notation zugrundeliegenden Konzepten vertraut sein, also z.B. mit »Speicher« und »Prozeß«. Der Vorteil beschränkt sich darauf, daß sie die Syntax nicht kennen müssen.

Wenn ein formales Modell vollständig paraphrasiert wird, entsteht ein umfangreicher, sehr unübersichtlicher Text. Ein generierendes Werkzeug ist nicht in der

Lage zu unterscheiden, welche Details wichtig sind und welche nicht. Auch einfache Modelle verfügen über lange natürlichsprachliche Repräsentationen.

Da die natürlichsprachliche Repräsentation maximal so viel Information enthalten kann wie das zugehörige formale Modell, gelten die in Kapitel 2.4 beschriebenen Einschränkungen der Ausdrucksmächtigkeit formaler Notationen auch für generierte Texte.

Parsende und generierende Ansätze

In der Literatur gibt es auch Ansätze, in denen parsende und generierende Ansätze kombiniert werden.

In (Rolland, Proix, 1992) wird ein Ansatz zur Spezifikation von Informationssystemen und das zugehörige Werkzeug OICSI bzw. ALECSI vorgestellt. Die natürlichsprachlichen Anforderungen werden mittels linguistischer Analysen in ein ER-Diagramm (»conceptual schema«) überführt. Zur Validierung wird aus dem ER-Diagramm ein natürlichsprachlicher Text generiert.

In (Dankel, et al., 1992) wird das System GATOR (*Gat*herer *o*f *R*equirements) vorgestellt. Der Systemanalytiker kommuniziert mit dem System über eine Benutzungsschnittstelle, die natürlichsprachliche Anweisungen verstehen kann. Auf diese Weise können Anfragen an den aktuellen Datenbestand und Änderungen durchgeführt werden. Dem System zugrunde liegt eine Wissensdatenbank, angefüllt mit Domänenwissen aus dem Bereich der Telekommunikation, spezialisiert auf Telefonanrufe und damit verbundene Funktionalitäten.

Die Kommentare über die parsenden und generierenden Ansätze gelten auch für die hier beschriebenen Ansätze.

Eingeschränkte Sprachen

In diesem Kapitel wird die Idee der eingeschränkten natürlichen Sprachen (»controlled languages«) vorgestellt. Den Ansätzen liegt die Überlegung zugrunde, daß ein Anforderungsdokument nur einmal geschrieben, aber sehr oft gelesen wird. Demzufolge ist es wirtschaftlich gerechtfertigt, den Erstellungsaufwand zu erhöhen, wenn im Gegenzug der Leseaufwand reduziert wird.

Grundidee ist, die natürliche Sprache (in den meisten Fällen Englisch) deutlich zu vereinfachen, sowohl den Wortschatz als auch die Grammatik.

Dieser Ansatz ist für den Verfasser eines Anforderungsdokuments gewöhnungsbedürftig, weil er sich erst auf die eingeschränkte Sprache einstellen muß. Er darf bestimmte, ansonsten erlaubte und vielleicht sogar als gut empfundene Sprachkonstrukte nicht benutzen. Er muß die eingeschränkte Sprache regelrecht lernen. Werkzeuge können dabei sehr hilfreich sein.

Für den Leser eines Anforderungsdokuments hat dieser Ansatz einige Vorteile:

- Er versteht die Sprache sofort, weil es sich um eine echte Teilmenge der natürlichen Sprache handelt. Er muß nichts neu lernen. Da sie in der Regel deutlich einfacher als die natürliche Sprache ist, ist sie auch für Nicht-Muttersprachler leicht zu verstehen.

- Die Anforderungsdokumente sind einfacher geschrieben und damit auch einfacher zu lesen und zu verstehen als beliebige natürlichsprachliche (englische) Texte. Probleme, die durch die Verwendung von Synonymen, Mehrdeutigkeiten, indirekten Querverweisen oder Füllwörtern entstehen, werden umgangen.
- Da mehr Aufwand in die Erstellung der Anforderungsdokumente gesteckt wird, hoffen die Autoren der eingeschränkten Sprachen, daß sich die Verfasser intensiver mit den Inhalten und der Sprache beschäftigen und somit die Qualität der Anforderungsdokumente insgesamt besser wird.

In der Literatur werden verschiedene eingeschränkte (englische) Sprachen vorgestellt: z.B. Simplified English (Kniffin et al., 1989), Attempto (Fuchs, Schwitter, 1995 und 1996), PROMIS-Ansatz (Lu, Jin, Wan, 1995) und ANLT (Osborne, MacNish, 1996).

In Attempto z.B. sind folgende Satzkonstruktionen erlaubt:

- einfache aktive Sätze, die aus Subjekt, Prädikat und Objekt bestehen,
- Bedingungssätze, die mit »If« anfangen, gefolgt von einem Hauptsatz,
- Fragen, auf die mit »Yes«/»No« geantwortet werden kann,
- Fragesätze, die mit einem wh-Fragewort (»where«, »who«, »what« ...) eingeleitet werden,
- Relativsätze, die das Subjekt oder Objekt betreffen,
- Negation von Nominalphrasen und
- Verknüpfungen mittels »and« und »or«.

Anaphorische Referenzen (Vorwärtsreferenzen) sind verboten.

Eingeschränkte Sprachen werden teilweise in großem Maßstab eingesetzt, Simplified English z.B. in der U.S. Air Force, bei Boeing und bei McDonnell-Douglas.

Der Anwendungsbereich der eingeschränkten Sprachen erstreckt sich nicht nur auf Anforderungsdokumente, sondern auf beliebige technische Dokumente. Die Qualität der technischen Dokumentation soll ganz allgemein verbessert werden. Spezifika von Anforderungsdokumenten werden nicht berücksichtigt.

Die Anmerkungen für parsende Ansätze gelten für eingeschränkte Sprachen als Spezialfälle der parsenden Ansätze ebenfalls.

Da eine eingeschränkte Sprache und die zugrundeliegende natürliche Sprache große Ähnlichkeiten aufweisen, ist ein Werkzeug zur Syntaxprüfung absolut notwendig. Jemand, der eine Sprache beherrscht, wird nicht so ohne weiteres bestimmte Teile der Sprache wieder verlernen, zumal er sie ja im »normalen« Leben weiterbenutzt.

In (Kniffin et al., 1989, S. 1137) werden einige Möglichkeiten beschrieben, wie ein Werkzeug den Systemanalytiker bei der Eingabe von Simplified-English-Texten unterstützen kann:

- Während des Schreibens wird geprüft, ob die benutzten Wörter im Lexikon definiert sind. Sobald ein nicht definiertes Wort benutzt wird, wird es markiert. Im Gegensatz dazu werden vom ADMIRE-Werkzeug die benutzten Wörter und Phrasen markiert.

- Es wird ein *inverser Thesaurus* verwaltet, in den oft falsch benutzte Wörter und Verweise auf die richtigen Wörter eingetragen werden können. Benutzt der Verfasser eines der falschen Wörter, schlägt der inverse Thesaurus vor, welches Wort statt dessen benutzt werden soll. Ähnliche Informationen stehen auch im ADMIRE-Ansatz zur Verfügung. Die »falschen« Wörter sind hierbei verworfene Glossareinträge. Die »richtigen« Wörter sind aktuelle Synonyme der verworfenen Glossareinträge.

10.2 Ergänzende Ansätze

10.2.1 Szenarien/Use Cases

Ein wichtiger Aspekt bei der Spezifikation eines Systems sind Arbeitsabläufe, also die Interaktionen zwischen dem System und seiner Arbeitsumgebung. Hierfür werden in den letzten Jahren verstärkt *Use Cases* (*Szenarien)* eingesetzt (Jacobson et al., 1995, Schneider, Winters, 1998).

Durch ein Szenario wird eine Sequenz von Aktionen (Ereignissen) beschrieben. Eine Aktion kann entweder von dem Zielsystem oder von einem System oder einer Person aus seiner Arbeitsumgebung ausgelöst werden. Wichtig hierbei ist, daß nicht eine einzelne Funktion oder ein einzelner Arbeitsschritt beschrieben wird, sondern ein ganzer Arbeitsablauf. Die Aktionen werden durch (strukturierte) natürliche Sprache beschrieben. Beispiele für solche Szenarien sind die »typischen Benutzungsszenarien« für das ADMIRE-Werkzeug in Kapitel 9.3.

Die größte Stärke der Szenarien ist, daß die Benutzung des Zielsystems und die Arbeitsabläufe im Vordergrund stehen. Für die Benutzer, die die Szenarien validieren sollen, ist diese Sichtweise sehr geeignet. Sie können gut abschätzen, wie das Zielsystem in ihre Arbeitsabläufe integriert wird.

Im Gegensatz zum ADMIRE-Ansatz, in dem versucht wird, möglichst atomare Informationseinheiten zu erstellen, werden in einem Szenario Informationen verschiedener Arten vermischt, insbesondere Abläufe, Ereignisse, Daten und Beschreibungen der Benutzungsoberfläche. Einige Informationen können trotz Verwendung der natürlichen Sprache nicht (sinnvoll) durch Szenarien beschrieben werden, z.B. bestimmte nicht-funktionalen Anforderungen. Anforderungen an die Portabilität des Zielsystems oder den prinzipiellen Aufbau der Benutzungsschnittstelle können schlecht innerhalb von Abläufen beschrieben werden. Konsistenz zwischen den einzelnen Szenarien ist schwer überprüfbar. Zusammengehörende Informationen sind oft über viele Szenarien verstreut.

Da auch in Szenarien hauptsächlich natürliche Sprache verwendet wird, könnten sie in den ADMIRE-Ansatz integriert werden, wobei aber einige offene Fragen bzgl. der Verwaltung der Szenarien und des Zusammenhangs zwischen Szenarien und den vorhandenen Eintragstypen zu klären sind. Einige der Prüfverfahren könnten direkt oder in abgewandelter Form auch auf Szenarien angewandt werden.

10.2.2 Notationszentrierte Werkzeuge

Wie die Diskussion in Kapitel 2.3.5 zeigt, gibt es viele (semi-) formale Spezifikationsnotationen. Für viele dieser Notationen gibt es »notationszentrierte« Werkzeuge, deren primärer Zweck darin besteht, Modelle in den jeweiligen Notationen zu erstellen und zu verwalten. Bekannte Werkzeuge sind z.B. Rational Rose (Rational Software Corporation), das die UML unterstützt, Innovator (MID), das verschiedene graphische Notationen unterstützt, u.a. UML, SA und ER, oder das SA-Werkzeug TeamWork (Sterling Software). Viele der (semi-) formalen Notationen sind graphisch (z.B. UML, SA), oder verfügen über ein reichhaltiges Repertoire an Nicht-Standard-Symbolen (z.B. Z).

Den notationszentrierten Werkzeugen ist gemeinsam, daß die (semi-) formale Notation im Vordergrund steht. Kommentare in Form von Freitexten können zu einzelnen Komponenten oder Diagrammen angegeben werden. Es besteht oft auch die Möglichkeit, Fremdtexte und -bilder zu importieren. Im Innovator z.B. müssen alle Informationen, die nicht durch die unterstützen Diagrammtypen beschrieben werden können, als Kommentare oder sog. externe Kapitel angegeben werden. Darunter fallen auch so wichtige Informationen wie z.B. nicht-funktionale Anforderungen und Schnittstellenbeschreibungen.

Die Prüfungen, die diese Werkzeuge anbieten, beziehen sich nur auf die unterstützte Notation: syntaktische und einfache kontextsensitive Prüfungen. Kommentare, importierte Texte und Bilder werden dabei nicht berücksichtigt. Mängel können nicht verwaltet werden.

Als ausschließliche Werkzeuge für die Requirements-Engineering-Phase eignen sich diese Werkzeuge nicht, weil wichtige Konzepte wie z.B. »Anforderungen« nicht unterstützt werden. Sie können aber gut als Ergänzung verwendet werden, um bestimmte Teilaspekte zu modellieren.

10.2.3 Ansätze zur Erstellung von Glossaren

Im folgenden werden einige Ansätze zur Erstellung von Lexika (Glossaren) und zum Auffinden potentieller Terme aus den Forschungsbereichen Linguistik und Requirements Engineering vorgestellt.

Ein in der Linguistik oft angewandtes Verfahren bei der Erstellung von Fachlexika ist die *Textkorpora-Analyse*. Voraussetzung ist, daß eine Menge repräsentativer Fachtexte (sog. *Textkorpora*) vorliegt. Dabei wird angenommen, daß Fachterme in den Fachtexten häufig benutzt werden. Die Fachtexte werden daraufhin untersucht, welche Wörter wie oft benutzt werden. Besonders häufig benutzte Wörter sind Kandidaten für solche Fachterme.

Die so ermittelte Kandidatenliste muß anschließend überarbeitet werden:

- Flexionen des gleichen Wortstamms werden zusammengefaßt.
- Häufige Wörter der zugrundeliegenden natürlichen Sprache wie z.B. Artikel und Präpositionen werden herausgefiltert.

Die reduzierte Liste wird von Fachleuten inspiziert. Die Entscheidung, ob ein Kandidat auch wirklich ein Fachterm ist, kann nur von Spezialisten des jeweiligen Fachgebiets getroffen werden.

Durch das bisher beschriebene Verfahren können nur Einwort-Terme gefunden werden. Typischerweise kommen in Fachsprachen aber auch Mehrwort-Terme vor. Auch Kandidaten für Mehrwort-Terme können aus den Textkorpora ermittelt werden, indem die Wörter nicht isoliert betrachtet werden, sondern geprüft wird, ob bestimmte Wörter gehäuft im näheren Umfeld von anderen Wörtern vorkommen.

Je größer die vorliegenden Textkorpora sind, desto genauer können die vorgenommenen Analysen sein. In (Bergenholtz, Pedersen, 1994) werden Untersuchungen von Textkorpora mit einem Umfang von bis zu 2,5 Millionen Wörtern beschrieben.

Die Textkorpora können den Fachlexikographen auch bei der Definition der Terme unterstützen. Er kann sich die Benutzungsstellen der Fachterme anschauen und versuchen, daraus Hinweise auf ihre Bedeutung zu finden. Diese Tätigkeiten werden oft auch durch spezielle Programme zur Verwaltung und Analyse des umfangreichen Textmaterials unterstützt (Weber, 1994).

Eine ähnliche Idee liegt auch dem Requirements-Engineering-Werkzeug AbstFinder (Goldin, Berry, 1994, Aguilera, Berry, 1990) zugrunde. Die Sätze in einem vorliegenden natürlichsprachlichen Anforderungsdokument werden jeweils paarweise verglichen. Gemeinsame Teilstrings, sog. Runs, werden identifiziert. Die Teilstrings müssen buchstabengenau übereinstimmen. Flexionen werden nicht erkannt.

Gegeben seien z.B. folgende Sätze:

1. »Die Priorität der Anforderung A1 ist hoch.«
2. »Das Attribut Priorität kann folgende Werte annehmen: hoch, niedrig.«
3. »Jeder Anforderung muß eine Priorität zugeordnet werden.«

Wenn der erste Satz mit dem zweiten verglichen wird, werden die Runs »Priorität« und »hoch« gefunden. Vergleicht man die Sätze 1 und 3, erhält man die Runs »Anforderung« und »Priorität«.

Das Werkzeug erzeugt einen Report, in dem für jeden Satz die mit mindestens einem anderen Satz gemeinsamen Phrasen ausgegeben werden. Für Satz 1 würde folgender Eintrag generiert: »Priorität hoch | Priorität Anforderung«. Die Analysen können konfiguriert werden: Für die Runs kann eine Mindestlänge vorgegeben werden und es kann eine Liste mit zu ignorierenden Phrasen angegeben werden.

Aufgabe des Systemanalytikers ist es, den Report durchzuarbeiten und wichtige Abstraktionen herauszuarbeiten. Diese Abstraktionen können als Kandidaten für Terme des Glossars oder, falls eine objektorientierte Analyse durchgeführt wird, als Kandidaten für Klassen, Assoziationen und Attribute verwendet werden. Das Werkzeug selbst findet die Abstraktionen nicht, es bereitet den zugrundeliegenden Text »nur« auf hilfreiche Weise auf.

In den hier beschriebenen Ansätzen wird von vorhandenen Texten ausgegangen, die analysiert werden, um Terme zu finden. In ADMIRE wird ein umgekehrter Ansatz verfolgt. Die Freitexte werden schon bei der Eingabe mit dem (vorhandenen) Glossar halbautomatisch abgeglichen.

Die Vorgehensweisen lassen sich gut kombinieren. Die im ADMIRE-Ansatz beschriebenen fokussierten Inspektionen werden jeweils bei Eingabe, Änderung oder Anzeigen eines Freitextes ausgeführt. Aus den Markierungen oder dem Ausblei-

ben der Markierungen kann der Systemanalytiker auf begriffliche Inkonsistenzen in dem Freitext oder auf Unvollständigkeiten im Glossar schließen. Die Textkorpora-Analyse kann zur (nachträglichen) Prüfung eines Anforderungsdokuments eingesetzt werden. Der Systemanalytiker wird auf häufig benutzte, im Glossar aber (noch) nicht definierte Phrasen aufmerksam gemacht.

10.2.4 Textverständlichkeit

Einige Autoren schlagen vor, die Verständlichkeit eines Freitextes anhand seiner syntaktischen Eigenschaften zu messen. Reiners (1996, S. 220, »Prüfe den Stil mit dem Zollstock«) verwendet folgende Maße:

- Anzahl Wörter pro Satz (W)
- Anzahl aktiver Verben je 100 Wörter (AV)
- Anzahl Personenbezeichner je 100 Wörter (PB)
- Anzahl abstrakter Hauptwörter je 100 Wörter (AH)

Aufgrund Reiners subjektiver Meinung können die Werte auf folgende Weise mit der gewünschten Größe »Textverständlichkeit« korreliert werden:

	W	AV	PB	AH
sehr leicht verständlich	1-13	> 14	> 12	0-4
leicht verständlich	14-18	13-14	10-11	5-8
verständlich	19-25	9-12	6-9	9-15
schwer verständlich	25-30	7-8	3-5	15-20
sehr schwer verständlich	> 30	0-6	0-2	> 20

Tabelle 44: Zuordnung der Maße W, AV, PB, AH zur Textverständlichkeit

In den Forschungsbereichen Psychologie und technische Dokumentation wurde dieser intuitive Ansatz erweitert und verfeinert. Es wurden Lesbarkeitsformeln aufgestellt, die z.B. folgende Parameter verwenden: die Wortlänge, die Anzahl Silben pro Wort, die durchschnittliche Satzlänge in Wörtern und Zeichen, den relativen Anteil langer Wörter oder den relativen Anteil Wörter, die nicht zu den häufigsten 3000 Wörtern der Sprache gehören. In (Lehner, 1994, S. 145ff) und (Groeben, 1982, S. 175ff) werden die wichtigsten Lesbarkeitsformeln vorgestellt. Im folgenden werden als Beispiele der Reading-Ease-Index und der Fog-Index beschrieben.

Der Reading-Ease-Index (*RI*) von Flesch (1948) berechnet sich nach folgender Formel:

$$RI = 206{,}85 - 0{,}846 \cdot WL - 1{,}105 \cdot SL$$

mit:

- *WL*: durchschnittliche Anzahl Silben pro 100 Wörter
- *SL*: durchschnittliche Satzlänge

Die Konstanten 206,85, 0,846 und 1,105 wurden empirisch ermittelt. *RI* nimmt typischerweise Werte zwischen 0 und 100 an, wobei 0 als minimal verständlich und 100 als optimal verständlich gilt (Groeben, 1982, S. 177). Der Reading-Ease-Index ist für englische Texte ausgelegt. Für deutsche Texte müssen die Konstanten angepaßt werden.

Der Fog-Index (*FI*) von Gunning (1968) ist folgendermaßen definiert:

$$FI = 0{,}4 \cdot (W + L)$$

mit

- *W*: Durchschnittliche Satzlänge (Anzahl der Wörter geteilt durch die Anzahl der Sätze in einem etwa 100-200 Wörter langen, statistisch ausgewählten Textabschnitt).
- *L*: relativer Anteil von Wörtern mit mehr als drei Silben, wobei Wörter, die mit Großbuchstaben beginnen und Verben mit den Endungen »-ed« und »-es« nicht mitgezählt werden.

Für technische Dokumente ist ein Wert von 12-16 akzeptabel (siehe Lehner, 1994, S. 167). Nach Fenton und Pfleeger (1994, S. 358) entspricht der Fog-Index der Anzahl Jahre Schulbildung, die jemand braucht, um den Text verstehen zu können. Auch hier gilt wieder, daß die Formel ursprünglich für englische Texte gedacht war und für deutsche Texte angepaßt werden muß.

Die Lesbarkeitsformeln können auf zwei Weisen verwendet werden:

- als Warnindex für unverständliche Freitextpassagen, und
- um Folgerungen für einen verständlichen Schreibtstil zu ziehen.

Wenn der Wert einer der Lesbarkeitsformeln »ungünstig« ist, liegt ein relativ starkes Indiz dafür vor, daß der zugrundeliegende Freitext schwer verständlich ist. Bei Anforderungsdokumenten müssen aber einige Besonderheiten beachtet werden. Die Terme des Glossars sind fachspezifische Terme, also in vielen Fällen komplizierte und lange Wörter. Wenn in der Formel ein Zusammenhang zwischen der Wortlänge und der Verständlichkeit enthalten ist, müssen die Terme des Glossars besonders behandelt werden. Wenn der relative Anteil der Wörter, die nicht zu den häufigen Wörtern einer Sprache gehören, in die Formel eingeht (wie z.B. in der Dale-Chall-Formel), dann müssen die Terme des Glossars zu den häufigen Wörtern gezählt werden, weil sie als verständlich angenommen werden.

Der umgekehrte Schluß, daß aus einem »günstigen« Wert leichte Verständlichkeit folgt, gilt in vielen Fällen nicht. Für Verständlichkeit spielt auch der Inhalt des Textes eine entscheidende Rolle, und der wird in den Formeln vollständig ignoriert. Ebenso werden die Vorkenntnisse der Leser nicht berücksichtigt.

Ein weiteres Problem mit den Lesbarkeitsformeln ist, daß unterschiedliche Formeln, angewandt auf den gleichen Text, oft unterschiedliche Ergebnisse liefern. In (Lehner, 1994, S. 161) werden die Unterschiede anhand einer Feldstudie aufgezeigt. Die Formeln messen weniger die Variable »Textverständlichkeit« als eher »Satzkomplexität« und »Wortschwierigkeit«.

Die Lesbarkeitsformeln sind objektiv und empirisch belegt (siehe Groeben, 1982, S. 183ff) und können teilweise automatisch erhoben werden. Das ADMIRE-Werkzeug könnte um Inhaltsprüfungen erweitert werden, die solche Lesbarkeitsfor-

meln anwenden. Die Formeln müßten aber an die o.g. Besonderheiten von Anforderungsdokumenten und an die deutsche Sprache angepaßt werden. Es müssen aber noch einige offene Punkte geklärt werden: Wie kann die Anzahl der Silben eines Wortes automatisch ermittelt werden? Welche Wörter gehören zu den häufigsten Wörtern der (für ein Projekt relevanten) Fachsprache?

10.2.5 Freitext-Analysen

Ergebnisse der Forschungen in den Bereichen Textverständlichkeit (Groeben, 1982, S. 185) und technische Dokumentation (Lehner, 1994, S. 51) sind, neben den Lesbarkeitsindizes, Stilregeln für verständliches Schreiben, z.B.:

- Es sollen möglichst kurze und geläufige Wörter benutzt werden.
- Terme sollen einheitlich benutzt werden (»Begriffshygiene«).
- Fremdwörter sollen möglichst selten benutzt werden. (In Fachsprachen bzw. projektspezifischen Sprachen zählt die Fach- oder Projektterminologie nicht zu den Fremdwörtern. Diese Wörter sollen benutzt werden.)
- Passivsätze sollen vermieden werden.
- Die Sätze sollen möglichst kurz sein. Die wichtigsten Informationen sollen am Satzanfang stehen.
- Sätze sollen aus einem Hauptsatz und maximal einem Nebensatz bestehen. Die wichtige Information soll im Hauptsatz enthalten sein.
- Einzelne Textabschnitte sollen in sich abgeschlossen sein, also möglichst keine Indikatoren enthalten, die auf andere Textabschnitte verweisen.

Sommerville und Sawyer (1997, S. 148f) fügen Requirements-Engineering-spezifische Stilregeln hinzu:

- In einem Satz soll niemals mehr als eine Anforderung enthalten sein. Das Wort »and« z.B. deutet auf mehrere Anforderungen in einem Satz hin.
- Die Terminologie soll konsistent benutzt werden, insbesondere Wörter wie »shall«, »should«, »will« und »must«. Jargon, Abkürzungen und Akronyme sollen nur benutzt werden, wenn sie von allen erwarteten Lesern verstanden werden oder im Glossar definiert sind.
- Natürlichsprachliche Aussagen sollen keine komplizierten oder geschachtelten Bedingungsstrukturen oder Zusammenhänge enthalten. Dafür sollen geeignetere Notationen verwendet werden.
- Die Einhaltung der Rechtschreibungs-, Zeichensetzungs- und Grammatikregeln ist für das Verständnis ebenfalls wichtig.

Die Einhaltung der wenigsten Stilregeln läßt sich automatisch prüfen. Es besteht aber die Möglichkeit, automatisch nach gewissen Indizien zu suchen, die auf eine Verletzung hinweisen. Im ADMIRE-Ansatz gibt es einige Inhaltsprüfungen, die nach solchen Indizien suchen, z.B. nach benutzten Konjunktionen, Indikatoren oder nach falsch benutzten Modalverben.

In (Wilson, Rosenberg, 1997 und Wilson, 1997) wird ein Werkzeug beschrieben, das natürlichsprachliche, englische Anforderungsdokumente automatisch analy-

siert. Ziel ist es, anhand einfacher Indizien Rückschlüsse auf die Qualität des Anforderungsdokuments zu ziehen. Zum einen wird nach bestimmten benutzten Wörtern gesucht, z.B. nach Imperativen (»shall«, »must«), Direktiven (»tables«, »figures«), WeakWords (»effective«, »easy«) und Optionen (»can«, »may«). Zum anderen werden einige Maße auf dem gesamten Dokument erhoben, z.B.:

- Umfang in Textzeilen und Anzahl Imperative. Aus der Anzahl der Imperative wird auf die Anzahl der Anforderungen geschlossen.
- Textstruktur bzw. Tiefe der Kapitelhierarchie.
- einige Lesbarkeitsmaße, z.B. Fleschs Reading-Ease-Index.

Das Werkzeug wird im Goddard Space Flight Center, genauer gesagt im Software Assurance Technology Center, eingesetzt. Der Autor gelangt zu der Schlußfolgerung, daß die automatischen Analysen aussagekräftige Bewertungen der Qualität von Anforderungsdokumenten liefern.

Im ADMIRE-Ansatz werden ähnliche Analysen durchgeführt, z.B. werden Freitexte auf die Benutzung von WeakWords untersucht. Ziel ist aber nicht eine nachträgliche Beurteilung der Qualität, sondern eine Suche nach Mängeln.

In (Rupp, Götz, 1997) wird ein Ansatz zur Analyse natürlichsprachlicher Anforderungsdokumente vorgestellt, in dem Ansätze aus der Linguistik und der Psychologie (Neurolinguistisches Programmieren) kombiniert werden. Die in einem Anforderungsdokument enthaltenen Sätze werden bestimmten linguistischen Auswertungen unterzogen. Typische Mängel, nach denen gesucht wird, sind Tilgungen, Generalisierungen und Verzerrungen. Der Ansatz definiert Vorgehensweisen, um solche Mängel zu erkennen.

An einem Beispiel soll gezeigt werden, wie Tilgungen erkannt werden können. In dem Satz »Das System soll Datenverluste melden.« kommt das Verb »melden« vor. Zu diesem Verb können folgende Fragen gestellt werden: »Wer meldet?«, »Was wird gemeldet?«, »Wem wird es gemeldet?«, »Wie wird es gemeldet?«, »Wo wird es gemeldet?«, »Wann wird es gemeldet?«, »Wie lange wird es gemeldet?«. Der obige Beispielsatz läßt die meisten dieser Fragen unbeantwortet. Damit liegen Indizien für Mängel vor.

Die Analysen sind sehr aufwendig und eignen sich nur für kurze, ca. zwei- bis dreiseitige Dokumente, z.B. Ausschreibungen. Ein mehrere hundert Seiten umfassendes Anforderungsdokument auf diese Weise zu analysieren, ist mit vertretbarem Aufwand nicht möglich. Einige der Analysen, z.B. die Suche nach Quantoren, können automatisiert werden und wurden in den ADMIRE-Ansatz integriert.

10.3 Bewertung

10.3.1 Zusammenfassung: Stärken und Schwächen der Arbeit

Ein zentrales Ziel dieser Arbeit war es, einen Requirements-Engineering-Ansatz zu entwickeln, der (auch) in Projekten in der Industrie einsetztbar sein soll. Aus den Ergebnissen einer im Rahmen dieser Arbeit durchgeführten Untersuchung in Industrieprojekten und aus Aussagen und Argumenten in der Literatur habe ich geschlossen, daß in vielen Projekten die natürliche Sprache als Hauptnotation zur

Spezifikation eingesetzt wird und daß ein Umstieg auf formalere Notationen oft nicht möglich oder gewollt ist. Deswegen ist die natürliche Sprache ein zentraler Bestandteil des ADMIRE-Ansatzes.

Als wichtigste Errungenschaften dieser Arbeit sehe ich folgende:

- Der ADMIRE-Ansatz ist universell einsetzbar. Er kann prinzipiell zur Spezifikation jedes beliebigen Systems verwendet werden, weil durch die natürliche Sprache beliebige Sachverhalte ausgedrückt werden können.
- Vom ADMIRE-Ansatz wird eine dokumentzentierte Arbeitsweise unterstützt. Das Informationsmodell schreibt vor, daß alle Einträge in Dokumenten organisiert werden. Die »normale« Arbeit mit ADMIRE besteht darin, Dokumente zu erstellen und zu bearbeiten. Systemanalytiker, die die Ergebnisse der Requirements-Engineering-Tätigkeiten bisher z.B. mit Textverarbeitungsprogrammen dokumentiert haben, können ihre Arbeitsweise zu einem großen Teil beibehalten.

  Die dokumentzentrierte Sicht kann aber auch verlassen werden, z.B. bei der Durchführung von Prüfverfahren oder bei der Anwendung von Filtern. In diesen Fällen wird der Inhalt eines Dokuments als (unstrukturierte) Menge von Einträgen aufgefaßt.
- Im Gegensatz zu vielen anderen Ansätzen für das Requirements-Engineering bietet ADMIRE die Möglichkeit, unvollständige, inkonsistente oder ganz allgemein mangelhafte Informationen einzugeben, zu verwalten und die Mängel zu dokumentieren. Mängel sind ein integraler Teil einer Dokumentation. Das spiegelt die Situation in realen Projekten wider, daß es in vielen Fällen nicht möglich ist, Mängel sofort zu beheben.
- Der ADMIRE-Ansatz ist flexibel. Das Informationsmodell wurde so entworfen, daß der Systemanalytiker beim Arbeiten mit ADMIRE möglichst wenig behindert wird. Dokumente können in beliebiger Reihenfolge erstellt werden. Einträge werden auch dann akzeptiert, wenn sie unvollständig sind oder sonstige Mängel aufweisen[1]. Prüfungen werden nur auf Verlangen des Systemanalytikers durchgeführt.
- Qualität spielt im ADMIRE-Ansatz eine sehr wichtige Rolle. Das Informationsmodell hat einige Eigenschaften, die den Systemanalytiker bei der Erstellung einer qualitativ »guten« Dokumentation unterstützen:
  - Die projektrelevante Terminologie kann in einem Glossar definiert werden.
  - Zur klaren Trennung von zu realisierendem Zielsystem und vorhandener Arbeitsumgebung können Kontextdiagramme (Systemmodelle) erstellt werden.
  - Es wird klar zwischen Ist-Zustand und Soll-Zustand unterschieden.

---

1. Die Vorgaben des Informationsmodells dürfen aber nicht verletzt werden. Das Informationsmodell enthält hauptsächlich strukturelle Vorgaben, die immer erfüllt sein müssen. Es enthält auch einige Vorgaben bzgl. der Vollständigkeit. So ist es z.B. nicht möglich, einen Eintrag mit einem leeren oder nicht eindeutigen Bezeichner einzutragen. Diese (nicht strukturellen) Vorgaben wurden aber so gewählt, daß nur das minimal Notwendige verlangt wird.

  - Die einzelnen Informationen können klar getrennt werden. Jeder Eintrag hat einen eindeutigen Bezeichner und kann somit eindeutig und einzeln referenziert werden.
  - Durch Attributierung können wichtige Zusatzinformationen zu den einzelnen Informationen erfaßt werden.
  - Durch Querverweise und Kategorisierung kann der Zusammenhang zwischen den Einträgen dokumentiert werden.
  - Aus oben genannten Gründen läßt das Informationsmodell zu, daß in einer Dokumentation bestimmte wünschenswerte Eigenschaften verletzt werden. Für jede dieser »verletzbaren« Eigenschaften wird ein Qualitätskriterium definiert.
- Zur Prüfung, ob eine Dokumentation die Qualitätskriterien erfüllt, wurden neue Prüfverfahren eingeführt und bekannte Prüfverfahren erweitert.
  - Die Ähnlichkeitssuche und fokussierte Inspektion, insbesondere die positive fokussierte Inspektion, sind in keinem mir bekannten anderen Ansatz vorgesehen.
  - Bei der Statusprüfung und der Inhaltsprüfung werden insbesondere auch die Inhalte der Freitexte geprüft, ohne daß, wie in vielen NLP-Ansätzen, die zu verwendende Sprache stark eingeschränkt wird oder ein linguistisches Lexikon vorliegen muß, das morphologische, syntaktische und semantische Informationen über jedes in den Freitexten benutzte Wort enthält.
  - Da wegen des hohen Anteils natürlicher Sprache Mängel nicht immer mit Sicherheit festgestellt werden können, wird in den Inhaltsprüfungen nach Indizien für Mängel gesucht. Die Entscheidung, ob wirklich ein Mangel vorliegt, wird den Systemanalytikern überlassen.
- Zur Präzisierung wurden das Informationsmodell, die Prüfkriterien der Inhaltsprüfungen, die Markierungsfunktionen der fokussierten Inspektionen, die Anzeigefunktionen der Statusprüfungen, die Filter der Inspektionen und die Ähnlichkeitsmaße der Ähnlichkeitssuche formalisiert.

Zur Validierung des ADMIRE-Ansatzes habe ich eine umfangreiche Beispieldokumentation erstellt. Als Anwendungsbeispiel habe ich das SESAM-2-Projekt ausgewählt, weil es, obwohl es ein Forschungsprojekt ist, viele Merkmale aufweist, die auch typisch für industrielle Projekte sind (siehe Kapitel 4.6). Da ich selbst an dem Projekt teilgenommen habe, konnte ich auf damals gewonnene Informationen zugreifen, auch auf interne Notizen und Protokolle.

Die vorhandenen Informationen habe ich benutzt, um eine Dokumentation für das SESAM-2-Projekt zu erstellen. In den früheren Kapiteln wurden Auszüge aus dieser Dokumentation als Beispiele verwendet.

Die Anwendung des ADMIRE-Ansatzes hat gezeigt,

- daß es möglich ist, umfangreiche und qualitativ hochwertige Dokumentationen zu erstellen, und
- daß die Prüfverfahren sinnvolle Indizien für Mängel liefern und den Systemanalytiker bei der Erkennung und Vermeidung von Mängeln effektiv unterstützen.

In der praktischen Anwendung des ADMIRE-Ansatzes haben sich aber auch einige Schwierigkeiten gezeigt:

- Durch die Verwendung natürlicher Sprache können viele Vorgaben des ADMIRE-Ansatzes relativ einfach unterlaufen werden. Der Systemanalytiker wird durch das ADMIRE-Werkzeug z.B. nicht daran gehindert, in einem Inhaltselement oder einer Annotation einen neuen Term einzuführen und zu definieren. In einem solchen Fall liefern die Prüfverfahren, die Freitexte auf benutzte Terme untersuchen, bzgl. dieses Terms keine sinnvollen Ergebnisse.
- Die Aufteilung der zu verwaltenden Informationen in einzelne atomare Einträge ist nicht immer einfach. Handelt es sich z.B. bei folgendem Satz um eine Anforderung oder um drei Anforderungen?

  »Ein SESAM-Modell enthält ein Schemamodell, ein Regelmodell und ein Situationsmodell.«

  Sollte das Inhaltselement also in drei Inhaltselemente aufgesplittet werden?

  »Ein SESAM-Modell enthält ein Schemamodell.«

  »Ein SESAM-Modell enthält ein Regelmodell.«

  »Ein SESAM-Modell enthält ein Situationsmodell.«

  Für beide Möglichkeiten gibt es gute Gründe.
- Wenn ein automatisches Prüfverfahren durchgeführt wird und wenn dabei keine Indizien für Mängel entdeckt werden, so besteht die Gefahr, daß der Systemanalytiker fälschlicherweise darauf schließt, daß die Dokumentation frei von Mängeln ist. Ähnlich wie bei Testverfahren kann durch die Prüfverfahren nur das Vorhandensein von Mängeln, nicht aber deren Abwesenheit festgestellt werden.

Im Zusammenhang mit der Prüfung natürlichsprachlicher Freitexte sind noch einige offene Punkte zu klären:

- In einigen Prüfverfahren wird untersucht, ob in den Freitexten bestimmte Wörter oder Phrasen benutzt werden. Die automatische Erkennung benutzter Wörter und Phrasen ist nicht trivial (siehe Diskussion in den Kapiteln 5.6 und 8.1.2). Das Informationsmodell bietet dem Systemanalytiker zwar die Möglichkeit, durch Angabe von Flexionen in den Erkennungsmechanismus einzugreifen. Einige Sonderfälle können dennoch nicht erkannt werden: z.B. diskontinuierliche Konstituenten (»schickt ab« statt »abschicken«). Bei der Erkennung von Mehrwort-Termen können Fehler auftreten, wenn in einem Freitext alle Wörter des Mehrwort-Terms einzeln benutzt werden, der Mehrwort-Term selbst aber nicht (siehe Kapitel 8.1.2). Bei der Erstellung der SESAM-2-Dokumentation hat sich gezeigt, daß falsche Erkennungen zwar selten sind, aber vorkommen.
- Mehrwort-Terme werden grundsätzlich nicht erkannt, wenn nicht alle Wörter des Terms in dem Freitext benutzt werden. Beispielsweise wird in dem Freitext »Das Attribut ist vom Typ String.« der Term »Attribut eines Dokuments« nicht erkannt. Ein Leser könnte evtl. aus dem Kontext schließen, daß dieser Term gemeint ist. Das Werkzeug verfügt aber nicht über die Möglichkeit, fehlende Informationen zu ergänzen.

- Mängel beziehen sich immer auf Einträge oder Attribute der Einträge, können aber nicht fein granularer angegeben werden. Wenn z.B. in einem Anforderungstext ein WeakWord benutzt wird, wird in einem bei einer Inhaltsprüfung erzeugten Mangeleintrag nicht festgehalten, wie oft oder an welchen Stellen das Wort in dem Freitext benutzt wird. Wenn es zwischen zwei Prüfungen gelöscht und an anderer Stelle im gleichen Freitext wieder eingefügt wird, liegt immer noch der gleiche Mangel vor. Diese Änderung wird bei den Prüfungen nicht festgestellt.

Einige Informationen lassen sich nur umständlich durch natürliche Sprache ausdrücken. Die SESAM-Modelle z.B. liegen in Form von textuellen Dateien vor. Ein wesentlicher Teil der SESAM-2-Dokumentation besteht darin, den Aufbau dieser Dateien zu beschreiben. Die prinzipiellen Zusammenhänge können gut durch natürliche Sprache beschrieben werden. Soll aber die konkrete Syntax angegeben werden, wie das z.B. in einem der ursprünglichen Spezifikationsdokumente des SESAM-Projekts der Fall war, dann eignen sich andere Notationen besser. Daß es prinzipiell möglich ist, andere Notationen in ADMIRE einzuführen, wird durch die graphische Notation für Systemmodelle gezeigt.

Die Spezifika von einigen Arten von Systemen werden durch ADMIRE nicht unterstützt, z.B.:

- die Aufteilung eines eingebetteten Systems in Hardware- und Software-Komponenten (Systementwurf) oder
- extreme Zuverlässigkeits- und Korrektheitsanforderungen bei sicherheitskritischen Systemen.

Die Erstellung der Dokumentation für das SESAM-2-Projekt hat gezeigt, daß der ADMIRE-Ansatz prinzipiell für größere Projekte und umfangreichere Dokumentationen geeignet ist. Der Nachweis, daß der ADMIRE-Ansatz auch im industriellen Umfeld bestehen kann, steht allerdings noch aus.

10.3.2 Ausblick

Der ADMIRE-Ansatz kann noch in vielerlei Hinsicht erweitert werden. Im folgenden werden einige Möglichkeiten aufgezeigt.

Das Informationsmodell könnte um eine *ToDo-Liste* erweitert werden. Die Systemanalytiker hätten dadurch die Möglichkeit, an einer zentralen Stelle Informationen über durchzuführende Tätigkeiten, zu klärende Fragen oder sonstige offene Punkte einzutragen. Eine ToDo-Liste könnte die Zusammenarbeit mehrerer Systemanalytiker erleichtern und würde eine Möglichkeit bieten, den Projektfortschritt besser abzuschätzen.

Oftmals ist es sinnvoll, auf alte Versionen einzelner Einträge, einzelner Dokumente oder der gesamten Dokumentation zurückzugreifen. Wenn das Pflichtenheft am Ende der Spezifikationsphase akzeptiert wird, wird es oft Bestandteil des Vertrags mit dem Kunden. Somit sollte es nicht mehr geändert werden. Andererseits müssen nachträgliche Änderungen möglich sein (siehe Kapitel 2.3.3), sonst wird das Pflichtenheft schnell inaktuell und damit unbrauchbar. Diese Probleme können durch eine *Versionsverwaltung* (configuration management, siehe z.B. (Whitgift, 1991)) gelöst werden. Für die als Vertragsgrundlage dienende Version des Pflichtenhefts würde eine »baseline« angelegt. Dadurch bestünde die Möglichkeit, die

Änderungen zwischen der Vertrags-Version und der aktuellen Version auf einfache Weise festzustellen. Zur Realisierung einer Konfigurationsverwaltung gibt es zwei prinzipielle Möglichkeiten:

- Das ADMIRE-Werkzeug könnte an ein externes Konfigurationsverwaltungswerkzeug angebunden werden.
- Es müßte eine eigene Konfigurationsverwaltung in das ADMIRE-Werkzeug integriert werden.

Im ADMIRE-Ansatz sind Kapitel als einziges Strukturierungsmittel für Anforderungsdokumente vorgesehen. Es könnten weitere Ordnungs- oder *Gliederungsmechanismen* unterstützt werden. Eine Möglichkeit hierfür bestünde darin, die Anforderungen in Hierarchien zu organisieren. Abstraktere Anforderungen würden eher in den oberen Ebenen der Hierarchie enthalten sein, sie verfeinernde konkrete Anforderungen eher in den unteren.

Die generierten Reporte des ADMIRE-Werkzeugs, die im HTML-Format vorliegen, enthalten *Hyperlinks*, z.B. zwischen den in Freitexten enthaltenen Wörtern und Phrasen und den sie definierenden Glossareinträgen, zwischen Haupteinträgen und ihren Ursprüngen und zwischen Mangeleinträgen und den Einträgen, die den Mangel aufweisen. Der Mechanismus der (automatisch generierten) Hyperlinks könnte auch im ADMIRE-Werkzeug realisiert werden. Hyperlinks würden die Navigation innerhalb einer Dokumentation erleichtern.

Die in Kapitel 8 beschriebenen Prüfverfahren könnten erweitert werden:

- Als weitere Inhaltsprüfungen oder fokussierte Inspektionen würden sich z.B. eine *Rechtschreibprüfung* oder die Erkennung von *Passiv*-Sätzen in Inhaltselementen anbieten. Passiv-Sätze sollten in Inhaltselementen vermieden werden, weil der Agierende (oft) nicht genannt wird.
- Die Inhaltsprüfungen und fokussierten Inspektionen könnten so erweitert werden, daß automatisch *Lösungsvorschläge* erzeugt werden. Wird z.B. ein nicht aktueller Glossareintrag in einem Freitext benutzt, dann könnte das ADMIRE-Werkzeug die Benutzung eines aktuellen synonymen Glossareintrags vorschlagen (soweit im Glossar einer enthalten ist).
- Die Ähnlichkeitssuche könnte auf folgende Weisen erweitert werden:

  Bisher funktioniert die Ähnlichkeitssuche so, daß zu einem gegebenen Eintrag die »ähnlichsten« Einträge eines Anforderungsdokuments gesucht und aufgelistet werden. Der Mechanismus könnte auch auf ein gesamtes Dokument angewandt werden, mit dem Ziel, das Dokument in (disjunkte) Mengen von »ähnlichen« Einträgen zu partitionieren. Diese Mengen könnten dann von den Systemanalytikern auf Inkonsistenzen oder Redundanzen untersucht werden.

  Die Ähnlichkeitssuche ist bisher nur auf Inhaltselementen und Annotationen definiert. Sie könnte auch auf weitere Eintragstypen ausgedehnt werden, z.B. auf Glossareinträge und Kapitel, deren Freitexte ebenfalls Widersprüche (untereinander oder zu anderen Einträgen) aufweisen können.

  Das Ähnlichkeitsmaß könnte so erweitert werden, daß nicht nur die gemeinsam benutzten Terme und Kategorien ausgewertet werden, sondern z.B. auch die den Inhaltselementen zugeordneten Systemmodelleinträge.

In einigen Fällen könnte das ADMIRE-Werkzeug durch (einfache) *Interpretationen der Freitexte* den Systemanalytiker beim Ausfüllen der Eingabemasken unterstützen. Wenn er z.B. eine Anforderung schreibt und dabei den Namen eines Zielsystems benutzt, könnte das Werkzeug automatisch einen Verweis (*sysentries*) von dem Inhaltselement auf den Systemmodelleintrag erzeugen. Der Systemanalytiker müßte diese automatisch erzeugten Verweise aber auch wieder entfernen können. Analog könnte das Werkzeug aus der Benutzung bestimmter Wörter oder Phrasen Rückschlüsse auf zu vergebende Kategorien ziehen.

Das Informationsmodell könnte so erweitert werden, daß die Systemanalytiker neue Arten von Inhaltselementen definieren können, um *Anpassungen* vorzunehmen und somit projektspezifische Besonderheiten besser zu berücksichtigen. Die neuen Inhaltselemente könnten sich z.B. durch zusätzliche Attribute von den vorhandenen Inhaltselementen unterscheiden.

In vielen Projektumgebungen liegen schon Anforderungsdokumente vor oder werden von anderen Personen geschrieben, oft mit Textverarbeitungsprogrammen wie z.B. Microsoft Word oder FrameMaker. Für eine konsistente Datenhaltung ist es sinnvoll, *Fremdtexte zu importieren*. Die im Originaldokument enthaltenen Absatzformate könnten verwendet werden, um Absätze zu Eintragstypen zuzuordnen. Voraussetzung ist, daß das Originaldokument bestimmte Bedingungen bzgl. der Verwendung der Absatzformate erfüllt oder entsprechend angepaßt wird.

Die Kategorisierung der Inhaltselemente könnte als Ausgangspunkt für Aufwandsschätzungen mit der *Function-Point-Methode* (Albrecht, Gaffney, 1983) verwendet werden. Hierfür müßten die Kategorien der Inhaltselemente an die Function-Point-Kategorien »externe Eingabe«, »externe Ausgabe«, »Anfrage«, »interne Datei« und »externe Datei« angepaßt werden. Die Kategorisierungen der Inhaltselemente und die Filtermechanismen des ADMIRE-Werkzeugs würden dann die Möglichkeit bieten, die zu jeder der FP-Kategorien gehörenden Inhaltselemente herauszusuchen. Daraus könnte die Anzahl der Function Points (FP) abgeschätzt und der zu erwartende Aufwand berechnet werden.

Glossar

Ablaufumgebung: Hardware und Software, unter denen das *Zielsystem* ausgeführt wird.

Adäquatheit: *Qualitätskriterium*, das auf *Inhaltselementen* definiert ist. Ein *Inhaltselement* ist hinreichend adäquat, wenn die *Anforderung*, die *Zusicherung*, das *Problem* oder das *Fakt* aus Sicht mindestens eines *Informanten* und bezogen auf die aktuelle Problemstellung angemessen beschrieben ist.

ADMIRE: Abkürzung für *A*dvanced *M*anagement of *I*nformal *Re*quirements; der in diesem Bericht vorgestellte Ansatz zur Erstellung, Verwaltung und Prüfung natürlichsprachlicher Spezifikationsdokumente.

ADMIRE-Informationsmodell: Datenmodell; beschreibt alle Informationstypen, die in einem dem ADMIRE-Ansatz folgenden *Prozeß* gesammelt und dokumentiert werden können.

ADMIRE-Prozeßmodell: beschreibt eine exemplarische, sinnvolle und auf das *Informationsmodell* abgestimmte Aufteilung des Requirements-Engineering-*Prozesses* in Phasen; deckt die *Problemanalysephase, Spezifikationsphase* und die *späteren Projektphasen* ab.

ADMIRE-Werkzeug: Werkzeug zur Verwaltung und Prüfung von *Dokumentationen*.

Ähnlichkeitssuche: *Prüfverfahren* des ADMIRE-Ansatzes. Zu einem gegebenen *Eintrag* werden »ähnliche« *Einträge* gesucht und aufgelistet. Die Ähnlichkeitssuche kann eingesetzt werden, um Inkonsistenzen und Redundanzen aufzuspüren.

All-Quantor: *Wort* oder *Phrase* (z.B. »alle«), durch deren Benutzung in einem Freitext ausgedrückt wird, daß bestimmte Sachverhalte ohne Ausnahme für alle Entitäten oder zu jedem Zeitpunkt oder an jedem Ort erfüllt sein müssen.

Änderungsmanagement: Vorgehensweise in den *späteren Projektphasen*, um kontrolliert Änderungen am *Pflichtenheft, Glossar* oder *Quellenkatalog* durchzuführen.

Änderungsmeldung: *externes Dokument*; beschreibt die gewünschten Änderungen des *Pflichtenhefts, Glossars* oder *Quellenkatalogs.*

Anforderung: eine gewünschte Eigenschaft des *Zielsystems*, die in einem Projekt realisiert werden soll. Anforderungen werden durch *Inhaltselemente* beschrieben.

Anforderung, funktionale: *Anforderung*, durch die eine Funktion oder ein Ablauf beschrieben wird.

Anforderung, nicht-funktionale: *Anforderung*, durch die eine gewünschte Eigenschaft oder Qualität des *Zielsystems* beschrieben wird; siehe auch *Benutzbarkeit, Effizienz, Funktionalität, Portierbarkeit, Wartbarkeit* und *Zuverlässigkeit.*

Anforderungsdokument: Oberbegriff für *Ist-Analysedokument*, *Anforderungssammlung*, *Lastenheft* und *Pflichtenheft*.

Anforderungssammlung: *Anforderungsdokument*, das in der *Problemanalysephase* erstellt wird; Ergebnis einer Informationserhebungstätigkeit. Anforderungssammlungen werden verwendet, um die Wünsche und Rahmenbedingungen einzelner *Informanten* oder *Parteien* zu dokumentieren.

Animation: Validierungstechnik. Das *Pflichtenheft*, das in einer *operationalen Notation* vorliegen muß, wird ausgeführt. Dadurch können die Prüfer feststellen, ob das Modell das gleiche Verhalten zeigt wie das gewünschte *Zielsystem*.

Annotation: Eintragstyp des *ADMIRE-Informationsmodells*; Annotationen können Erklärungen, Begründungen oder Beispiele enthalten.

Annotationsfenster: Fenster der *GUI*, in dem alle *Annotationen* einer *Dokumentation* verwaltet werden können.

Arbeitsumgebung: Menge der *Systeme* und Personen, die mit einem (*Ziel-*)*System* interagieren.

Assertion: synonym zu *Zusicherung*.

Assoziation: binäre, gerichtete, benannte Beziehung zwischen *Klassen* des formalen *ADMIRE-Informationsmodells*.

atomar: siehe *Atomizität*.

Atomizität: *Qualitätskriterium*, das auf *Inhaltselementen*, *Glossareinträgen* und *Quellenkatalogeinträgen* definiert ist. Ein *Inhaltselement* ist hinreichend atomar, wenn es nicht sinnvoll in mehrere Inhaltselemente aufgespaltet werden kann. Ein *Glossareintrag* ist atomar, wenn durch ihn genau einem Term eine Bedeutung zugewiesen wird. Ein *Quellenkatalogeintrag* ist atomar, wenn durch ihn auf genau eine *Informationsquelle* verwiesen wird.

Auftraggeber: Rolle in einem Requirements-Engineering-Prozeß. Mit den Auftraggebern wird der Vertrag geschlossen. Sie bezahlen das Projekt, legen die Rahmenbedingungen fest und sind befugt, Entscheidungen zu treffen.

Autor: Person, die eine *Dokumentation* erstellen und ändern darf.

Benutzbarkeit: Oberbegriff für *Produktqualitäten*, die die Benutzerführung und Bedienbarkeit eines *Systems* durch eine Person betreffen.

Benutzer: Rolle in einem Requirements-Engineering-Prozeß. Die Benutzer sind diejenigen, die später mit dem *Zielsystem* arbeiten werden. Das können ebenso »normale« Anwender sein wie Operateure, Administratoren oder Personen, die indirekt mit dem *Zielsystem* interagieren, z.B. der Chef einer Sekretärin, die einen Brief unter Verwendung des *Zielsystems* (in diesem Fall eines Textverarbeitungsprogramms) schreibt.

Benutzt-Beziehung: Beziehung zwischen einer *Phrase* und einem *Freitext*. Falls die *Phrase* nur ein *Wort* enthält, kann die Beziehung folgendermaßen definiert werden: Ein *Freitext* benutzt ein *Wort*, wenn der *Freitext* eine *Flexion* des *Wortes* (Direkt-Benutzt-Beziehung) oder eine *Flexion* eines *Synonyms*, *Hyponyms* oder *Hyperonyms* des *Wortes* enthält (Indirekt-Benutzt-Beziehung). Enthält die *Phrase* mehrere *Wörter*, müssen einige Sonderfälle berücksichtigt werden (siehe Kapitel 5.6).

Brainstorming: allgemeine Informationserhebungsmethode. Ziel einer Brainstorming-Sitzung ist es, zuerst möglichst viele Ideen zu generieren und später die »guten« Ideen auszuwählen. Brainstorming wird im *Requirements Engineering* oft in der *Problemanalysephase* angewandt.

Check: Teilschritt, den das *ADMIRE-Werkzeug* bei einer *Inhaltsprüfung* ausführt. Auf die ausgewählten *Einträge* werden die ausgewählten *Inhaltsprüfungen* angewandt (siehe auch *Recheck*).

Checkliste: enthält Vorgaben für den Inhalt, die Struktur oder die Qualität eines *Anforderungsdokuments*; kann zur Erstellung und Prüfung eines *Anforderungsdokuments* verwendet werden. Aus den *Qualitätskriterien* des *ADMIRE*-Ansatzes können Checklisten für die *Validierung* und *Verifikation* abgeleitet werden.

Definition: Teil eines *Glossareintrags*. Durch die Definition wird einem *Term* eine Bedeutung zugewiesen.

Dokument: Die Informationen einer *Dokumentation* werden in Dokumenten organisiert. Folgende Arten von Dokumenten werden unterschieden: *Anforderungsdokument, Glossar, Quellenkatalog* und *Mängelkatalog*.

Dokument, externes: Dokument, das nicht Teil einer *Dokumentation* ist. *Quellenkatalogeinträge* der Kategorie »externes Dokument« können auf sie verweisen.

Dokumentation: Instanz des *ADMIRE-Informationsmodells*.

Dokumenteintrag*:* Oberbegriff für *Pflichtenheft, Lastenheft, Anforderungssammlung, Ist-Analysedokument, Glossar,* und *Quellenkatalog*. Weil Dokumenteinträge zur Definition der *Qualitätskriterien* eingeführt wurden, gehören *Mängelkataloge* nicht zu den Dokumenteinträgen.

Dokumentfenster: Fenster des *GUI*, in dem alle *Anforderungsdokumente* einer *Dokumentation* verwaltet werden können.

Domänenanalyse: Tätigkeit in der *Problemanalysephase*. Hierbei werden Informationen über einen Anwendungsbereich als Ganzes gesammelt und es wird ein *Domänenmodell* erstellt. In späteren Projekten des gleichen Anwendungsbereichs kann auf dieses Wissen zurückgegriffen werden.

Domänenmodell: Ergebnis(dokument) der *Domänenanalyse*; enthält typischerweise Daten und generische Anforderungen, die für konkrete Systeme vervollständigt oder angepaßt werden müssen.

Effizienz: Oberbegriff für *Produktqualitäten*, die das Laufzeitverhalten und den Ressourcenbedarf eines *Systems* betreffen.

Eindeutigkeit: Qualitätskriterium, das auf *NL-Einträgen* des *ADMIRE-Informationsmodells* definiert ist. Ein *NL-Eintrag* ist hinreichend eindeutig, wenn alle darin enthaltenen *Freitexte* für alle zu erwartenden Leser genau eine Bedeutung haben.

Eintrag: Teil des *ADMIRE-Informationsmodells*; Oberbegriff für *Inhaltselement, Annotation, Systemmodell, Glossareintrag, Quellenkatalogeintrag, Mangeleintrag, Anforderungsdokument, Kapitel, Glossar, Quellenkatalog* und *Mängelkatalog*.

Eintrag, aktueller: *Eintrag*, dessen *Zustand* nicht »verworfen« ist. *Einträge* (außer *Annotationen*), die kein Zustandsattribut haben, sind immer aktuell. Eine

Annotation ist aktuell, wenn sie an mindestens einen aktuellen *Eintrag* gebunden ist.

Einwort-Term: *Term*, der aus einem *Wort* besteht, z.B. »Spieler«.

Einzeleintrag: Oberbegriff für *Inhaltselement*, *Annotation*, *Systemmodell*, *Glossareintrag*, *Quellenkatalogeintrag*, *Kapitel* oder *Anforderungsdokument*. Weil Einzeleinträge zur Definition der *Qualitätskriterien* eingeführt wurden, gehören *Mangeleinträge* nicht zu den Einzeleinträgen.

Entwickler: Rolle in einem Requirements-Engineering-Prozeß. Die Entwickler haben die Aufgabe, das *Zielsystem* zu realisieren. Hierzu gehören alle Personen, die an dem Entwicklungsprozeß beteiligt sind, z.B. *Systemanalytiker*, Entwerfer, Programmierer, Projektleiter und Qualitätssicherungsbeauftragte.

Entwicklungsumgebung: Hardware und Software (z.B. Programmiersprache, Compiler), mittels derer die Entwicklung des *Zielsystems* durchgeführt wird.

Ereignis: Beschreibung eines bemerkenswerten Vorkommens zu einem bestimmten Zeitpunkt an einer bestimmten Stelle.

Ereignis, Ausgabe-: *Ereignis*, das vom *Zielsystem* erzeugt wird und an ein *System* oder eine Person der *Arbeitsumgebung* gerichtet ist.

Ereignis, Eingabe-: *Ereignis*, das in der *Arbeitsumgebung* erzeugt wird und an das *Zielsystem* gerichtet ist.

Ereignis, internes bedingtes: *Ereignis*, das innerhalb des *Zielsystems* immer dann erzeugt wird, wenn eine bestimmte Bedingung erfüllt ist.

Ereignis, internes zeitliches: *Ereignis*, das innerhalb des *Zielsystems* zu bestimmten Zeitpunkten erzeugt wird.

Fakt: eine vorhandene Eigenschaft eines *Systems* oder einer Person. Fakten sind immer erfüllt. Fakten sind z.B. Aussagen über den *Ist-Zustand* oder gegebene Randbedingungen, unter denen die *Anforderungen* realisiert werden sollen. Fakten werden durch *Inhaltselemente* beschrieben.

Flexion: siehe *Flexionsbeziehung*.

Flexionsbeziehung: Beziehung zwischen zwei *Wörtern*. Ein *Wort* ist eine *Flexion* eines anderen *Wortes*, wenn es eine *Grundform* gibt, von der beide *Wörter* durch Konjugation oder Deklination erzeugt werden können.

Flexionseditor: Fenster des *GUI*, in dem die *Flexionen* eines *Glossareintrags* verwaltet werden können.

Filter: Funktion, durch die aus einer *Dokumentation Einträge* ausgewählt werden, die bestimmten, durch die Funktion vorgegebenen Bedingungen genügen.

Freitext: beliebig lange, evtl. leere Folge von *Wörtern*, *Phrasen*, Leerzeichen, Satzzeichen oder Sonderzeichen (siehe auch Kapitel 6.1.1)

Funktion (eines Software-Systems): beschreibt, wie Eingabedaten oder Eingabeereignisse in Ausgabedaten oder Ausgabeereignisse transformiert werden. Hierbei kann ein evtl. vorhandener interner Zustand berücksichtigt oder geändert werden.

Funktionalität: Oberbegriff für *Produktqualitäten*, die die Genauigkeit, Sicherheit (security), Angemessenheit und Interoperabilität eines *Systems* betreffen.

Glossar: *Dokument*, das die für alle Projektbeteiligten verbindliche Projektterminologie festlegt; enthält eine Menge von *Glossareinträgen*.

Glossareintrag: Eintragstyp des *ADMIRE-Informationsmodells*. In einem Glossareintrag wird einem *Term* durch eine *Definition* eine Bedeutung zugewiesen.

Glossarfenster: Fenster des *GUI*, in dem ein *Glossar* verwaltet werden kann.

Grundform: Wortform, in der ein *Einwort-Term* vorliegen sollte. Bei konjugierbaren Wörtern ist der Infinitiv die Grundform, bei deklinierbaren Wörtern der Nominativ Singular (oder Plural, falls die Singularform nicht existiert). Für *Mehrwort-Terme* gibt es keine festen Regeln zur Bildung der Grundform.

GUI: Abkürzung für Graphical User Interface; *Schnittstelle* des *ADMIRE-Werkzeugs* zu den *Systemanalytikern*.

Haupteintrag: Eintragstyp des *ADMIRE-Informationsmodells*; Obergriff für alle *Einträge*, zu denen *Ursprünge* angegeben werden können: *Inhaltselemente*, *Systemmodelle*, *Annotationen*, *Glossareinträge*, *Quellenkatalogeinträge* und *Mangeleinträge*.

Hauptfenster: Fenster, das beim Starten des ADMIRE-Werkzeugs angezeigt wird.

Hauptmenü: Menü des *Hauptfensters*.

Homonym: Eigenschaft einer *Phrase*. Eine *Phrase* ist homonym, wenn sie mehrere Bedeutungen hat.

HTML-Schnittstelle: *Schnittstelle* des *ADMIRE-Werkzeugs*; Dateischnittstelle, über die generierte Reporte ausgegeben werden können. Die generierten Dateien liegen im HTML-Format vor.

HTML-Eingabe-Schnittstelle: *Schnittstelle* des *WWW-Browsers*; Dateischnittstelle, über die HTML-Dateien eingelesen werden können.

Hyperonym: siehe Hyponymiebeziehung.

Hyponym: siehe Hyponymiebeziehung.

Hyponymiebeziehung: Beziehung zwischen zwei *Phrasen*. Die *Phrase* p_1 ist hyponym zu p_2, wenn zwischen p_1 und p_2 eine Ist-Ein-Beziehung vorliegt (p_1 ist ein spezielles p_2). Dadurch ergibt sich eine Beziehung der Über- bzw. Unterordnung. Beispiel: »Katze« ist ein spezielles »Tier«. Die übergeordnete *Phrase* (»Tier«) wird *Hyperonym*, die untergeordnete (»Katze«) wird *Hyponym* genannt.

Indikator: *Wort*, das nur in seinem Kontext Sinn ergibt, weil es sich auf andere Teile eines Freitextes bezieht. Indikatoren sind z.B. Personal- (»er«), Possessiv- (»seiner«) und Demonstrativpronomen (»dieser«) sowie einige Adverben des Ortes (»dort«) und der Zeit (»bald«).

Informationsmodell: synonym zu *ADMIRE-Informationsmodell*.

Informationsquelle: ein *Informant*, eine *Partei* oder ein *externes Dokument*; kann relevante Informationen für das aktuelle Projekt liefern bzw. enthalten.

Informant: *Informationsquelle*; eine Person, die für das aktuelle Projekt relevante Informationen liefern kann. Dazu zählen auch Personen, die direkt oder indirekt durch die Einführung des *Zielsystems* positiv oder negativ betroffen sind.

Inhaltselement: Eintragstyp des *ADMIRE-Informationsmodells*. Ein Inhaltselement ist eine eindeutig referenzierbare und soweit möglich in sich abgeschlos-

sene Informationseinheit, die ein Charakteristikum eines *Systems*, einer *Schnittstelle*, einer *Person* oder eines *Projekts* beschreibt. Enthält eine *Anforderung*, ein *Fakt*, eine *Zusicherung* oder ein *Problem*.

Inhaltsprüfung: automatisches *Prüfverfahren* des ADMIRE-Ansatzes. Anhand gegebener *Prüfkriterien* wird eine *Dokumentation* auf Indizien für *Mängel* durchsucht.

Inspektion: visuelles *Prüfverfahren* des ADMIRE-Ansatzes. Bei einer Inspektion wird ein Teil einer *Dokumentation* nach bestimmten Gesichtspunkten ausgewählt, ausgedruckt und *Informanten* zur Prüfung vorgelegt. Die eigentliche Prüfung wird von den *Informanten* durchgeführt. Die Ergebnisse werden in einem *Protokoll* festgehalten und nach Abschluß der Prüfung in die *Dokumentation* integriert.

Inspektion, fokussierte: *Prüfverfahren* des ADMIRE-Ansatzes, durch das der *Systemanalytiker* beim Schreiben und Lesen eines *Freitextes* eine visuelle Rückmeldung erhält. Wörter und *Phrasen*, die eine besondere Bedeutung haben (siehe *Glossar, Wortliste*) werden markiert. Bei einer positiven fokussierten Inspektion werden *Phrasen* markiert, deren Benutzung erwünscht ist. Bei einer negativen fokussierten Inspektion werden *Phrasen* markiert, die nicht *benutzt* werden sollen.

Interview: allgemeine Informationserhebungsmethode; wird häufig in der *Problemanalysephase* eingesetzt. Die *Systemanalytiker* befragen *Informanten* anhand vorbereiteter Fragen.

Ist-Analyse: Tätigkeit während der *Problemanalysephase*. Ziel einer Ist-Analyse ist es, den *Ist-Zustand* zu erheben.

Ist-Analysedokument: *Anforderungsdokument*, das in der *Problemanalysephase* erstellt werden kann; beschreibt den *Ist-Zustand*.

Ist-Zustand: die Situation, wie sie zu Projektbeginn vorgefunden wird. Zur Beschreibung des Ist-Zustands gehören Aussagen über die manuellen Arbeitsabläufe bzw. über die Abläufe mit einem zu ersetzenden *System*, über die aktuelle *Arbeitsumgebung* und über vorhandene *Probleme*.

Kapitel: Eintragstyp des *ADMIRE-Informationsmodells*. Kapitel dienen zur Gruppierung der *Inhaltselemente*. Ein *Anforderungsdokument* enthält beliebig viele Kapitel, angeordnet in einer *Kapitelhierarchie*.

Kapitelhierarchie: beliebig breite und tiefe Hierarchie von *Kapiteln*; die Wurzel ist ein *Anforderungsdokument*.

Kardinalität: Angabe zu einer *Kommunikationskante* eines *Systemmodells* über die Anzahl der *Systeme* oder Personen, die gleichzeitig über diese Kommunikationskante miteinander interagieren dürfen oder müssen.

Kategorie: Attribut von *Inhaltselementen*, *Glossareinträgen* und *Mangeleinträgen*, durch das die Art des Eintrags charakterisiert wird. Die Mengen der möglichen Kategorien für *Inhaltselemente* und *Glossareinträge* sind in jeweils einer *Kategorisierungshierarchie* angeordnet.

Kategorisierungshierarchie: Hierarchie von *Kategorien*. *Kategorien*, die weiter unten in der Hierarchie angeordnet sind, partitionieren die übergeordneten *Kategorien*.

Klasse: Komponente eines *Klassendiagramms*. Eine Klasse hat Attribute und Methoden. Zur Beschreibung des *ADMIRE-Informationsmodells* werden nur Klassen verwendet, die keine Methoden haben.

Klassendiagramm: *semi-formale Notation* zur Beschreibung komplexer, beliebiger Datenstrukturen. Die Struktur des *ADMIRE-Informationsmodells* wird durch Klassendiagramme beschrieben, die *Klassen, Assoziationen* und *Vererbung*sbeziehungen enthalten.

Kommunikationskante: Kantentyp in einem *Systemmodell*. Durch eine Kommunikationskante wird ausgedrückt, daß *Systeme* und/oder Personen direkt Informationen austauschen können.

Konfigurationsfenster: Fenster des *GUI*, in dem eingestellt werden kann, welche konkreten *Inhaltsprüfungen, Statusprüfungen* und *fokussierten Inspektionen* durchgeführt werden sollen.

Konjunktion: *Wort* oder *Phrase*, die Sätze oder Teilsätze miteinander verknüpft, z.B. »und«, »oder«, »entweder oder«, »wenn«, »aber«, »obwohl«.

Konsistenz: *Qualitätskriterium*, das auf *Einzeleinträgen* und auf Mengen von *Einzeleinträgen* definiert ist. Ein *Einzeleintrag* ist hinreichend konsistent, wenn sich seine Attributwerte nicht widersprechen und wenn sie zum Eintragstyp passen. Eine Menge von *Einzeleinträgen* ist hinreichend konsistent, wenn keine Teilmenge einen Widerspruch enthält und wenn keine der im Informationsmodell zugelassenen Inkonsistenzen vorhanden ist (siehe Kapitel 5.10).

Konsistenz, externe: *Qualitätskriterium*, das auf Mengen von *Einzeleinträgen* und *Quellenkatalogeinträgen* (der Kategorie »externes Dokument«) definiert ist. Eine Menge von *Einzeleinträgen* ist hinreichend extern konsistent, wenn keine Teilmenge einen Widerspruch zu den Aussagen der *externen Dokumente* enthält.

Kontextdiagramm: synonym zu *Systemmodell*.

Korrektheit: *Qualitätskriterium*, das auf *Einzeleinträgen* des *ADMIRE-Informationsmodells* definiert ist. Ein *Einzeleintrag* ist hinreichend korrekt, wenn die Attribute und Verweise des *Einzeleintrags* objektiv richtig sind, oder, falls die Korrektheit nicht objektiv beurteilbar ist, wenn es eine Person gibt, die diesen *Einzeleintrag* für richtig hält, oder wenn es ein für das Projekt relevantes *externes Dokument* gibt, aus dem der *Einzeleintrag* abgeleitet werden kann.

Kunde: Rolle in einem Requirements-Engineering-Prozeß; Oberbegriff für *Benutzer* und *Auftraggeber*.

Lastenheft: *Anforderungsdokument*, in dem die Wünsche und Rahmenbedingungen aus Sicht aller befragten *Informanten* und projektrelevante Informationen aller ausgewerteten *externen Dokumente* dokumentiert werden.

Mangelart: Attribut eines *Mangeleintrags*, durch das angegeben wird, ob der *Mangeleintrag* manuell oder automatisch erzeugt wurde.

Mangeleintrag: Eintragstyp des *ADMIRE-Informationsmodells*. Durch Mangeleinträge können *Einträge* einer Dokumentation als mangelhaft gekennzeichnet werden.

Mängelkatalog: *Dokument*, das alle *Mangeleinträge* einer *Dokumentation* enthält.

Mangelkategorie: Attribut eines *Mangeleintrags*, das angibt, welches *Qualitätskriterium* durch den beschriebenen Mangel verletzt wird.

Mangelverweis: Attribut eines *Mangeleintrags*, durch das angegeben wird, welche *Einträge* der *Dokumentation* den beschriebenen Mangel aufweisen.

Mängelkatalogfenster: Fenster des *GUI*, in dem der *Mängelkatalog* verwaltet werden kann.

Mehrdeutigkeit: Gegenteil von *Eindeutigkeit*.

Mehrwort-Term: *Term*, der aus mehr als einem *Wort* besteht, z.B. »angehender Projektleiter«.

Metaanforderung: *Anforderung* an den Inhalt, den Aufbau oder die Qualität einer *Dokumentation*.

Metaspezifikation: Teil einer *Dokumentation*; enthält die *Vorlagen* für *Anforderungsdokumente*, die *Kategorisierungshierarchien* für *Inhaltselemente* und *Glossareinträge*, die Mengen der möglichen *Prioritäten* und *Stabilitäten* für *Inhaltselemente* und die *Wortliste*.

Metaspezifikationsfenster: Fenster des *GUI*, in dem die *Metaspezifikation* angezeigt werden kann.

Minimalität: Qualitätskriterium, das auf *NL-Einträgen* und auf *Anforderungsdokumenten* definiert ist. Ein *NL-Eintrag* ist hinreichend minimal, wenn die enthaltenen *Freitexte* keine unnötigen oder kürzer formulierbaren Teile (z.B. Füllwörter) enthalten. Ein *Anforderungsdokument* ist hinreichend minimal, wenn es keine unnötigen Einträge enthält.

Modalverb: Hilfsverb, durch das der Modus (z.B. Indikativ, Optativ) eines Satzes festgelegt wird.

Nachvollzogenheit: Ein *Haupteintrag* ist nachvollzogen, wenn er über seine *Ursprungs*verweise direkt oder indirekt auf einen *Quellenkatalogeintrag* verweist.

Natürliche Sprache: *informale* textuelle *Notation*, die nur durch den Wortschatz sowie die Orthographie- und Grammatikregeln der jeweiligen Sprache (z.B. Deutsch) eingeschränkt wird.

nicht redundant: *Qualitätskriterium*, das auf Mengen von *Inhaltselementen* definiert ist. Eine Menge von *Inhaltselementen* ist hinreichend nicht redundant, wenn jede Aussage in maximal einem der *Inhaltselemente* vorkommt.

nicht überspezifiziert: *Qualitätskriterium*, das auf einzelnen *Inhaltselementen* der Kategorie »Anforderung« definiert ist. Ein *Inhaltselement* der Kategorie »Anforderung« ist hinreichend nicht überspezifiziert, wenn die enthaltene *Anforderung* nicht einschränkender ist, als es aus Sicht der Informanten wirklich notwendig ist.

NL-Eintrag: Einzeleintrag, der einen Freitext enthält; Oberbegriff für *Inhaltselement*, *Annotation*, *Glossareintrag*, *Quellenkatalogeintrag* und *Kapitel*.

Norm: enthält Vorgaben für Aufbau und Inhalt eines *Anforderungsdokuments* (*Produktnorm*) oder für den Software-Entwicklungsprozeß (Prozeßnorm). Siehe auch *Vorlage*.

Normkonformität: *Qualitätskriterium*, das auf *Anforderungsdokumenten* definiert ist. Ein *Anforderungsdokument* ist normkonform, wenn es genau so strukturiert ist, wie es durch die Norm vorgegeben wird (siehe Kapitel 5.10).

Notation, formale: Notation, deren Syntax und Semantik definiert sind.

Notation, informale: Notation, deren Syntax und Semantik nicht definiert sind.

Notation, mathematische: *formale Notation*, die nicht unbedingt ausführbar ist.

Notation, operationale: *formale Notation*, die ausführbar ist.

Notation, semi-formale: *Notation*, deren Syntax definiert ist, deren Semantik jedoch nicht oder nur teilweise definiert ist.

Partei: Menge von *Informanten*, die für das Projekt ähnliche Informationen liefern können, eine ähnliche Sichtweise vertreten oder über ähnliche Befugnisse verfügen.

Pflichtenheft: *Anforderungsdokument*, in dem das *Zielsystem* spezifiziert wird; Hauptergebnis der *Spezifikationsphase*.

Phrase: nicht leere Folge von *Wörtern*.

Portierbarkeit: Oberbegriff für *Produktqualitäten*, die die Übertragung eines *Systems* auf eine andere Plattform (Hardware, Betriebssystem ...) betreffen.

Prädikatenlogik: *formale Notation*, durch die logische Bedingungen beschrieben werden können; wird zur Beschreibung von *Prüfkriterien* und zur Beschreibung von Bedingungen an das *ADMIRE-Informationsmodell* verwendet.

Präzision: *Qualitätskriterium*, das auf *Inhaltselementen*, *Glossareinträgen* und *Quellenkatalogeinträgen* definiert ist. Ein *Inhaltselement*, ein *Glossareintrag* oder ein *Quellenkatalogeintrag* ist hinreichend präzise, wenn alle enthaltenen *Freitexte* so genau wie für die Problemstellung erforderlich sind und wenn in den Inhaltselementen quantitative Angaben soweit möglich verwendet werden.

Priorität: Attribut eines *Inhaltselements* der Kategorie »Anforderung« oder »Problem«, durch das seine Wichtigkeit im Vergleich zu anderen *Inhaltselementen* (der Kategorien »Anforderung« und »Problem«) beschrieben wird.

Problem: etwas, das in dem Projekt gelöst oder verbessert werden soll. Probleme werden durch *Inhaltselemente* beschrieben.

Problemanalyse: Teildisziplin des *Requirements Engineerings*, die sich insbesondere mit der Erhebung und Beschreibung von Informationen aus Sicht von *Informanten* beschäftigt.

Problemanalysephase: erste Teilphase der *Requirements-Engineering-Phase*, in der der *Ist-Zustand* analysiert und der *Soll-Zustand* erhoben wird. Hierfür werden relevante *Informationsquellen* identifiziert, Informationen (insbesondere *Anforderungen*) erhoben und folgende *Dokumente* erstellt: *Anforderungssammlungen*, *Ist-Analysedokument(e)*, *Lastenheft(e)*, *Glossar* und *Quellenkatalog*.

Produktqualität: Menge aller möglichen Qualitäten für ein (Software-) System; kann nach ISO/IEC (1991) in die Kategorien *Benutzbarkeit*, *Effizienz*, *Funktionalität*, *Portierbarkeit*, *Wartbarkeit* und *Zuverlässigkeit* unterteilt werden.

Projektphasen, spätere: alle Phasen des aktuellen Projekts nach der *Requirements-Engineering-Phase*.

Projektstand: Attribut eines *Inhaltselements* der Kategorie »Anforderung«, durch das angegeben wird, ob die enthaltene Anforderung zum aktuellen Zeitpunkt von dem Zielsystem vollständig, teilweise oder nicht erfüllt wird.

Protokoll: *externes Dokument*; wird bei einer Informationserhebungs-, Entscheidungsfindungs-, *Validierungs-* oder *Verifikations*tätigkeit erstellt. Die enthaltenen Informationen werden nach Ende der Tätigkeit in die *Dokumentation* integriert.

Prototyping: Verfahren für die *Validierung* und *Verifikation* von *Anforderungsdokumenten*. Ziel ist es, möglichst früh im Software-Entwicklungsprozeß eine lauffähige Version (eines Teils) des *Zielsystems* zu erstellen, um Analysen durchzuführen oder Rückmeldungen von den *Informanten* zu erhalten.

Prozeß: Instanz eines *Prozeßmodells*.

Prozeßmodell: synonym zu *ADMIRE-Prozeßmodell*.

Prüfkriterium: Kriterium, das bei einer *Inhaltsprüfung* angewandt wird; gibt vor, nach welchen Indizien (für Mängel) gesucht wird.

Prüfungsfenster: Fenster des *GUI*, in dem die zu prüfenden *Einträge* der *Dokumentation* festgelegt und die *Inhaltsprüfungen*, *Statusprüfungen* und die *Ähnlichkeitssuche* gestartet werden können.

Prüfverfahren: Oberbegriff für *Inspektion*, *fokussierte Inspektion*, *Inhaltsprüfung*, *Statusprüfung* und *Ähnlichkeitssuche*.

Qualitätskriterium: Bedingung, die *Einträge* einer *Dokumentation* erfüllen sollen. Im *ADMIRE-Prozeßmodell* wird beschrieben, welche *Einträge* zu welchen Zeitpunkten welche Qualitätskriterien erfüllen sollen (siehe auch *Korrektheit*, *Vollständigkeit*, *Adäquatheit*, *Konsistenz*, *externe Konsistenz*, *nicht redundant*, *nicht überspezifiziert*, *Realisierbarkeit*, *Überprüfbar*keit, *Atomizität*, *Verständlichkeit*, *Eindeutigkeit*, *Präzision*, *Minimalität* und *Normkonformität*).

Quellenkatalog: *Dokument*, das Informationen über die identifizierten *Informationsquellen* enthält; enthält beliebig viele *Quellenkatalogeinträge*.

Quellenkatalogeintrag: Eintragstyp des *ADMIRE-Informationsmodells*. Ein Quellenkatalogeintrag enthält Informationen über eine *Informationsquelle*.

Quellenkatalogfenster: Fenster des *GUI*, in dem ein *Quellenkatalog* verwaltet werden kann.

Querverweis, textueller: *Substring* eines *Freitextes*, der buchstabengetreu mit dem eindeutigen Bezeichner eines *Eintrags* der *Dokumentation* übereinstimmt.

Realisierbarkeit: *Qualitätskriterium*, das auf Mengen von *Inhaltselementen* der Kategorie »Anforderung« definiert ist. Eine Menge von *Inhaltselementen* der Kategorie »Anforderung« ist realisierbar, wenn es eine Implementierung gibt, die alle enthaltenen *Anforderungen* gleichzeitig erfüllt.

Recheck: Teilschritt, den das *ADMIRE-Werkzeug* bei einer *Inhaltsprüfung* ausführt. Vorhandene »aktuelle« und »potentielle« automatisch erzeugte *Mangeleinträge* werden daraufhin überprüft, ob der beschriebene Mangel in der *Dokumentation* immer noch vorliegt (siehe auch *Check*).

Redundanz: siehe *nicht redundant*.

Requirement: synonym zu *Anforderung*.

Requirements Engineering: Teildisziplin des Software Engineerings, die sich mit der Erhebung von Informationen (insbesondere *Anforderungen*) und der Spezifikation von *Zielsystemen* beschäftigt.

Requirements-Engineering-Phase: Phase vor dem Entwurf, in der als Hauptergebnis ein *Pflichtenheft* erstellt wird. Die Requirements-Engineering-Phase wird oft in eine *Problemanalyse*- und eine *Spezifikationsphase* unterteilt.

Requirements Specification: synonym zu *Lastenheft*.

Review: allgemeines Verfahren zur Prüfung beliebiger *Dokumente*; wird im *Requirements Engineering* häufig zur *Validierung* eines *Anforderungsdokuments* eingesetzt. Bei einem Review wird das zu prüfende *Dokument* von Gutachtern anhand der Referenzunterlagen beurteilt. Die Ergebnisse werden in einer Review-Sitzung zusammengetragen und protokolliert.

Schnittstelle: Teil eines *Systems*, über den es mit seiner direkten *Arbeitsumgebung* kommunizieren kann. Ein *System* kann über seine Schnittstellen Eingaben empfangen und Ausgaben erzeugen.

Sicht: Menge aller in einem *Anforderungsdokument* enthaltenen *Einträge*, die bestimmte, durch einen *Filter* vorgegebene Kriterien erfüllen.

Sichtenfenster: Fenster des *GUI*, in dem *Sichten* verwaltet werden können.

SimpleWord: *Wort*, das bei der Erkennung von benutzten *Mehrwort-Termen* in einem *Freitext* ignoriert wird. Wenn in einem *Freitext* z.B. nach »Attribut eines Dokuments« gesucht wird, wird nur nach »Attribut« und »Dokuments« gesucht, das SimpleWord »einer« wird ignoriert.

Soll-Zustand: die gewünschte Situation nach Projektdurchführung. Sind *Ist-Zustand* und Soll-Zustand gegeben, so ist die Differenz zwischen beiden genau das, was durch das Projekt realisiert werden soll.

Spezifikationsphase: zweite Teilphase der *Requirements-Engineering-Phase*; folgt der *Problemanalysephase*. In dieser Phase wird das *Lastenheft* überarbeitet. Unvollständigkeiten und Vagheiten werden beseitigt. Das *Zielsystem* wird spezifiziert. Ergebnisdokumente sind das *Pflichtenheft*, das aktualisierte *Glossar* und der aktualisierte *Quellenkatalog*.

Stabilität: Attribut eines *Inhaltselements*, durch das beschrieben wird, als wie wahrscheinlich Änderungen an dem *Inhaltselement* eingeschätzt werden.

Stakeholder: synonym zu *Informant*.

Statusprüfung: *Prüfverfahren* des ADMIRE-Ansatzes zur Erkennung von Unvollständigkeiten; erzeugt einem Überblick darüber, welche Informationen in der *Dokumentation* mit welcher Häufigkeit vorkommen.

Substring-Beziehung: Beziehung zwischen zwei *Freitexten*. Der *Freitext* p ist ein Substring des *Freitextes* s, wenn p buchstabengetreu in s enthalten ist, wobei Groß- und Kleinschreibung nicht unterschieden werden.

Synonym: siehe *Synonymiebeziehung*.

Synonymiebeziehung: Beziehung zwischen zwei *Phrasen*. Zwei *Phrasen* sind synonym, wenn sie, bezogen auf das aktuelle Projekt, die gleiche Bedeutung haben, also wechselseitig verwendet werden können.

SysAdmin-Schnittstelle: *Schnittstelle* des *ADMIRE-Werkzeugs* zu den *Systemadministratoren*.

System: Einheit; Menge von Elementen (Teilsystemen), die durch bestimmte Kriterien von ihrer Umgebung (*Arbeitsumgebung*) abgegrenzt sind, die über Beziehungen verknüpft sein können und die zur Erreichung eines bestimmten Ziels oder Zwecks dienen. Ein *System* ist in eine *Arbeitsumgebung* eingebettet und kommuniziert mit den *Systemen* oder Personen der *Arbeitsumgebung* über *Schnittstellen*. Im *ADMIRE*-Ansatz wird zwischen *vorgegebenen Systemen* und *Zielsystemen* unterschieden.

System der Arbeitsumgebung: synonym zu *vorgegebenes System*.

System, vorgegebenes: *System*, das nicht in dem aktuellen Projekt realisiert werden soll (siehe auch *Zielsystem*).

Systemanalytiker: Rolle in einem Requirements-Engineering-*Prozeß*. Die Systemanalytiker führen die meisten der in einem (Software-) Projekt anfallenden Requirements-Engineering-Tätigkeiten durch oder sorgen dafür, daß sie durchgeführt werden.

Systemmodell: *Eintragstyp* des *ADMIRE-Informationsmodells*. Ein Systemmodell beschreibt entweder die Einbettung eines Zielsystems in seine *Arbeitsumgebung* (*Soll-Analyse*) oder die Umgebung, wie sie zu Projektbeginn vorgefunden wird (*Ist-Analyse*). Es enthält eine Menge von *Systemmodelleinträgen*. Durch ein Systemmodell kann eine klare Grenze zwischen *Zielsystem* und *Arbeitsumgebung* gezogen werden.

Systemmodelleintrag: Komponente eines *Systemmodells*; verweist auf ein *System*, eine Person (bzw. deren Rolle) oder eine *Schnittstelle*.

Systemmodellfenster: Fenster des *GUI*, in dem ein *Systemmodell* verwaltet werden kann.

Systemumgebung: synonym zu *Arbeitsumgebung*.

Task-Analyse: Tätigkeit der *Problemanalysephase*. Hierbei werden die Arbeitsabläufe untersucht, in die das *Zielsystem* eingebettet werden soll.

TBD: Abkürzung für »to be determined«; Marker, um Teile eines *Anforderungsdokuments* als unvollständig zu markieren. *Freitexte*, die »TBD« als *Substring* enthalten, gelten als unvollständig.

Term: Attribut eines *Glossareintrags*. Einem Term wird durch eine *Definition* eine Bedeutung zugewiesen.

Term des Anwendungsbereichs: *Term*, der in der *definierten* Bedeutung von mindestens einem *Informanten* benutzt wird.

Term, projektspezifischer: *Term*, der in dem Projekt neu gebildet oder mit einer projektspezifischen Bedeutung belegt wurde.

Terminologie: Menge der im *Glossar* definierten *Terme*.

Testfall: Attribut eines *Inhaltselements* (der Kategorie »Anforderung«, »Fakt« oder »Zusicherung«); beschreibt ein Kriterium, das ein *System*, eine Person oder ein

Projekt erfüllen muß, damit das zugehörige *Inhaltselement* als erfüllt betrachtet werden kann.

Überprüfbarkeit: *Qualitätskriterium*, das auf *Inhaltselementen* der Kategorien »Anforderung«, »Fakt« oder »Zusicherung« definiert ist. Ein *Inhaltselement* der Kategorie »Anforderung«, »Fakt« oder »Zusicherung« ist hinreichend überprüfbar, wenn es eine endliche, kosteneffektive Technik gibt, mit der geprüft werden kann, ob die Systeme, Schnittstellen, Personen oder das Projekt die in dem *Inhaltselement* geforderte Eigenschaft aufweisen.

Überspezifikation: siehe *nicht überspezifiziert*.

UML: Abkürzung für *Unified Modeling Language*.

Unified Modeling Language: normierte, *semi-formale Notation* für die objektorientierte Analyse und den objektorientierten Entwurf. Eine mit formaler Semantik unterlegte Teilmenge der UML-*Klassendiagramme* wird zur Beschreibung des formalen *ADMIRE-Informationsmodells* verwendet.

Ursprung: Verweis eines *Haupteintrags* auf einen anderen *Eintrag*, von dem die in dem *Haupteintrag* enthaltene Information abgeleitet wurde. Ein Ursprung verweist entweder auf einen *Quellenkatalogeintrag* oder einen *Eintrag* eines *Anforderungsdokuments*.

Validierung: Tätigkeit, durch die festgestellt werden soll, ob die in einer *Dokumentation* enthaltenen Informationen den »wahren« Wünschen und Rahmenbedingungen der *Informanten* genügen. Es soll die Frage beantwortet werden, ob das »richtige« *System* entwickelt wird. Im Gegensatz zur *Verifikation* müssen an der Validierung *Informanten* beteiligt werden.

Vererbung: strukturelle Beziehung zwischen *Klassen* des formalen *ADMIRE-Informationsmodells*. Eine *Klasse* erbt alle Attribute und Assoziationen der Oberklasse.

Verifikation: Tätigkeit, durch die festgestellt werden soll, ob eine *Dokumentation* bestimmte *Qualitätskriterien* erfüllt. Im Gegensatz zur *Validierung* kann die Verifikation allein von den *Systemanalytikern* ohne Beteiligung der *Informanten* durchgeführt werden.

Verständlichkeit: *Qualitätskriterium*, das auf *NL-Einträgen* definiert ist. Ein *NL-Eintrag* ist hinreichend *verständlich*, wenn alle enthaltenen *Freitexte* so geschrieben sind, daß jeder erwartete Leser deren Bedeutung erfassen kann, wenn er zusätzlich auf die anderen *Einträge* des gleichen *Dokuments* und des *Glossars* zugreifen kann.

Vollständigkeit: *Qualitätskriterium*, das auf *Einzeleinträgen* und auf *Dokumenteinträgen* definiert ist. Ein *Einzeleintrag* bzw. *Dokumenteintrag* ist hinreichend vollständig, wenn er alle für das Projekt relevanten und zur Eintragsart bzw. Dokumentart passenden Informationen enthält (siehe auch Kapitel 5.10).

Vorlage: Schablone für die Struktur eines *Anforderungsdokuments*; gibt die *Kapitelhierarchie* vor; enthält eine Vorlagen-Kapitelhierarchie.

Vorlagen-Kapitel: einzelnes *Kapitel* einer *Vorlage*.

Wartbarkeit: Oberbegriff für *Produktqualitäten*, die die Änderbarkeit, Stabilität, Testbarkeit oder Analysierbarkeit eines *Systems* betreffen.

WeakWord: *Wort* oder *Phrase*, deren *Benutzung* in einem *Freitext* darauf schließen läßt, daß der *Freitext* mit hoher Wahrscheinlichkeit unpräzise ist.

Wort: eine nicht leere Folge von Buchstaben, Ziffern und den Zeichen Punkt (».«), Bindestrich (»-«) und Unterstrich (»_«).

Wort, maximales: ein *Wort* innerhalb eines *Freitexts*, das buchstabengetreu in dem *Freitext* enthalten ist und zu dem es kein längeres *Wort* an der gleichen Stelle gibt, das ebenfalls in dem *Freitext* enthalten ist. Beispielsweise ist das *Wort* »Dokuments« ein maximales *Wort* in dem *Freitext* »Attribut eines Dokuments«, das *Wort* »Doku« aber nicht.

Worteintrag: ein Element der *Wortliste*.

Wortliste: Teil der *Metaspezifikation*. In die Wortliste werden alle *Wörter* und *Phrasen* eingetragen, die im *ADMIRE*-Ansatz eine spezielle Bedeutung haben. Jeder *Worteintrag* hat einen *Worttyp*.

Worttyp: dient der Charakterisierung einzelner *Worteinträge* der *Wortliste*. Im Informationsmodell sind folgende Worttypen definiert: »*SimpleWord*«, »*All-Quantor*«, »falsche *Konjunktion*«, »andere *Konjunktion*«, »*Modalverb*«, »$Modalverb_{Anf}$«, »*Indikator*« und »*WeakWord*«.

WWW-Browser: *vorgegebenes System* in der *Arbeitsumgebung* des *ADMIRE-Werkzeugs*, das über seine *HTML-Eingabe-Schnittstelle* in das HTML-Format konvertierte *Dokumente* und *Sichten* einlesen, darstellen und ausdrucken kann.

WWW-GUI: *Schnittstelle* des *WWW-Browsers* zu den *Systemanalytikern* und sonstigen *Informanten*.

Zielanalyse: Tätigkeit in der *Problemanalysephase*. Hierbei werden die langfristigen, fundamentalen Ziele von *Informanten* ermittelt.

Zielsystem: *System*, das im aktuellen Projekt realisiert werden soll. Im ADMIRE-Ansatz ist ein Zielsystem immer ein Software-*System*.

Zusicherung: eine gewünschte Eigenschaft eines *Systems* oder einer Person der *Arbeitsumgebung*. Von Zusicherungen wird angenommen, daß sie während des Einsatzes des *Zielsystems* erfüllt sind. Zusicherungen werden durch *Inhaltselemente* beschrieben.

Zuverlässigkeit: Oberbegriff für *Produktqualitäten*, die das Verhalten eines *Systems* in Extrem- oder Fehlersituationen betreffen, z.B. Robustheit gegen Fehleingaben, Häufigkeit von Fehlern, Fähigkeit, nach einem Systemabsturz fortzufahren.

Zuverlässigkeit eines Prüfergebnisses: Attribut einer *Inhaltsprüfung*, durch das angegeben wird, ob die bei der *Inhaltsprüfung* gefundenen Indizien mit hoher oder niedriger Wahrscheinlichkeit auf Mängel schließen lassen.

Zustand: Attribut eines *Inhaltselements*, eines *Glossareintrags*, eines *Quellenkatalogeintrags* oder eines *Mangeleintrags*, durch das die Stellung des Eintrags im Requirements-Engineering-*Prozeß* beschrieben wird. Mögliche Werte sind »verworfen« und »aktuell«. Bei *Inhaltselementen* gibt es zusätzlich den Wert »zukünftig«.

Index

A

Abkürzung 92
Ablaufumgebung 85, 235
Abnahmetest 21
AbstFinder 224
Abstract 121
abstract 121
accept (ext) 138
accept (ext) with changes 138
accept (int) 138
accept (int) with changes 138
achievable 131
Adäquatheit 23, 101, 132, 175, 235
adequate 130
ADMIRE 14, 235
ADMIRE-Werkzeug 16, 63, 235
Ähnlichkeitsmaß 164
Ähnlichkeitssuche 70, 163, 235
ALECSI 220
All-Quantor 168, 235
Alphabet 109
Änderbarkeit 23
Änderungsdatum 114
Änderungsmanagement 35, 137, 235
Änderungsmeldung 137, 148, 235
Anforderung 20, 77, 119, 235
  funktionale 20, 235
  generische 30
  nicht-funktionale 20, 235
Anforderungsdokument 65, 73, 121, 149, 236
Anforderungssammlung 65, 73, 121, 236
Animation 36, 236
ANLT 221
Annahme, implizite 32
Annotation 65, 77, 87, 121, 236
Annotation 121
Annotationsfenster 196, 236
Annotiertheit 23
ApplGlossaryEntry 130
Arbeitsumgebung 28, 76, 88, 236
ASCII 110
Assertion 119
associatedpersons 116
associatedsystems 119
Assoziation 111, 236
atomic 131
Atomizität 104, 132, 182, 236
Attempto 221
Auftraggeber 27, 236
Aufzählung
  hierarchische 55
  mit Schlüsselwörtern 55
  numerierte 55
  punktierte 55
Ausführbarkeit 24
Ausgabedatum 84
Ausgabeereignis 85, 238
author 114
Automat, endlicher 39
automatic 124
Autor 37, 75, 96, 114, 236

B

Backus-Naur-Form 39
Backus-Naur-Form, erweiterte 39
baselined 139
Baumdiagramm 55
Benutzbarkeit 84, 236
Benutzer 27, 236
Benutzt-Beziehung 93, 236
  direkt gemeinsame 164
  direkte 236
  indirekt gemeinsame 164
  indirekte 236
Benutzt-Prädikat
  Direkt- 113
  Indirekt- 114
Benutzungshandbuch 21
Bezeichner 75
Bildschirmabzug 55
Blank 110
BNF 39
Boolean 109
Brainstorming 33, 237
Buchstabe 109

C

c 165
changedate 114
Chapter 122
chaptercount 173
Check 156, 237
Checkliste 36, 237
Class 118, 120, 121
class 123
classcount 173
classes 117, 118, 120
Code and Fix 13
complete 130
conform 131
conjunctionfree 184
connected 178
Connector 118
consistent 130

contains 113, 124
contclass 124
ContElement 119, 122
contents 119, 121
controlled language 220
ContState 119
cooporate prototyping 34
correct 130
creationdate 127
cstemplate 124
current 116, 118, 119, 177
CurrentContElement 129
CurrentDocSourceEntry 128
CurrentMainEntry 129
CurrentRequirement 129
CurrentSourceEntry 128
CurrentState 121

D

DASERT 217
Date 110
Datenbeschreibungssprache 50
Datenflußdiagramm 40
Datenmodellierer 57
Datum 110
DClass 126
decisions 123
Default 80
default 119
Defect 123
DefectCatalogue 121, 123
DefectClass 124
DefectKind 124
DefectState 124
Definition 91, 117, 237
definition 117
descr 119
description 123
»designer-as-apprentice«-Ansatz 34
Digit 110
directuse 113, 159, 160, 161
DISERT 217
DocAnnotation 129
DocEntry 129
DocParagraph 130
DocSourceEntry 116
Document 121
DocumentEntry 130
Dokument 73, 121, 140, 237
Dokument, externes 96, 116, 140, 237
Dokumentation 73, 237
Dokumenteintrag 99, 130, 237
Dokumentfenster 194, 237
domain 118
Domänenanalyse 30
Domänenmodell 30
done 119, 124
DOORS 214
Drittanbieter 57

E

ε 110
Eason's Approach 33
EClass 126
EEClass 126
Effizienz 84, 237
Eindeutigkeit 23, 104, 132, 185, 237
Eingabedatum 84
Eingabeereignis 85, 238
Einleitung 76, 122
Eintrag 114, 237
  aktueller 177, 237
Eintragstyp 114
Einwort-Term 91, 238
Einzeleintrag 99, 130, 238
elektronisch gespeichert 24
entries 121, 123
Entry 114
EntryAnnotation 129
Entscheidungen 123
Entscheidungsbaum 40
Entscheidungstabelle 40
Entwickler 27, 238
Entwicklungsumgebung 85, 238
Entwurf 21
Entwurfsunabhängigkeit 23
EPOS 216
equivalent 157
ER-Diagramm 39
Ereignis 85, 238
  Ausgabe- 85, 238
  Eingabe- 85, 238
  internes bedingtes 85, 238
  internes zeitliches 85, 238
ETHICS 33
externalconsistent 130
extsourcecount 173
extsources 115

F

Fact 119
Fakt 78, 119, 238
false 109
FClass 126
Fehlererkennungsprozedur 36
Filter 163, 238
Flexion 117, 238
Flexionsangabe 117
Flexionsbeziehung 113, 238
Flexionseditor 196, 238
Flußdiagramm 55
focus groups 34
Fog-Index 226
formalcomplete 170
formalconsistent 176
FrameMaker 213

Freitext 74, 110, 238
Functional Specification 52
Funktion 83, 238
Funktionalität 84, 238
future 119
future workshops 34

G

GATOR 220
glclass 124
Gleichheit
  von Einträgen 143
  von Wörtern 74
GlEntry 117
Glossar 73, 91, 117, 121, 134, 149, 239
Glossareintrag 91, 117, 239
Glossarfenster 194, 239
Glossary 117, 121
GlossaryAnnotation 129
GlossaryEntry 130
GlScope 118
GlState 118
Grammatik, kontextfreie 39
Grundform 91, 113, 239
GUI 191, 239
Gutachter 37

H

Haupteintrag 115, 239
Hauptfenster 194, 239
Hauptmenü 194, 239
heading 122
Homonym 92, 239
HTML-Eingabe-Schnittstelle 191, 239
HTML-Schnittstelle 191, 239
Hyperonym 92, 114, 239
hyperonymes 117
Hyperonym-Verweis 117
Hyponym 92, 114, 239
hyponyme 114
Hyponymiebeziehung 92, 239

I

id 114, 115, 125
Indicator 186
indicatorfree 187
Indikator 186, 239
indirectsources 115
indirectsubclasses 126
indirectuse 159, 161
inflectedword 117
Inflection 117
inflection 113, 159
inflections 117
Informant 26, 239
Informationsmodell 16, 63, 73, 127, 235, 239
Informationsquelle 26, 95, 116, 239
Inhaltselement 77, 78, 119, 239
Inhaltsprüfung 69, 155, 240
initial 138, 139
Innovator 223
Inspektion 38, 70, 162, 240
  fokussierte 69, 158, 240
  negative fokussierte 159, 240
  positive fokussierte 159, 240
Interaktionsdiagramm 39
Interface 118
Interview 33, 240
introduction 122
intsourcecount 174
intsources 115, 168
isauthor 116
Ist-Analyse 30, 240
Ist-Analysedokument 65, 73, 121, 240
Ist-Zustand 30, 240

J

JAD 34
Joint Application Design 34

K

Kapitel 122, 240
Kapitelhierarchie 75, 240
Kapitelnummer 76
KAPS 217
Kardinalität 89, 111, 119, 240
Kategorie
  eines Glossareintrags 117, 240
  eines Inhaltselements 120, 240
  eines Mangeleintrags 97, 123, 240, 242
Kategorisierungshierarchie 83, 124, 240
KI 216
kind 123
Klasse 111, 241
Klassendiagramm 39, 241
Knappheit 24
Kommunikationskante 89, 118, 241
Kommuniziert-Mit-Beziehung 140
Konfigurationsfenster 196, 241
Konjunktion 241
  anreihende 182
  kausale 183
  konzessive 183
  restriktive 183
Konsistenz 22, 101, 176, 241
  externe 23, 102, 131, 180, 241
Kontextdiagramm 241
Korrektheit 22, 100, 131, 167, 241
Kunde 27, 241
Künstliche Intelligenz 216

L

Lastenheft 28, 66, 73, 121, 134, 241
Latex 213
Layout 87
Leerzeichen 110
Lernprozeß 32
LESD 217
Lesen 36

Letter 109
Liest/Schreibt-Beziehung 141
Lösungsidee 123

M

MainEntry 115
MainTerm 129
Manager 37
Mangelart 97, 123, 241
Mangeleintrag 97, 123, 241
Mängelkatalog 73, 121, 123, 241
Mängelkatalogfenster 196, 242
Mangelverweis 97, 123, 242
manual 124
maxword 114
Mehrdeutigkeit 242
Mehrwort-Term 91, 242
Menge 109
  strukturierte 111
Mengendefinition
  extensionale 109
  intensionale 109
Metaanforderung 87, 242
Metadatum 50
Metaspecification 124
Metaspezifikation 124, 242
Metaspezifikationsfenster 196, 242
Metaview 219
Microsoft Word 213
minimal 131
Minimalität 104, 132, 189, 242
ModalVerb 185
Modalverb 185, 242
Model 127
model 128
Moderator 37

N

N 109
Nachvollziehbarkeit
  rückwärtsgerichtete 23
  vorwärtsgerichtete 23
Nachvollzogenheit 167, 242
name 116, 118
natural language processing 216
Natürliche Sprache 39, 242
need change 139
NLDocEntry 129
NL-Eintrag 99, 130, 242
NLEntry 130
NLP 216
Norm 24
Normkonformität 105, 132, 189
Notation 38
  formale 38, 242
  informale 38, 243
  mathematische 38, 243
  operationale 38, 243
  semi-formale 38, 243
notoverspecified 130
notredundant 130

O

OICSI 220
open 119, 124
OtherConjunction 183

P

P(S) 109
Paragraph 110
paragraphs 128
Partei 27, 96, 243
partly done 119
PartySourceEntry 116
PDL 40
PersClass 126
PersonSourceEntry 116
Petri-Netz 40
Pflichtenheft 21, 28, 66, 73, 121, 134, 243
Phrase 74, 110, 243
Phrase 110
Portierbarkeit 84, 243
position 122
potential 124
Potenzmenge 109
Prädikatenlogik 40, 111, 243
Präzision 23, 104, 132, 188, 243
precise 131
PredefinedSystem 118
prio 120
Priorität 80, 120, 124, 243
Priority 120
priority 124
PrjClass 126
PrjState 119
Problem 77, 119, 243
Problem 119
Problemanalyse 243
Problemanalysephase 28, 133, 243
Produktnorm 24
Produktqualität 84, 243
Program Design Language 40
project 118
projectname 127
Projektphase, spätere 69, 133, 137, 244
Projektplanung 86
Projektstand 80, 119, 244
Projektvorgabe 86
Protokoll 162, 244
Prototyping 36, 244
Prozeß 244
Prozeßmodell 16, 63, 133, 235, 244
Prozeßnorm 25
Prüfkriterium 69, 155, 244
Prüfling 37
Prüfungsfenster 196, 244
Prüfverfahren 16, 63, 155, 244
Pseudocode 40
pstate 119

Q

QClass 126
QFD 34
qsactivity 123
Qualität 83
Qualitätenbaum 84
Qualitätskriterium 99, 130, 149, 244
Qualitätssicherung 87
Qualitätssicherungsmaßnahme 123
Quality Function Deployment 34
Quantor 168
quantorfree 169
Quelle 116
Quellenkatalog 66, 73, 95, 116, 121, 134, 154, 244
Quellenkatalogeintrag 95, 116, 244
Quellenkatalogfenster 196, 244
Querverweis
  textueller 75, 244
querverwiesen 24

R

R 109
Rational Rose 223
rctemplate 124
RDT 214
Reading-Ease-Index 225
Realisierbarkeit 23, 132, 181, 244
RECAP 218
Recheck 156, 244
Redundanz 23, 103, 132, 180, 242
ref 116
Referenzunterlagen 37
rejected 116, 118, 119, 124
Relevanz 22
ReqAnalysis 121
ReqCollection 121
ReqDocument 121
request change 139
Requirement 245
  Behavioral 20
  Non Behavioral 20
  Non Operational 20
  Operational 20
Requirement 119
Requirements Creep 34
Requirements Engineering 13, 245
Requirements Management 214
Requirements Specification 52, 245
Requirements-Engineering-Phase 19, 245
Requisite Pro 214
REVIEW 219
Review 37, 245
  technisches 37
Richtlinie 87
rightmode 186
Role 118
Rolle 118
Round Robin Review 38
rstemplate 124
RTM 214

S

Σ 109
SA 40
SADT 217
Schnittstelle 245
Schnittstellenkante 88, 118
scope 117
Sekretär 37
Sequenzdiagramm 39
SESAM 71
Sicht 193, 245
Sichtenfenster 194, 245
SimEntry 164
SimpleWord 94, 245
SimpleWord 160
Simplified English 221
SingleEntry 130
Skizze 55
Soft Systems Methodology 33
Software Engineering 13
Soll-Zustand 245
solution 123
SourceCatalogue 116, 121
SourceEntry 116
SourceState 116
Specification 121
Spezifikation 20
Spezifikationsphase 28, 133, 136, 245
sptemplate 124
SSM 33
Stabilität 81, 120, 124, 245
Stakeholder 26, 245
Stand der Praxis 49
Star Office 213
state 116, 117, 119, 123
Statecharts 39
Statusprüfung 70, 162, 173, 245
structuralconform 190
Structured Analysis 40
Strukturiertheit 23
subchapters 123
substring 113
Substring-Beziehung 113, 245
subterm 117
Suchfenster 194
Synonym 92, 114, 245
synonyme 114
synonymes 117
Synonymiebeziehung 92, 246
Synonym-Verweis 117
SysAdmin-Schnittstelle 192, 246
sysentries 120
sysentrycount 173
SysModelEntry 118
System 246
  der Arbeitsumgebung 246

vorgegebenes 88, 118, 246
Systemanalytiker 25, 246
SystemModel 118, 121
Systemmodell 76, 88, 118, 121, 246
Systemmodelleintrag 88, 118, 246
Systemmodellfenster 194, 246
Systemumgebung 246
Szenario 36, 39, 222

T

Tabelle 55
TargetInterface 118
TargetSystem 118
Task-Analyse 30, 246
TBD 75, 110, 246
TBD 123
TChapter 124
TClass 118
TeamWork 223
Template 124
temporale Logik 40
Term 91, 117, 246
des Anwendungsbereichs 91, 246
projektspezifischer 91, 246
term 117, 125
termcount 173
Terminologie 246
TestCase 119
Testfall 21, 82, 119, 246
tests 119
Textkorpus 223
Thesaurus, inverser 222
title 121
TPriority 121
traced 168
true 109
tsubchapters 125
TVolatility 121
types 125

U

Überprüfbarkeit 23, 103, 132, 182, 247
Überschrift 76, 122
Überspezifikation 103, 132, 180, 242
UML 40, 111, 247
unambiguous 131
understandable 131
Unified Modeling Language 40, 247
uppercase 113
Uppercase-Funktion 113
Ursprung 79, 97, 115, 247
use 114
Use Case 39, 222
UsedGlossaryEntry 130
UsedMainTerm 130

V

validated (need change) 139
Validierung 35, 145, 247
Vanilla Review 38
Vererbung 111, 247
verifiable 131
verified (need change) 139
Verifikation 35, 144, 247
Verständlichkeit 23, 104, 132, 184, 247
Verwendbarkeit 23
Vital Link 214
vola 120
Volatility 120
volatility 124
Vollständigkeit 22, 100, 131, 170, 247
Vorlage 76, 124
Vorlagen-Kapitel 76, 124
Vorlagen-Kapitelhierarchie 76

W

Walkthrough 38
Wartbarkeit 84, 247
Wartung 21
WeakWord 188, 248
WeakWord 188
weakwordfree 188
Wegwerfprototyp 37
»What versus how«-Dilemma 21
Wiederverwendbarkeit 24
Word 110
WordEntry 124
WordType 125
Wort 74, 110
maximales 94
Worteintrag 124, 248
Wortliste 94, 124, 183, 186, 187, 188, 248
Worttyp 94, 125, 183, 186, 187, 188, 248
WrongConjunction 183
WWW-Browser 248
WWW-GUI 248

X

XTie-RT 214

Z

Z 40
Zeichen 110
Zielanalyse 30, 248
Zielsystem 19, 88, 118, 248
Ziffer 110
Zusicherung 78, 119, 248
Zustand 248
eines Glossareintrags 93, 117
eines Inhaltselements 79, 119
eines Mangeleintrags 98, 123
eines Quellenkatalogeintrags 96, 116
interner 84
Zustandsübergangsdiagramm 39
Zuverlässigkeit 84, 248
der Prüfergebnisse einer Inhaltsprüfung 155, 248

Literatur

Aguilera, C.; Berry, D.M. (1990): The Use of a Repeated Phrase Finder in Requirements Extraction. Journal of Systems and Software, 13 (3), S. 209-230.

Albrecht, A.J.; Gaffney, J.E. (1983): Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. IEEE Transactions on Software Engineering; 9 (6), S. 639 - 648.

Balzer, R. (1991): Tolerating Inconsistencies. Proceedings of the 13th International Conference on Software Engineering, 13. - 16. Mai 1991, Austin (Texas, USA), S. 158-165.

Bergenholtz, H.; Pedersen, J. (1994): Zusammensetzung von Textkorpora für die Fachlexikographie. In (Schaeder, Bergenholtz, 1994, S. 161-176).

Beyer, H.R.; Holtzblatz, K. (1995): Apprenticing with the Customer. Communications of the ACM, 38 (5), S. 45-52.

Biebow, B.; Szulman, S. (1992): DISERT: A Tool for the Detection of Incoherences in Software Engineering Requirements Texts. Proceedings of the 5th International Conference on Software Engineering and Its Applications, 7. - 11. Dezember 1992, Toulouse (Frankreich), S. 577-588.

Biebow, B.; Szulman, S. (1994): Acquisition and Validation of Software Requirements. Knowledge Acquisition, 6 (4), S. 343-367.

Boehm, B.W. (1976): Software Engineering. IEEE Transactions on Computers, 25 (12), S. 1226-1241.

Boehm, B.W. (1984): Verifying and Validating Software Requirements and Design Specifications. IEEE Software, 1 (1), S. 75-88.

Boehm, B.W.; Horowitz, E.; Lee, M.J. (1995): Software Requirements Negotiating and Renegotiating Aids: A Theory-W Based Spiral Approach. Proceedings of the 17th International Conference on Software Engineering, 23. - 30. April 1995, Seattle (Washington, USA), S. 243-253.

Booch, G.; Rumbaugh, J.; Jacobson, I. (1998): The Unified Modeling Language User Guide; Addison-Wesley (Reading).

Borillo, M.; Borillo, A.; Castell, N.; Latour, D. (1992): Applying Linguistic Engineering to Spatial Software Engineering: the Traceability Problem. Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 92), 3. - 7. August 1992, Wien (Österreich), S. 593-595.

Bowen, J.P.; Butler, R.W.; Dill, D.L.; Glass, R.L.; Gries, D.; Hall, A.; Hinchey, M.G.; Holloway, C.M.; Jackson, D.; Jones, C.B.; Lutz, M.J.; Parnas, D.L.; Rushby, J.; Wing, J.; Zave, P. (1996): An Invitation to Formal Methods. IEEE Computer 29 (4), S. 16-30.

Bras, M.; Toussaint, Y. (1993): Artificial Intelligence Tools for Software Engineering: Processing Natural Language Requirements. Proceedings of the 8th International Conference on Applications of Artificial Intelligence in Engineering, S. 275-290.

British Standards Institution (1986): British Standard Guide to Specifying User Requirements for a Computer-Based System. BS 6719: 1986, nachgedruckt in (Dorfman, Thayer, 1990, S. 39-51).

Caine, S.; Gordon, E.K. (1975): PDL - a Tool for Software Design. AFIPS National Computer Conference, S. 271-276.

Castell, N.; Slavkova, O.; Toussaint, Y.; Tuells, A. (1994): Quality Control of Software Specifications Written in Natural Language. Proceedings of the 7th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, 31. Mai - 3. Juni 1994, Austin (Texas, USA), S. 37-44.

LeCharlier, B.; Flener, B. (1998): Specifications are Necessarily Informal or: Some More Myths of Formal Methods. Journal of Systems and Software, 40, S. 275-296.

Chen, P. (1977): The Entity Relationship Model: Toward a Unifying View of Data. ACM Transactions on Database Systems, 1 (1), S. 9-36.

Chvalovsky, V. (1983): Decision Tables. Software - Practice and Experience, 13, S. 423-449.

Coad, P.; Yourdon, E. (1991): Object-Oriented Analysis. 2. Auflage. Prentice Hall (Engelwood Cliffs).

Cordes, D.W.; Carver, D.L. (1988): Knowledge Base Applications with Software Engineering: A Tool for Requirements Specifications. Proceedings of the 1st International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, 1. - 3. Juni 1988, Tullahoma (USA), S. 266-272.

Craigen, D.; Gerhart, S. (1995): Formal Methods Reality Check. IEEE Transactions on Software Engineering, 21 (2), S. 90-98.

Dalianis, H. (1992): A Method for Validating a Conceptual Model by Natural Language Discourse Generation. Proceedings of the 4th International Conference on Advanced Information Systems Engineering (CAiSE '92), 12. - 15. Mai 1992, Manchester (England), S. 425-444.

Dankel, D.D.; Walker, W.; Schmalz, M.; Nielsen, K. (1992): A Model for Capturing Requirements. Proceedings of the 5th International Conference on Software Engineering and Its Applications, 7. - 11. Dezember 1992, Toulouse (Frankreich), S. 589-598.

Davis, A.M. (1993): Software Requirements: Objects, Functions & States. Prentice Hall (Engelwood Cliffs).

Davis, A.M.; Overmyer, S.; Jordan, K.; Caruso, J.; Dandashi, F.; Dinh, A.; Kincaid, K.; Ledeboer, G.; Reynolds, P.; Sitaram, P.; Ta, A.; Theofanos, M. (1993): Identifying and Measuring Quality in Software Requirements and Design Specifications. Proceedings of the 1st International Software Metrics Symposium, S. 141-152.

Dorfman, M. (1990): System and Software Requirements Engineering. In (Thayer, Dorfman, 1990, S. 4-16).

Dorfman, M.; Thayer, R.H. (1990): Standards, Guidelines and Examples on System and Software Requirements Engineering. IEEE Computer Society (New York).

Dorfman, M.; Thayer, R.H. (1997): Software Engineering. IEEE Computer Society (New York).

Drappa, A.; Deininger, M.; Ludewig, J.; Melchisedech, R. (1995): Modeling and Simulation of Software Projects. Proceedings of the 20th Annual Software Engineering Workshop, Greenbelt (Maryland, USA), S. 269 - 275.

Duden (1998): Die Grammatik. 6. Auflage. Bibliographisches Institut (Mannheim).

Edwards, M.L.; Flanzer, M.; Terry, M.; Landa, J. (1995): RECAP: A Requirements Elicitation, Capture, and Analysis Process Prototype Tool for Large Complex Systems. Proceedings of the 1st IEEE International Conference on Engineering of Complex Computer Systems, 6. - 10. November 1995, Ft. Lauderdale (Florida, USA), S. 278-281.

Faulk, S. (1997): Software Requirements: A Tutorial. In (Dorfman, Thayer, 1997, S. 82-101).

Fenton, N.; Pfleeger, S.L.; Glass, R. (1994): Science and Substance - A Challenge to Software Engineers. IEEE Software, 11 (4), S. 86-95.

Finney, K. (1996): Mathematical Notation in Formal Specification: Too Difficult for the Masses? IEEE Transactions on Software Engineering, 22 (2), S. 158-159.

Flesch, R.A. (1948): A New Readability Yardstick. Journal of Applied Psychology, 32, S. 221-223.

Frühauf, K.; Ludewig, J.; Sandmayr, H. (2000): Software-Prüfung: Eine Anleitung zum Test und zur Inspektion. 4. Auflage. vdf Hochschulverlag (Zürich).

Fuchs, N.E.; Schwitter, R. (1995): Attempto Controlled Natural Language for Requirements Engineering. Proceedings of the 7th International Workshop on Logic Programming Environments, Dezember 1995, Portland (Oregon, USA).

Fuchs, N.E.; Schwitter, R. (1996): Attempto Controlled English (ACE). Proceedings of the 1st International Workshop on Controlled Language Applications, 26. - 27. März 1996, Leuven (Niederlande).

Gaudel, M.-C. (1994): Formal Specification Techniques. Proceedings of the 16th International Conference on Software Engineering, 16. - 21. Mai 1994, Sorrento (Italien), S. 223-227.

Gause, D.C.; Weinberg, G.M. (1989): Exploring Requirements: Quality before Design. Dorset House (New York).

Gehani, N. (1982): Specifications: Formal and Informal - A Case Study. Software - Practice and Experience, 12, S. 433-444.

Goguen, J.A. (1996): Formality and Informality in Requirements Engineering. Proceedings of the 2nd International Conference on Requirements Engineering, 15. - 18. April 1996, Colorado Springs (Colorado, USA), S. 102-108.

Goldin, L.; Berry, D.M. (1994): AbstFinder, a Prototype Abstraction Finder for Natural Language Text for Use in Requirements Elicitation: Design, Methodology, and Evaluation. Proceedings of the 1st International Conference on Requirements Engineering, 18. - 22. April 1994, Colorado Springs (Colorado, USA), S. 84-93.

Groeben, N. (1982): Leserpsychologie: Textverständnis - Textverständlichkeit. Aschendorff (Münster).

Gunning, H. (1968): The Technique of Clear Writing. McGraw-Hill (New York).

Hall, A. (1990): Seven Myths of Formal Methods. IEEE Software, 7 (5), S. 11-20.

Harel, D. (1987): Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming, 8, S. 231-274.

Harvell, R. (1997): What is a Requirement? In (Thayer, Dorfman, 1997, S. 23-29).

Hatley, D.J.; Pirbhai, I.A. (1987): Strategies for Real-Time System Specification. Dorset House (New York).

Herzog-Tabar, E. (1996): Vergleich von Spezifikationsverfahren am Beispiel einer Airbagsteuerung. Diplomarbeit Nr. 1396, Fakultät Informatik, Universität Stuttgart.

Hofstadter, D.R. (1991): Gödel Escher Bach: ein endloses geflochtenes Band. dtv (München).

Hollocker, C.P. (1990): A Review Process Mix. In (Thayer, Dorfman, 1990, S. 485-491).

Hsia, P.; Davis, A.M.; Kung, D.C. (1993): Status Report: Requirements Engineering. IEEE Software, 10 (6), S. 75-79.

IEEE Computer Society (1990): IEEE Standard Glossary of Software Engineering Terminology. IEEE Computer Society (New York).

IEEE Computer Society (1998): IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830-1998.

Iscoe, N. (1993): Domain Modeling - Overview & Ongoing Research at EDS. Proceedings of the 15th International Conference on Software Engineering, 17. - 21. Mai 1993, Baltimore (Maryland, USA), S. 198-200.

ISO/IEC (1991): ISO/IEC 9126:1991 Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their Use.

Jackson, M. (1995): Requirements and Specifications: A Lexicon of Practice, Principles, and Prejudices. Addison-Wesley (Reading).

Jacobson, I.; Christerson, M.; Jonsson, P.; Övergaard, G. (1995): Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley (Reading).

Jarke, M.; Pohl, K.; Dömges, R.; Jacobs, S.; Nissen, H.W. (1994): Requirements Information Management: The NATURE Approach. Ingénerie des Systèmes d'Informations, 2 (6).

Jepson, B.; Hughes, D. J. (1998): Official Guide to MiniSQL 2.0. Wiley (Chichester).

Jones, C. (1998): Conflict and Litigation Between Software Clients and Developers. American Programmer, 11 (4), S. 10-21.

Kang, K.C.; Ko, K.-I. (1995): PARTS: A Temporal Logic-Based Real-Time Software Specification and Verification Method. Proceedings of the 17th International Conference on Software Engineering, 23. - 30. April 1995, Seattle (Washington, USA), S. 169-176.

Kniffin, J.D.; Kincaid, J.P.; Lang, S.; Thomas, M. (1989): Computer Aids for Authoring Technical Text Written in Controlled English. Proceedings of the Human Factors Society 33rd Annual Meeting, 16.-20. Oktober 1989, Denver (Colorado, USA), S. 1134-1138.

Krauß, S.; Drappa, A. (1998): Einsatz von Ada 95 in einem Forschungsprojekt: Erfahrungen aus Software-Engineering-Sicht. In: Keller, H.B. (Hrsg.): Entwicklung von Software-Systemen mit Ada. Wissenschaftliche Berichte FZKA 6177, Forschungszentrum Karlsruhe, S. 43-53.

Lam, W.; McDermid, J.A. (1997): A Summary of Domain Analysis Experience by Way of Heuristics. ACM SIGSOFT Software Engineering Notes, 22 (3), S. 54-64.

Lamport, L. (1985): LATEX - A Document Preparation System. Addison-Wesley (Reading).

Lauber, R.J. (1982): Development Support Systems. IEEE Computer, 15 (5), S. 36-46.

Lauber, R. (1985): EPOS-Einführung: Einführende Darstellung des entwicklungsunterstützenden projekmanagement-orientierten Spezifikationssystems. 5. Auflage. Universität Stuttgart, Institut für Regelungstechnik und Prozeßautomatisierung.

Lauber, R. (1985a): EPOS-Kurzbeschreibung. Universität Stuttgart, Institut für Regelungstechnik und Prozeßautomatisierung.

Lehman, M.M. (1980): Programs, Life Cycles, and Laws of Software Evolution. Proceedings of the IEEE, 68 (9), S. 1060-1076.

Lehner, F. (1994): Software-Dokumentation und Messung der Dokumentenqualität. Hanser (München).

Lu, R.; Jin, Z.; Wan, R. (1995): Requirements Specification in Pseudo-Natural Language in PROMIS. Proceedings of the 19th Annual International Computer Software and Applications Conference, 9. - 11. August 1995, Dallas (Texas, USA), S. 96-101.

Ludewig, J. (1982): Computer-Aided Specification of Process Control Systems. IEEE Computer, 15 (5), S. 12-20.

Ludewig, J.; Glinz, M.; Matheis, H. (1985): Software-Spezifikation durch halbformale, anschauliche Modelle. Proceedings of the GI/OCG/ÖGI-Jahrestagung 1985, Wien (Österreich), S. 193-204.

Ludewig, J. (1993): Sprachen für das Software Engineering. Informatik-Spektrum, 16, S. 286-294.

Ludewig, J. (Hrsg.) (1994): SESAM - Software Engineering Simulation durch animierte Modelle. Bericht 5/94, Fakultät Informatik, Universität Stuttgart.

Ludewig, J. (1997): Software Engineering: Vorläufiges Skript zur Vorlesung Software Engineering, Fakultät Informatik, Universität Stuttgart.

Macaulay, L.A. (1996): Requirements Engineering. Springer (Berlin).

DeMarco, T. (1982): Structured Analysis and System Specification. Prentice Hall (New York).

McMenamin, S.M.; Palmer, J.F. (1984): Essential Systems Analysis. Yourdon (New York).

Melchisedech, R.; Deininger, M.; Drappa, A.; Hoff, H.; Krauß, S.; Li, J.; Ludewig, J.; Mandl-Striegnitz, P. (1996): SESAM - A Software Engineering Education Tool Based on Graph Grammars. Bulletin of the European Association for Theoretical Computer Science (EATCS), 58, S. 198-221.

Melchisedech, R. (1998): Investigation of Requirements Documents Written in Natural Language. Proceedings of the Conference on European Industrial Requirements Engineering (CEIRE), London (England), nachgedruckt in Requirements Engineering Journal, 3 (2), Springer (Berlin), S. 91-97.

Meyer, B. (1985): On Formalism in Specification. IEEE Software, 2 (1), S. 6-26.

Mindrup, U. (1998): Konzeption und Realisierung eines Werkzeugs zur Erstellung und Verwaltung natürlichsprachlicher Spezifikationen. Diplomarbeit Nr. 1629, Fakultät Informatik, Universität Stuttgart.

Moret, B. (1982): Decision Trees and Diagrams. ACM Computing Surveys, 14 (4), S. 593-623.

Nagl, M. (1992): Einführung in die Softwaretechnik. Skript zur gleichnamigen Vorlesung im WS 91/92, Fakultät Informatik, RWTH Aachen.

Nagl, M. (1990): Softwaretechnik: Methodisches Programmieren im Großen. Springer (Berlin).

Naur, P.; Randell, B. (Hrsg.) (1969): Software Engineering - Report on a Conference Sponsored by the NATO SCIENCE COMMITTEE, 7.-11. Oktober 1968. Garmisch (Deutschland), Science Affairs Division NATO, Belgien.

Nicklas, D. (1999): Umfrageergebnis Requirements School. Internes Dokument, Abteilung Software Engineering, Fakultät Informatik, Universität Stuttgart.

Norris, M. (1986): Z (A Formal Specification Method). STARTS Report, National Computing Centre (NCC), nachgedruckt in (Thayer, Dorfman, 1990, S. 345 - 369).

Osborne, M.; MacNish, C.K. (1996): Processing Natural Language Software Requirements Specifications. Proceedings of the 2nd International Conference on Requirements Engineering, 15. - 18. April 1996, Colorado Springs (Colorado, USA), S. 229-237.

Ousterhout, J.K. (1994): Tcl and Tk. Addison-Wesley (Reading).

Parnas, D.L. (1998): Formal Methods Technology Transfer will Fail. Journal of Systems and Software, 40, S. 195-198.

Partsch, H. (1991): Requirements Engineering. Oldenbourg (München).

Peterson, J. (1977): Petri Nets. ACM Computing Surveys, 9 (3), 1977, S. 223-252.

Porter, A.A.; Votta, L.G. (1994): An Experiment to Assess Different Defect Detection Methods for Software Requirements Inspections. Proceedings of the 16th International Conference on Software Engineering, 16. - 21. Mai 1994, Sorrento (Italien), S. 103-112.

Raggett, D. (1997): HTML 3.2 Reference Specification, W3C Recommendation 14-Jan-1997. Online-Dokument: http://www.w3.org/TR/REC-html32.

Reiners, L. (1996): Stilfibel. 28. Auflage. dtv (München).

Richter, M. (1998): Konzeption und Implementierung von Verfahren zur (halb-) automatischen Prüfung natürlichsprachlicher Spezifikationen. Diplomarbeit Nr. 1654, Fakultät Informatik, Universität Stuttgart.

Robertson, S.; Robertson, J. (1999): Mastering the Requirements Process. Addison-Wesley (Harlow).

Rolland, C.; Proix, C. (1992): A Natural Language Approach for Requirements Engineering. Proceedings of the 4th International Conference on Advanced Information Systems Engineering (CAiSE '92). 12. - 15. Mai 1992, Manchester (England), S. 257-277.

Roman, G.-C. (1985): A Taxonomy of Current Issues in Requirements Engineering. IEEE Computer, 18 (4), S. 14-23.

Ross, D.T.; Schomann, K.E. jr. (1977): Structured Analysis for Requirements Definition. IEEE Transactions on Software Engineering, 3 (1), S. 6-15.

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W. (1991): Object-Oriented Modeling and Design. Prentice Hall (New York).

Rupp, C.; Götz, R. (1997): Sprachliche Methoden des Requirements Engineering (NLP). Internes Papier der Sophist GmbH, Nürnberg.

Sailor, D.J. (1990): System Engineering: An Introduction. In (Thayer, Dorfman, 1990, S. 35-47).

Salek, A.; Sorenson, P.G.; Tremblay, J.P.; Punshon, J.M. (1994): The REVIEW System: From Formal Specifications to Natural Language. Proceedings of the 1st International Conference on Requirements Engineering, 18. - 22. April 1994, Colorado Springs (Colorado, USA), S. 220-229.

Schaeder, B.; Bergenholtz, H. (1994): Fachlexikographie. Gunter Narr Verlag (Tübingen).

Scharer, L. (1981): Pinpointing Requirements. Datamation 1981, nachgedruckt in (Thayer, Dorfman, 1990, S. 17-22).

Schneider, G.; Winters, J.P. (1998): Applying Use Cases: A Practical Guide. Addison-Wesley (Reading).

Sommerville, I.; Sawyer, P. (1997): Requirements Engineering: A Good Practice Guide. Wiley (New York).

Swartout, W.; Balzer, R. (1982): On the Inevitable Intertwining of Specification and Implementation. Communications of the ACM, 25 (7), S. 438-440.

Tavolato, P.; Vincena, K. (1984): A Prototyping Methodology and Its Tool. In: Budde, R. (Hrsg.): Approaches to Prototyping. Springer (Berlin).

Thayer, R.H.; Dorfman, M. (1990): System and Software Requirements Engineering. IEEE Computer Society (New York).

Thayer, R.H.; Dorfman, M. (1997): Software Requirements Engineering. 2. Auflage. IEEE Computer Society (New York).

Ward, P.T.; Mellor, S.J. (1985): Structured Development for Real-Time Systems. Yourdon (New York).

Weber, N. (1994): Maschinelle Hilfen bei der Herstellung, Verwaltung und Überarbeitung von Fachwörterbüchern. In (Schaeder, Bergenholtz, 1994, S. 191-207).

Weidenhaupt, K.; Pohl, K.; Jarke, M.; Haumer, P. (1998): Scenarios in System Development. IEEE Software, 15 (2), S. 34-45.

Weinberg, G.M. (1988): Rethinking Systems Analysis and Design. Little, Brown and Co. (Boston).

Whitgift, D. (1991): Methods and Tools for Software Configuration Management. Wiley (Chichester).

Wilson, W.M.; Rosenberg, L.H.; Hyatt, L.E, (1997): Automated Analysis of Requirements Specifications. Proceedings of the 19th International Conference on Software Engineering, 17. - 23. Mai 1997, Boston (Massachusetts, USA), S. 161-171.

Wilson, W.M. (1997): Writing Effective Requirements Specifications. Online-Dokument des Goddard Space Flight Centers, Software Assurance Technology Center.

Wirth, N. (1985): Programmieren in Modula-2. Springer (Berlin).

Yeh, R.T.; Ng, P. (1990): Software Requirements - A Management Perspective. In (Thayer, Dorfman, 1990, S. 450-461).

Yu, E.S.K. (1997): Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. Proceedings of the 3rd International Symposium on Requirements Engineering, 6. -10. Januar 1997, Annapolis (Maryland, USA), S. 226-235.

Zave, P. (1990): A Comparison of the Major Approaches to Software Specification and Design. In (Thayer, Dorfman, 1990, S. 197-199).

Zave, P.; Jackson, M. (1993): Conjunction as Composition. ACM Transactions on Software Engineering and Methodology, 2 (4), S. 379-411.