Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Automatisierung von Requirements Traceability mittels Information Retrieval

MAGISTERARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften

im Rahmen des Studiums

Masterstudium Informatikmanagement (066 922)

eingereicht von

DI Florian Schnitzhofer

Matrikelnummer 0255427

an der

Fakultät für Informatik der Technischen Universität Wien

Betreuung

Betreuer: a.Univ.-Prof. Dr. Paul Gruenbacher

Wien, 30.11.2010 _____ _____

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

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

Wien, 30. November 2010

DI Florian Schnitzhofer

Abstract

Recovering requirements traceability is a resource and time consuming process that is difficult to automate. This work investigates the application of information retrieval algorithms for automated detection of traceability links between requirements. It will be assumed that the recognition rate will depend on the type of the traceability link. The author investigates the following types of traceability links: structured dependencies, constraint dependencies, and cost/value dependencies. To evaluate the hypothesis a vector space model and latent semantic indexing algorithm are analyzing two case studies with template based requirements. The result shows that structured dependencies are best recognized, despite a generally low detection rate.

Abstract

Das Erstellen von Requirements Traceability ist ein ressourcen- und zeitintensiver Vorgang der schwer automatisierbar ist. Diese Arbeit untersucht den Einsatz von Information Retrieval Algorithmen zur automatisierten Erkennung von Beziehungen zwischen Anforderungen. Es wird angenommen, dass der Beziehungstyp Einfluss auf die Analyseergebnisse hat. Dafür werden folgende drei Beziehungstypen eingeführt: strukturelle Abhängigkeiten, beschränkende Abhängigkeiten und kosten- und wertrelevante Abhängigkeiten. Zur Überprüfung der These werden ein Vector-Space-Model und ein Latent-Semantic-Indexing Algorithmus auf zwei Fallstudien mit schablonenartigen Anforderungen angewendet. Die Auswertung der Ergebnisse zeigt, dass trotz der generell geringen Erkennungsrate die strukturellen Abhängigkeiten besser erkannt werden als andere Beziehungstypen.

Inhaltsverzeichnis

1 Einleitung 8

2 Anforderungsmanagement und Requirements Traceability 11

2.1 Anforderungsdefinition 11

2.2 Anforderungsmanagement 16

2.3 Formulierung von Anforderungen 17

2.4 Requirements Traceability 19

2.5 Typen von Intra-Requirement Traces 22

3 Information Retrieval Algorithmen 24

3.1 Vector-Space-Model 24

3.1.1 Zählen der Wörter vs. Gewichtung der Wörter 25

3.1.2 Berechnung der Ähnlichkeit 26

3.2 Latent Semantic Indedexing 28

3.2.1 Zählen der Wörter 28

3.2.2 Singulärwertzerlegung 30

3.2.3 Dimensionsreduktion 31

3.2.4 Berechnung der Ähnlichkeit 33

4 Information Retrieval für Requirement Traceability: Existierende Forschungsarbeiten 36

5 Automatisierung der Intra-Requirements Traceability 44

5.1 Herausforderungen 45

5.2 Vorgehensweise 46

5.2.1 Aufbereitung der Anforderung 47

5.2.2 Analyse der Abhängigkeiten zwischen den Anforderungen 49

5.2.3 Validieren der Ergebnisse 52

6 Beschreibung der Lösung 56

6.1 Anforderungsmanagementsystem - Starbase (Borland) CaliberRM 56

6.2 Technische Architektur ... 58
6.3 Technischer Aufbau des Werkzeugs ... 60
7 Fallstudien ... 63
7.1 Wohnungspartnerbörse ... 63
7.1.1 Beispielanforderung ... 63
7.1.2 Glossar ... 64
7.2 Pacemaker ... 68
8 Auswertung und Analyse der Ergebnisse ... 70
8.1 Vector-Space-Model ... 71
8.2 Latent-Semantic-Indexing ... 74
8.2.1 Laufzeitverhalten ... 75
8.2.2 Ermittlung eines optimalen Werts für die Dimensionsreduktion ... 76
8.3 Vergleich der Ergebnisse ... 76
9 Didaktische Möglichkeiten zur Verbesserung der Ergebnisse ... 79
9.1 Gemeinsames Vokabular durch ein einheitliches Glossar ... 79
9.2 Vereinheitlichung des Schreibstils ... 80
9.3 Gleicher Wissensstand von allen Anforderungsexperten ... 81
10 Zusammenfassung und Ausblick ... 82
11 Literatur ... 84
12 Anhang - Requirement Traceability Werkzeug ... 88
12.1.1 OSGI - Standard ... 88
12.1.2 Serververbindungs-Assistent ... 90
12.1.3 Anforderungsdarstellung ... 93
12.1.4 Analyse-Assistent ... 96
12.1.5 Modi des Werkzeugs ... 98
12.1.6 Ausgabe der Informationen ... 99

1 Einleitung

Eine Schlüsselkomponente bei der erfolgreichen Implementierung von großen Softwareprojekten[1] ist es die Anforderungen an das Zielsystem zu verstehen und deren Abhängigkeiten zu dokumentieren. Es zählt mittlerweile zur gängigen Praxis, Beziehungen von Anforderungen zu Design-, Implementierungs- und Testdokumenten zu erstellen. Ein weniger beachtetes Themengebiet ist das Auffinden von Abhängigkeiten zwischen Anforderungen. Sehr große Softwareprojekte können sehr viele dokumentierte Anforderungen besitzen. Diese werden von unterschiedlichen Anforderungsexperten erhoben und in einem Anforderungsmanagementsystem verwaltet. Bei einigen tausenden Anforderungen ist es für einzelne Personen sehr schwer den Überblick zu behalten und Abhängigkeiten zwischen den Anforderungen zu finden und zu dokumentieren. Moderne Anforderungsmanagementsysteme unterstützen die Anforderungsexperten beim Dokumentieren dieser Abhängigkeiten, doch sie bieten keine Hilfestellung beim Auffinden von diesen Abhängigkeiten.

Diese Arbeit analysiert den Einsatz und die Eignung von Information Retrieval Algorithmen (IR Algorithmen) zum automatischen Erkennen von Requirements Traceability. Es werden das Vector-Space-Model und der Latent-Semantic-Indexing Algorithmus untersucht und die Ergebnisse anhand von zwei realistischen Fallstudien validiert. Es wird die Annahme getroffen, dass der Abhängigkeitstyp der Anforderung Auswirkungen auf die Erkennungsrate der Algorithmen hat. Zur Überprüfung dieser Annahme werden drei Abhängigkeitstypen definiert: strukturelle Abhängigkeiten, beschränkende Abhängigkeiten, und kosten- und wertrelevante Abhängigkeiten.

Zur Analyse der Algorithmen wird ein Werkzeug basierend auf der Eclipse-Entwicklungsumgebung implementiert. Die Anforderungen werden im Anforderungsmanagementsystem Borland CaliberRM verwaltet und vom Werkzeug geladen. Vor der Verarbeitung der IR-Algorithmen werden die Anforderungstexte aufbereitet. Dabei

[1] Zur Bestimmung der Größe von IT-Projekten gibt es mehrere Vorgehensweisen (z.B.: Function-Point-Analyse, Aufwand in Personentage). In diesem Fall kann die Definition von John08 genommen werden, der Projekte mit ca. 250 Use-Cases als groß definiert.

werden die Texte in atomare Wörter (Token) aufgeteilt, auf ihren Wortstamm zurückgeführt und die Stoppwörter entfernt. Im nächsten Schritt können Abkürzungen oder ähnliche Wörter ersetzt werden. Die Analyse der Anforderungen erfolgt durch die IR-Algorithmen. Dabei wird ein Ähnlichkeitswert zwischen den Anforderungen berechnet. Übersteigt dieser Wert einen vorher festgelegten Grenzwert, so wird von einer Abhängigkeit zwischen diesen zwei Anforderungen ausgegangen.

Zur Validierung der Ergebnisse werden zwei Fallstudien in das Anforderungsmanagementsystem eingespielt. Die erste Fallstudie spezifiziert einen Teil einer Wohnungspartnerbörse im Internet. Es werden exemplarisch 160 Anforderungen ausgewählt und als drei Referenzprojekte angelegt. Ein Anforderungs-Experte trägt für jedes Projekt die Abhängigkeiten zwischen den Anforderungen ein. Die Projekte unterscheiden sich durch den dokumentierten Abhängigkeitstyp. In einem Projekt werden nur die strukturellen Abhängigkeiten eingetragen, im zweiten die beschränkenden und im dritten die kosten- und wertrelevante Abhängigkeiten. Durch dieses Vorgehen ist es möglich die Algorithmen jeweils auf eine Abhängigkeitsart zu untersuchen. Die Ergebnisse werden als Recall (Trefferquote) und Precision (Genauigkeit) berechnet.

Der letzte Teil dieser Arbeit beschäftigt sich mit den didaktischen Möglichkeiten zur Verbesserung der erreichten Ergebnisse. Viele dieser Maßnahmen müssen bereits zum Zeitpunkt der Erstellung der Anforderungen berücksichtigt werden. Die wichtigste Komponente zur Verbesserung der Erkennungsrate ist eine Normierung des verwendeten Vokabulars und der gleiche Wissensstand aller Anforderungsexperten.

In den folgenden Absätzen werden die verfolgten Ziele dieser Arbeit beschrieben. Das Hauptziel besteht im automatisierten Erkennen von Abhängigkeiten zwischen Anforderungen. Der Vorteil der Automatisierung besteht in der raschen Analyse von großen Datenmengen. In der Praxis muss ein Anforderungsanalytiker die Ergebnisse des Algorithmus nochmals kontrollieren. Dennoch macht es einen Unterschied, ob alle Anforderungen geprüft werden müssen, oder nur eine vom Werkzeug getroffene Auswahl. Ziel ist es eine genaue und eingeschränkte Vorauswahl zu treffen, sodass der Anforderungsanalytiker mit weniger Zeitaufwand die Beziehungen zwischen den Anforderungen analysieren kann.

Ein abgeleitetes Ziel ist die optimale Toolunterstützung für diesen Prozess. Das Werkzeug muss sich in die Anforderungsmanagement-Toollandschaft integrieren und den Benutzer bei der Analyse unterstützen.

Bei großen Anforderungsdatensätzen spielt das Thema der Qualitätssicherung eine große Rolle. Ein Ziel ist es bestehende Anforderungsdatenbanken mit bestehenden Verknüpfungen zu kontrollieren.

Die Erreichung des Hauptziels wird mittels bekannter Kennzahlen (Recall und Precision) überprüft. Dabei werden die Ergebnisse aus dem Analysealgorithmus mit einem Testdatensatz von bereits verbundenen Anforderungen verglichen. Für die automatische Erkennung der Abhängigkeiten kommen zwei unterschiedliche Information Retrieval Algorithmen zum Einsatz. Diese werden im Zuge dieser Arbeit einem sinnvollen Vergleich unterzogen.

In den folgenden Absätzen wird der Aufbau der Arbeit beschrieben: In den ersten zwei Kapiteln wird dem Leser der nötige wissenschaftliche Hintergrund zu den Themen Anforderungsmanagement und Requirements Traceability gegeben. Kapitel 3 beschreibt die Information Retrieval Algorithmen und leitet die verwendeten Formeln her. In Kapitel 4 wird auf relevante Arbeiten mit ähnlichen Problemstellungen eingegangen.

In Kapitel 5 wird das wissenschaftliche Vorgehen beschrieben und es werden die einzelnen Verarbeitungsschritte dokumentiert. Die implementierte Lösung und den technischen Aufbau erläutert Kapitel 6.

Die Beschreibung der Fallstudien befindet sich im Kapitel 7 vor der Auswertung und Analyse der Ergebnisse. Kapitel 8 zeigt die Ergebnisse der Studie und stellt den Vergleich beider Algorithmen auf. Kapitel 9 beschreibt abschließend die didaktischen Möglichkeiten zu Verbesserung der Ergebnisse. Im Anhang findet sich die Nutzerbeschreibung des implementierten Werkzeugs.

Alle Inhalte dieser Arbeit sprechen Frauen und Männer gleichermaßen an. Zur besseren Lesbarkeit wird nur die männliche Sprachform verwendet. Die Bezeichnungen „Mitarbeiter“, „Benutzer“, „Architekt“ o. ä. sind deshalb als geschlechtsneutral anzusehen.

2 Anforderungsmanagement und Requirements Traceability

Anforderungen stellen ein zentrales Element in der Entwicklung von neuen Softwaresystemen dar. Speziell bei der Neuentwicklung von großen Informationssystemen hängt eine erfolgreiche Umsetzung von der Qualität und der Vollständigkeit der Anforderungen ab. Im kommenden Kapitel wird der Begriff Anforderung definiert und im Softwareentwicklungsprozess eingeordnet. Danach folgen eine Erörterung möglicher Beziehungen zwischen Anforderungen, Design-Dokumenten, Code und Testbedingungen, sowie eine Klassifizierung dieser Abhängigkeitstypen. Im weiteren Verlauf der Arbeit wird das Wort Anforderung synonym für Software Anforderung verwendet.

Das folgende prominente Beispiel soll die Wichtigkeit von Anforderungsmanagement und den Zusammenhang von Anforderungen und deren Verständnis für den Projekterfolg verdeutlichen [Berg02b]:

„Am 11. März 1993 wurde das Taurus Projekt an der Londoner Börse nach sechs Jahren Entwicklungszeit gestoppt. Taurus hätte als Backbone-System den Handel an der Börse revolutionieren sollen. Das Projekt kostete der Börse 130 Millionen Dollar und externen Firmen nochmals 600 Millionen Dollar. Es wurde nie ein einziges Modul implementiert, da sich die verschiedenen Stakeholder nicht auf die gemeinsamen Anforderungen einigen konnten und die gewünschte Performance nie geliefert werden konnte. Das Scheitern hatte viele Gründe, mitunter ein Grund war die große Anzahl von Anforderungen und deren Abhängigkeiten, sowie die Auflösung dieser Abhängigkeiten.“

2.1 Anforderungsdefinition

Eine wichtige Voraussetzung für die weiteren Ausführungen ist ein einheitliches Verständnis des Begriffs Software Anforderung [John08]. Eine Definition wird im IBM Rational Unified Process (RUP) geliefert [Eeles05]:

“A requirement describes a condition or capability to which a system must conform; either derived directly from user needs, or stated in a contract, standard, specification, or other formally imposed document.”

IEEE definiert eine Software Anforderung als Summe der folgenden Definitionen [IEEE1998]:

„A software capability needed by a user to solve a problem or to achieve an objective;

A software capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document;

The set of all software requirements that forms the basis for subsequent development of the software or software component;

A collective term for the following types of software requirements: functional requirements, performance requirements, external interface requirements, design constraints, and quality attributes."

Diese Definition von IEEE ist in manchen Bereichen sehr spezifisch. Sie definiert die Unterteilung der Anforderungen in verschiedene Anforderungstypen und erwähnt die Überprüfung der Anforderungen gegen einen Vertrag, die Spezifikation oder andere Dokumente. Eine ausreichende Definition ergibt sich jedoch erst in Kombination mit den Qualitätskriterien der Anforderungen. Diese werden am Ende des Kapitels 2.1 diskutiert.

Software Anforderungen werden typischerweise von Anforderungsexperten (Business Analysts) neu erhoben oder aus den Wünschen und Bedürfnissen der Kunden/Benutzer abgeleitet. Die Herkunft dieser Grobanforderungen kann vielfältig sein. Eine Vorgehensweise in der modernen Softwareentwicklung ist, dass man die Systemanforderungen aus Geschäftsprozessen ableitet. Weitere Ursprünge für Grobanforderungen können Feature Requests (zusätzlich geforderte Funktionalität) der Benutzer oder regulatorische Änderungen sein.

Die Grobanforderungen zeichnen ein grobes Zielbild der zukünftigen Anwendung und müssen in iterativen Schritten in eine Sammlung von Software Anforderungen überführt werden. Diese Anforderungssammlung erhebt den Anspruch das Zielsystem vollständig zu beschreiben. Im IEEE Standard 830 - 1998 wird dieses Vorgehen und die Aufbereitung von Anforderungen definiert. Zusätzlich werden Gründe für das Dokumentieren von Software Anforderungen diskutiert. In klassischen Softwareentwicklungsprojekten bilden die Anforderungen eine gemeinsame Basis zwischen Auftraggeber und Auftragnehmer. Die

Abnahme des finalen Softwareprodukts erfolgt dann anhand der spezifizierten Anforderungen. Gut dokumentierte Anforderungen reduzieren den Entwicklungsaufwand und senken die Testkosten. Weitere Vorteile von vollständigen Anforderungen sind die verbesserte Planung und Schätzung von Projekten oder Änderungen, das vereinfachte Betreiben und Portieren der finalen Anwendung und die Abgrenzung des Umfanges (Scope) der zu entwickelnden Software [Rupp09].

Die Definition über gute und vollständige Anforderungen wird in der Literatur sehr ausführlich und kontrovers diskutiert. Als Grundprinzipien haben sich folgende Qualitätskriterien für Anforderungen etabliert [IEEE1998]:

Korrekt (Correct)

Die erhobenen Anforderungen müssen korrekt sein. Hierunter versteht man, dass die Anforderungen mit den Sachverhalten der Umgebung übereinstimmen. Auch wenn dieses Prinzip sehr klar und unmissverständlich klingt, kann durch subjektive Wahrnehmung der Sachverhalt unterschiedlich interpretiert und dokumentiert werden. Im oben genannten Taurus-Beispiel gab es verschiedene Stakeholder-Gruppen mit unterschiedlichen Sichten auf das System. Anforderungen die für eine Gruppe korrekt waren, wurden von der anderen Gruppe als nicht korrekt wahrgenommen [Berg02b]. Die Aufgabe des Anforderungs-Analytikers liegt darin die Anforderungen soweit abzustimmen und zu dokumentieren, dass sie von allen Stakeholdern als korrekt wahrgenommen werden. Das Prinzip der Korrektheit sollte auch für Beziehungen zwischen Anforderungen gelten.

Eindeutig (Unambiguous)

Anforderungen müssen eindeutig sein. Diese Eigenschaft gilt als erfüllt sobald jede Anforderung genau eine Bedeutung hat. Wichtig ist auch, dass die Anforderungen als ganzes ein unmissverständliches Bild des Zielsystems zeichnen.

Vollständig (Complete)

Anforderungen müssen vollständig sein. Die Hauptaufgabe von Anforderungen ist die ganzheitliche Spezifikation eines Software-Systems. Ein Grundparadigma der modernen modellgetriebenen Softwareentwicklung geht davon aus, dass man ein

vollständiges Spezifikationsmodell durch endliche Transformationsschritte in das Zielsystem überführen kann [Almei07]. Für Anforderungen bedeutet dies, sie müssen vollständig die Grenzen und die gewünschten Inhalte des Systems beschreiben [Stach73]. Diese verkürzte und meist noch sehr allgemeine Spezifikation wird dann in den weiteren Softwareentwicklungsphasen weiter verfeinert und mit immer mehr Informationen angereichert. Sind in den Anforderungen noch viele Informationen implizit enthalten, so werden diese bei späteren Transformationen explizit gemacht.

Verifizierbar (Verifiable)

Für die erfolgreiche Abnahme und den Test des Systems müssen die Anforderungen verifizierbar sein. Abnahme- und Testbedingungen können direkt an den Anforderungen stehen oder separat formuliert werden. Im weiteren Verlauf dieser Arbeit wird davon ausgegangen, dass Anforderungen und Testbedingungen getrennt und zeitlich versetzt spezifiziert werden. Zuerst werden die Anforderungen erhoben und neutral formuliert und im weiteren Verlauf des Softwareentwicklungsprozesses werden daraus Testbedingungen abgeleitet. Dies erhöht zwar den Aufwand verbessert jedoch auch die Qualität des Produkts und dient als zusätzliche Qualitätssicherung für die Anforderungen.

Konsistent (Consistent)

Bei steigender Anzahl der Anforderungen steigt auch die Wahrscheinlichkeit von Inkonsistenzen in den Anforderungen. Ein wichtiger Faktor ist die Anzahl der Anforderungsanalytiker, die die Anforderungen erheben. Bei großen Projekten wird die Anforderungsspezifikation unter mehreren Personen aufgeteilt, um Parallelität und effizienteres Arbeiten zu ermöglichen. Speziell an den Schnitträndern der Themengebiete können so leicht Inkonsistenzen entstehen. Diese Schnittstellenprobleme sind durch erhöhte Kommunikation zwischen den beteiligten Anforderungsanalytiker und ausführlichen Qualitätssicherungsmaßnahmen (z.B.: Peer-Review) zu minimieren. Sollten im weiteren Verlauf dennoch Inkonsistenzen entdeckt werden, so müssen die Anforderungen zu einem späteren Zeitpunkt angepasst werden.

Modifizierbar (Modifiable)

Um dieses späteren Anpassungen zu ermöglichen, müssen die Anforderungen modifizierbar verfasst sein. Nicht nur bei der Formulierung der Anforderungen ist dieses Prinzip zu bedenken, sondern auch bei der Festlegung des Anforderungsmanagementprozesses. Der Prozess muss zu jedem Zeitpunkt Änderungen an den Anforderungen zulassen. Dies kann über Change Requests (Änderungsanträge) oder direkte Änderungen geregelt werden. Wichtig ist ein ständiges Pflegen und Aktualisieren der Anforderungen.

Verwendbar für laufenden Betrieb

Eine große Rolle spielt die Aktualität der Anforderungen während des laufenden Betriebs der Anwendung. De-facto-Standards wie die Information Technology Infrastructure Library - ITIL definieren Prozesse für den laufenden Betrieb und den Einsatz von Support-Hierarchien. Ein zentraler Bestandteil dabei ist die gültige Dokumentation des Systems und dazu zählen auch die aktuellen Anforderungen. Ein laufendes System wird im Regelfall kontinuierlichen Verbesserungen und Weiterentwicklungen unterliegen. In diesem Fall müssen neue Anforderungen zu den bestehenden Anforderungen hinzugefügt oder bestehende Anforderungen geändert werden. Diese ständigen Veränderungen am Anforderungsset binden Ressourcen und kosten Geld, sind aber in der Praxis allgegenwärtig. Hier können automatisierte Vorgänge, wie das automatische Auffinden von Abhängigkeiten, großen Mehrwert bewirken.

Nachvollziehbar (Traceable)

Anforderungen müssen nachvollziehbar (traceable) sein. Bei der Entwicklung eines Softwaresystems werden viele Entwicklungsartefakte erstellt und große Mengen von Informationen verwaltet. Die in Modellen enthaltenen Informationen unterliegen zahlreichen Transformationsschritten. Diese können manuell aber auch automatisiert ablaufen. Ziel eines modernen Anforderungsmanagements ist es, sämtliche Informationsstände miteinander zu verknüpfen und somit die Nachvollziehbarkeit zu sichern.

Der IEEE-Standard 830-1984 gibt folgende Definition für Nachvollziehbarkeit (Traceability) [IEEE1998]:

"A software requirements specification is traceable if (i) the origin of each of its requirements is clear and if (ii) it facilitates the referencing of each requirement in future development or enhancement documentation."

Damit wird zwischen zwei verschiedenen Arten von Nachvollziehbarkeit unterschieden. Einerseits muss der Ursprung einer Anforderung dokumentiert werden. Auf der anderen Seite müssen die Verknüpfungen zu den Spezifikations- und Entwicklungsartefakten und neuen Anforderungen verwaltet werden. Die verschiedenen Arten von Verknüpfungen, Vorgehensweisen und die Vorteile von Traceability werden im zweiten Teil dieses Kapitels behandelt. Die Worte Nachvollziehbarkeit, Verfolgbarkeit und Traceability werden im weiteren Verlauf synonym verwendet.

2.2 Anforderungsmanagement

Der Weg von einer wagen Zielvorstellung bis zu den fertig dokumentierten Anforderungen hängt stark vom gewählten Vorgehensmodell ab. Doch auch fest geschriebene Anforderungen können sich im Laufe der Projektzeit ändern. In seiner „Estimation Sotware Costs" Studie geht Jones von einer Änderungsrate von 2% pro Monat bei Softwareprojekten aus [Jones98]. Im Zuge dieser Arbeit wird von einem auf dem V-Modell basierenden Softwareentwicklungsprozess ausgegangen. Die Zeit für die Anforderungsanalyse wird dabei meist zu Beginn des Projektes fixiert und erhöht somit die initiale Qualität und Vollständigkeit der erhobenen Anforderungen. Die bekannteste Variante ist das V-Modell XT und wurde vom deutschen Bundesministerium für Inneres entwickelt. Das XT steht dabei für eXtreme Tailoring und bedeutet dass die modularen, aufeinander aufbauenden Vorgehensbausteine projektspezifisch angepasst werden. Ein Vorgehensbaustein ist eine eigenständige Einheit und je nach Projekttyp werden diese Aktivitäten aneinander gereiht. Die Anforderungserhebung findet sich im Bereich der „Systementwicklung" in den Vorgehensbausteinen „Anforderungserstellung" (siehe Abbildung 1: „Requirements Specified") und „Systemerstellung" (siehe Abbildung 1: „System Specified" und „System Designed") wieder.

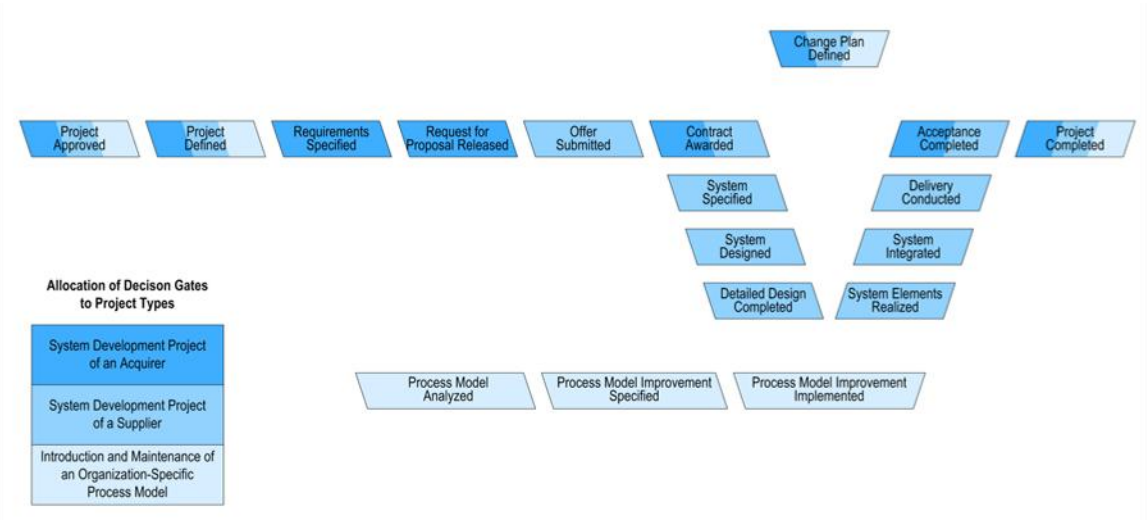


Abbildung 1: V-Modell XT - Entscheidungspunkte für die Projektdurchführungsstrategie [VMXT]

2.3 Formulierung von Anforderungen

In den vorhergehenden Kapiteln wurde eine Definition von Anforderungen gegeben und die notwendigen Qualitätskriterien definiert. Im Weiteren wird auf den Aufbau und die Formulierung von Anforderungen eingegangen.

Für die Dokumentation von Anforderungen wurden unterschiedliche Methoden und Darstellungsweisen entwickelt. Einige Verfahren empfehlen die Darstellung der Anforderungen mittels Diagrammen. In Kombination aus Zustands-, Ablauf- und Komponentendiagrammen lassen sich die gewünschten Zielanwendungen beschreiben. Aus Erfahrungen mit praktischen Kundenprojekten schließt der Autor dieser Arbeit, dass diese ausgefeilte Art der Spezifikation zu einem sehr frühen Zeitpunkt der Anforderungsanalyse (noch) nicht möglich ist. In der sehr unstrukturierten Anfangsphase des Entwicklungsprozesses werden viele Anforderungen mit Prosa dokumentiert.

Die weitere Arbeit beschränkt sich auf den Umgang mit Prosa-Anforderungen. Die größte Hürde der prosasprachlichen Spezifikation ist der Interpretationsspielraum der menschlichen Sprache [Rupp09]. Wichtig für eine Anforderungsdokumentation in natürlicher Sprache ist ein gemeinsames "Sprachuniversum". Dieses kann als Glossar gepflegt werden und dient als Grundlage für alle erfassten Anforderungen. Chris Rupp beschreibt in ihrem Buch „Requirements-Engineering und –Management“ einen schablonenbasierten Ansatz für die Dokumentation von Anforderungen. Das schablonenartige Erfassen erhöht das gleiche Verständnis und verringert den Aufwand beim Schreiben. Die Schablonen-Grammatik zwingt

die Autoren zu einem einheitlichen Stil und unterbindet ausschweifende und missverständliche Formulierungen. Ein wichtiger Vorteil des Schablonensystems von Rupp ist die leichte Erlernbarkeit der syntaktischen Vorgabe. Somit sind auch weniger erfahrene Anforderungsanalytiker in der Lage hochwertige Anforderungen zu formulieren.

Die Schablonenschreibweise wurde für diese Arbeit gewählt, da davon ausgegangen wird, dass sich durch diese Standardisierung die Analyseergebnisse der Information Retrieval Algorithmen verbessern.

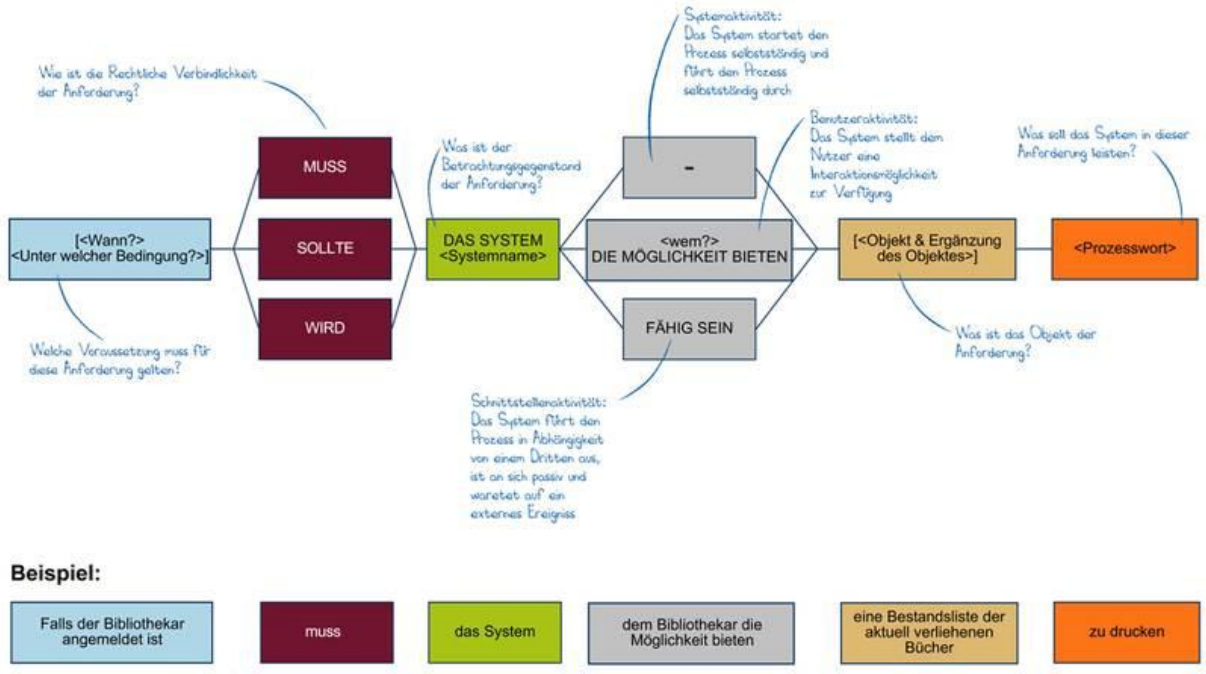


Abbildung 2: Anforderungsschablone zur einheitlichen Spezifikation [Rupp09]

Die Anforderungsschablone besteht aus sechs Modulen. Der vordere Teil beschreibt unter welcher Bedingung die Anforderung ihre Gültigkeit erhält. Mit dem Verb wird die Wichtigkeit der Anforderung festgelegt. Dabei kann zwischen den Wörtern „wird“, „kann“/ „sollte“ und „muss“ gewählt werden. Zusätzlich zu ihrer eigentlichen semantischen Bedeutung drücken diese Wörter die Wichtigkeit der Anforderung aus (wird > kann > muss). Der nächste Block beschreibt das System oder die Systemkomponente für die die Anforderung gilt. Der mittlere Block des Anforderungssatzes bestimmt die Art der

Funktionalität. Systemaktivitäten können durch Benutzerinteraktionen, selbständige Systemaktivitäten oder Schnittstellen angestoßen werden. Diese Art der Ausführung ist für eine Anforderung wichtig und muss in späteren Phasen der Implementierung klar ersichtlich sein. Das Satzende bilden das Objekt und das Prozesswort. Das Objekt und die Ergänzung des Objekts konkretisierten die Anforderung und vermeiden somit Missverständnisse. Das Prozesswort spezifiziert die geforderte Funktionalität. Diese Prozesswörter sollten zentral definiert werden (z.B.: in einem Glossar), sodass jeder Stakeholder das gleiche darunter versteht. Beim praktischen Einsatz der Schablone kann auch eine leicht veränderte Variante zum Einsatz kommen. Die Bedingung am Anfang des Satzes kann ausgelassen werden wenn sie nicht benötigt wird. Dann beginnt der Satz mit dem Block „Das System“ und als nächstes das Verb („wird“, „kann“/„sollte“, „muss“).

Bei großen industrialisierten Softwareentwicklungsprojekten werden sehr große Mengen an Anforderungen (mehrere Hunderte bis Tausende Anforderungen) definiert. Folgt man einem der etablierten Softwareentwicklungsmodelle, so werden die Anforderungen vor der Implementierung definiert und sollten im Laufe des Softwareentwicklungsprozesses möglichst stabil bleiben. Die wichtigste Herausforderung dabei besteht in der Beherrschung der Komplexität und in der Verfolgung von Anforderungen. Das nächste Kapitel beschäftigt sich mit dieser Problematik und erläutert die verschiedenen Methodiken zum Umgang mit Abhängigkeiten von Anforderungen.

2.4 Requirements Traceability

Pinheiro beschreibt im Buch “Perspectives on Software Requirements” das Grundprinzip von Anforderungs-Traceability [Pinh04]: *“The concept of requirements tracing is quite simple: to follow relationships or links”*. Dieser einfache Satz bringt den Inhalt und die Ziele der Nachvollziehbarkeit sehr gut auf den Punkt. Ausgehend von Anforderungen gibt es Beziehungen und Verlinkungen zu anderen Artefakten. Eine Abhängigkeit oder „Trace“ ist definiert als eine Markierung, Verlinkung, Kennzeichnung, die ein geschehenes Ereignis oder eine Verbindung anzeigt. In der weiteren Arbeit werden die Ausdrücke Requirement Traceability, Anforderungs-Traceability und Anforderungsabhängigkeiten (Definition siehe Kapitel 2.5) synonym verwendet.

Anforderungs-Traceability ist die Verfolgung von Anforderungen durch den gesamten Softwareentwicklungsprozesses. Gotel und Finkelstein definieren Anforderungs-Traceability wie folgt [Gotel94]:

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

Sie machen explizit darauf aufmerksam, dass nicht die reine Verlinkung sondern auch die Beschreibung der Beziehung von Bedeutung ist. Beziehungsinformationen sollen die Einflüsse von Artefakten verdeutlichen und diese nachvollziehbar machen. Ein Aspekt der Anforderungs-Traceability ist die Rückwärts Traceability (Backward Traceability). Hierbei wird die Anforderung zu ihrem Ursprung in Beziehung gesetzt. Dies können Personen, Institutionen, Gesetze oder Ziele sein. Vorwärts Traceability (Forward Traceability) ist die Möglichkeit eine Anforderung mit einem Entwurfs- oder Implementierungsartefakt im Softwareentwicklungsprozesses in Beziehung zu setzen. In diesem Fall zieht man eine Verbindung zu Design-Dokumenten, Implementierungsartefakten, Testbedingungen oder Systemdokumentationen. Vorwärts-Traceability bildet die Grundlage für Risikoabschätzungen und Auswirkungsanalyse (Impact Analysis) von neuen Anforderungen oder Anforderungsänderungen. Auch für eine koordinierte Abnahme des Systems müssen Testfälle auf Anforderungen zurückgeführt werden können. Nur so ist es möglich sicherzustellen, dass auch alle Anforderungen erfüllt sind.

Anforderungen sind das Resultat eines Auswahl- und Analyse-Prozesses. Das Tracing von von Anforderungen bevor sie in die Spezifikation einfließen wird als Pre-Requirements-Traceability bezeichnet. Post-Requirements-Traceability beschreibt die Beziehungen einer Anforderung die nach ihrer Einbindung in die Anforderungsspezifikation auftreten [Pinh04].

Für die weitere Arbeit ist die Unterscheidung zwischen Intra- und Extra-Requirements Traceabilty von großer Bedeutung. In der Literatur wird Extra-Requirements Traceabilty auch als Inter-Requirements Traceability bezeichnet [Anqu08]. Die Möglichkeit zwischen Anforderungen auf dem gleichen Abstraktionsniveau Beziehungen zu erstellen wird als Intra-Requirement Traceability bezeichnet [Anqu08]. Beziehungen von Anforderungen zu externen

Artefakten bezeichnet man als Extra Requirements Traceability [Pinh04]. Diese Arbeit beschäftigt sich mit Beziehungen und Abhängigkeiten zwischen Anforderungen (Intra-Requirements Traceability). Diese sind besonders wichtig für die Anforderungsanalyse bei Anforderungsänderungen oder Weiterentwicklungen [Anqu08].

Die Definitionen von den oben genannten Traceability Möglichkeiten sind an einigen Stellen überlappend und sollen in der folgenden Grafik gegenüber gestellt werden [Pinh04].

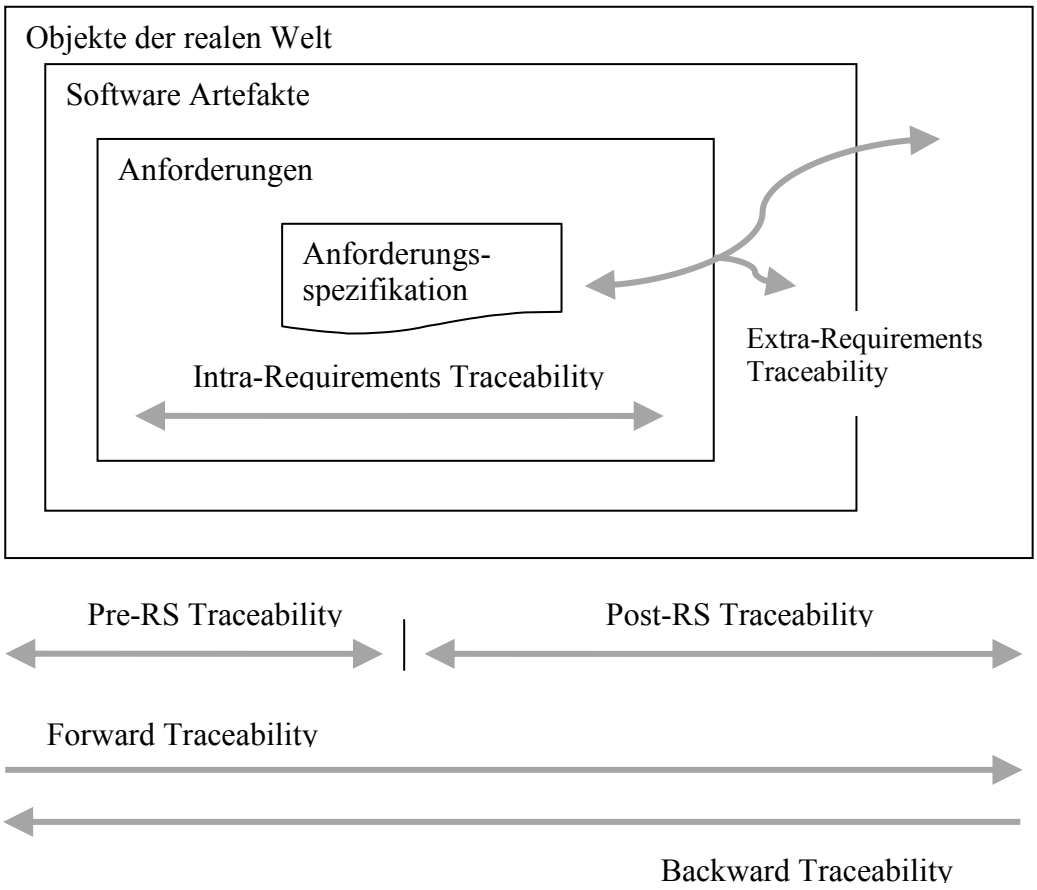


Abbildung 3: Möglichkeiten der Traceability [Pinh04] [Anqu08]

In den vorherigen Kapiteln wurde Traceability als eine wichtige Eigenschaft von Anforderungen definiert und die verschiedenen Möglichkeiten der Verfolgbarkeit aufgezeigt. Ramesch [Rame01] und Pohl [Phol09] geben folgende Gründe für den praktischen Einsatz von Anforderungs-Traceability an: Die Nachvollziehbarkeit von Anforderungen ermöglicht

die Nachweisbarkeit, dass alle Anforderungen gemäß den Zielen im System umgesetzt und getestet wurden. Eine der wichtigsten Implikationen von durchgängiger Anforderungs-Traceability ist die Identifikation von überflüssigen Systemkomponenten und von überflüssigen Anforderungen. Studien haben gezeigt, dass der Aufwand für nachträgliche Anpassungen am System nur schwer abschätzbar sind und meist unterschätzt werden [Rame01]. Für diesen Fall ist es wichtig eine Auswirkungsanalyse auf Basis der Traceability-Informationen durchzuführen. Gute Nachvollziehbarkeit von Anforderungen bietet ein besseres Verständnis des Gesamtsystems und kann so die Wiederverwendung von Systemkomponenten fördern. Die Zurechenbarkeit von Aufwand zu Zielen, sowie ein risikobasierter Testansatz werden mit Hilfe dieser Informationen möglich. Schlussendlich verbessert sich die Ursachenforschung und Aufwandsabschätzung für die Fehlerbehebung und macht Abhängigkeiten von Fehlern deutlich.

2.5 Typen von Intra-Requirement Traces

Der weitere Inhalt dieser Arbeit fokussiert sich speziell auf die Intra-Requirements Traceability. Dabei kann man zwischen folgenden Typen von Abhängigkeiten unterscheiden:

- Strukturelle Abhängigkeiten (Structured Dependencies)
- Beschränkende Abhängigkeiten (Constraint Dependencies)
- Kosten- und wertrelevante Abhängigkeiten (Cost/Value Dependencies)

Bei der strukturellen Abhängigkeit von Anforderungen geht man von einer baumähnlichen Anforderungsstruktur aus. In der Literatur spricht man auch von einer Verfeinerung der Anforderungen oder hierarchischem Aufbau der Anforderungen [Rupp09]. Diese Art der Anforderungsstruktur resultiert aus dem Anforderungserfassungsprozess. Im ersten Schritt dokumentiert man die Grobanforderungen. Diese können sich zum Beispiel auf Ebene der Use Cases (Anwendungsfälle) befinden. Mittels nachfolgender Iterationsschritte werden die Anforderungen von einer Detailebene zur folgenden Detailebene verfeinert. Die unteren Anforderungsebenen ergänzen oder verfeinern den Inhalt der darüber liegenden Anforderungen und werden im Verbund verstanden. Die Ursprungsanforderungen sowie alle weiteren Anforderungsebenen sind gleichzeitig gültig und alle Ebenen gelten als Teil der Gesamtspezifikation. Strukturelle Abhängigkeiten sind die Teilbäume unter den

Grobanforderungen. Diese Abhängigkeiten gelten von oben nach unten und somit muss jede Feinanforderungen einer Grobanforderung (Elternanforderung) zuordenbar sein.

Die Anforderungen des Fallbeispiels sind hierarchisch gegliedert und geben eine klare Vorstellung von strukturellen Abhängigkeiten. Die Struktur wurde auf vier Ebenen beschränkt, wobei auf oberster Ebene die Use Cases beschrieben werden. Die zweite Ebene betrachtet die einzelnen Anwendungsfälle aus Sicht des Benutzers. Ebene drei und vier dienen der Spezifikation der Detailinformationen und der Systemkomponenten.

Beschränkende Abhängigkeiten (Constraint Dependencies) können zwischen allen Anforderungsebenen entstehen. Bei dieser Art der Abhängigkeit werden Beeinflussungen und Beschränkungen zwischen den Anforderungen dokumentiert. Es gibt mehrere Ursachen die Beschränkungen zwischen Anforderungen erzeugen können. Auf der einen Seite führen fachliche oder funktionale Abhängigkeiten zwischen Anforderungen zu erhöhtem Konfliktpotential. Auf der anderen Seite können Anforderungen auch Ausnahmen, Randbedingungen oder Realisierungsvorgaben von anderen Anforderungen sein. In einer späteren Phase im Softwareentwicklungsprozess sind diese Querverweise von besonderer Bedeutung. Bei Änderungen einer Anforderungskomponente sind alle verbundenen Anforderungen zu analysieren und gegebenenfalls auch anzupassen. Der Systemtest kann durch diesen Abhängigkeitstyp zusätzliches Fehlerpotential erkennen und das Zusammenspiel der betroffenen Systemkomponenten verbessert testen.

Bei kosten- und wertrelevanten Abhängigkeiten (Cost/Value Dependencies) stehen die Anforderungen in einem Kostenverhältnis zueinander. In der Praxis bestimmen diese Anforderungen und ihre Abhängigkeiten den Umfang des Projektes. Beim kosten- und wertrelevanten Abhängigkeitstyp kann eine Abhängigkeit auch den gewonnenen Wert für die Applikation oder dem Auftraggeber beeinflussen. Kosten und Wert einer Anforderung hängen sehr stark von der Realisierung und der Architektur des Zielsystems ab. Kosten- und wertrelevante Abhängigkeiten treten oft bei Realisierungsvorgaben von Anforderungen auf.

3 Information Retrieval Algorithmen

Das folgende Kapitel beschäftigt sich mit zwei Algorithmen aus dem Bereich Information Retrieval und beschreibt deren Funktionsweise. Der Vector-Space-Model Algorithmus gilt als grundlegender Algorithmus zum Erkennen von Ähnlichkeiten zwischen zwei Textdokumenten. Das Latent-Semantic-Indexing wurde durch intensives Forschen an Suchalgorithmen entwickelt und identifiziert Muster und Abhängigkeiten zwischen Wörtern in Textdokumenten. In der weiteren Arbeit wird der Einsatz und Eignung dieser Algorithmen für das automatische Erkennen von Abhängigkeiten zwischen Anforderungen geprüft. Mit einer Gegenüberstellung der Evaluationsergebnisse wird eine Aussage über den Eignungsgrad dieser Algorithmen zur Abhängigkeitserkennung von Anforderungen getroffen.

3.1 Vector-Space-Model

Das Vector-Space-Model (VSM) ist ein Standardverfahren, Textdokumente durch ihre Wörter als Punkte in einem hochdimensionalen und metrischen Vektorraum zu repräsentieren [Salt75].

In einem ersten Schritt wird das Textdokument in einzelne Wörter aufgeteilt. Jedes Wort stellt dabei eine Dimension dar und die Anzahl der Wörter ergibt die Position im Vektorraum. Die nachfolgende Darstellung zeigt die Repräsentation von zwei Anforderungen im Vektorraum [Natt05].

	container	containerization	item	level	main	package	price	print	process	purchase	sale	sequence	statistics	type
D_a	1	2	2	0	0	1	3	0	7	0	0	0	0	2
D_b	6	0	5	4	2	0	1	2	0	1	1	0	8	1

Tabelle 1: Exemplarisches Beispiel für einen Vector Space zwischen zwei Anforderungen [Natt05]

3.1.1 Zählen der Wörter vs. Gewichtung der Wörter

Die genaue Position der Vektoren hängt von der Gewichtung der Wörter ab. Es gibt mehrere Möglichkeiten diesen Algorithmus zu implementieren.

Die einfachste Variante ist das Vorkommen der Wörter zu zählen und damit die Position festzulegen.

Für den Vektors $v_d = [w_{1,d}, w_{2,d}, \dots, w_{N,d}]^T$ aus dem Dokument d gilt,

$$n_{w_t,d} \dots Häufigkeit\ von\ Wort\ w\ im\ Dokument\ d$$

$$w_{t,d} = n_{w_t,d}$$

Dieses Vorgehen stößt bei großen Häufigkeiten an seine Grenzen. Eine einfache Möglichkeit dieses Problem auszugleichen, ist das Gewichten der Wörter mit folgender Formel.

$$w_{t,d} = 1 + \ln(n_{w_t,d})$$

Im klassischen Vector-Space-Model von Salton, Wong und Yang [Salt75] kam das Term Frequency-Inverse Document Frequency (tf-idf) Verfahren zur Gewichtung der Wörter zum Einsatz. Dieses Modell setzt voraus, dass man globale Parameter, wie die gesamte Dokumentengröße zur Verfügung hat.

Die Methode besteht aus der Berechnung der Term Frequency (Termfrequenz) und der Inverse Document Frequency (Inverse Dokumenthäufigkeit). Die Termfrequenz eines gegebenen Dokuments gibt einen Hinweis auf die Bedeutung des Wortes für das Dokument. Die Inverse Document Frequency berechnet die allgemeine Bedeutung des Wortes für die Gesamtmenge der betrachteten Textdokumente.

Die Gewichtung des Vektors $v_d = [w_{1,d}, w_{2,d}, \dots, w_{N,d}]^T$ aus dem Dokument d gilt,

$$w_{t,d} = tf_{t,d} * idf_i$$

Berechnung der Term Frequency

$$tf_{t,d} = \frac{n_{wt,d}}{\sum_k n_{k,d}}$$

$$\sum_k n_{k,d} \quad \ldots Summe\ aller\ Wörter\ im\ Dokument\ d$$

Berechnung der Inverse Document Frequency:

$$idf_i = log \frac{|D|}{1 + |\{d: t_i \in d\}|}$$

$$|D| \ldots Summe\ aller\ untersuchten\ Dokumente$$

$$1 + |\{d: t_i \in d\}| \ldots Summe\ aller\ Dokumente\ in\ denen\ t_i\ vorkommt$$

3.1.2 Berechnung der Ähnlichkeit

Die Ähnlichkeit zwischen zwei Dokumenten erhält man durch Berechnung des Cosinus-Werts. Dieser Wert gibt den Winkel zwischen zwei Vektoren an.

Folgendes Beispiel dient zur Veranschaulichung des Vector-Space-Models. Das Beispiel beinhaltet drei unterschiedliche Wörter (t_1, t_2, t_3). Diese werden von drei Dokumenten (D_a, D_b, D_c) mit verschiedenen Häufigkeiten verwendet. Abbildung 4 stellt diesen drei-dimensionalen Vektorraum bildlich dar.

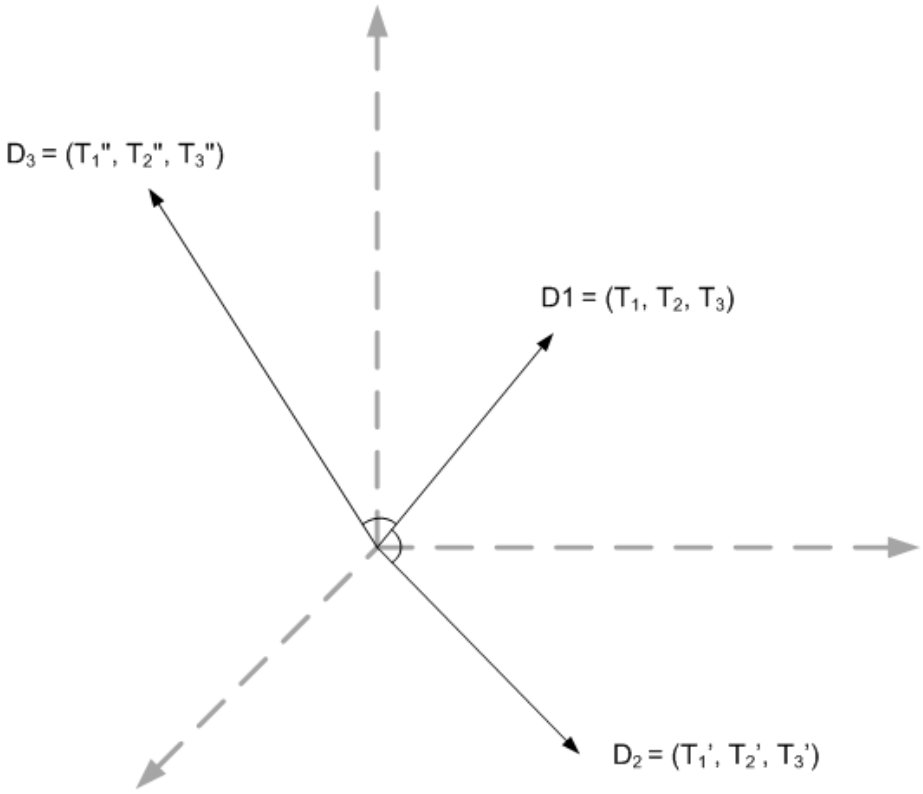


Abbildung 4: Exemplarische Darstellung eines drei dimensionalen Vektorraums

Die Winkel zwischen Dokument D_a und D_b wird mittels der Cosinus-Formel berechnet:

$$\cos(D_a, D_b) = \frac{D_a * D_b}{|D_a| * |D_b|}$$

$D_a * D_b$ berechnet das Skalarprodukt zwischen den zwei Dokumentvektoren. Dieses wird durch Multiplikation von allen Wort-Häufigkeiten und anschließender Aufsummierung berechnet.

Für die Berechnung der Häufigkeiten der Wörter wurde in dieser Arbeit die Formel $w_{t,d} = 1 + \ln(n_{w_t,d})$ gewählt. Diese ist schnell zu berechnen und unabhängig vom Gesamten Dokumentenraum.

Da der Winkel zwischen zwei Vektoren eine Aussage zur Lage von Vektoren zueinander trifft, kann man diesen Wert auch zur Ähnlichkeitsberechnung heranziehen. Damit ergibt sich folgende Formel zur Berechnung der Ähnlichkeit mit Hilfe des Vector-Space-Models:

$$sim(D_a, D_b) = \cos(D_a, D_b) = \frac{\sum_i [1 + \log_2 D_a(i)] * [1 + \log_2 D_b(i)]}{\sqrt{\sum_i [1 + \log_2 D_a(i)]^2} * \sqrt{\sum_i [1 + \log_2 D_b(i)]^2}}$$

3.2 Latent Semantic Indedexing

Bei Latent-Semantic-Indexing (LSI) handelt es sich um ein zusätzliches Verfahren, das auf dem Vector-Space-Model Verfahren aufsetzt. Der Algorithmus ist patentgeschützt und wurde 1990 erstmals durch Deerwaster beschrieben [Deer90]. Beim Vector-Space-Model werden die einzelnen Wörter unabhängig voneinander betrachtet. Dadurch entsteht das Problem, dass einzelne Wörter das Ergebnis maßgeblich beeinflussen, auch wenn diese nur eine Nebenrolle im Dokument spielen. Latent-Semantic-Indexing versucht die einzelnen Wörter eines Dokuments im Zusammenhang zu sehen und gewinnt somit zusätzliche semantische Information. Das Wort „Weihnachtsferien“ in einem Dokument über Buchhaltungssoftware würde somit einen geringeren Score erhalten, da es semantisch nicht in Zusammenhang mit den Wörtern der Buchhaltungssoftware steht.

Der LSI Algorithmus behandelt die Auswertung von Wörtern und Dokumenten als ein statistisches Problem. Der originale Wortlaut der ursprünglichen Veröffentlichung von [Deer90] lautet wie folgt:

“The proposed approach tries to overcome the deficiencies of term-matching retrieval by treating the unreliability of observed term-document association data as a statistical problem. We assume there is some underlying latent semantic structure in the data that is partially obscured by the randomness of word choice with respect to retrieval. We use statistical techniques to estimate this latent structure, and get rid of the obscuring "noise". A description of terms and documents based on the latent semantic structure is used for indexing and retrieval.”

3.2.1 Zählen der Wörter

Wie schon beim VSM beschrieben, müssen auch bei diesem Algorithmus die Dokumente in Wörter aufgeteilt und gezählt oder gewichtet werden. Es kommen identische Gewichtungsmethoden wie beim VSM zum Einsatz. Für die Berechnung der Werte im implementierten Algorithmus wurde das Wörter zählende Verfahren (Primitive Term Count Model) angewandt.

Mit diesen Werten wird die Termfrequenz Matrix A erstellt. Prinzipiell ist diese Matrix der gemeinsame Nenner zwischen VSM und LSI Algorithmus.

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Das folgende Beispiel stammt aus dem Buch „Information Retrieval: Algorithms and Heuristics“ von David A. Grossman [Gross04] (Korrigierte Werte übernommen von [Garc06]). Da der LSI-Algorithmus viel im Umfeld von Suchmaschinen verwendet wird, ist auch das Beispiel eine Suchabfrage.

Der Suchstring besteht aus den drei Wörtern:

- q: gold silver truck

Die Dokumentensammlung besteht aus drei Sätzen:

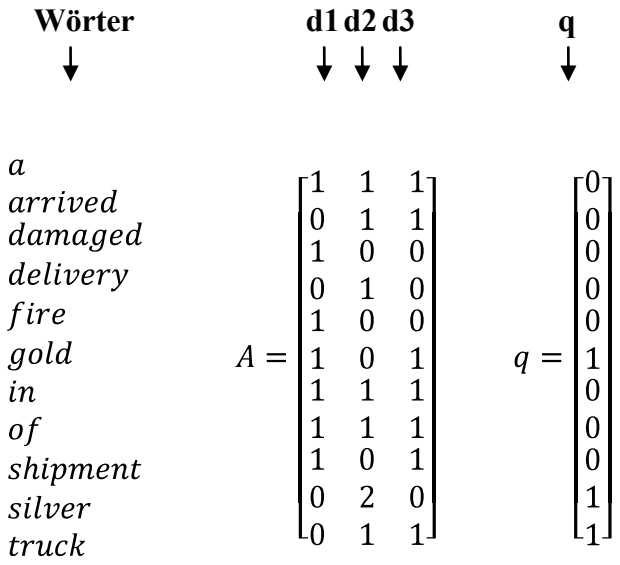
- d1: Shipment of gold damaged in a fire.
- d2: Delivery of silver arrived in a silver truck.
- d3: Shipment of gold arrived in a truck.

Die Vorverarbeitung der Wörter erfolgt in folgender Reihenfolge:

1. Text wird in Wörter aufgeteilt
2. Wörter werden in Kleinbuchstaben umgewandelt
3. Wörter werden alphabetisch sortiert

Stoppwörter werden nicht ignoriert und die Wörter werden nicht auf ihren Wortstamm zurückgeführt (stemming).

Die ermittelten Wörter der Dokumentensammlung werden gezählt und damit die Termfrequenz Matrix A erstellt:



Die weitere Verarbeitung erfolgt mit der Matrix A. Der Suchvektor q wird gegen Ende wieder mit in die Berechnung aufgenommen.

3.2.2 Singulärwertzerlegung

Schon bei einer Dokumentensammlung von drei Sätzen erreicht die Matrix A eine beachtliche Größe. Diese wird über die Singulärwertzerlegung in Matrizen aus ihren Eigenvektoren und Eigenwerten aufgespaltet. Bei der Anwendung des LSI-Algorithmus geht man davon aus, dass die Matrix A aus semantisch wichtigen Dimensionen (Hauptdimensionen) und für die Bedeutung des Dokuments unwichtige Dimensionen besteht. Diese Hauptdimensionen müssen erhalten bleiben, wobei die anderen Dimensionen zusammengefasst werden können. Durch das Vergleichen der semantischen Konzepte einer Dokumentensammlung, kann somit eine deutliche Verringerung der Komplexität und damit der Laufzeit erreicht werden. Diese dimensionsreduzierte, approximierende Matrix A wird als Semantischer Raum bezeichnet.

Die Matrix A wird in die drei Komponenten U, S, Vund V^T zerlegt (Singulärwertzerlegung):

$$Gleichung\ 1\colon A = U * S * V\text{^}T$$

Die orthogonalen Matrizen U und V enthalten die Eigenvektoren von A^TA bzw. AA^T. Die Matrix S besteht aus einer Diagonalmatrix von Singulärwerten, die sich als Wurzeln der Eigenwerte von A^TA berechnen.

Für das Beispiel ergibt das folgende Matrizen:

$$A = \begin{bmatrix} -0{,}4201 & 0{,}0748 & -0{,}0460 \\ -0{,}2995 & -0{,}2001 & 0{,}4078 \\ -0{,}1206 & 0{,}2749 & -0{,}4538 \\ -1{,}576 & -0{,}3046 & -0{,}2006 \\ -0{,}1206 & 0{,}2749 & -0{,}4538 \\ -0{,}2626 & 0{,}3794 & 0{,}1547 \\ -0{,}4201 & 0{,}0748 & -0{,}0460 \\ -0{,}4201 & 0{,}0748 & -0{,}0460 \\ -0{,}2626 & 0{,}3794 & 0{,}1547 \\ -0{,}3151 & -0{,}6093 & -0{,}4013 \\ -0{,}2995 & -0{,}2001 & 0{,}4078 \end{bmatrix} \qquad S = \begin{bmatrix} 4{,}0989 & 0 & 0 \\ 0 & 2{,}3616 & 0 \\ 0 & 0 & 1{,}2737 \end{bmatrix}$$

$$V = \begin{bmatrix} -0{,}4945 & 0{,}6492 & -0{,}5780 \\ -0{,}6458 & -0{,}7194 & -0{,}2556 \\ -0{,}5817 & 0{,}2469 & 0{,}7750 \end{bmatrix} \qquad V^T = \begin{bmatrix} -0{,}4945 & -0{,}6458 & -0{,}5817 \\ 0{,}6492 & -0{,}7194 & 0{,}2469 \\ -0{,}5780 & -0{,}2556 & 0{,}7750 \end{bmatrix}$$

3.2.3 Dimensionsreduktion

Das Aufspalten von Matrix A ist ein Zwischenschritt der für die Dimensionsreduktion benötigt wird. Das Ziel ist es die größten (und wichtigsten) k Singulärwerte zur erhalten und die anderen zu eliminieren. Die Größe von k muss individuell für jede Dokumentensammlung ermittelt werden. Die Dimensionsreduktion wird über die Eigenwerte in der erzeugten Matrix S gesteuert. Das geschieht durch sukzessives Weglassen des jeweils kleinsten Eigenwertes bis zu einer unbestimmten Grenze k.

Für das Beispiel werden zwei Singulärwerte ($k = 2$) verwendet; auch Rang 2 Approximation genannt. Die Matrizen U und V werden auf ihre ersten zwei Spalten reduziert und Matrix S auf die ersten zwei Zeilen reduziert.

$$k = 2$$

$$A \sim A_k = \begin{bmatrix} -0{,}4201 & 0{,}0748 \\ -0{,}2995 & -0{,}2001 \\ -0{,}1206 & 0{,}2749 \\ -1{,}576 & -0{,}3046 \\ -0{,}1206 & 0{,}2749 \\ -0{,}2626 & 0{,}3794 \\ -0{,}4201 & 0{,}0748 \\ -0{,}4201 & 0{,}0748 \\ -0{,}2626 & 0{,}3794 \\ -0{,}3151 & -0{,}6093 \\ -0{,}2995 & -0{,}2001 \end{bmatrix} \qquad S \sim S_k = \begin{bmatrix} 4{,}0989 & 0 \\ 0 & 2{,}3616 \end{bmatrix}$$

$$V \sim V_k = \begin{bmatrix} -0{,}4945 & 0{,}6492 \\ -0{,}6458 & -0{,}7194 \\ -0{,}5817 & 0{,}2469 \end{bmatrix} \qquad V^T \sim {V^T}_k = \begin{bmatrix} -0{,}4945 & -0{,}6458 & -0{,}5817 \\ 0{,}6492 & -0{,}7194 & 0{,}2469 \end{bmatrix}$$

Der nächste Schritt ist die Abbildung der Suchanfrage (q) in den oben berechneten semantischen Raum [Berry95].

Die Matrix S ist symmetrisch entlang ihrer Diagonale und daher gilt: $S^T = S$

Zur Berechnung der Ähnlichkeit zwischen einem Dokument und der Suchanfrage muss die $Gleichung$ 1 umgeformt werden:

$$Gleichung\ 2\text{: } A^T = (USV^T)^T = VSU^T$$

$$Gleichung\ 3\text{: } A^T US^{-1} = VSU^T US^{-1}$$

$$Gleichung\ 4\text{: } V = A^T US^{-1}$$

Die Matrix A besteht aus $n > 1$ Dokumenten und die Matrix V bestehtaus n Zeilen. Die Zeilen von V stehen dabei jeweils für die Koordinaten des Dokumenten-Vektors. Für einen Dokumenten-Vektor d kann man die $Gleichung$ 4 neu anschreiben:

$$Gleichung\ 5\text{: } d = d^T US^{-1}$$

Im LSI wird eine Suchanfrage (q) gleich behandelt wie ein Dokument und daher kann q mit der gleichen Formel berechnet werden:

$$Gleichung\ 6\text{: } q = q^T U S^{-1}$$

Zur Verbesserung der Laufzeitkomplexität wird im letzten Schritt die Dimensionsreduktion in der Berechnung berücksichtigt:

$$Gleichung\ 7\text{: } d = d^T U_k {S_k}^{-1}$$

$$Gleichung\ 8\text{: } q = q^T U_k {S_k}^{-1}$$

$Gleichung\ 7$ und 8 beinhalten die neuen Vektorkoordinaten im dimensionsreduzierten Vektorraum. Die Ähnlichkeit zwischen den Vektoren d und q kann nun mittels Cosinus-Winkel Berechnung erfolgen:

3.2.4 Berechnung der Ähnlichkeit

$$Gleichung\ 9\text{: } sim(q,d) = sim(q^T U_k {S_k}^{-1}, d^T U_k {S_k}^{-1})$$

Für das Referenzbeispiel ergeben sich die Vektoren der drei Dokumente aus den Zeilen der Matrix V:

$$d_1 = [-0{,}4945, 0{,}6492]$$

$$d_2 = [-0{,}6458, -0{,}7194]$$

$$d_3 = [-0{,}5817, 0{,}2469]$$

Die Suchanfrage erfolgt durch Auflösen der $Gleichung\ 8$ ($k = 2$):

$$q = q^T U_k {S_k}^{-1}$$

$$q = [0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1] \begin{bmatrix} -0{,}4201 & 0{,}0748 \\ -0{,}2995 & -0{,}2001 \\ -0{,}1206 & 0{,}2749 \\ -1{,}576 & -0{,}3046 \\ -0{,}1206 & 0{,}2749 \\ -0{,}2626 & 0{,}3794 \\ -0{,}4201 & 0{,}0748 \\ -0{,}4201 & 0{,}0748 \\ -0{,}2626 & 0{,}3794 \\ -0{,}3151 & -0{,}6093 \\ -0{,}2995 & -0{,}2001 \end{bmatrix} \begin{bmatrix} \frac{1}{4{,}0989} & 0 \\ 0 & \frac{1}{2{,}3616} \end{bmatrix}$$

$$q = [-0{,}2140 \quad -0{,}1821]$$

Per Definition ist das Ergebnis einer Suchanfrage eine gereihte Liste der Dokumente in denen die Suchwörter gefunden wurden. Für die Berechnung gilt es nun die Vektoren der Dokumente (d_1, d_2, d_3) mit dem Vektor der Suchanfrage q zu vergleichen und die Dokumentvektoren in sortierter Reihenfolge zurückzugeben. Die folgenden Schritte sind dem Vector-Space-Model-Verfahren entnommen und ermöglichen es uns die Ähnlichkeit zwischen den Vektoren zu bestimmen. Für das Beispiel gilt die folgende Berechnung:

$$\text{sim}(q, d) = \cos(q, d) = \frac{q \cdot d}{|q||d|}$$

$$\text{sim}(q, d_1) = \frac{(-0{,}2140) * (-0{,}4945) + (-0{,}1821) * (0{,}6492)}{\sqrt{(-0{,}2140)^2 + (-0{,}1821)^2} * \sqrt{(-0{,}4945)^2 + (0{,}6492)^2}} = -0{,}0541$$

$$\text{sim}(q, d_2) = \frac{(-0{,}2140) * (-0{,}6458) + (-0{,}1821) * (-0{,}7194)}{\sqrt{(-0{,}2140)^2 + (-0{,}1821)^2} * \sqrt{(-0{,}6458)^2 + (-0{,}7194)^2}} = 0{,}9910$$

$$\text{sim}(q, d_3) = \frac{(-0{,}2140) * (-0{,}5817) + (-0{,}1821) * (0{,}2469)}{\sqrt{(-0{,}2140)^2 + (-0{,}1821)^2} * \sqrt{(-0{,}5817)^2 + (0{,}2469)^2}} = 0{,}44781$$

Damit ergibt sich für die Dokumente folgende absteigende Reihenfolge:

$$d_2 > d_3 > d_1$$

Anhand der nachfolgenden grafischen Darstellung wird diese Reihenfolge deutlich. Der Vektor q ist dem Vektor d_2 am Ähnlichsten. Das bedeutet der Winkel ist nur geringfügig unterschiedlich. Der Vektor d_1 hingegen hat eine andere Ausrichtung und weist den größten Unterschied zu d auf.

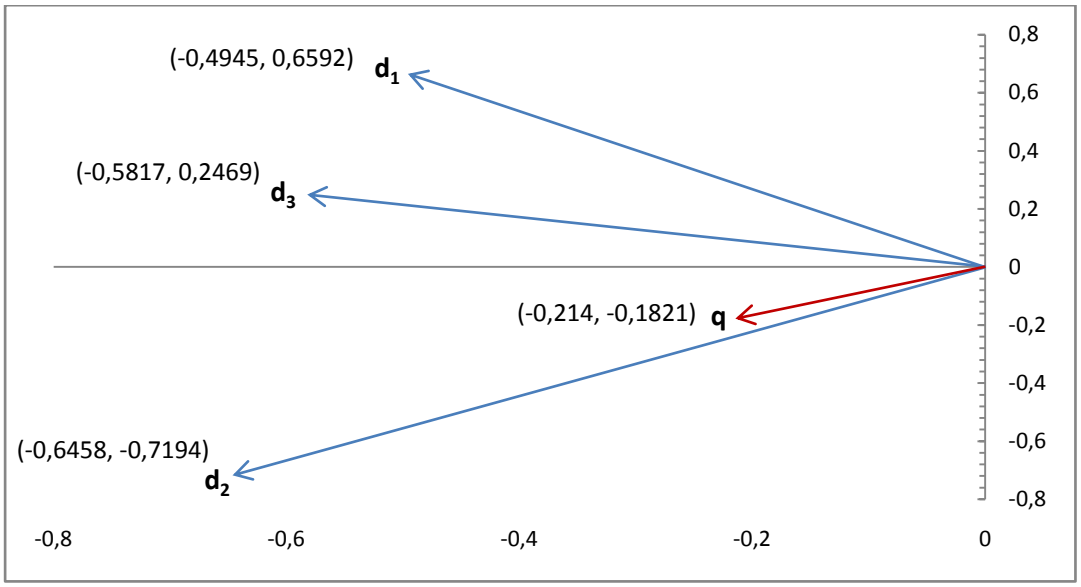


Abbildung 5: Grafische Repräsentation des Ergebnisses

Der Unterschied von LSI zum anfangs erwähnten VSM liegt in der Berechnung des Vektorraums. Bei VSM werden die Vektoren unabhängig voneinander berechnet und im Vektorraum dargestellt. Beim LSI Algorithmus haben diese Vektoren Abhängigkeiten untereinander. Durch die zusätzlichen Verarbeitungsschritte werden semantische Abhängigkeiten mit in den Vektorraum integriert und so aussagekräftigere Vektoren erzeugt.

4 Information Retrieval für Requirement Traceability: Existierende Forschungsarbeiten

Im Kontext dieser Arbeit sind zwei Themengebiete von Interesse: Requirements Traceability und der Einsatz von Information Retrieval Techniken für Requirements Traceability. In diesem Kapitel werden jeweils wichtige wissenschaftliche Arbeiten aus beiden Themengebieten analysiert und kurz auf die Ergebnisse eingegangen.

Eine der am meisten zitierten wissenschaftlichen Arbeiten auf dem Gebiet der Anforderungs-Traceability wurde von Gotel und Finkelstein [Gotel94]. Neben der Definition von Requirements Traceability (siehe Kapitel Requirements Traceability) definieren sie darin auch das erste Mal Pre und Post Requirements Specification Traceability. Ihre wissenschaftlichen Untersuchungen basieren auf einer Kombination von empirischen und klassischen Analysetätigkeiten. Zur Informationsbeschaffung verwendeten sie Fragebögen, Interviews, Focus Groups und Analyseworkshops zur Beobachtung von Teilnehmern.

In ihrer Analyse gingen die Autoren auf die vorhandene Werkzeugunterstützung für die Nachverfolgbarkeit von Anforderungen ein. Sie unterteilten die Werkzeuge in vier Kategorien:

- **Universelle Werkzeuge:** Werkzeuge die man manuell konfigurieren muss, aber dann für den individuellen Zweck geeignet sind. Beispiele dafür sind Office Anwendungen.
- **Spezialisierte Werkzeuge:** Werkzeuge die definierte Bereiche des Anforderungsmanagements erfüllen.
- **Werkzeugsammlungen (Workbenches):** Workbenches bezeichnet Anforderungsmanagement-Software die große Teile des Themengebiets unterstützt und teilweise automatisiert. Typischerweise mit zentraler Datenbank zur Speicherung der Daten.
- **Arbeitsumgebungen:** Arbeitsumgebungen unterscheiden sich von den Workbenches durch die Unterstützung des vollen Softwareentwicklungszyklus.

Das Fazit ihrer Untersuchung vor 16 Jahren war, dass kein Werkzeug zur damaligen Zeit eine adäquate Unterstützung lieferte. Seither haben sich viele große Hersteller des Themas

angenommen und zahlreiche weitere Tools implementiert. Dennoch befinden wir uns noch in einer konsolidierenden Marktsituation. Zum einen wurde Borland von Microfocus übernommen und zum anderen habt IBM den Hersteller vom Anforderungsmanagementtool DOORS (http://www.ibm.com/software/awdtools/doors/productline/) übernommen. Weiterhin sind die Hersteller bemüht große Arbeitsumgebungen mit vollem Funktionsumfang zu entwickeln.

In ihrer Arbeit wird die Frage gestellt, warum es noch immer ein Problem mit Anforderungs-Traceability gibt. Die Antwort sehen die Autoren in einer unzureichenden Definition von Requirements Traceability. Ihre Untersuchungen haben gezeigt, dass in den unterschiedlichen Situationen der Umfang und die Interpretation von Anforderungs-Traceability abwichen und somit keine einheitliche Basis gewährleistet wurde. Mit ihrer Veröffentlichung versuchen sie eine einheitliche Definition des Problems zu geben (siehe Kapitel Requirements Traceability). Auf diesem Gebiet wurde in den letzen Jahren sehr viel geforscht und die Definitionen des Problemfeldes haben sich weitestgehend stabilisiert. Ein weiteres Problem das Gotel und Finkelstein aufzeigen, ist die unterschiedliche Erwartungshaltung der Stakeholder gegenüber den Zielen von Anforderungs-Traceability. Zahlreiche Publikationen (z.B: [Pinh04], [Hayes03], [Rupp09], [Salt75]) geben unterschiedliche Gründe und Typen für Traceability an. Wichtig ist die Differenzierung zwischen den verschiedenen Typen von Requirements Traceability. Selbst moderne Anforderungswerkzeuge verfolgen unterschiedliche Ziele bei der Requirements Traceability Unterstützung.

Gotel und Finkelstein argumentieren, dass die Verbesserung von Pre RS Traceability wichtig sei. Post RS Traceability werde schon gut unterstützt und könne somit aus wissenschaftlicher Sicht vernachlässigt werden. Keine Aussage treffen sie über Intra-Requirements Traceability. Dieses Themengebiet ist die mittlere Schnittmenge von Pre und Post RS Traceability und muss somit beide Seiten betrachten. In ihrer Arbeit empfehlen sie künstliche Intelligenz Algorithmen oder Experten-System Technologie einzusetzen, um automatisch Traces zwischen den Informationen zu ziehen: *„Work related to requirements for end-use of pre-RS traceability mainly falls into the areas of information retrieval, artificial intelligence, and human computer interaction.”*

Eine wissenschaftliche Arbeit die sich mit diesem Thema auseinandersetzt ist der Artikel von Johan Natt och Dag von der Lund University und seinen Kollegen [Natt05].

Bei der Entwicklung von großen Softwareprodukten gibt es eine enorme Menge von Anforderungen. Natt teilt diese in zwei Kategorien: Kundenwünsche und Produktanforderungen [Natt05]. Die Herausforderung im Bereich der Traceability ist es, diese Anforderungskategorien zu verknüpfen und auch wenn sich Anforderungen verändern diese Beziehung aktuell zu haltenb. Ziel dieser Pre RS Traceability ist die Priorisierung von Anforderungen und eine korrekte Release-Planung. Natt och Dag kommt zum gleichen Schluss wie Gotel und Finkelstein und sieht die Lösung des Problems in der Automatisierung durch Information Retrieval Algorithmen. Bei Projekten mit mehreren tausenden Anforderungen und ähnlich vielen Traces wäre es nicht mehr finanziell leistbar diese manuell zu erstellen und zu aktualisieren. Die Verbindung zwischen Kundenwunsch und Produktanforderung liegt im gleichen Inhalt und beide referenzieren die gleiche Funktionalität des Systems. Trotz ähnlicher Semantik können sich beide Anforderungstypen durch ihren Prosastil und durch die verwendeten Wörter unterscheiden. Natürlich gibt es keine Algorithmen die den Zusammenhang eindeutig identifizieren können. Durch den Einsatz von modernen Natural Language Processing (NLP) Techniken kann man jedoch semantische Ähnlichkeiten ermitteln. Diese Ähnlichkeitsanalyse erfolgt durch lexikalische Analysen der Anforderungen. Eine globale Annahme für diese These ist ein gemeinsam vorliegender Sprachraum, sodass gleiche Funktionalitäten auch gleich benannt werden.

Die Autoren der Arbeit haben sich für die Verwendung des Vector Space Models zur Repräsentation der Anforderungen entschieden. Dabei werden die Anforderungen auf ihre wichtigsten Wörter reduziert und dieses in einem gemeinsamen Vektorraum aufgespannt. Je geringer der Winkel zweier Vektoren ist, desto größer die Ähnlichkeit der Anforderungen. Zur Berechnung des Winkels wurde die Cosinus-Funktion verwendet.

Der Ablauf der Berechnung erfolgt nach folgendem Schema. Zuerst werden die Anforderungen für die Verarbeitung aufbereitet. Dabei muss jede Anforderung folgende Stationen durchlaufen:

- **Flattened:**
  Zuerst werden alle Anforderungsinformationen vereinheitlicht und zusammengeführt, sodass jede Anforderung als Text-Dokumente vorliegt.
  Unwichtige Information fällt bei dieser Station raus und wird nicht berücksichtigt.

- **Tokenized:**
  Als nächstes wird jede Anforderung in eine Liste von Wörtern (Token) umgewandelt. Dabei werden alle Satzzeichen gelöscht.
- **Stemming:**
  Die einzelnen Wörter werden auf ihre sprachliche Grundform zurückgeführt. Durch diese Vorgehensweise werden Wortendungen normalisiert, Mehrzahl in Einzahl umgewandelt und andere Lexikalische Ausprägungen auf die Grundform transformiert.
- **Stop Word Removal:**
  Im letzten Schritt werden häufig genutzte Wörter entfernt. Diese semantisch unbedeutende „Stop Words“ sind für die Analyse überflüssig und können somit ignoriert werden.

Die folgende Grafik zeigt die verschiedenen Stationen anhand einer Beispielanforderung:

Stage 1: Flattened	Stage 2: Tokenized	Stage 3: Stemmed	Stage 4: Stop words removed
Pricing and Containerization Specifically what I am interested in is containerization and pricing. For a prospect I am working with (pretty much a distributor of electonic components) I need pricng by type of package by cusotmer type(wholesale or retail).	*pricing and containerization specifically what i am interested in is containerization and pricing for a prospect i am working with pretty much a distributor of electonic components i need pricng by type of package by cusotmer type wholesale or retail*	*price and containerization specifically what i be interest in be containerization and price for a prospect i be work with pretty much a distributor of electonic component i need pricng by type of package by cusotmer type wholesale or retail*	*price containerization specifically containerization price prospect pretty distributor electonic component pricng type package cusotmer type wholesale retail*

Tabelle 2: Phasen der Aufbereitung eines Anforderungstexts [Natt05]

Nach der Aufteilung der Anforderung in Token erfolgt die Weiterverarbeitung nach dem Vector-Space-Model. Jeder Anforderungstext wird als Vektor in einem n-Dimensionalen Vektorraum dargestellt. Der Winkel zwischen den Vektoren gibt die Ähnlichkeit der beiden Anforderungen an. Der wichtigste Schritt ist einen Schwellwert zu finden, ab wann eine Ähnlichkeit signifikant wird. Dies ist wichtig, da das Ziel die automatische Ermittlung von Abhängigkeiten zwischen den Anforderungen ist. Übersteigt der Winkelwert diesen Schwellenwert gibt es eine Abhängigkeit zwischen den Anforderungen.

Die Autoren evaluierten ihr vorgeschlagenes Verfahren anhand eines realen Sets von Anforderungen. Es handelte sich dabei um ein Software-Framework mit 250 Modulen und

10.000 Komponenten. Die Anforderungen wurden unterteilt in Kundenwunschanforderungen (Market Requirements) und Produktanforderungen (Business Requirements). Über die Jahre wuchs die Anzahl der Anforderungen auf 2.400 Market Requirements (MR) und 1.100 Business Requirements (BR). Ziel war es die bisher manuell geprüften Anforderungsabhängigkeiten zwischen MR und BR automatisch zu berechnen. Zur Überprüfung ihrer Ergebnisse wurde ein Testset erstellt indem die Verknüpfungen manuell durch einen Experten erstellt wurden. Der Algorithmus erreichte einen Recall von 51 Prozent. Das bedeutete für die Praxis eine gewaltige Zeitersparnis, da das Tool ca. die Hälfte der Beziehungen schon richtig vorschlägt. Als zukünftige Forschungsarbeit beschreiben sie Tätigkeiten zur Verbesserung des Algorithmus und der verbesserten Verwertung von semantischen Relationen.

Ein Verbesserung der Ergebnisse verspricht die Verwendung des Latent Semantic Indexing Algorithmus [Marc03]. Jane Huffman Hayes und ihren Kollegen von der University of Kentucky stellten sich die zentrale Frage, ob das Anforderungs-Traceability Problem mit Hilfe von Information Retrieval (IR) Algorithmen zu lösen ist und welcher Algorithmus der Beste ist [Hayes03].

Einer der aufwändigsten und ressourcenintensivsten Aktivitäten beim Erkennen von Abhängigkeiten zwischen Anforderungen ist das manuelle Ermitteln der Abhängigkeitskandidaten. Das Requirements Traceability Tool sollte in der Lage sein diese Abhängigkeiten zu erkennen und dem Benutzer vorzuschlagen. Ziel von Hayes Studie war es die Qualität der vorgeschlagenen Abhängigkeiten zu optimieren und eine Lösung zu implementieren die auch aus Laufzeitarchitektonischer Sicht praxistauglich ist. Das Team teilte seine Aufgaben in folgende sechs Aktivitäten auf:

1. Analyse des Anforderungs-Tracing als IR-Problem; Auswahl eines geeigneten IR-Algorithmus,
2. Aufbereitung des Anforderungstextes,
3. Analyse des Algorithmus-Ergebnisses,
4. Strategien zur Verbesserung des Ergebnisses und
5. Vergleich des Algorithmus mit einem menschlichen Analytiker.

Ein Großteil der Tätigkeiten zur Auffindung von Abhängigkeiten zwischen Anforderungen beschäftigt sich mit der manuellen oder automatischen Suche nach Schlüsselwörtern. Hayes argumentiert, dass für diese Tätigkeiten Information Retrieval Algorithmen optimiert seien und deshalb eine logische Automatisierungsmöglichkeit darstellen. Für ihre Untersuchung verwertet sie die Abhängigkeiten von Design-Dokumenten zu Anforderungen des Softwareprodukts. Wie schon Natt och Dag sieht auch Hayes den Schlüssel zur Automatisierung von Requirements Traceabilty in der Ähnlichkeit von Anforderungen zu Design-Dokumenten. Eine Abhängigkeit zwischen Anforderungen und Design-Dokument wird definiert sobald dieser Ähnlichkeitswert einen manuell festgelegten Grenzwert übersteigt. Obwohl Information Retrieval Algorithmen sehr gut für das definierte Problem funktionieren, so gibt es auch zwei Probleme:

- Domain Größe: IR-Algorithmen arbeiten auf sehr großen Datenmengen und je mehr Daten sie erhalten desto besser werden die Ergebnisse. Die Menge an Anforderungen ist für IR-Algorithmen meist vergleichsweise gering und auch die Länge einer Anforderung übersteigt meist keine zwei bis drei Sätze. Diesem Problem muss man mit einer guten Auswahl der IR-Algorithmen begegnen. Durch die kleine Datenmenge ist es möglich einen komplexeren Algorithmus zu wählen und dadurch eine Verbesserung des Ergebnisses zu bewirken.
- Unabhängigkeit der Anfrage: Viele der besten IR-Algorithmen wurden für Suchanfragen im Internet entwickelt. Eine der grundlegenden Annahmen ist, dass jede Anfrage unabhängig voneinander ist. Bei einem Set von Anforderungen trifft diese Annahme nicht zu, da manche Anforderungen voneinander abhängen und semantisch zusammen gehörig sind. Hayes argumentiert an dieser Stelle, dass speziell zwischen Grob- und Detailanforderungen Zusammenhänge bestehen, die sie mittels einer Matrix wieder aus dem Ergebnis rechnen könnte. Für diese Arbeit wurde ein anderer Ansatz gewählt. Alle Anforderungen sind per Definition voneinander unabhängig. Die strukturellen Abhängigkeiten wurden per Testversuch gesondert untersucht (siehe Kapitel 8).

Die Aufbereitung der Anforderungen für den Algorithmus erfolgte nach dem gleichen Muster wie bei Natt och Dag beschrieben. Anforderungen wurden nach Tokens aufgeteilt, Stoppwörter wurden entfernt, die Token wurden auf ihre Grundform gebracht und als

Vektoren gespeichert. Als IR-Algorithmus kam der Vanilla Vector Retrieval Algorithmus zum Einsatz. Hierbei handelt es sich um einen klassischen Vector-Space-Model Algorithmus. Mit zwei Erweiterungen versuchten die Autoren den Algorithmus effizienter zu gestalten:

- Schlüsselwortsequenzen: Zusätzlich zum „normalen“ Vektormodell wurden technische Schlüsselwörter definiert. Diese wurden dem Algorithmus zusätzlich vor der Berechnung zur Verfügung gestellt. Somit konnten diese Schlüsselwörter ihren semantischen Zusammenhang behalten und wurden als ein Token verarbeitet. Ein Beispiel ist das Wort „ecs production environment“. Der Aufbereitungsalgorithmus hätte diese Wörter in drei Token „ecs“, „production“ und „environment“ aufgeteilt und somit wäre der semantische Zusammenhang verloren gegangen.
- Wörterbuch: Bei der zweiten Erweiterung handelt es sich um ein eigens aufgebautes Wörterbuch. Jeweils zwei Tokens werden mit einem Ähnlichkeitswert verbunden. Dabei wird folgendes Tripel gespeichert: (k_i, k_j, a_{ij}). k_i und k_j sind dabei die Wörter und a_{ij} der ermittelte Ähnlichkeitswert. Ein Beispiel für solch ein Tripel: („corrupted data“, „missing packets“, 1,0). Bei der Emittlung des Cosinus-Winkels zwischen den Vektoren fließt diese Ähnlichkeitskonstante ein. Angenommen das Wörterbuch sei T, die Anforderung $r = (r_1, \dots, r_{N)}$ und das Design-Dokument d= $(w_1, \dots, w_{N)}$ dann berechnet sich der Abhängigkeitswert mit folgender Formel

$$sim_T(d,r) = \frac{\sum_{i=1}^{N} w_i * r_i + \sum_{(k_i,k_j,a_{ij})\epsilon T} a_{ij} * (w_i * r_j + w_j * r_i)}{\sqrt{\sum_{i=1}^{N} {w_i}^2 * \sum_{i=1}^{N} {r_i}^2}}$$

Die Evaluierung des Algorithmus und der Erweiterungen zeigte, dass die Ergebnisse ohne Erweiterungen nicht weiter verwertbar gewesen wären. Alle drei Komponenten in Kombination erreichten einen Recall-Wert von 85% bei einer Precision von 40%. Die Vergleiche der Algorithmen wurden auf einem Testset von Anforderungen und Design-Dokumenten vorgenommen. Ein erfahrener Analytiker erstellte manuell die Abhängigkeitsverbindungen zwischen den Dokumenten. Das Testset hatte eine Größe von 19 High-Level Anforderungen zu 50 Low-Level Anforderungen (Design-Dokumente). Trotz des kleinen Testsets benötigte der Analytiker über eine Stunde um die Abhängigkeiten festzustellen. Die Laufzeit des Algorithmus betrug wenige Sekunden. Hayes folgert aus den Untersuchungen, dass trotz der guten Ergebnisse ein Anforderungsexperte die finale

Abhängigkeits-Linkliste kontrollieren und bestätigen muss. Kritisch muss man das kleine Testset von Anforderungen sehen. In der Arbeit wurden keine Aussagen über die Größe des Wörterbuchs oder der technischen Schlüsselwörter gegeben. Bei großen Mengen von Anforderungen könnte sich die Erstellung dieser Listen als sehr aufwändig erweisen.

Die aktuelle Forschung beschäftigt sich mit Methodiken zur weiteren Verbesserung der IR-Algorithmen.

In seiner Arbeit kombiniert Asuncion einen Information Retrieval Algorithmus mir der Machine-Learning Technik "Topic Modeling" [Asun10]. Dabei werden die Anforderungen als eine Mischung von verschiedenen zugrunde liegenden Themen ("Topics") betrachtet. Ziel der Bemühungen ist es den Anforderungsanalytiker durch diese zusammengefassten Themengebiete bei der Erstellung der Beziehungen zwischen den Anforderungen zu unterstützen. Auch in anderen Artikel [Duan07], [Hayes08] wurde versucht mittels Clustering Methoden die Genauigkeit der IR-Algorithmen zu verbessern und zusammengehörige Anforderungen zu gruppieren.

Doch nicht nur die Algorithmen können verbessert werden, sondern auch die Verarbeitungsprozesse. In seinem Paper versucht Lucia festzustellen, wie sich der Prozess auf die Tracing Leistungen von Anforderungsanalytikern auswirkt [Lucia08]. Es wird zwischen der "One-Shot" Methode und einem iterativen Vorgehen unterschieden. Bei der "One-Shot" Methode stellt das IR-basierte Traceability Tool dem Anwender die ab dem Grenzwert gekürzte Liste von vermuteten Traces zur Verfügung. Im iterativen Verfahren kann der Anforderungsanalytiker diesen Grenzwert inkrementell nachjustieren. Das Ergebnis der Arbeit zeigt, dass der Prozess keinen Einfluss auf die aufzuwendende Zeit hat. Die gefundenen Beziehungen zwischen den Anforderungen wurden durch den inkrementellen Prozess verbessert.

In dieser Arbeit wurde der Grenzwert als fixiert angegeben und die „One-Shot“ Methode implementiert. Die Auswertung der Ergebnisse durch einen Anforderungsanalytiker ist nicht Teil des Umfangs.

5 Automatisierung der Intra-Requirements Traceability

Beispiele wie das zu Anfangs erwähnte Taurus Projekt belegen die Wichtigkeit der Anforderungs-Traceability für den Erfolg eines Projektes. Mit wachsender Größe des Softwareprodukts ergeben sich zusätzliche Herausforderungen. Mit der Größe nimmt die Anzahl der zu erfassenden Anforderungen zu. Die erhobenen Anforderungen haben aber weiterhin Abhängigkeiten untereinander und zu den Zielen des Systems. Bei steigender Anzahl der Anforderungen werden mehr Anforderungsexperten parallel an der Anforderungserhebung arbeiten. Im Domain-Driven-Design Ansatz geht man von einer fachlich getriebenen Systemarchitektur aus [Evan04]. Für die Anforderungserhebung bedeutet das, dass man die Themengebiete in fachliche Domains einteilen kann. Die Anforderungsexperten werden diesen Domains zugeteilt und erheben in ihren Domains die entsprechenden Anforderungen. Dieser Umstand skaliert mit der wachsenden Größe des Softwaresystems und reduziert die Gesamtkomplexität. Probleme können an den Schnittstellen von Domains entstehen. Eine weitere Herausforderung sind Domain übergreifende Anforderungen. Diese können starke Abhängigkeiten zu Anforderungen aus verschiedenen Domains haben.

Die starke Trennung der Aufgaben kann die beteiligten Personen „blind“ für übergreifende Abhängigkeiten machen. Bei einer unüberschaubaren Anforderungsanzahl ist es für eine einzelne Person nicht mehr möglich alle Abhängigkeiten zu erfassen. Der dadurch steigende Abstimmungsaufwand für diese Aufgabe muss vor dem Projektmanagement gerechtfertigt werden. Kritisch sind Anforderungsabhängigkeiten bei bestehenden Projekten. In der Praxis kommt es vor, dass der Hersteller eines Softwaresystems nicht der Betreiber ist. Laufende Änderungen oder Verbesserungen am System werden somit von unterschiedlichen Teams gemacht. Im Idealfall sind die Anforderungen gut dokumentiert und werden mit dem System übergeben. Trotzdem ist es wichtig, dass die Abhängigkeiten zwischen den Anforderungen spezifiziert sind. Bei Änderungen erkennt der Anforderungsexperte Abhängigkeiten zu anderen Systemkomponenten und kann diese auflösen.

Für die oben beschriebenen Probleme gibt es nur wenige Werkzeugunterstützung am Markt. Moderne Anforderungsmanagementsysteme (z.B.: Borland CaliberRM (www.borland.com/caliber), DOORS (http://www.ibm.com/software/awdtools/doors/productline/), IBM Rational RequisitePro (http://www.ibm.com/software/awdtools/reqpro/) erlauben zwar das Erstellen

von Abhängigkeiten zwischen Anforderungen, das automatische Erkennen der Abhängigkeiten wird allerdings nicht unterstützt.

Die Arbeit setzt an dieser Stelle an und beschreibt eine Lösung zur automatischen Erkennung von Intra-Requirements Abhängigkeiten. Die Anforderungen werden in natürlicher Sprache verfasst und in einem Anforderungsmanagementtool verwaltet. Das Zielsystem ist mit dem Anforderungsmanagementtool verbunden und greift auf die darin gespeicherten Anforderungen zu. Ein Analysealgorithmus analysiert alle Anforderungen und ermittelt eventuelle Abhängigkeiten. Stellt der Algorithmus eine Abhängigkeit zwischen zwei Anforderungen fest, wird diese Abhängigkeit im Anforderungsmanagementtool gespeichert. Der Anforderungsexperte erhält die Ergebnisse in einer Requirements Traceability Matrix oder als grafische Darstellung zur Verfügung gestellt. Diese Darstellung ist wichtig, da die Ergebnisse immer von einem Anforderungsexperten kontrolliert werden müssen.

5.1 Herausforderungen

Die größte Herausforderung bei der Automatisierung stellt die menschliche Sprache dar. Alle Anforderungen sind in natürlicher Sprache verfasst und damit nicht eindeutig und formalisierbar. Die hier verwendeten Anforderungen folgen den syntaktischen Regeln der Schablonengrammatik. Diese wurde in Kapitel 2.3 beschrieben und hat das Ziel alle Anforderungen im selben Stil zu dokumentieren und Mehrdeutigkeiten zu minimieren. Wichtig für diesen Stil der Anforderungserhebung ist ein gemeinsames Sprachuniversum, bzw. Glossar. Das Glossar definiert die grundlegenden Begriffe und schafft so eine gemeinsame Basis für alle Anforderungsexperten. Der Umstand, dass möglichst viele Begriffe normiert sind und an denselben Stellen gleich verwendet werden, kommt den Information Retrieval Algorithmen prinzipiell entgegen.

Eine weitere Herausforderung ist die Verwaltung der Anforderungen mit allen Anforderungs-Abhängigkeiten. Wie im Eingangskapitel erwähnt, gibt es verschiedene Arten von Abhängigkeiten zwischen Anforderungen. Strukturelle Abhängigkeiten lassen sich sehr gut durch eine baumartige Strukturierung der Anforderungen dokumentieren. Zwischen den zwei Typen beschränkende Abhängigkeiten und kosten- und wertrelevante Abhängigkeiten werden keine Unterscheidungen gemacht.

Wichtig für eine erfolgreiche Analyse ist die Aufbereitung der Anforderungen. Eine Anforderung aus unserem Testdatenbestand besteht aus einem sprechenden Titel, der Anforderungsbeschreibung und Attributen. Die Attribute werden großteils für das Anforderungsmanagement und die Release-Planung benötigt. Ebenfalls zu den Attributen gehören Versionsinformationen und verantwortliche Personen. Auf die Abhängigkeiten zwischen den Anforderungen haben die Attribute keine Auswirkungen. Als verwertbare Elemente bleiben somit der Anforderungstitel und die Beschreibung über. Beide Elemente werden zusammengefügt und fließen als ein Anforderungstext in die Verarbeitungslogik mit ein.

Die Arbeit geht davon aus, dass Information Retrieval Algorithmen für das Erkennen von Abhängigkeiten von Anforderungen prinzipiell geeignet sind. Dennoch gibt es einige Herausforderungen im Umgang mit IR-Algorithmen zu bewältigen. Zwei bereits in [Heyes03] beschriebene Problemfelder sind die Domain Größe und die Unabhängigkeit der Anfrage. Für IR-Algorithmen ist ein großer Datensatz wünschenswert, da dann statistische Fehler minimiert werden. Eine vorgeschlagene Lösung ist die Wahl eines komplexeren Algorithmus zur Berechnung. Bei der Latent-Semantic-Indexing Berechnung konnte durch diesen Umstand ein höher dimensionierter Vektorraum gewählt werden. Dennoch zeigten die Ergebnisse dieser Arbeit, dass bei größeren Anforderungsmengen die Laufzeit zu einem wichtigen Faktor werden kann. Auch die zu verarbeitende Menge an Wörter war ausreichend für den Information Retrieval Algorithmus.

5.2 Vorgehensweise

Die Anforderungen werden im Anforderungsmanagementtool Borland CaliberRM erfasst [Caliber]. Das Haupteinsatzgebiet dieses Werkzeug ist das Verwalten von Anforderungen und damit erfüllt es die gewünschten Vorraussetzungen. Die Anforderungen werden auf einem zentralen Server gespeichert und können per Client aufgerufen und bearbeitet werden. Da die Client-Software von Borland kostenpflichtig ist und der Source Code nicht verändert werden kann, wurde ein einfacher Client neu programmiert.

Folgende Anforderungen werden an die Client-Software gestellt:

- Die Client-Software muss die Anforderungen vom Server laden und anzeigen.

- Ein Algorithmus muss die Anforderungen analysieren und Abhängigkeiten zwischen den Anforderungen markieren.
- Das Werkzeug muss die vorgeschlagenen Abhängigkeiten mit einer Testdatenbank vergleichen und Kennzahlen des verwendeten Algorithmus berechnen.
- Die Client-Software muss sich mit verschiedenen Projekten am Server verbinden und diese Verbindung speichern.

Das neu implementierte Werkzeug erfüllt diese Anforderungen und ermöglicht dem Benutzer die Analyse der gespeicherten Software-Anforderungen. Die Anforderungen werden am Server in Projekten zusammengefasst. Die Client-Software analysiert immer nur die Anforderungen eines einzelnen Projektes. Bei den Anforderungen handelt es sich um natürlichsprachige Texte und daher ist folgende Vorverarbeitung für den Information Retrieval Algorithmus notwendig.

5.2.1 Aufbereitung der Anforderung

Die Aufbereitung der Anforderungen gliedert sich in folgende Phasen:

1. Laden der Anforderungen

Bei erstmaligem Verbinden mit dem Server muss auf dem Client die Verbindung konfiguriert werden. Danach ruft die Software alle Anforderungen eines Projekts vom Server ab. Von den Anforderungen werden der Anforderungstitel und die Anforderungsbeschreibung für die weitere Verarbeitung verwendet. Alle unwichtigen Attribute einer Anforderung werden für die weitere Bearbeitung ignoriert, da sie keine zusätzlich verwertbare Information beinhalten. Am Server wird die Anforderungsbeschreibung im HTML-Format gespeichert. Im ersten Verarbeitungsschritt wird der Inhalt extrahiert und ins Textformat umgewandelt. Bilder, Skizzen und Diagramme werden in diesem Schritt gelöscht, da der Algorithmus nur reinen Text verarbeiten kann. Anforderungstitel und Anforderungsbeschreibung werden als ein gemeinsamer Text abgespeichert. Dieser Vorgang wird in der Literatur als „Flattening“ bezeichnet [Natt05].

2. Aufspaltung in Token

Vor der weiteren Verarbeitung wird der Anforderungstext in einzelne Wörter aufgespaltet. In der Literatur wird häufig der Begriff Token für Wort verwendet und der Prozessschritt als „Tokenization“ bezeichnet [Natt05]. Bei der Aufspaltung der Sätze werden Satzzeichen entfernt. Beispiele für Satzzeichen sind der Punkt (.), das Komma (,) der Strichpunkt (;) und jegliche Formen von Klammern (). Das Endergebnis ist eine Wortliste für jede Anforderung.

3. Rückführung auf den Wortstamm

Die Liste mit Wörtern beinhaltet die unveränderten Wörter des Satzes. Für die Analyse mit einem Information Retrieval Algorithmus ist es wichtig die Vergleichbarkeit der Wörter zu gewährleisten. Dafür müssen die Wörter auf ihren Wortstamm zurückgeführt werden. „Stemming“ wird auch als Grundformenreduktion oder Normalformenreduktion bezeichnet und beschreibt das Verfahren die verschiedenen morphologischen Varianten eines Wortes auf den gemeinsamen Wortstamm zurückzuführen. Eines der bekanntesten Stemming-Verfahren ist der Algorithmus von Porter [Savo06]. Martin Porter implementiert seinen Algorithmus für sämtliche Sprachen und veröffentlichte diese im "Snowball"-Projekt [Porter80]. Für das Analyse-Werkzeug verwende ich die Java-Version des Snowball Projektes. Der Algorithmus ist nicht fehlerfrei, dennoch werden die Wörter auf eine gemeinsame Basis gebracht. Für den Zweck der Vergleichbarkeit ist dieser Algorithmus ausreichend.

4. Entfernung der Stoppwörter

In diesem Prozessschritt wird die vollständige Liste aller Wörter einer Anforderung auf die notwendigen Wörter reduziert. Dies erfolgt durch die Identifizierung von Stoppwörtern in den Sätzen. Stoppwörter werden meist durch grammatikalische oder syntaktische Regeln in die Sätze eingefügt und ermöglichen keine Rückschlüsse auf den Inhalt des Satzes. Für den Information Retrieval Algorithmus werden diese Stoppwörter aus dem Text entfernt. Stoppwörter treten in jedem Satz sehr zahlreich auf und werden in jedem Satz wiederholt. Aus diesem Grund würden sie bei der Analyse der Anforderungen keinen Mehrwert bieten und einen hohen Aufwand verursachen.

Für die Deutsche Sprache übliche Stoppwörter sind bestimmte Artikel ('der', 'die', 'das'), unbestimmte Artikel ('einer', 'eine', 'ein'), häufig gebrauchte Präpositionen (z. B. 'an', 'in', 'von') und Konjunktionen (z. B. 'und', 'oder', 'doch') [Goetze08].

5. Substitution der Wörter

Eine natürliche Sprache besitzt für die meisten Dinge mehrere Wörter um dieselbe Sache zu beschreiben. Teilweise mit geringfügigen Unterschieden aber dennoch demselben semantischen Inhalt. Ein Beispiel dafür ist „Computer“, „Rechner“ und „Rechenmaschine“. Eine Annahme dieser Arbeit ist, dass verschiedene Personen die Anforderungen erhoben haben. Es wurde Richtlinien für die Verwendung von Ausdrücken ausgegeben (Glossar) und man geht von einem gemeinsamen Sprachuniversum aus. Dennoch kommt es in der Praxis vor, dass Wörter unterschiedlich gebraucht werden oder verschiedene Wörter für dieselbe Sache verwendet werden. Aus diesem Grund wird als letzter Verarbeitungsschritt die Wörtersubstitution eingeführt. Hierbei werden die Wörter in ihrer Grundform mit einer Liste verglichen. Kommt ein Wort in dieser Substitutionsliste vor, wird sie durch das Pendant ersetzt. Damit können alle „Rechenmaschinen“ durch „Computer“ ersetzt werden. Dieses einfache Verfahren erzielt nochmals eine Verbesserung des nach gelagerten Information Retrieval Algorithmus.

Durch Rechtschreibfehler kann es möglich sein, dass Wörter unterschiedlich aufgespaltet werden und somit das Ergebnis verzerren. Bei häufigen Rechtschreibfehlern hat es sich etabliert, diese über die Substitutionsliste korrigieren zu lassen.

5.2.2 Analyse der Abhängigkeiten zwischen den Anforderungen

Nachdem die Anforderungstexte die Vorverarbeitungsschritte durchlaufen haben, liegen die Anforderungen als Liste von Tokens vor. Zur weiteren Verarbeitung baut das System ein „Wörterbuch“ von allen Anforderungen auf. Hierbei werden die Wörter des gesamten Projekts gezählt und auch die Anforderungslisten mit zusätzlicher Information über die Token angereichert.

In einem weiteren Schritt werden die Anforderungen eines Referenzprojekts geladen. Bei diesem Projekt wurden die Abhängigkeiten zwischen den Anforderungen von einem Anforderungsexperten erstellt. Diese Information wird mit den Anforderungen gespeichert und zu einem späteren Zeitpunkt mit den berechneten Ergebnissen verglichen.

Nach diesem Schritt beginnt die individuelle Vorbereitung der Anforderungen für die verschiedenen Information Retrieval Algorithmen. Es wurden zwei IR-Algorithmen implementiert:

- ***Vector-Space-Model***

  Der Vector-Space-Model Algorithmus benötigt keine weitere Vorbereitung der Anforderungen. Die Anforderungen werden in einer Liste gespeichert. Diese Liste beinhaltet für jede Anforderung eine weitere Liste mit den entsprechenden Token.

  Für die Anwendung des Vector-Space-Model auf das beschriebene Problem muss der Algorithmus geringfügig modifiziert werden. Der ursprüngliche Algorithmus ist für die Anwendung auf Suchanfragen ausgelegt. Hierbei vergleicht man einen Suchstring mit sämtlichen Dokumenten und erhält Wahrscheinlichkeitswerte, nach denen man die einzelnen Dokumente reihen kann. Für das automatische Erkennen von Abhängigkeiten zwischen Anforderungen muss man jede Anforderung mit jeder vergleichen. Der Algorithmus besteht aus zwei Schleifen, die jeweils durch die Anforderungsliste iterieren. Dabei wird jeder Anforderungsvektor mit dem anderen verglichen und der Ähnlichkeitswert berechnet. Der implementierte Algorithmus ist im Kapitel 3 Information Retrieval Algorithmen beschrieben.

  Nach dem Berechnen eines Ähnlichkeitsfaktors muss dieser noch mit einem festgelegten Grenzwert verglichen werden (siehe Kapitel 5.2.3). Wenn der Ähnlichkeitsfaktor den Grenzwert übersteigt, dann handelt es sich um eine Abhängigkeit zwischen den beiden Anforderungen. Diese festgestellte Abhängigkeit wird mit dem Testdatensatz verglichen. Bei positiver Übereinstimmung gilt die Abhängigkeit zwischen diesen Anforderungen als korrekt ermittelt. Es werden zwei Werte gespeichert: die Summe aller vorgeschlagenen Abhängigkeiten und die korrekt erkannten Abhängigkeiten. Für Analysezwecke kann man die Ähnlichkeitswerte in eine Textdatei schreiben lassen.

In einem zusätzlichen Schritt wurde versucht das Vector-Space-Model mit einem Bayes Klassifizierung Algorithmus zu kombinieren. Während der Implementierung stellte sich heraus, dass dieser Ansatz keine akzeptable Laufzeit erreichen kann und einige Annahmen stellten sich als grundlegend falsch heraus. Aus diesem Grund wurde die Implementierung gestoppt und der Latent-Semantic-Indexing Algorithmus implementiert.

- ***Latent-Semantic-Indexing***

  Der Latent-Semantic-Indexing Algorithmus wurde ähnlich dem Vector-Space-Model Algorithmus für das Erkennen von Abhängigkeiten zwischen Anforderungen angepasst. Im Kapitel 3 Information Retrieval Algorithmen wurde der Algorithmus mit einer Suchanfrage q beschrieben. In der tatsächlichen Implementierung werden die einzelnen Anforderungen miteinander verglichen. Somit nimmt jede Anforderung einmal die Position von q ein und wird mit allen anderen Anforderungen verglichen.

  Bei Latent-Semantic-Indexing ist es wichtig die Matrix A zu erstellen. Dies geschieht mit dem Aufbau des „Wörterbuchs“. Als nächster Schritt wird die Singulärwertzerlegung abgearbeitet. Dabei wird die Matrix A in die drei Komponenten U, S und V^T zerlegt:

  $$A = U * S * V^T$$

  Für die komplexen Matrizenberechnungen wurde das Matrizen-Framework „Jama“ eingesetzt [Jama05]. Dieses Framework hat bereits Java-Funktionen für die Singulärwertzerlegung (Singular Value Decomposition) implementiert, die in der Arbeit verwendet wurden. Die Funktionen für die Dimensionsreduktion mussten neu implementiert werden. Die berechneten Matrizen des LSI werden mit dem „Wörterbuch“ abgespeichert, sodass die Werte für die weitere Berechnung vorliegen.

  Der LSI-Algorithmus nimmt jede Anforderung und baut daraus einen Vektor nach der folgenden Formel:

  $$d = d^T U_k {S_k}^{-1}$$

Der Vorteil dieses Algorithmus ist, dass diese Vektoren nicht nur die Werte der einzelnen Token beinhalten. Dank des Latent-Semantic-Indexing beinhalten sie auch Teile der semantischen Information aus den Anforderungen. Dies liegt in der Annahme begründet, dass einzelne Anforderungen sich mit der gleichen Thematik beschäftigen. Die Wörter einer Anforderung gehören auch inhaltlich zusammen und diese Information wird bei dieser Berechnungsart berücksichtigt.

$$sim(d_1, d_2) = sim({d_1}^T U_k {S_k}^{-1}, {d_2}^T U_k {S_k}^{-1})$$

Mit der Cosinus-Winkel Berechnung wird der Ähnlichkeitswert zwischen den Anforderungsvektoren ermittelt. Dieses Vorgehen wurde im Kapitel 3 „Information Retrieval Algorithmen“ beschrieben.

Auch bei diesem Algorithmus wird der ermittelte Ähnlichkeitswert mit einem festgelegten Grenzwert verglichen. Sobald der Ähnlichkeitswert den Grenzwert übersteigt, wird von einer Abhängigkeit zwischen den Anforderungen ausgegangen. Eine ermittelte Abhängigkeit wird im Anforderungsmangement-Tool CaliberRM eingetragen. Der Anforderungs-Experten kann die Ergebnisse mit einer Anforderungs-Matrix oder einer grafischen Baumdarstellung kontrollieren. Für Kontrollzwecke werden die Ähnlichkeitswerte zusätzlich in einer Textdatei gespeichert.

5.2.3 Validieren der Ergebnisse

Ein Ziel der Arbeit ist das Vergleichen der Algorithmen zur Erkennung der Abhängigkeiten zwischen den Anforderungen. Für diesen Zweck wurde ein eigenes Testprojekt mit Beziehungen zwischen den Anforderungen angelegt. Die Anforderungen dieses Testprojekts sind identisch mit dem analysierten Projekt, sodass ein Vergleichen der Ergebnisse möglich ist.

Als Kennzahlen werden die Werte „Precision“ (Genauigkeit) und „Recall“ (Trefferquote) berechnet. Diese Werte geben Auskunft über die Güte der ermittelten Ergebnisse.

Für das beschriebene Projekt geht man von einem Set von Anforderungen Ψ aus. Für eine Anforderung d gibt es R_d richtige Abhängigkeiten. Der Information Retrieval Algorithmus

berechnet n_d Abhängigkeiten für die Anforderung d, von denen r_d richtig berechnet wurden [Hayes06]. Für beide Berechnungsarten liegen die Werte zwischen $[0, 1]$.

- ***Recall***

  Der $recall$ (Trefferquote) berechnet eine Prozentzahl von richtig gefundenen Abhängigkeiten zwischen den Anforderungen. Für ein einzelnes Dokument d ergibt sich der $recall$ mit folgender Formel:

$$recall = \frac{r_d}{R_d} = \frac{korrekt \ \wedge gefunden}{korrekt} \%$$

  Für die weitere Arbeit wird der $recall$ für das gesamte Set von Anforderungen berechnet. Dieser berechnete $recall$ trifft eine Aussage über den Prozentsatz von allen richtig gefundenen Abhängigkeiten zwischen den Anforderungen:

$$recall = \frac{\sum_{d \in \Psi} r_d}{\sum_{d \in \Psi} R_d}$$

  Ein $recall = 1$ sagt aus, dass alle korrekten Abhängigkeiten gefunden wurden. Er gibt keine Aussage darüber, ob zusätzlich nicht korrekte Abhängigkeiten gefunden wurden.

- ***Precision***

  Die $precision$ (Genauigkeit) berechnet die Genauigkeit der ermittelten Abhängigkeiten.

$$precision = \frac{r_d}{n_d} = \frac{korrekt \ \wedge gefunden}{gefunden} \%$$

  Für die spätere Auswertung wird die $precision$ für das gesamte Set von Anforderungen berechnet. Dieser berechnete $precision$ Wert trifft eine Aussage über den Anteil von relevanten Abhängigkeiten zwischen den Anforderungen:

$$precision = \frac{\sum_{d \in \Psi} r_d}{\sum_{d \in \Psi} n_d}$$

Ein $precision = 1$ sagt aus, dass alle gefundenen Abhängigkeiten korrekt sind. Er gibt keine Aussage darüber, ob zusätzliche korrekte Abhängigkeiten vorliegen, die nicht gefunden wurden.

- ***Grenzwert***

  Bei der Analyse der Algorithmen wurde ein Grenzwert beschrieben mit dessen Hilfe die Abhängigkeiten bestimmt werden können. Sobald ein Ähnlichkeitswert diesen Grenzwert ε übersteigt, geht man von einer Abhängigkeit zwischen den beiden Anforderungen aus. Es wurden für beide Algorithmen unterschiedliche Grenzwerte angesetzt. Beim Vector-Space-Model ist ein weit verbreiteter Wert $\varepsilon = 0{,}7$. Dieser Wert entspricht einem 45° Winkel zwischen den Vektoren [Marc03]. Je höher dieser Wert gewählt wird, desto ähnlicher müssen sich die Wörter in den Anforderungen sein. Für die Auswertung der Arbeit wurde dieser Wert unterschiedlich angepasst, um eventuell bessere Werte bestimmen zu können.

  Für den Latent-Semantic-Indexing Algorithmus wurde ein Grenzwert $\varepsilon = 0{,}07$ gewählt. Testversuche haben ergeben, dass mit diesem Wert die besten Ergebnisse für das Testset ermittelt werden. Die Testversuche werden in Kapitel 8 „Auswertung und Analyse der Ergebnisse“ beschrieben.

  Für Analysezwecke ist es Möglich alle Ähnlichkeitswerte zwischen den Anforderungen in eine Datei auszugeben. Wenn man die Werte mit Farben hinterlegt ergibt sich folgende Darstellung der analysierten Abhängigkeiten:

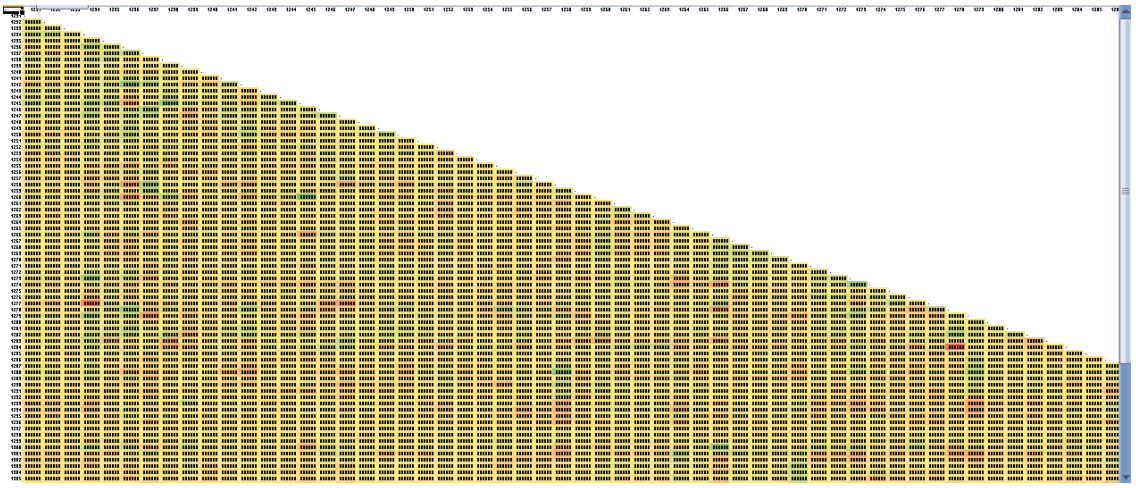


Abbildung 6: Grafische Aufbereitung der Abhängigkeitsergebnisse zwischen den Anforderungen

In der ersten Zeile und Spalte befinden sich die IDs der Anforderungen. Grüne Felder bedeuten eine hohe Wahrscheinlichkeit, dass die beiden Anforderungen eine Abhängigkeit besitzen und rote Felder, dass sie keine haben.

Wenn die Darstellung vergrößert wird, bietet sich für alle Anforderungen ein klares Bild der Abhängigkeit.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	-	1231	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247	1248
2	1231	-																	
3	1232	0,05099	-																
4	1233	0,02458	0,01992	-															
5	1234	0,06300	0,07100	0,01495	-														
6	1235	0,03631	0,01642	0,01444	0,02313	-													
7	1236	-0,01161	0,00501	-0,00667	0,11835	0,04758	-												
8	1237	-0,01154	-0,01425	-0,00657	-0,01581	0,05158	0,08997	-											
9	1238	0,05856	0,03847	0,02169	0,06218	0,06796	0,02570	0,00070	-										
10	1239	0,01475	0,00730	0,00079	0,03178	0,00345	0,01872	0,00187	0,01402	-									
11	1240	0,02387	0,01323	0,01160	0,01102	0,01528	-0,01484	-0,01786	0,01828	0,01259	-								
12	1241	0,01916	0,01852	0,00539	0,04667	0,03461	0,05350	0,03518	0,04071	0,00411	0,01252	-							
13	1242	-0,02900	-0,01723	-0,01170	0,01928	0,04672	0,17631	0,10904	0,01036	-0,02223	-0,01877	0,04723	-						
14	1243	0,04885	0,03962	0,02049	0,04794	0,02990	-0,00600	-0,00105	0,04088	0,00491	0,01551	0,01808	-0,01901	-					
15	1244	0,04491	0,02712	0,02015	0,01883	0,03736	-0,01676	0,01234	0,04820	0,00024	0,01851	0,01809	-0,01538	0,03512	-				
16	1245	0,08326	0,05340	0,02615	0,10271	0,09102	-0,05457	0,00408	0,16202	0,03188	0,03428	0,04179	-0,03396	0,05990	0,06738	-			
17	1246	0,00907	0,00822	-0,00228	0,05280	0,02055	0,05122	0,12534	0,00433	-0,01768	-0,00727	0,04278	0,05034	0,01796	0,01358	0,00703	-		
18	1247	0,02525	0,03175	0,00383	0,10735	0,01507	0,07050	0,09293	0,02255	-0,05183	-0,00234	0,05543	0,04151	0,02698	0,02074	0,06152	0,09787	-	
19	1248	-0,01001	-0,00307	-0,00522	0,01717	0,00105	0,02633	0,00489	0,00833	-0,01028	-0,00845	0,01509	0,02812	-0,00323	-0,00839	0,00013	0,01474	0,01577	-
20	1249	0,00027	0,01073	-0,00332	0,03506	0,01764	0,07559	0,00862	0,00470	-0,01294	-0,01495	0,01951	0,06197	0,00008	-0,00964	-0,01331	0,00741	0,01702	0,01927
21	1250	-0,02322	-0,01359	-0,00950	0,00469	0,02848	0,09084	0,02151	0,00891	-0,01317	-0,00978	0,02835	0,08773	-0,01822	-0,01526	-0,00964	0,00620	0,01037	0,02549
22	1251	0,02438	0,02376	0,00588	0,05917	0,03495	0,06817	0,03236	0,01918	0,01210	0,00221	0,02867	0,04245	0,02451	0,00715	0,00227	0,03638	0,02275	0,00428
23	1252	0,03918	0,04050	0,01442	0,05790	-0,00004	0,00003	-0,01923	0,04959	0,03099	0,01044	0,01023	-0,03109	0,02991	0,02116	0,05374	0,00051	0,01884	0,00257
24	1253	-0,01634	-0,01615	-0,00962	-0,00685	0,02436	0,04352	0,05452	-0,00088	-0,00439	-0,01236	0,01956	0,05274	-0,01092	-0,00559	-0,01347	0,02979	0,01219	0,01001
25	1254	-0,00756	-0,00579	-0,00662	0,00600	-0,00484	0,00324	0,03716	-0,01696	-0,00589	0,00354	0,01639	0,01044	-0,00079	-0,00287	-0,01287	0,04067	0,03190	0,00352
26	1255	-0,00472	-0,00421	-0,00197	-0,01490	-0,01814	-0,04047	-0,00297	-0,01982	0,01325	0,04573	0,02707	-0,01216	-0,00558	-0,00575	-0,01626	-0,00491	0,00912	-0,00370

Abbildung 7: Vergrößerte Darstellung der Analyseergebnisse

Die Tabellen zeigen einen Analyselauf des Latent-Semantic-Indexing Algorithmus auf die Fallstudie Pacemaker (erklärt in Kapitel 7.2 Pacemaker).

6 Beschreibung der Lösung

In diesem Kapitel wird das Umfeld des implementierten Werkzeugs vorgestellt. Es wird auf die Architektur der Lösung eingegangen und die einzelnen Komponenten erklärt. Das Gesamtsystem besteht aus einer Server-Client Architektur. Die Anforderungen werden auf einem kommerziellen Anforderungsmanagementsystem gespeichert. Das implementierte Werkzeug greift auf diese Anforderungen zu und führt die Analysen durch. Festgestellte Abhängigkeiten zwischen den Anforderungen werden wieder zurück auf den Server gespielt.

Die Beschreibung des eigentlichen Werkzeugs inklusive Beschreibung der Benutzung befindet sich im Anhang.

6.1 Anforderungsmanagementsystem - Starbase (Borland) CaliberRM

Das Verwalten einer großen Anzahl von Anforderungen wird erst durch ein professionelles Anforderungsmanagementsystem möglich gemacht. Für diese Arbeit fiel die Entscheidung auf das Produkt Starbase (Borland) CaliberRM [Caliber]. CaliberRM ist ein Anforderungsmanagementsystem zur Verwaltung, Validierung und Versionierung von Anforderungen. Die Anforderungen werden auf dem CaliberRM Server in der proprietären Versant-Datenbank gespeichert. Diese ist im CaliberRM-Produkt enthalten und wird mit dem Produkt mit installiert. Die Versant-Datenbank ist einer objektorientierten Datenbank, die auf einem Windows 2000 oder NT-Server installiert werden muss. Die richtige Installation des CaliberRM Servers ist von großer Bedeutung, da die Datenbankwartung sehr aufwändig ist und es wenige freie Informationen über die Datenbank gibt.

CaliberRM kann Anforderungen in einer – analog zu Microsoft Windows - baumähnlichen Struktur oder als Netzwerkdiagramm anzeigen. In der Baumdarstellung wird als Wurzel das Projekt angezeigt. Darunter werden die Anforderungen in einer logischen, hierarchischen Weise angezeigt. Beispielsweise können die Anforderungen in Funktionsbereiche unterteilt werden. Für die eindeutige Identifizierung wird ein automatischer Schlüssel an jede Anforderung vergeben. Diese kann in der Baumstruktur angezeigt werden.

Anforderungen können direkt in CaliberRM eingegeben oder importiert werden. Bei der manuellen Eingabe von Anforderungen überprüft CaliberRM automatisch alle Wörter auf Rechtschreibfehler. Diese können ähnlich wie in Microsoft Word ausgebessert werden.

Zusätzlich wird jedes eingegebene Wort gegen die projektspezifischen Wörterbücher geprüft. Der Administrator kann beim Erstellen eines Anforderungsprojekts diese Wörterbücher importieren. Diese Funktionalität forciert die Verwendung einer einheitlichen Terminologie und reduziert somit die Gefahr von Missverständnissen bei der Implementierung. Anforderungen können in CaliberRM aus Microsoft Word und/oder Excel importiert werden. Der Import-Assistent kann die Anforderungen je nach Import-Dateityp, durch Schlüsselwörter, Trennzeichen oder Überschriften identifizieren. CaliberRM ermöglicht auch den Export von Anforderungen in Microsoft Access, Microsoft Word oder HTML-Dateien.

Das wichtigste Argument für den Einsatz von CaliberRM ist die gut ausgebaute Traceability Funktion. Anforderungen können ohne großen Aufwand mit anderen Anforderungen verknüpft werden. Zusätzlich kann man Anforderungen mit externen Dokumenten verknüpfen. Die eingetragenen Abhängigkeiten lassen sich in einem Abhängigkeitsraster (Matrix) oder in einem grafischen Netzwerkdiagramm darstellen. Zur Hervorhebung von Abhängigkeit können diese „verdächtig“ markiert werden. Im CaliberRM Client werden diese Traceability-Links mit roter Farbe angezeigt.

Die anderen Features des Produkts kamen in dieser Arbeit nicht zum Einsatz und werden deshalb nicht an dieser Stelle beschrieben. Bei einer Integration des Analyse-Werkzeugs in einem praktischen Projekt kann jedoch der volle Funktionsumfang der CaliberRM Suite genutzt werden. Die nächsten Absätze analysieren die Stärken und Schwächen von CaliberRM und beschreiben typische Projekte bei denen die Software zum Einsatz kommt.

Zu den eindeutigen Stärken des Tools gehört, dass die Anforderungen in einer Datenbank gespeichert werden. Dies erhöht im Gegensatz zu einer nicht datenbankbasierten Anwendung die Performance. Der integrierte Glossar ermöglicht eine Rechtschreibprüfung und gewährt eine konsistente Anforderungsterminologie. Ein weiterer Vorteil der Anwendung ist, dass die Anforderungen von CaliberRM direkt mit Drittanwendungen verknüpft werden können.

Der größte Nachteil von CaliberRM ist, dass es keine ausreichenden online Hilfestellungen gibt und der Support des Herstellers inakzeptabel ist. Für externe Weiterentwicklungen stellte sich die verwendete Datenbank als Schwäche des Tools heraus. Diese ist Proprietär und es kann nicht über die Datenbank auf die Anforderungen zugegriffen werden. Da es sich bei

CaliberRM um ein kommerzielles Tool handelt, entstehen durch die Anschaffung und die Lizenzen beim Benutzer hohe Kosten.

Einsatzgebiet für CaliberRM

- Projekte, die bereits ein funktionierendes Anforderungsmanagement haben
- Teamgröße größer als 5 Personen
- Projekte, die mit anderen Starbase-Produkten arbeiten
- Projekte mit Einzel- und mehreren Standorten, da die Anforderungen zentralisiert gespeichert werden können
- Projekte, die CMM Level 2-Zertifizierung erfordern oder höher

Zusätzlich zur CaliberRM Client-Software stellt der Hersteller ein Eclipse-Plugin zur Verfügung. Dieses Plugin ist proprietär und es ist nicht möglich die bestehende Funktionalität zu erweitern. Aus diesem Grund wurde ein neues Plugin für Eclipse geschrieben und um die Analysefunktionalität ergänzt.

Die Entwicklungsumgebung Eclipse wird von sehr vielen Softwareentwicklungsprojekten eingesetzt und ist somit meist auf den Projekten vorhanden. Der modulare Aufbau von Eclipse ermöglicht es Erweiterungen (Plug-Ins) für dieses Framework zu schreiben. Für diese Arbeit wurde das Eclipse Framework in der Version 3.3.0 verwendet. Die ältere Version musste verwendet werden, um mit den Borland Produkten kompatibel zu sein. Für die CaliberRM Suite stellt der Hersteller eine Entwicklerschnittstelle zur Verfügung. Das CaliberRM SDK ermöglicht den Zugriff und die Manipulation von sämtlichen gespeicherten Anforderungen, deren Attribute, Projekten oder Benutzern. Das SDK kann in Java oder .Net Programmen verwendet werden.

Eine ausführliche Beschreibung des Requirement Traceability Werkzeugs befindet sich in Kapitel 12 Anhang - Requirement Traceability Werkzeug.

6.2 Technische Architektur

Der in diesem Kapitel beschriebene technische Aufbau wurde bei der Analyse der Algorithmen verwendet. Beim Einsatz des Requirement Traceability Werkzeugs in der Praxis wird dieser Aufbau abweichen. Die Basiskomponenten werden jedoch bestehen bleiben.

Zu den Basiskomponenten zählt ein Server, der die Anforderungen speichert. Ein klassisches Anforderungsmanagementsystem muss dem Benutzer Zugriff auf die Anforderungen bieten. Die zentrale Komponente stellt das Requirement Traceability Werkzeug dar.

Für diese Arbeit wurde als Anforderungsmanagementsystem die Borland CaliberRM Suite genommen. Die Installation des Borland CaliberRM Servers stellte sich als Herausforderung dar. Der Borland CaliberRM Server verlangt als Mindestanforderung ein Server Betriebssystems von Windows, doch die verwendete Testumgebung basiert auf einem Windows XP Betriebssystem. Mit Hilfe der Virtualisierungssoftware "Virtual PC" wurde eine virtuelle Windows Server 2003 Instanz eröffnen. Danach war die Installation und Nutzung des Borland CaliberRM Servers einwandfrei möglich.

Auf dem Client werden die Borland CaliberRM Client-Software und ein vorkonfiguriertes Eclipse mit dem Requirement Traceability Werkezug Plugin verwendet. Beide Clients werden für den Zugriff auf den CaliberRM Server konfiguriert, sodass sie die Anforderungen abrufen und manipulieren können.

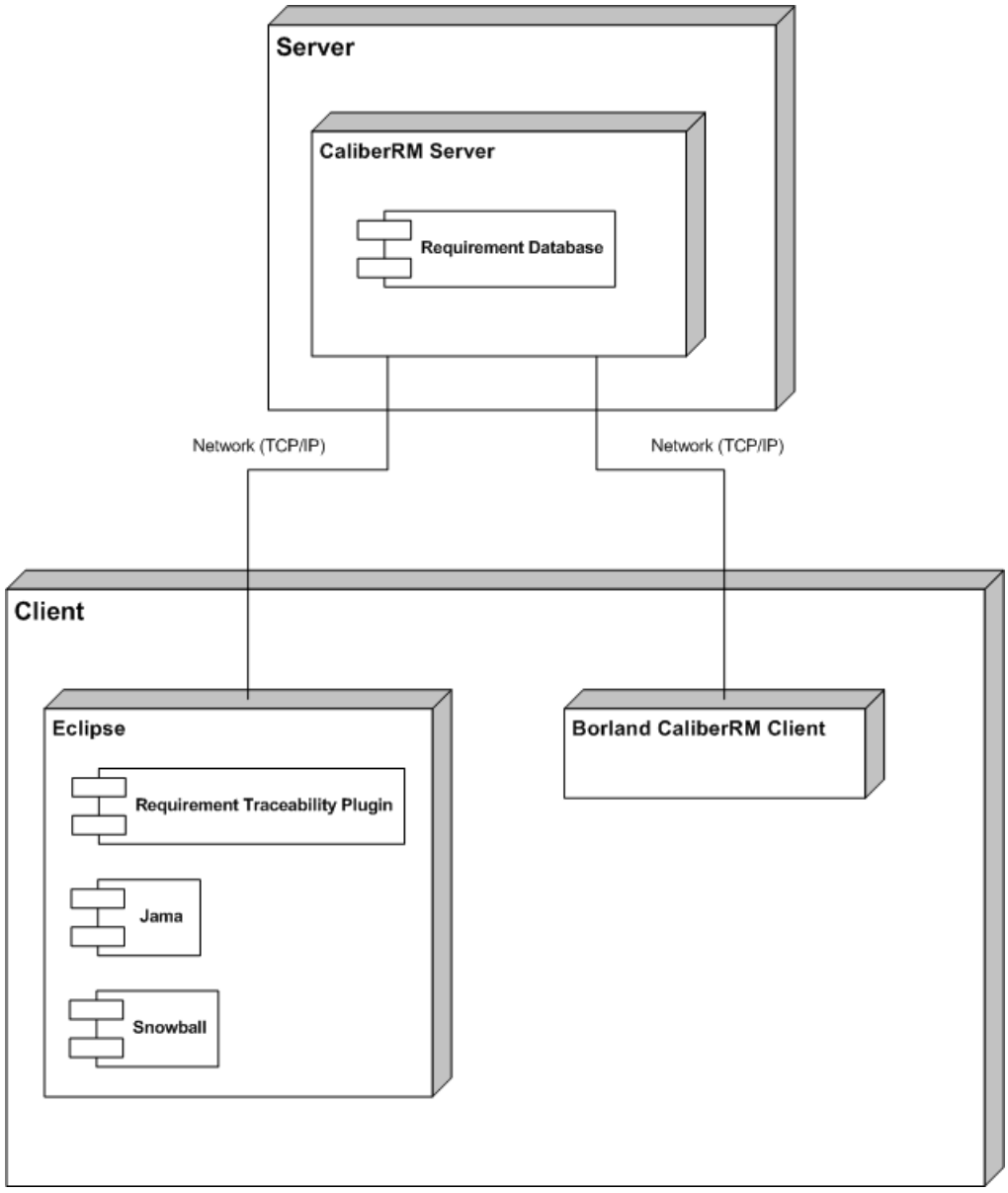


Abbildung 8: Deployment Diagramm des Requirement Traceability Werkzeugs

6.3 Technischer Aufbau des Werkzeugs

Das Plugin besteht aus mehren Modulen. Der Schnitt in diese Module soll die Komplexität reduzieren und die Kompatibilität erhöhen. Als Basis-Modul dient das Connection Handling. Hier wurden die Funktionen für die Verwaltung der Serververbindung implementiert. Gemeinsam mit der der CaliberRM Functions Komponente sind diese Klassen sehr stark auf die Verwendung der Borland CaliberRM Suite ausgerichtet. Die CaliberRM Functions fassen alle Funktionen zur Verwaltung der Anforderungen zusammen. Durch diese Kapselung sollte

es mit minimalem Aufwand möglich sein, die Anforderungen auch von einem anderen Anforderungsmanagementsystem oder von einer Datenbank zu beziehen.

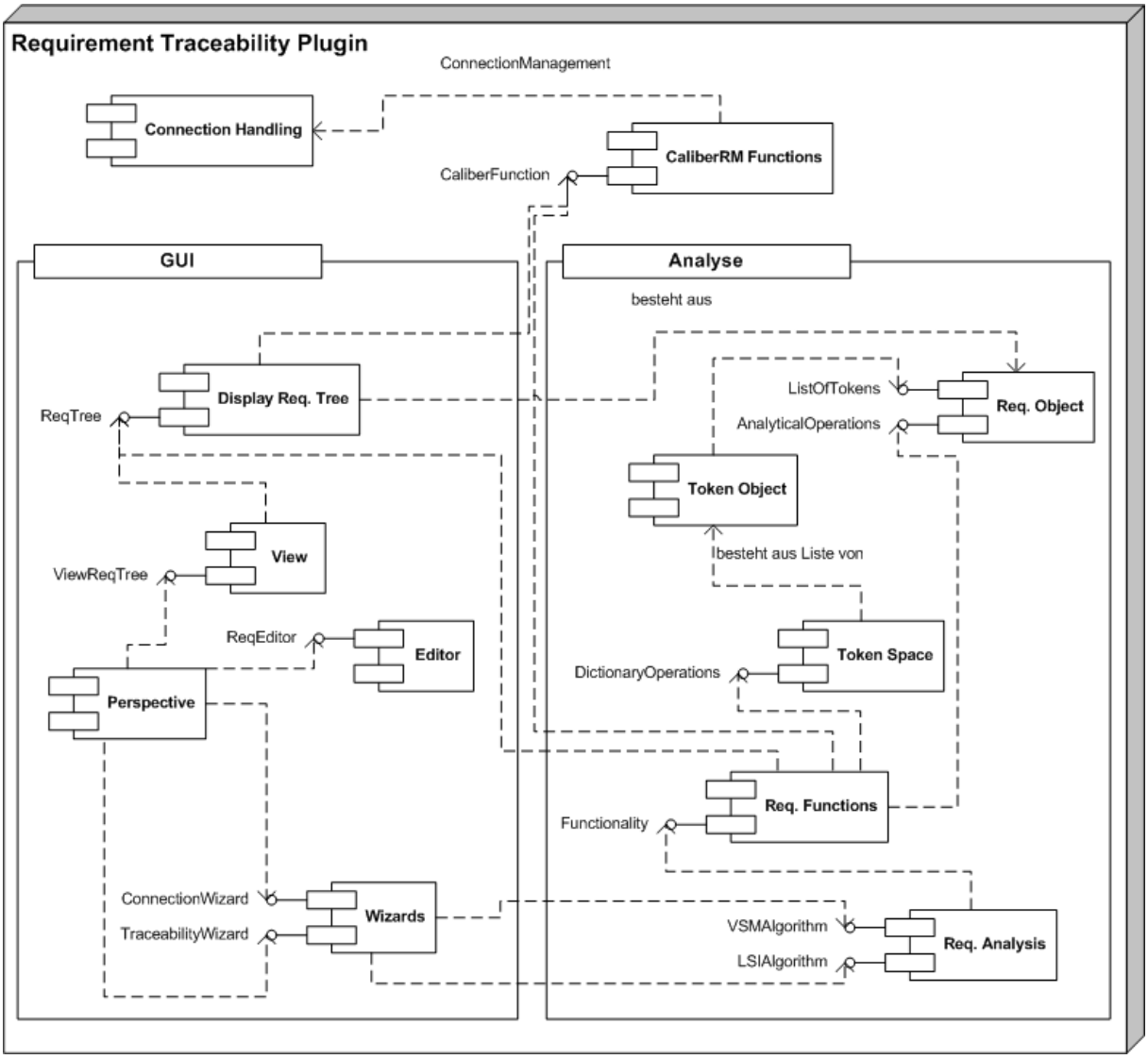


Abbildung 9: Komponenten Diagramm des Requirement Traceability Werkzeugs

Darauf aufbauend teilt sich die technische Architektur in die Darstellung der Daten (GUI-Komponente) und in die Analyse der Daten (Analysekomponente). Die Darstellungsklassen werden ausführlich im Anhang beschrieben. Die Komponente Display Req. Tree dient der Repräsentation aller Anforderungen in einer abstrakten Baumstruktur. Diese wird sowohl von der grafischen Oberfläche als auch von den Analyse-Modulen verwendet. Der Display Tree besteht aus Requirement Objects. Jede Anforderung wird als ein Requirement Object

repräsentiert. Diese Objekte beinhalten alle Informationen und Methoden für eine Anforderung. Während der Verarbeitung einer Anforderung wird der Anforderungstext in einen Token Space umgewandelt. Dieser Token Space besteht aus einer Liste von Wörtern und bietet den anderen Klassen verschiedenste Manipulationsoperationen an. Die einzelnen Wörter werden als Token Objects repräsentiert. Somit besteht ein Token Space aus einer Liste von Token Objects. Das Wörterbuch im Analysealgorithmus wurde ebenfalls mittels Token Space Object realisiert. Aus diesem Grund befinden sich auch die Wörterbuch-Analysefunktionen im Token Space Object.

Die eigentlichen Analysefunktionen sind unter den Requirements Functions zusammengefasst. Hier befinden sich die Operationen für die Analyse-Algorithmen und die mathematischen Berechnungen zur Validierung der Algorithmen. Der Ablauf der Analyse wird durch das Requirement Analysis Modul gesteuert. Dieses Modul wurde als Thread implementiert, da die Berechnung unabhängig von der grafischen Oberfläche erfolgen sollte. Das Analyse-Modul verarbeitet die Anforderungen und meldet den aktuellen Status an den Analyse-Wizard. Durch dieses Vorgehen bekommt der Benutzer den Fortschritt angezeigt.

Für die Berechnung der Algorithmen wurden zwei externe Java-Pakete eingebunden. Das Snowball-Paket beinhaltet Funktionen zum Stemming von Wörtern. Es handelt sich dabei um die Java-Implementierung des Porter-Stemming Algorithmus [Porter80]. Das Paket beinhaltet den Support für verschiedenste Sprachen. In dieser Implementierung wurden die Sprachpakete für Deutsch und Englisch verwendet. Für die effiziente Berechnung der Matrizenoperationen im Latent-Semantic-Indexing Algorithmus kam das Matrizenpaket Jama zum Einsatz [Jama05]. Diese Java-Bibliothek stellt die gängigsten Matrizenfunktionen zur Verfügung. Unter anderem auch die Singulärwertzerlegung. Die Implementierung zeichnet sich durch ihre ausgezeichnete Performance aus.

Das Requirements Traceability Werkzeug kann durch das Konfigurations-Modul in verschiedene Modi versetzt werden. Diese Modi werden im Kapitel 12.1.5 beschrieben.

7 Fallstudien

Das Requirements Traceability Werkzeug wurde mit zwei Fallstudien getestet. Die Anforderungen der ersten Fallstudie spezifizieren eine Wohnungspartnerbörse.

7.1 Wohnungspartnerbörse

Die Fallstudie wurde einer realen Anwendung nachempfunden, die Anforderungsinhalte sind jedoch erfunden. Der Inhalt der Anforderungen beschreibt einen kleinen Teilbereich einer Wohnungspartnerbörse. Diese Online-Plattform hat den Zweck, Wohnungssuchende und Wohnungsanbieter zu vermitteln. Das Portal soll eine einfache Suche über alle freien WG-Plätze ermöglichen. Der Business Case hinter diesem fiktiven Portal könnten die Werbeeinnahmen oder Premium Kunden sein. Die Wohnungspartnerbörse wird von der „Fiktiv Wohnungsvermittlungs GmbH“ betrieben. Diese stellt Immobilienmakler an, die die Vermittlungsgeschäfte überwachen und für Premium Kunden abwickeln.

Jede Anforderung wurde im Anforderungsmanagementsystem Borland CaliberRM eingegeben. Eine Anforderung besteht aus einem Titel und dem Anforderungstext. Der Anforderungstext wurde mit der Anforderungsschablone von Kapitel 2.3 erfasst. Für die Fallstudie wurden nur funktionale Anforderungen erhoben. Durch das schrittweise verfeinern der Anforderungen entsteht die hierarchische Baumstruktur. Zur verbesserten Darstellung der Ebenen wurden für jede Hierarchiestufe eine Römische Ziffern im Titel eingefügt.

7.1.1 Beispielanforderung

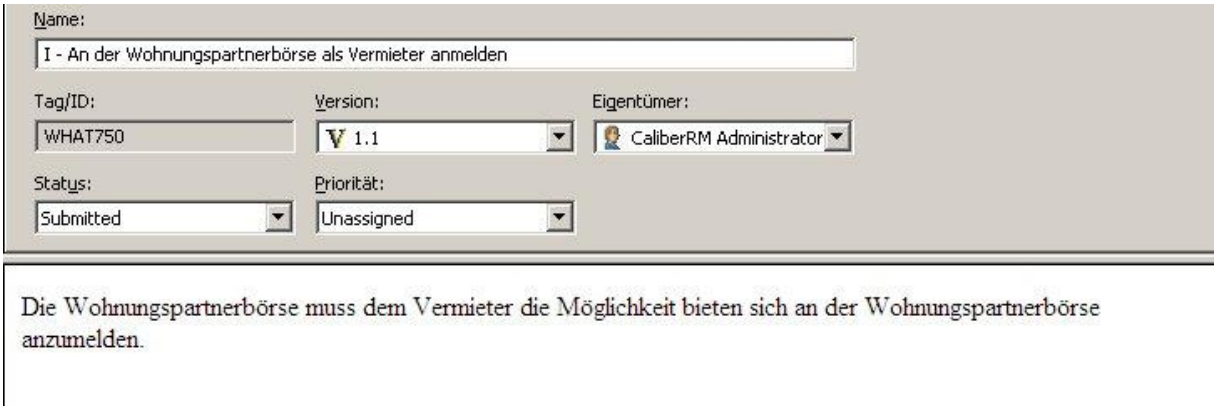


Abbildung 10: Beispielanforderung in Borland CaliberRM

Die zusätzlichen Attribute für eine Anforderung wurden für die Fallstudie nicht ausgefüllt, da der Algorithmus diese nicht zur Analyse verwendet. In der ersten Zeile befindet sich der Titel der Anforderung und im untersten Textfeld die Anforderungsspezifikation. Dieser Text folgt den syntaktischen Regeen der Anforderungsschablone aus Kapitel 2.3. Für diesen Ansatz müssen die verwendeten Wörter standardisiert werden. Vor dem Verfassen der Anforderungen wurde ein Glossar fixiert. Darin werden die wichtigsten Wörter beschrieben. Die Anforderungen basieren auf diesen Wörtern.

7.1.2 Glossar

Wohnungspartnerbörse	**Name des Online-Portals**
Online neue Wohnung suchen	Meldung an die Wohnungsargentur, dass man eine neue Wohnung sucht
Wohnungsplatzangebot	Der Vermieter bietet online einen Wohnungsplatz an
WG-Partnerprofil	Der WG-Suchende erstellt ein Profil zur WG Suche
Immobilienmakler	Makler der „Fiktiv Wohnungsvermittlungs GmbH“
Quicksearch	Einfache Suche nach WG-Plätzen
Hierarchische Suche	Suche über einschränkende Auswahlfelder
Wohnungssuchprofil	Identisch mit dem WG-Partnerprofil (wird nur online verwendet)
WG-suchend	Status für Makler - bedeutet, dass noch nach Wohnung gesucht wird
semantisches suchen	Die Wohnungsplatzsuche erfolgt mittels einer semantischen Suchmaschine. Diese findet die optimale Wohnung oder Mieter.
Treffen	Wenn eine geeignete Wohnung gefunden wurde, wird ein Treffen arrangiert.
2er-WG	Spezielles Wohnungsangebot für 2-Personen Wohnungen
WG-Partner	Person die eine Wohnung sucht
Wohnungsplatz	Ein freier Wohnungsplatz
WG-Suchhelfer	Online Assistent zur Wohnungssuche

Vermieter	Vermieter einer Wohnung /eines WG-Platzes

Tabelle 3: Glossar für Fallstudie Wohnungspartnerbörse

Die erste Hierarchiestufe des Anforderungsbaums beschreibt die groben Anforderungen. Insgesamt befinden sich darunter drei Verfeinerungsstufen.

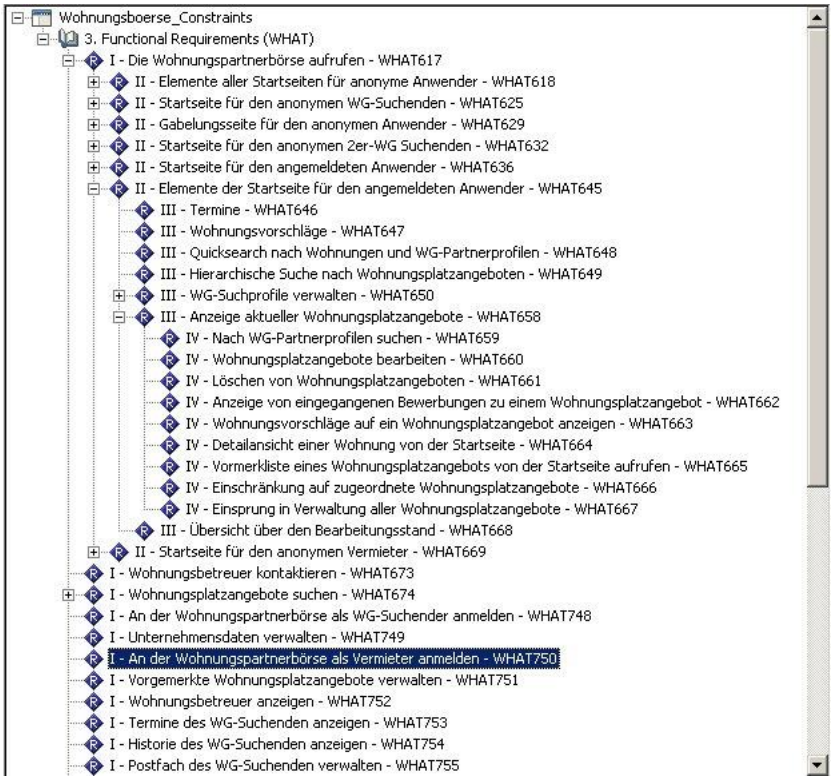


Abbildung 11: Anforderungsbaum der Fallstudie Wohnungspartnerbörse

Der Testaufbau besteht aus einem Anforderungsset mit allen Anforderungen. Für die Evaluierung der Algorithmen müssen die Anforderungen mit Abhängigkeiten im System eingetragen sein. Die Arbeit untersucht die verschiedenen Abhängigkeitstypen und unterscheidet dabei zwischen drei Arten von Abhängigkeiten:

- Strukturelle Abhängigkeiten (Structured Dependencies)
- Beschränkende Abhängigkeiten (Constraint Dependencies)
- Kosten- und wertrelevante Abhängigkeiten (Cost/Value Dependencies)

Für jeden Abhängigkeitstyp wurde eine Kopie des ursprünglichen Anforderungssets erstellt. Ein Anforderungsexperte konnte so für jedes der drei Repositories die entsprechenden Abhängigkeiten ausfüllen. Der nachfolgende Screenshot zeigt diese Projekte durch das rote Rechteck hervorgehoben.

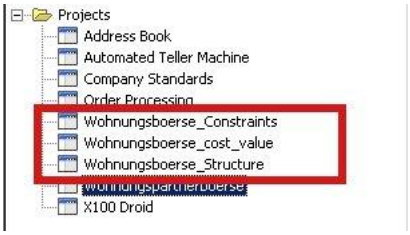


Abbildung 12: Projektübersicht für Evaluierung der verschiedenen Anforderungstypen

Die Abhängigkeiten zwischen den Anforderungen werden im CaliberRM Client verwaltet. Die Software bietet die Möglichkeit bei einer Abhängigkeit die Richtung mit einzugeben. Für diese Arbeit wurde dieses Feature jedoch nicht genutzt, da prinzipielle Abhängigkeiten zwischen den Anforderungen untersucht wurden und für diesen Fall die Richtung der Abhängigkeit als nicht relevant angesehen wird.

Die Darstellung der Abhängigkeiten kann als Abhängigkeitsbaum oder als Requirement Traceability Matrix erfolgen. Die Matrix-Ansicht hat sich während der praktischen Arbeit als übersichtlicher erwiesen. Die nachfolgenden Screenshots zeigen beide Ansichten.

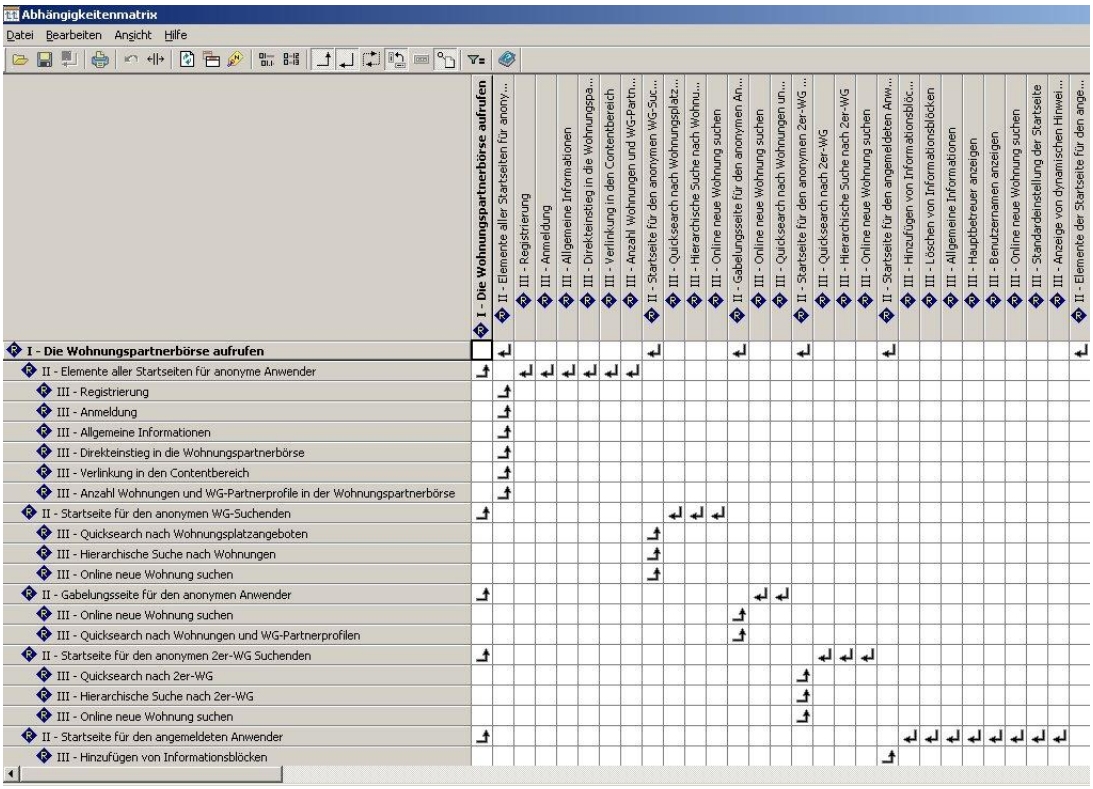


Abbildung 13: Requirements Traceability Matrix für die Fallstudie Wohnungspartnerbörse

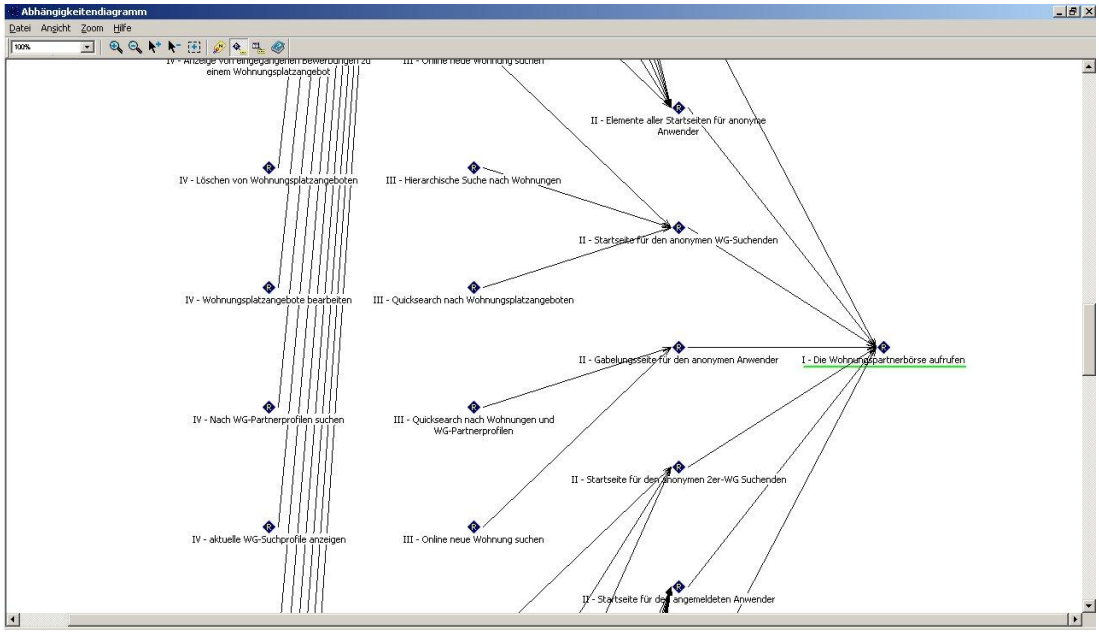


Abbildung 14: Grafische Repräsentation der Requirements Traceability Matrix der Fallstudie Wohnungspartnerbörse

Diese Fallstudie besteht aus 160 Prosa-Anforderungen. Es wurden je nach Repository folgende Abhängigkeiten manuell eingestellt:

- Strukturelle Abhängigkeiten: 163
- Beschränkende Abhängigkeiten: 60
- Kosten- und wertrelevante Abhängigkeiten: 77

7.2 Pacemaker

Diese Fallstudie basiert auf der Spezifikation eines Herzschrittmachers (in Englisch: Pacemaker). Es handelt sich dabei um eine sehr realitätsnahe Spezifikation in Englischer Sprache. Das Anforderungsdokument umfasst 35 Seiten und beinhaltet 98 Anforderungen. Im Internet zur Verfügung gestellt wurde die Spezifikation vom Boston Scientific Institut. Der Import der Anforderungen in das Anforderungsmanagementsystem CaliberRM war über die Transformation in ein Microsoft Word Format möglich. Dafür musste das PDF-Dokument als Textdatei abgespeichert werden und alle Formatierungsfehler manuell ausgebessert werden. Der Text von tabellarischer Spezifikation wurde trotz Formatierungsverlust beibehalten, da auch diese Information durch den Analysealgorithmus ausgewertet wurde.

Die Abhängigkeiten für das Validieren der Ergebnisse mussten manuell gezogen werden. Die Kapitel des Anforderungsdokuments legen die hierarchischen Abhängigkeiten zwischen den Anforderungen fest. Die beschränkenden Abhängigkeiten und kosten- und wertrelevanten Abhängigkeiten mussten vom Autor analysiert und hinzugefügt werden. Die folgende Grafik zeigt einen Ausschnitt des Abhängigkeitsgrafen einer Anforderung. Durch die Aneinanderreihung dieser Abhängigkeiten erkennt man die Wichtigkeit der Anforderung für das Gesamtsystem.

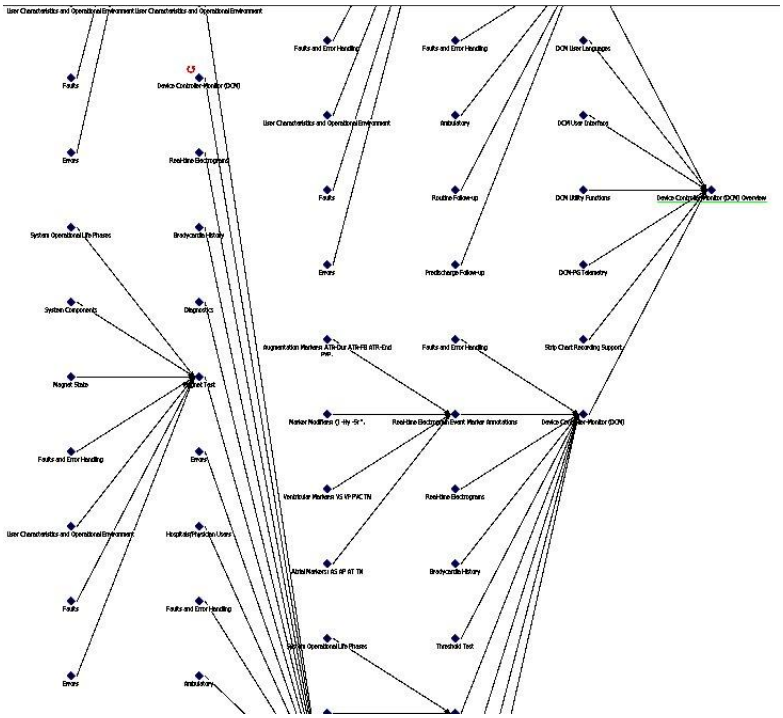


Abbildung 15: Grafische Repräsentation der Requirements Traceability Matrix der Fallstudie Pacemaker

Diese Fallstudie besteht aus 98 Prosa-Anforderungen. Es wurden je nach Repository folgende Abhängigkeiten manuell eingestellt:

- Strukturelle Abhängigkeiten: 59
- Beschränkende Abhängigkeiten: 84
- Kosten- und wertrelevante Abhängigkeiten: 105

8 Auswertung und Analyse der Ergebnisse

Im folgenden Kapitel werden die Algorithmen auf beide Fallstudien angewendet und die Ergebnisse ausgewertet und interpretiert. Folgende Untersuchungen wurden mit den Algorithmen gemacht:

- Interpretation der Recall und Precision Werte
- Ermittlung des optimalen Grenzwerts ε
- Laufzeitverhalten der Algorithmen
- Ermittlung eines optimalen Werts für die Dimensionsreduktion
- Vergleich der Algorithmen
- Interpretation der Analyseergebnisse der Fallstudien

Die wichtigsten Indikatoren für die Auswertung der Algorithmen waren der Recall und der Precision Wert. Die folgende Tabelle zeigt die detaillierten Ergebnisse der Fallstudie Wohnungspartnerbörse. In den ersten zwei Spalten werden die Ergebnisse des Vector-Space-Model-Algorithmus dargestellt. Die Analyseergebnisse des Latent-Semantic-Indexing Algorithmus werden in den rechten zwei Spalten dargestellt. Die Tabelle ist nach den drei Abhängigkeitstypen sortiert und lässt einen direkten Vergleich zwischen diesen zu. Unterhalb des Prozentwerts befinden sich die korrekt gefundenen Abhängigkeiten und stellen sie in Relation mit den tatsächlich korrekten oder den gesamt gefundenen Abhängigkeiten.

		VSM		LSI		
	ε	0,7	0,5	0,07	0,08	
Cost/Value	Recall	16,88%	45,45%	15,58%	14,29%	
		13/77	35/77	12/77	11/77	(korrekt gefunden) / korrekt
	Precision	7,26%	2,71%	1,61%	1,96%	
		13/179	35/1291	12/745	11/561	(korrekt gefunden) / gefunden
Structure	Recall	12,88%	52,15%	34,36%	27,61%	
		21/163	85/163	56/163	45/163	(korrekt gefunden) / korrekt
	Precision	11,73%	6,58%	7,52%	8,02%	
		21/179	85/1291	56/745	45/561	(korrekt gefunden) / gefunden
Constraint	Recall	3,33%	26,67%	10,00%	6,67%	
		2/60	16/60	6/60	4/60	(korrekt gefunden) / korrekt
	Precision	1,12%	1,24%	0,81%	0,71%	
		2/179	16/1291	6/745	4/561	(korrekt gefunden) / gefunden

Tabelle 4: Ergebnisse der Algorithmen gegliedert nach Abhängigkeitstypen für Fallstudie Wohnungspartnerbörse

Die Berechnung von Precision und Recall erfolgt über den Vergleich der gefunden Abhängigkeiten zu den tatsächlichen Abhängigkeiten. Die Ergebnisse der Algorithmen können mit diesem Vorgehen in plakativen Prozentzahlen ausgedrückt werden und somit die Praxistauglichkeit nachweisen. Bei der Justierung der Algorithmen spielt das Festlegen des Grenzwerts ε eine große Rolle. Die Tabelle 4 zeigt nicht nur die Prozentzahl der Algorithmen, sondern auch die tatsächliche Trefferanzahl. In der Praxis wäre es für einen Anforderungs-Experten keine Lösung, wenn er 1291 Vorschläge für Abhängigkeiten erhält. Unter diesen zahlreichen Vorschlägen befinden sich 85 richtig erkannte Abhängigkeiten. Trotzdem müsste der Analytiker alle vorgeschlagenen Beziehungen analysieren. Für die Auswertung der Fallstudien wurde deshalb von einem Grenzwert ausgegangen, der eine realistische Anzahl von vorgeschlagenen Abhängigkeiten ausgibt. Dieser Grenzwert wurde für die Vergleichbarkeit fixiert. Die detaillierten Ergebnisse werden jedoch immer mit variablen Grenzwerten dargestellt, da man so ein besseres Verständnis für die Genauigkeit erhält. Die grafische Repräsentation der Ergebnisse lässt ein einfaches Vergleichen der Algorithmen zu.

Anhand der Fallstudie Pacemaker wurde die Verwendung der Substitution der Wörter getestet. Da die Anforderungstexte sehr medizinisch sind, wird dem Leser am Anfang ein Abkürzungsverzeichnis bereitgestellt. Dieses Glossar wurde als Substitutionsliste für den Algorithmus aufgenommen.

```
AP;Atrial Pace
AS;Atrial Sense
ARP;Atrial Refractory Period
ATR;Atrial Tachycardia Response
AV;Atrial-to-Ventricular
BOL;Beginning Of (battery) Life
BPM;Beats Per Minute
cc;Cardiac Cycle(s)
...
```

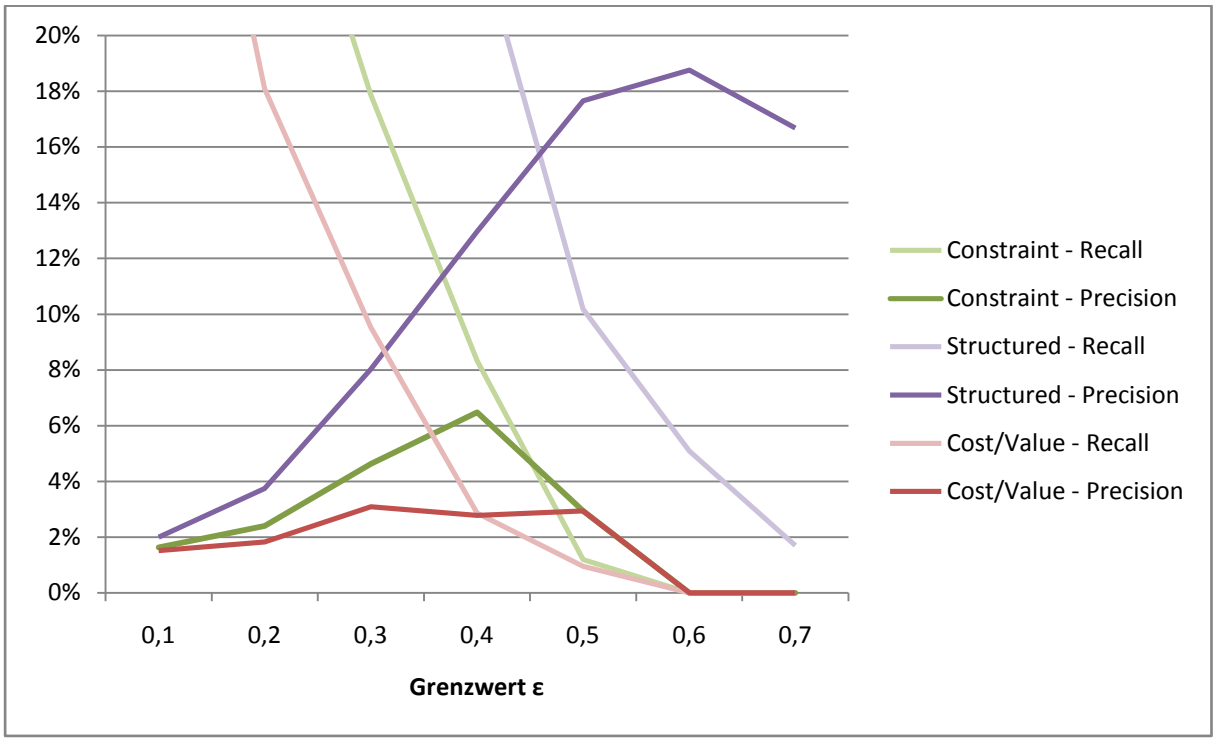
Tabelle 5: Ausschnitt des Abkürzungsverzeichnisses

Durch die Verwendung der Substitutionsliste verschlechterten sich die Analyseergebnisse. Aus diesem Grund wird diese Substitutionsliste für den Algorithmus nicht verwendet.

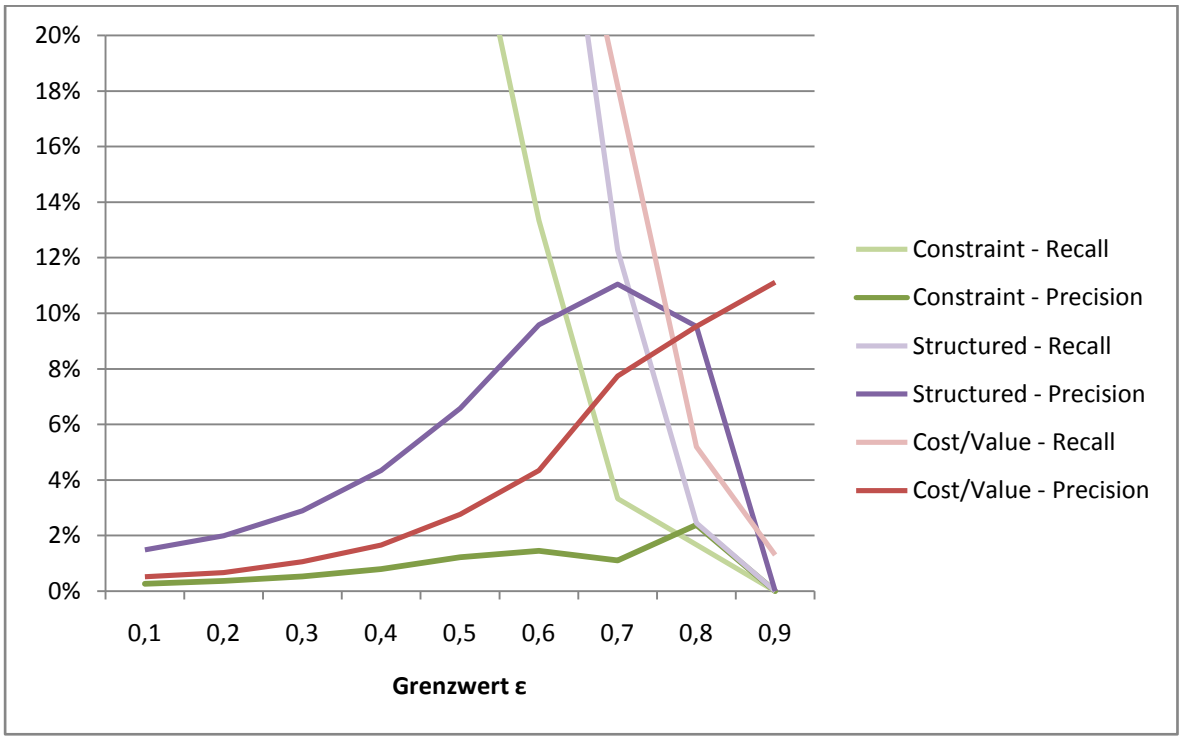
8.1 Vector-Space-Model

Die nachfolgenden Auswertungen der Algorithmen wurden mit beiden Fallstudien erstellt. Die Grafiken zeigen jeweils eine Fallstudie und stellen die Ergebnisse der Abhängigkeitstypen gegenüber. Die X-Achse enthält die Grenzwerte im Intervall von 0,1 bis

0,7 und die Y-Achse die Prozentzahlen für Recall und Precision. Für jeden Abhängigkeitstyp werden diese Werte mittels einer Geraden für alle Grenzwerte dargestellt.



Grafik 1: Grenzwertentwicklung bei VSM-Algorithmus für Fallstudie Pacemaker



Grafik 2: Grenzwertentwicklung bei VSM-Algorithmus für Fallstudie Wohnungspartnerbörse

Die Auswertung der Fallstudien zeigt eine klare Abhängigkeit der Ergebnisse mit dem Typ der Abhängigkeit. Am Besten werden die strukturelle Abhängigkeiten (Structured Dependencies) erkannt. Die anderen Abhängigkeitstypen liegen weit abgeschlagen bei einer Genauigkeit von nur wenigen Prozent. Beschränkende Abhängigkeiten (Constraint Dependencies) schneiden trotzdem besser ab als die kosten- und wertrelevanten Abhängigkeiten (Cost/Value Dependencies).

Gründe für dieses Ergebnis:

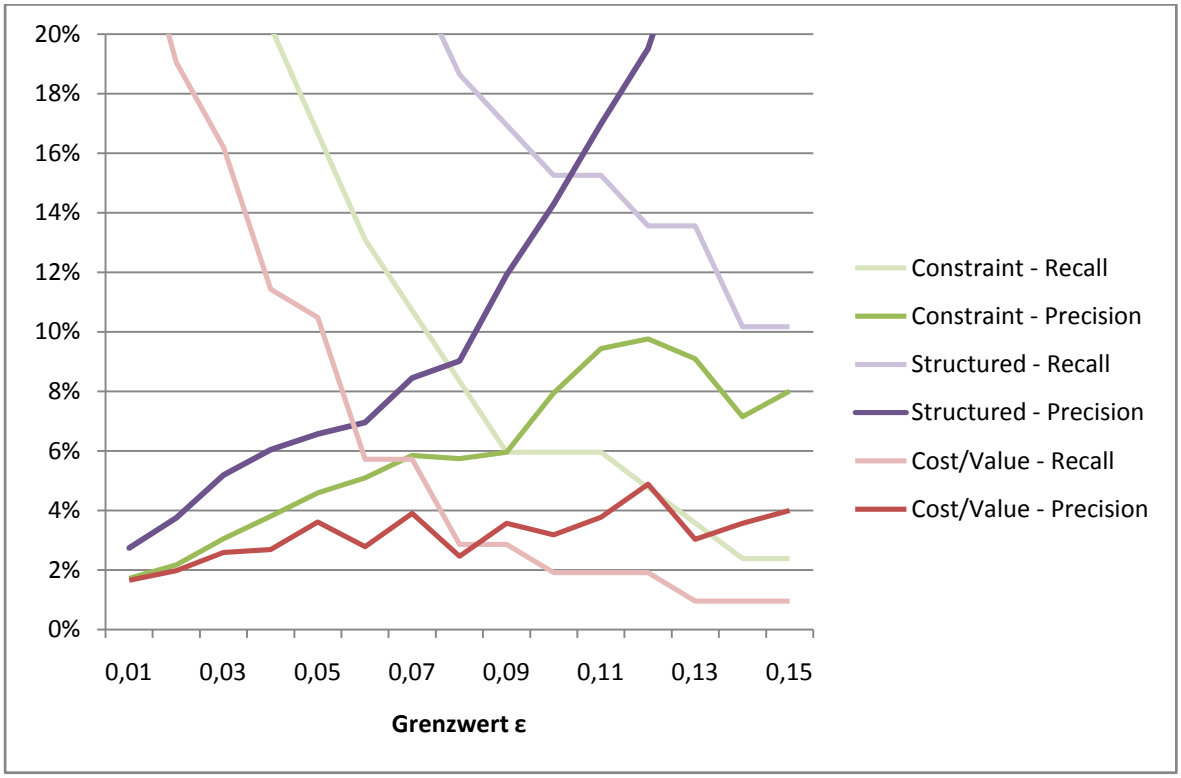
- Die strukturellen Abhängigkeiten ergeben sich großteils aus der ursprünglichen Struktur des Anforderungsdokuments und wurden vom Autor der Anforderungen so berücksichtigt.

- Strukturelle Abhängigkeiten verfeinern meist die Angaben in der unteren Hierarchieebene. Dabei werden die gleichen Wörter verwendet und deshalb erreicht der Algorithmus bessere Ergebnisse.

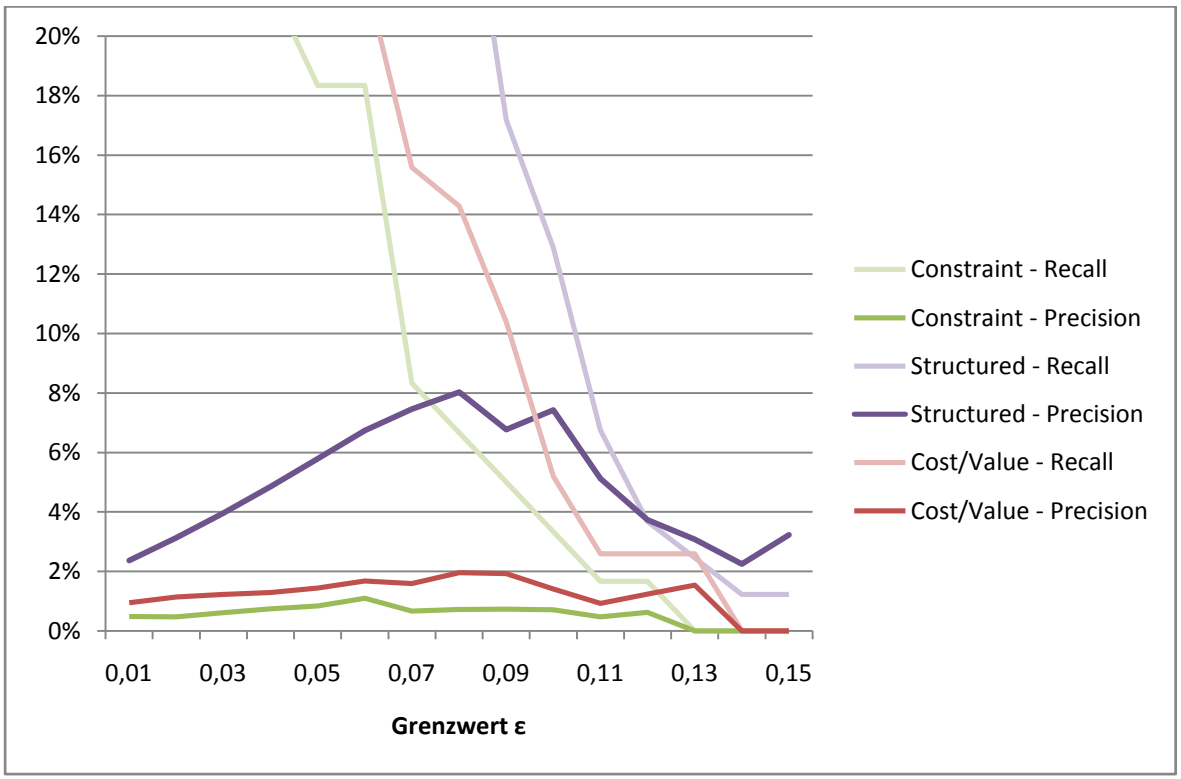
- Beschränkende und kosten- und wertrelevante Abhängigkeiten haben keinen direkten Zusammenhang zu den verwendeten Wörtern in den Anforderungen. Sie werden aus dem inhaltlichen (semantischen) Zusammenhang gezogen.

- Die beschränkenden Abhängigkeitstypen mussten vom Autor der Arbeit gezogen werden. Dabei können sich Fehler eingeschlichen haben und das Ergebnisse negativ beeinflusst haben. Dennoch wird hier davon ausgegangen, dass diese Fehlerrate keine signifikante Auswirkung auf das Ergebnis hatte.

8.2 Latent-Semantic-Indexing



Grafik 3: Grenzwertentwicklung bei LSI-Algorithmus für Fallstudie Pacemaker

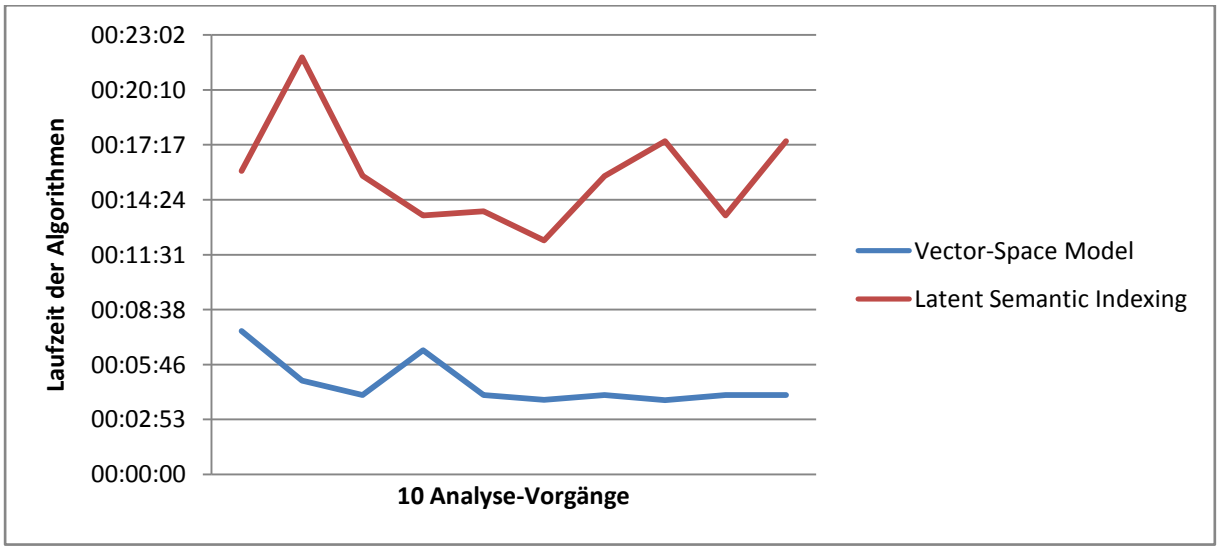


Grafik 4: Grenzwertentwicklung bei LSI-Algorithmus für Fallstudie Wohnungspartnerbörse

Bei der Auswertung des Latent-Semantic-Indexing Algorithmus zeigt sich ein ähnliches Bild, wie beim Vector-Space-Model Algorithmus. Die strukturellen Abhängigkeiten werden um einiges besser erkannt als die beiden anderen Abhängigkeitstypen. Im direkten Vergleich der Charts fällt die Verschiebung der Grenzwerte ε abhängig vom Abhängigkeitstyp auf. Recall und Precision treffen sich bei den strukturellen Abhängigkeiten immer bei einem höheren Grenzwert. Mit diesem Wissen lassen sich die Ergebnisse abhängig vom Abhängigkeitstyp nochmals verbessern. Für strukturelle Abhängigkeiten könnte ein höherer Grenzwert angenommen werden, als für Cost/Value Abhängigkeiten.

8.2.1 Laufzeitverhalten

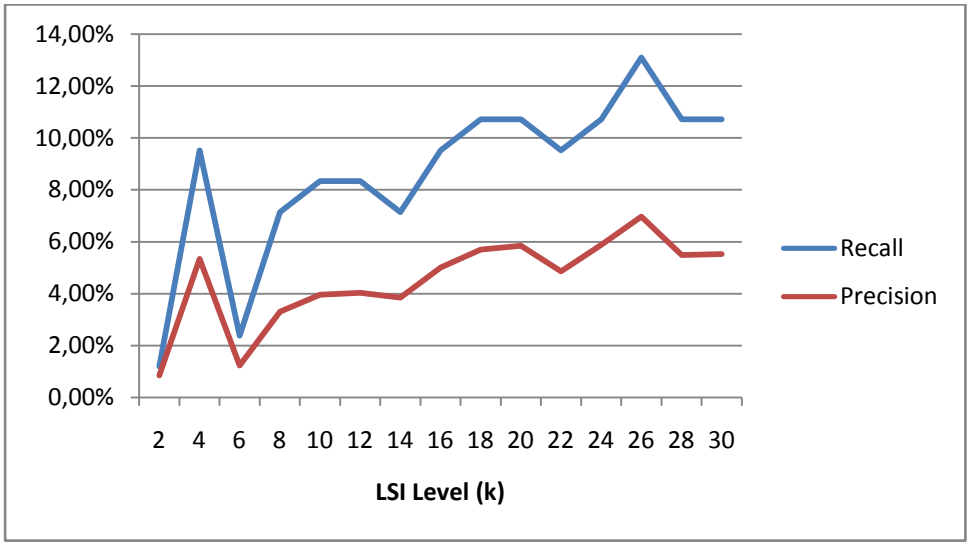
Aufgrund des einfacheren Algorithmus erreicht der Vector-Space-Model Algorithmus einen signifikante Geschwindigkeitsvorsprung gegenüber dem Latent-Semantic-Indexing Algorithmus. Die Laufzeit wird bei jedem Analysevorgang protokolliert und ermöglicht somit diese Auswertung. Die nachfolgende Grafik stellt das Laufzeitverhalten von 10 exemplarischen Analysevorgängen als Chart dar.



Grafik 5: Gegenüberstellung des Laufzeitverhaltes

Beim Latent-Semantic-Indexing Algorithmus hängt die Laufzeit auch von der gewählten Dimensionsreduktion ab.

8.2.2 Ermittlung eines optimalen Werts für die Dimensionsreduktion

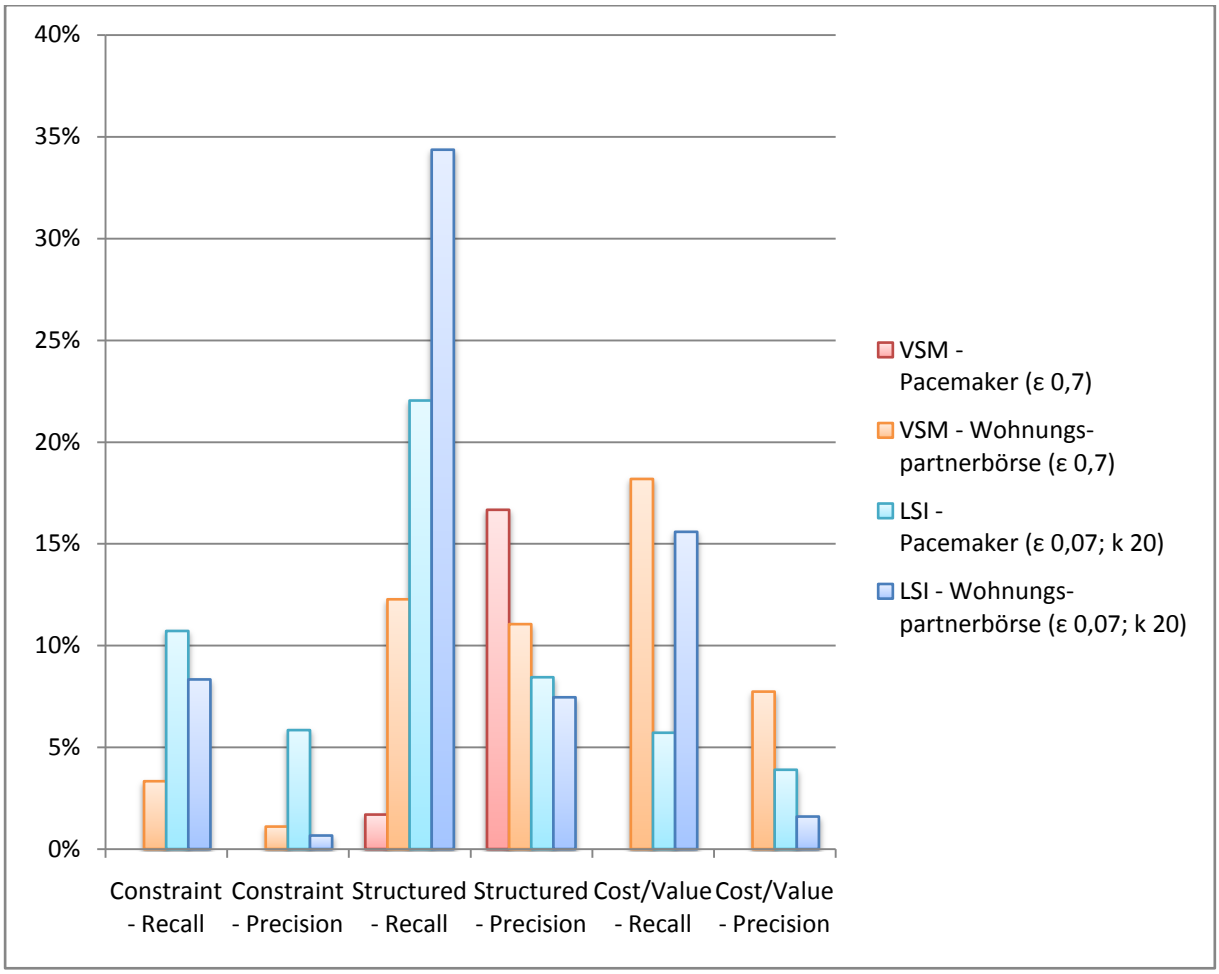


Grafik 6: Entwicklung des Recall und Precision mit verändertem k bei Dimensionsreduktion

Die Auswertung der Ergebnisse des LSI-Algorithmus zeigt, dass eine Erhöhung der Matrix Dimension nur geringe Auswirkung auf das Ergebnis hat. In allen Auswertungsläufen wird von einem LSI-Level (k) von 20 ausgegangen. Dieser Wert wird auch in der Literatur empfohlen [Berry95].

8.3 Vergleich der Ergebnisse

Im Vergleich beider Algorithmen schneiden beide Algorithmen ähnlich ab. Dieses Ergebnis ist auf die gewählten Fallstudien zurückzuführen. Alle Anforderungsbeispiele wurden mit einem einheitlichen Glossar verfasst. Die Wörter werden in gleichen Situationen immer gleich verwendet. Dadurch konnte der Latent-Semantic-Indexing-Algorithmus seinen Vorteil der semantischen Analyse nicht ausspielen.



Grafik 7: Vergleich der Algorithmen unter Berücksichtigung der Abhängigkeitstypen

Die Grafik 7 zeigt die finale Gegenüberstellung der Algorithmen und hebt nochmals die Abhängigkeitstypen hervor. Dabei zeigt sich, dass die strukturellen Abhängigkeiten am besten erkannt werden. Die Erkennungsrate ist bei den beschränkenden Abhängigkeiten und den kosten- und wertrelevanten Abhängigkeiten sehr gering.

In diesem Zusammenhang sind auch die Ergebnisse der publizierten wissenschaftlichen Arbeiten zu hinterfragen. Die Algorithmen wurden teilweise identisch oder ähnlich implementiert und dennoch versprechen die Artikel signifikant bessere Analysewerte. [Hayes03] erreichte je nach Optimierungsgrad Werte für Recall von 17,6% bis 85,4% und Precision von 5,3% bis 40,6%. [Natt05] erreichte Recall Werte von 51% bei einer Vorschlagsliste von 10 zugeordneten Beziehungen. Die einfachste Erklärung für dieses Phänomen ist das zusätzliche Verarbeiten von domainspezifischer Information. In einigen

Arbeiten werden Domain-Wörter als unteilbare Einheiten im Algorithmus verarbeitet und somit die Werte verbessert. Diese Maßnahme brachte in den untersuchten Beispielen nur wenig Verbesserung, da die Wörter schon durch das Glossar normiert wurden. Außerdem sind die Abhängigkeitstypen in den Publikationen meist struktureller Natur, da sehr oft Verfeinerungen der Anforderungen als Abhängigkeiten in die Analyse eingehen. Somit konnte die Arbeit zeigen, dass das Erkennen von beschränkenden Abhängigkeiten komplexer ist, als in der Literatur beschrieben.

Die Anforderungen aus der Fallstudie Wohnungspartnerbörse unterscheiden sich von den Anforderungen des Pacemakers in der Länge und im syntaktischen Aufbau. Beide Algorithmen konnten in beiden Fallstudien nur wenige Abhängigkeiten richtig erkennen. Trotzdem waren die Ergebnisse der Pacemaker Fallstudie besser. Der Grund dafür könnten die längeren Pacemaker Anforderungstexten sein. Information Retrieval Algorithmen arbeiten besser, je mehr Daten sie auswerten. Dabei wird der statistische Fehler immer geringer. Aus dieser These kann man ableiten, dass die automatische Erkennung der Abhängigkeiten bei längeren Anforderungstexten besser funktioniert als bei kurzen Sätzen.

9 Didaktische Möglichkeiten zur Verbesserung der Ergebnisse

In diesem Kapitel werden Maßnahmen zur Verbesserung der Ergebnisse diskutiert, die durch Schulungen oder prozessuale Umstellungen erzielt werden können. Prinzipiell verfolgen alle Maßnahmen ein gemeinsames Ziel: Die Verbesserung der Qualität der Anforderungen [Fabb98]. Dieses Ziel ist im Zusammenhang mit Softwareanforderungen sehr wichtig, da die Qualität der Anforderungen sehr starke Auswirkung auf die Qualität des finalen Softwareprodukts hat.

In den vorherigen Kapiteln dieser Arbeit wurde von Methoden zur Analyse von Anforderungen geschrieben. In der Fachliteratur wird dieses Vorgehen als Natural Language Processing (NLP) bezeichnet. Macias und Pulman haben NLP-Techniken auf die Erstellung von natürlichsprachigen Anforderungen angewandt [Pulman95]. Zusammengefasst ergeben sich daraus folgende Prinzipien für die Erstellung:

- Das verwendete Vokabular muss fixiert und unter den Stakeholdern abgestimmt sein.
- Ein gemeinsamer Schreibstil muss gefunden werden, sodass die Anforderungen leichter verstanden werden.
- Vieldeutigkeit der Aussagen muss vermieden werden. Wörter, Sätze und Inhalte müssen syntaktisch und semantisch eindeutig sein.
- Die Informationen, die durch die Anforderungen transportiert werden müssen klar und eindeutig sein.

Im Folgenden wird auf die einzelnen Punkte kurz eingegangen und mögliche Vorgehensweisen zur Erreichung dieser Prinzipien vorgestellt:

9.1 Gemeinsames Vokabular durch ein einheitliches Glossar

Die Verwendung eines gemeinsamen Glossars zum Verfassen der Anforderungen kann die Ergebnisse des Algorithmus erheblich verbessern. Alle Information-Retrieval-Algorithmen arbeiten mit der Analyse der verwendeten Wörter. Die Sätze werden in ihre Wörter aufgetrennt und diese werden untereinander verglichen. Ungeschulte Personen neigen zur Vermeidung von Wortwiederholungen und bewirken dadurch eine Verschlechterung der

Analyseergebnisse. Die gefundenen Wörter haben dann keinen starken semantischen Zusammenhang und werden nicht als identisch identifiziert. Abhängigkeitsinformation gehen somit verloren. Bei der Verwendung von normierter Sprache kann dieses Phänomen verringert werden.

Wichtige Aspekte bei der Verwendung eines einheitlichen Glossars sind die Veröffentlichung und die Schulung der Glossarinformationen. Jede Aktualisierung im Glossar muss an alle Stakeholder kommuniziert werden. Prinzipiell zählen alle am Softwareentwicklungsprozess beteiligten Personen zu den Stakeholdern. Als besonders wichtig gilt die Schulung der Anforderungsexperten und der Fachbereiche, da diese stark bei der Initialen Erstellung der Anforderungen beteiligt sind. In gemeinsamen Workshops müssen alle Beteiligten auf den gemeinsamen Wortschatz geschult werden. Wichtig ist ein gemeinsames Verständnis der semantischen Bedeutung von jedem Wort. Durch diese Vereinheitlichung und Normierung des Sprachgebrauchs verringern sich die Missverständnisse und es trägt zur Qualitätssteigerung der entwickelten Software bei.

Moderne Anforderungsmanagementsysteme ermöglichen die Definition von Glossaren und unterstützen die Anforderungsexperten bei der Verwendung dieser normierten Wörter. Eine Best-Practice in diesem Bereich ist die Definition von Wörtern die nicht verwendet werden dürfen. Auch die verbotenen Wörter werden als Glossar repräsentiert und bei Verwendung werden diese als „unerwünscht“ markiert.

9.2 Vereinheitlichung des Schreibstils

Der gemeinsame Schreibstil wurde in dieser Arbeit durch die Anwendung einer Anforderungsschablone erreicht. Dieses Vorgehen wurde bereits ausführlich in den vorherigen Kapiteln beschrieben. Im praktischen Einsatz erwiesen sich die Schulung und die Akzeptanz dieser Schablonenschreibweise als große Herausforderung. Die beteiligten Personen gaben als Rückmeldung an, die Schablonenschreibweise sei umständlich zu schreiben und zu lesen. Durch gezielte Vorteilsübersetzung war es schlussendlich möglich alle Beteiligten vom Einsatz dieser Vorgehensweise zu überzeugen. Diese Überzeugungsarbeit ist ein wichtiger Bestandteil der anfänglichen Schulungsphase. Da die Vorteile der Schablonenschreibweise für die einzelnen Personen nicht sofort sichtbar sind, muss eine klare Definition der Ziele gegeben werden.

9.3 Gleicher Wissensstand von allen Anforderungsexperten

In ihrer Arbeit “The Impact of Educational Background on the Effectiveness of Requirements Inspections: An Empirical Study” untersuchen Carver und seine Kollegen die Auswirkungen des Bildungsabschlusses auf den Review-Prozess von Anforderungen [Carver98]. Prinzipiell gelten diese Ergebnisse auch für den Erstellungsvorgang und dem Verstehen von Anforderungen. Mit der Arbeit wurde versucht die Annahme zu beweisen, dass der Ersteller und sein Wissensstand sehr großen Einfluss auf die Qualität der Anforderung haben.

Eine Erkenntnis aus der Studie war, dass Personen die Erfahrung im Verfassen von schriftlichen Anforderungen hatten, signifikant mehr Fehler fanden als Personen ohne solche Erfahrungen. Dieses Ergebnis belegt, dass man das Arbeiten mit Anforderung erlernen kann und zeigt wie wichtig eine entsprechende Ausbildung für Anforderungsexperten ist.

Eine interessante Beobachtung war, dass das Bildungsniveau zwischen Master- und PhD-Studenten keinen signifikanten Einfluss auf die Ergebnisse hatten. Die Ergebnisse zeigten ebenfalls, dass die Personen mit Abschlüssen in IT fremden Masterstudien (z.B.: Ingenieurwesen, Naturwissenschaft oder Wirtschaft) signifikant mehr Fehler in den Anforderungen fanden, als Inspektoren mit Abschlüssen in Datenverarbeiten Masterstudien (z.B.: Informatik, Software-Engineering oder Elektrotechnik).

Für das Verfassen von Anforderungen liegen keine entsprechenden Studien vor. Es kann aus den vorliegenden Ergebnissen geschlossen werden, dass die Erfahrung im Umgang mit Anforderungen auch beim Verfassen von Anforderungen den größten qualitativen Unterschied macht.

10 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war es Abhängigkeiten zwischen Anforderungen automatisch zu erkennen. Es wurden die Information Retrieval Algorithmen - Vector-Space-Model und Latent-Semantic-Indexing analysiert und in einem Requirement Traceability Werkzeug implementiert. Das Hauptaugenmerk bei der automatischen Erkennung der Abhängigkeiten zwischen den Anforderungen lag auf den Abhängigkeitstypen. Es wurde zwischen folgenden drei Abhängigkeitstypen unterschieden: strukturelle Abhängigkeiten, beschränkende Abhängigkeiten, und kosten- und wertrelevante Abhängigkeiten.

Insgesamt liegt die Erkennungsrate für beide Fallstudien auf einem enttäuschend niedrigen Niveau. Die Ergebnisse zeigen aber sehr klar, dass strukturelle Abhängigkeiten besser durch die automatisierten Algorithmen gefunden werden. Die in der Literatur von [Hayes03] und [Natt05] erzielten Ergebnisse geben sehr gute Werte für die automatische Erkennung von Anforderungsabhängigkeiten an. Diese Recall und Precision Werte sind jedoch kritisch zu hinterfragen, da es sich bei den Abhängigkeitstypen meist um strukturelle Abhängigkeiten handelt. Wie diese Arbeit zeigen konnte, ist das Erkennen von beschränkenden Abhängigkeiten oder kosten- und wertrelevanten Abhängigkeiten komplexer und erzielt keine praxistauglichen Ergebnisse. Information Retrieval Algorithmen, wie der Vector-Space-Model oder Latent-Semantic-Indexing Algorithmus, analysieren die Abhängigkeiten von Anforderungen auf Grund der gemeinsam verwendeten Wörter. Während bei strukturellen Abhängigkeiten die Anforderungen häufig auf den gleichen Wörtern basieren, haben beschränkende Abhängigkeiten keine syntaktische Repräsentation in den Anforderungstexten. Die beschränkenden Abhängigkeiten beruhen auf inhaltlichen Informationen, die ein Algorithmus ohne zusätzliche domainspezifische Information nicht akzeptabel auswerten kann. Auch der Versuch einen Teil der semantischen Information mit dem Latent-Semantic-Indexing Algorithmus auszuwerten brachte nicht die erhofften Ergebnisse. Gründe dafür sind die kurzen Anforderungstexte. Es wird angenommen, dass sich die Werte des Algorithmus mit wachsender Anforderungstextlänge verbessern.

Für einen praxistauglichen Einsatz dieses Verfahrens müsste man mehrere Ansätze miteinander kombinieren. Ein weiteres Forschungsfeld liegt in der Kombination der Information Retrieval Technik mit anderen Traceability Techniken (wie etwa

szenariobasierten Ansätzen [Egyed05]). Der Algorithmus würde zusätzlich zum Anforderungstext auch die Art der Anforderung auswerten. Abhängig vom Typ der Anforderung könnte man den Grenzwert ε dynamisch bestimmen. Somit erhöht man die Wahrscheinlichkeit einer Abhängigkeit zwischen zwei Anforderungen von einem bestimmten Typ und könnte die Genauigkeit des Algorithmus erhöhen.

Der abschließende Teil dieser Arbeit beschäftigte sich mit den didaktischen Möglichkeiten zur Verbesserung der erreichten Ergebnisse. Die einfachste Möglichkeit der Verbesserung ist das Verwenden eines gemeinsamen Vokabulars. Dadurch wird die Trefferquote der Algorithmen erhöht und die Qualität der Anforderungen steigt. Diese Maßnahme reduziert auftretende Missverständnisse durch die natürliche Sprache und erhöht somit die Qualität des Endprodukts. Weitere Verbesserungen werden durch intensives Schulen der beteiligten Personen erreicht. Ziel sollte es sein einen gemeinsamen Schreibstil zu finden und alle Beteiligten auf einen gleichen Wissenstand zu bringen.

11 Literatur

[Almei07] Almeida, Jo, a. P.; Iacob, M.-E. & Eck, P. (2007), 'Requirements traceability in model-driven development: Applying model and transformation conformance', *Information Systems Frontiers* **9**(4), 327--342.

[Anqu08] Anquetil, N.; Grammel, B.; da Silva, I. G. L.; Noppen, J.; Khan, S. S.; Arboleda, H.; Rashid, A. & Garcia, A. (2008), Traceability for Model Driven, Software Product Line Engineering, *in* 'ECMDA Traceability Workshop Proceedings', SINTEF, Norway, pp. 77--86.

[Asun10] Asuncion, H. U.; Asuncion, A. U. & Taylor, R. N. (2010), Software traceability with topic modeling, *in* 'ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering', ACM, New York, NY, USA, pp. 95--104.

[Berg02b] Bergman, M.; King, J. L. & Lyytinen, K. (2002), 'Large-scale requirements analysis as heterogeneous engineering', 357--386.

[Berry95] Berry, M. W.; Dumais, S. T. & O'Brien, G. W. (1995), 'Using linear algebra for intelligent information retrieval', *SIAM Rev.* **37**(4), 573--595.

[Caliber] Borland Software Corporation (a Micro Focus Company), 2010, 'CaliberRM - Enterprise Software Requirements Management System', www.borland.com/ caliber.

[Carver98] Carver, J. C.; Nagappan, N. & Page, A. (2008), 'The Impact of Educational Background on the Effectiveness of Requirements Inspections: An Empirical Study', *IEEE Transactions on Software Engineering* **34**, 800-812.

[Cock03] Cockburn, A. (2003), *Agile Software-Entwicklung. Die Prinzipien der agilen Software-Entwicklung dargestellt und erläutert.*, The MIT Press.

[Deer90] Deerwester, S. C.; Dumais, S. T.; Landauer, T. K.; Furnas, G. W. & Harshman, R. A. (1990), 'Indexing by Latent Semantic Analysis', *Journal of the American Society of Information Science* **41**(6), 391--407.

[Duan07] Duan, C. & Cleland-Huang, J. (2007), Clustering support for automated tracing, *in* 'ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering', ACM, New York, NY, USA, pp. 244--253.

[Eeles05] Eeles, P. (2005), 'Capturing Architectural Requirements', Website, http://www.ibm.com/developerworks/rational/library/4706.html.

[Egyed05] Egyed, A. & Grünbacher, P. (2005), 'Supporting Software Understanding with Automated Requirements Traceability', *International Journal of Software Engineering and Knowledge Engineering* **15**(5), 783-810.

[Evan04] Evans, E. (2004), *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley.

[Fabb98] Fabbrini, F.; Fusani, M.; Gervasi, V.; Gnesi, S. & Ruggieri, S. (1998), 'Achieving Quality in Natural Language Requirements'.

[Garc06] Garcia, E. (2006), 'SVD and LSI Tutorial 4: Latent Semantic Indexing (LSI) How-to Calculations', Website, http://www.miislita.com/information-retrieval-tutorial/svd-lsi-tutorial-4-lsi-how-to-calculations.html.

[Gotel94] Gotel, O. C. Z. & Finkelstein, A. C. W. (1994), An Analysis of the Requirements Traceability Problem, *in* , pp. 94--101.

[Gross04] Grossman, D. A. & Frieder, O. (2004), *Information Retrieval: Algorithms and Heuristics*, Springer, Dordrecht.

[Goetze08] Götze, M. (2008), 'Deutsche Stopwords', Website, http://solariz.de/649/deutsche-stopwords.htm.

[Hayes03] Hayes, J. H.; Dekhtyar, A. & Osborne, J. (2003), Improving Requirements Tracing via Information Retrieval, *in* 'RE '03: Proceedings of the 11th IEEE International Conference on Requirements Engineering', IEEE Computer Society, Washington, DC, USA, pp. 138.

[Heyes06] Hayes, J. H.; Dekhtyar, A. & Sundaram, S. K. (2006), 'Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods', *IEEE Trans. Softw. Eng.* **32**(1), 4--19.

[Hayes08] Hayes, J. H.; Antoniol, G. & Guéhéneuc, Y.-G. (2008), 'PREREQIR: Recovering Pre-Requirements via Cluster Analysis', *Reverse Engineering, Working Conference on* **0**, 165-174.

[IEEE1998] (1998), 'IEEE Recommended Practice for Software Requirements Specifications', *IEEE Std 830-1998* , i.

[Jama05] Mathworks and the National Institute of Standards & Technology (2005), 'Jama: A Java matrix package', Website, http://math.nist.gov/javanumerics/jama/.

[John08] Johnson, R.; Roussos, G. & Vetti-Tagliati, L. (2008), 'Requirements analysis for large scale systems', *J. Object Tech* **Vol.7**(8).

[Jones98] Jones, C. (1998), 'Estimating Software Costs'

[Kruch03] Kruchten, P. (2003), *The Rational Unified Process*, Addision-Wesley Longmann, Amsterdam.

[Lucia08] De Lucia, A.; Oliveto, R. & Tortora, G. (2008), IR-Based Traceability Recovery Processes: An Empirical Comparison of "One-Shot" and Incremental Processes, *in* 'ASE '08: Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering', IEEE Computer Society, Washington, DC, USA, pp. 39--48.

[Pulman95] Macias, B. & Pulman, S. G. (1995), 'A Method for Controlling the Production of Specifications in Natural Language', *Comput. J.* **38**(4), 310-318.

[Marc03] Marcus, A. & Maletic, J. I. (2003), Recovering documentation-to-source-code traceability links using latent semantic indexing, *in* 'ICSE '03: Proceedings of the 25th International Conference on Software Engineering', IEEE Computer Society, Washington, DC, USA, pp. 125--135.

[Natt05] Natt och Dag, J.; Gervasi, V.; Brinkkemper, S. & Regnell, B. (2005), 'A Linguistic-Engineering Approach to Large-Scale Requirements Management', *IEEE Software* **22**, 32-39.

[Pinh04] Pinheiro, F. A. C. (2004), *Perspectives on Software Requirements (Kluwer Internation Series in Engineering and Computer Science*, Kluwer Academic Publishers, chapter 5, pp. 91-115.

[Phol09] Pohl, K. & Rupp, C. (2009), *Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level*, Vol. 1, dpunkt.

[Porter80] Porter, M. F. (1980), 'An Algorithm for Suffix Stripping', *Program* **14**(3), 130--137.

[Rame01] Ramesh, B. & Jarke, M. (2001), 'Toward Reference Models for Requirements Traceability', *IEEE Trans. Softw. Eng.* **27**(1), 58--93.

[Rupp09] Rupp, C. (2009), *Requirements - Engineering und -Management*, Vol. 5, Carl Hanser Verlag München Wien.

[Salt75] Salton, G.; Wong, A. & Yang, C. S. (1975), 'A vector space model for automatic indexing', *Commun. ACM* **18**(11), 613--620.

[Savo06] Savoy, J. (2006), Light stemming approaches for the French, Portuguese, German and Hungarian languages, *in* 'SAC '06: Proceedings of the 2006 ACM symposium on Applied computing', ACM, New York, NY, USA, pp. 1031--1035.

[Stach73] Stachowiak, H. (1973), *Allgemeine Modelltheorie*, Springer.

[VMXT] 'V-Modell XT', Bundesministerium des Innern, Technical report, IT-Beauftragte der deutschen Bundesregierung für Informationstechnik, http://www.cio.bund.de/cln_102/DE/IT-Methoden/V-Modell_XT/v-modell_xt_node.html.

12 Anhang - Requirement Traceability Werkzeug

Dieses Kapitel beschreibt den Aufbau und die Nutzung des Requirement Traceability Werkzeugs. Es basiert auf dem OSGi Standard und wurde als Eclipse-Plugin für die Zielplatform Eclipse 3.1 bis 3.3 implementiert. Es war nicht möglich eine aktuelle Eclipse-Plattform zu verwenden, da die Programmierschnittstelle von CaliberRM nur für diese Plattform verfügbar war. Aus diesen Tatsachen ergeben sich Plugin-Abhängigkeiten zum Eclipse-Framework und zum Borland CaliberRM SDK.

12.1.1 OSGI - Standard

Das OSGi-Framework ist eine modulare Service-Plattform für die Java-Programmiersprache. Dabei werden die einzelnen Komponenten als Bundle bezeichnet. Diese Bundle stellen ihre Schnittstelle als Service zur Verfügung. Das Eclipse Framework wurde nach diesem OSGi Standard aufgebaut. Alle Eclipse-Plugins implementieren diesen OSGi Standard. Aus diesem Grund befinden sich unter den Abhängigkeiten des Werkzeugs auch OSGi Bundles. Um den OSGi-Standard zu erfüllen, muss eine Activator Basisklasse implementiert werden. In dieser Klasse werden Routinen für den Start und das Ende des Bundles implementiert.

Für diese Arbeit wurde ein Eclipse-Plugin implementiert. Zusätzlich zu den OSGi-Dateien benötigt man die Plugin.xml Datei. In dieser Datei werden die Eclipse-Plugin spezifischen Informationen gespeichert. Die Activator Klasse wird zwar formal für das Bundle benötigt, die eigentliche Verarbeitungslogik wird aber als Extension in das Eclipse-Plugin eingebunden. Für das Requirement Traceability Werkzeug werden eine eigene Perspective, eine View, eine Property-Page und ein Editor als Extension definiert.

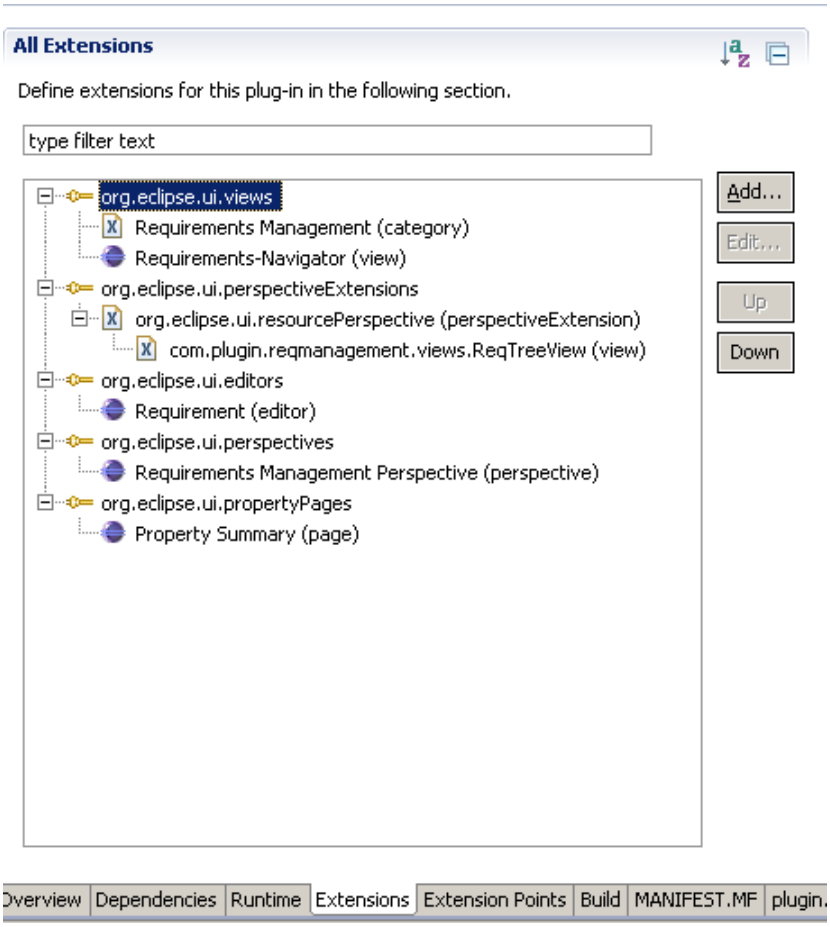


Abbildung 16: Übersicht der eingebundenen Erweiterungen für das Eclipse Plugin

Für die übersichtliche Darstellung der Anforderungen wurde eine eigene Analyse-Perspective implementiert. Eine Perspektive in Eclipse ist eine benannte und gespeicherte Organisation von Ansichten (Views), Menüs und Symbolleisten. Das Standard-Eclipse hat verschiedene Perspectives vorkonfiguriert. Diese dienen zum Java-Programmieren, für die Plugin-Entwicklung oder für die Versionierung. Die Analyse-Perspective öffnet am rechten Rand die baumartige Anforderungsdarstellung, in der Mitte öffnet sich der Anforderungseditor und am rechten unteren Rand befinden sich die Anforderungseigenschaften.

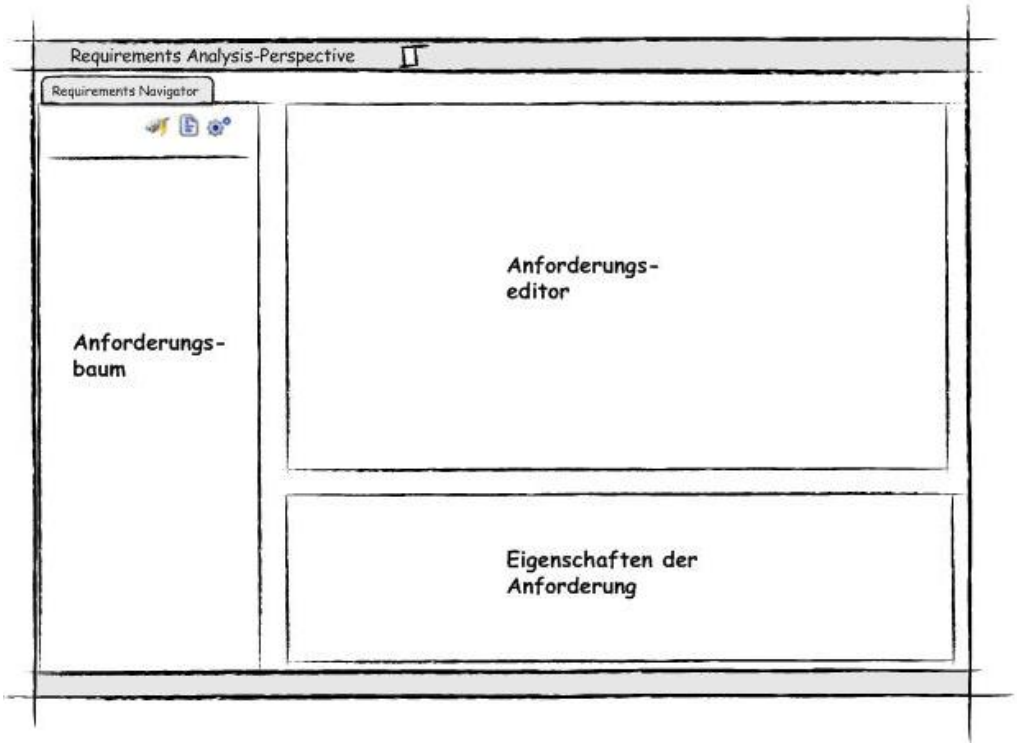


Abbildung 17: Prototyp des Requirement Traceability Werkzeugs

Die initiale Benutzerinteraktion erfolgt mit den Symbolen die über dem Anforderungsbaum angeordnet sind. Diese Symbole stellen die drei wichtigsten Aktionen dar:

 Verbindung zum Server herstellen.

 Konfiguration der Serververbindung und Auswahl des Projektes für die Analyse

 Start der Analyse

Wird das Werkzeug zum ersten Mal gestartet muss eine Serververbindung konfiguriert werden. Wenn keine Serververbindung besteht wird anstatt des Anforderungsbaums der Satz: „No Server Connection!“ angezeigt. Durch Auswahl des Konfigurationssymbols wird der Serververbindungs-Assistent gestartet.

12.1.2 Serververbindungs-Assistent

Dieser Assistent dient zur Konfiguration der Servereinstellungen. Die Eingaben werden permanent in einer Datei auf der Festplatte gespeichert. Somit bleiben die Einstellungen auch

nach einem Neustart des Programms erhalten. Die erste Seite des Assistenten dient der Eingabe der Serverdaten. Es werden ein Verbindungsname, die Internetadresse und der Port des Servers eingegeben. Für die Authentifizierung am Borland CaliberRM Server müssen Benutzername und Passwort eingegeben werden.

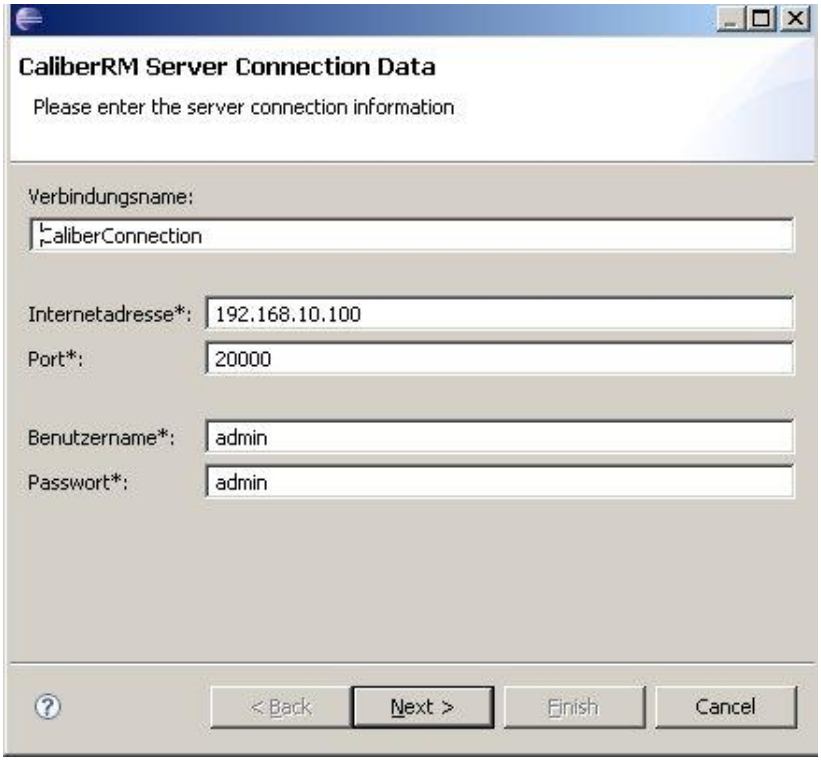


Abbildung 18: Serververbindungs-Assistent - Seite zur Eingabe der Server- und Benutzerdaten

Der Assistent erstellt beim Betätigen der „Next“-Schaltfläche eine Verbindung zum Server und ruft alle verfügbaren Projekte ab. Diese werden in einer einfachen Listendarstellung angezeigt. Der Benutzer muss ein Projekt für die Analyse auswählen. Für diese Arbeit und die Auswertung der Ergebnisse wurden verschiedene Projekte mit den gleichen Anforderungen angelegt. Je nach Abhängigkeitstyp wurde das Projekt anders benannt. Im folgenden Screenshot sind die letzten drei Projekte mit einem Schlüsselwort für den Abhängigkeitstyp dargestellt.

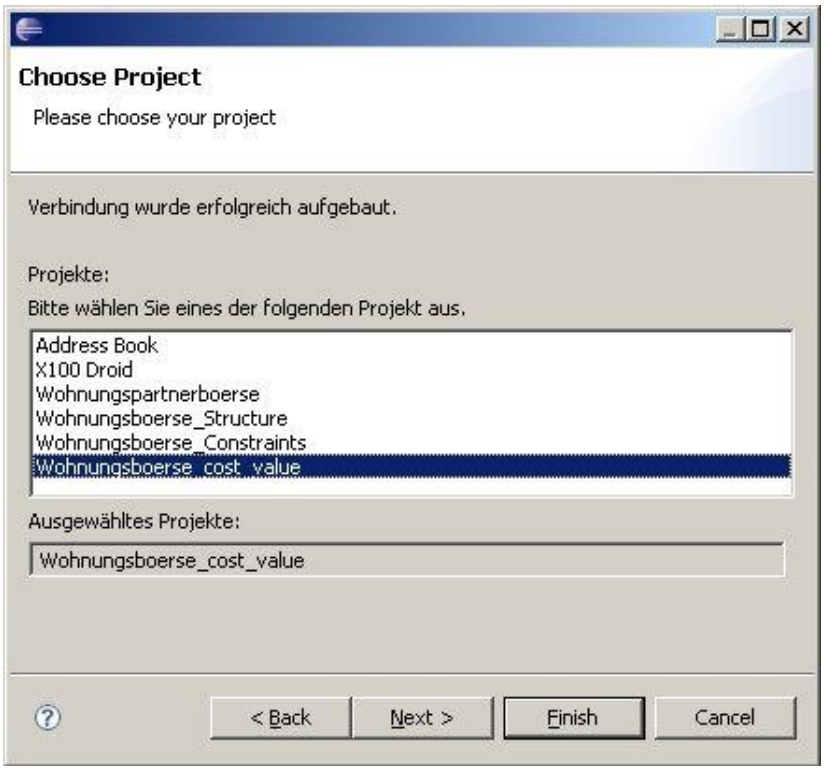


Abbildung 19: Serververbindungs-Assistent - Auswahl des Projektes

Die Anforderungen der Projekte könnten verschiedene Baselines besitzen. Auch diese werden durch den Assistenten vom Server abgerufen und zur Auswahl dargestellt. Sobald der Benutzer eine Baseline auswählt und die „Finish“-Schaltfläche betätigt, werden die Eingaben gespeichert und der Assistent wird beendet. Nun sind alle getroffenen Einstellungen im Werkzeug gespeichert.

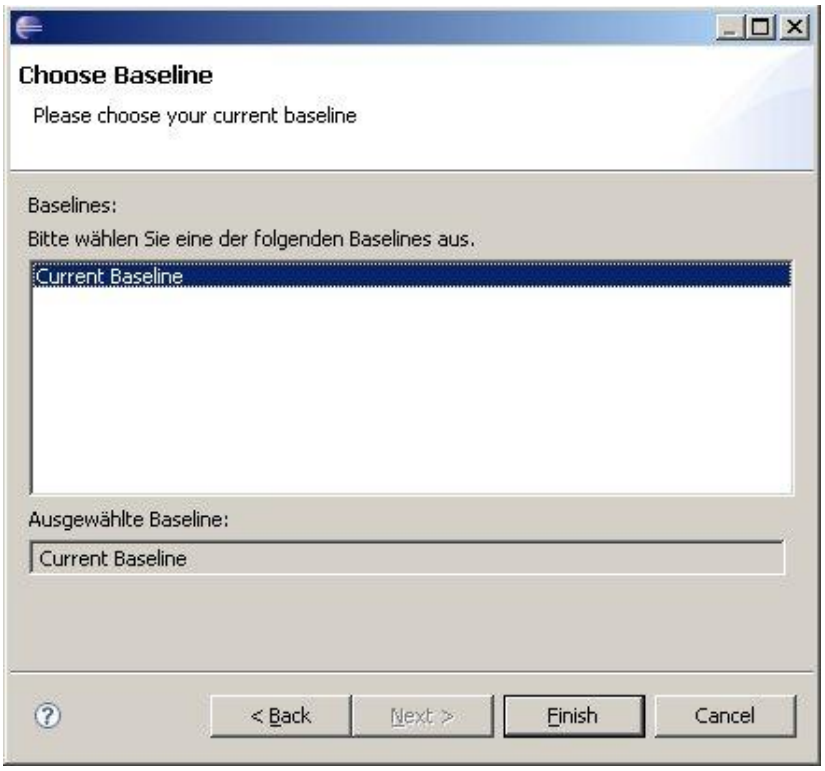


Abbildung 20: Serververbindungs-Assistent - Auswahl der gewünschten Baseline

Nach Beendigung des Assistenten wird keine Serververbindung erstellt. Erst durch Betätigen des „Verbinden“-Symbols kann der Benutzer eine Verbindung zum Server erstellen. Der Client baut die Verbindung zum Server auf und ruft den Anforderungsbaum auf. Der Anforderungsbaum wird in einer eigenen View dargestellt. Alle wichtigen Fensterbereiche im Eclipse-Framework werden als Views (Ansichten) implementiert.

12.1.3 Anforderungsdarstellung

Nach dem Abrufen werden die Anforderungen auf der ersten Ebene dargestellt. Sobald der Benutzer das Plus-Symbol anwählt wird die nächste Ebene von Anforderungen nachgeladen. Mit dieser Maßnahme wird die Last am Anforderungsserver minimal gehalten. Anforderungen die weiteren Anforderungen beinhalten werden als Ordner dargestellt.

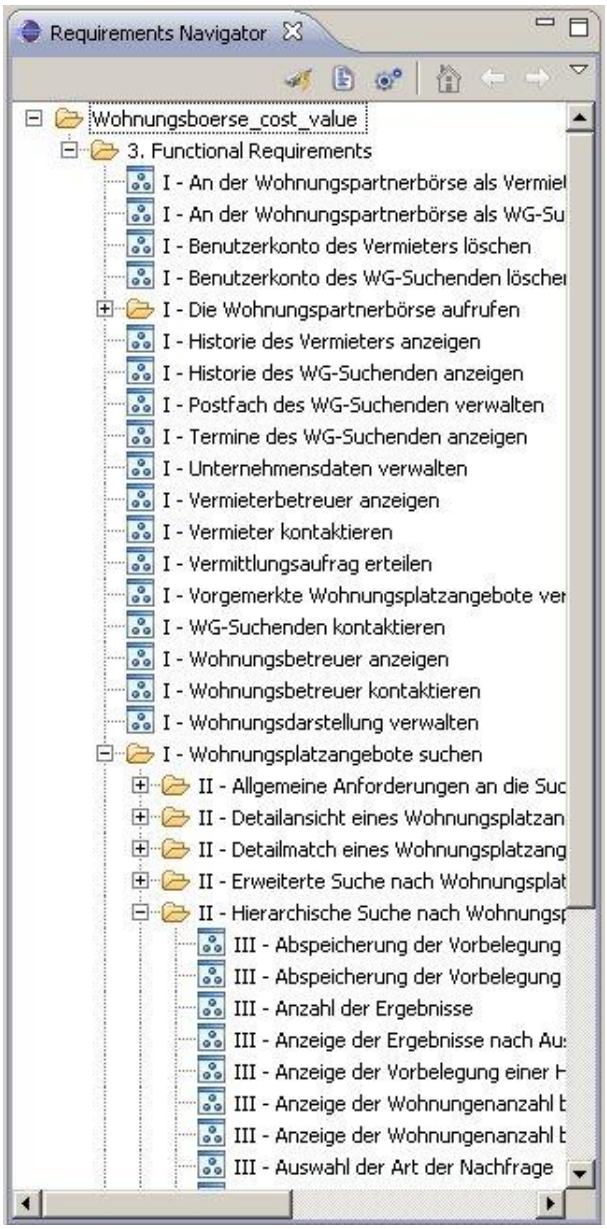


Abbildung 21: Anforderungsdarstellung - Die Anforderungen werden in einer Baumstruktur dargestellt

Für die Baumdarstellung werden die Anforderungen mit einer Wrapper-Klasse implementiert. Diese übernimmt die Darstellung als Baum und definiert die Aktionen bei einem Mausklick. Durch die Instanzierung der vordefinierten Baum-Klassen bleibt im Hintergrund der eigentliche Anforderungsbaum bestehen. Alle Funktionen des Werkzeugs können somit auf einen Anforderungsbaum zugreifen. Bei Doppelklick auf eine Anforderung öffnet sich diese in einem eigenen Editor in der Bildschirmmitte.



Abbildung 22: Anforderungs-Editor - Zur Bearbeitung der Anforderungen

Die Darstellung der Anforderung zeigt den Namen und die Beschreibung der Anforderungen. Beide Attribute werden auch im Analyse-Algorithmus verwendet. Die restlichen Attribute werden als Properties (Eigenschaften) am unteren Bildschirmrand angezeigt. Der nachfolgende Screenshot zeigt die Baumdarstellung, einen geöffneten Editor und die Attribute der Anforderung.

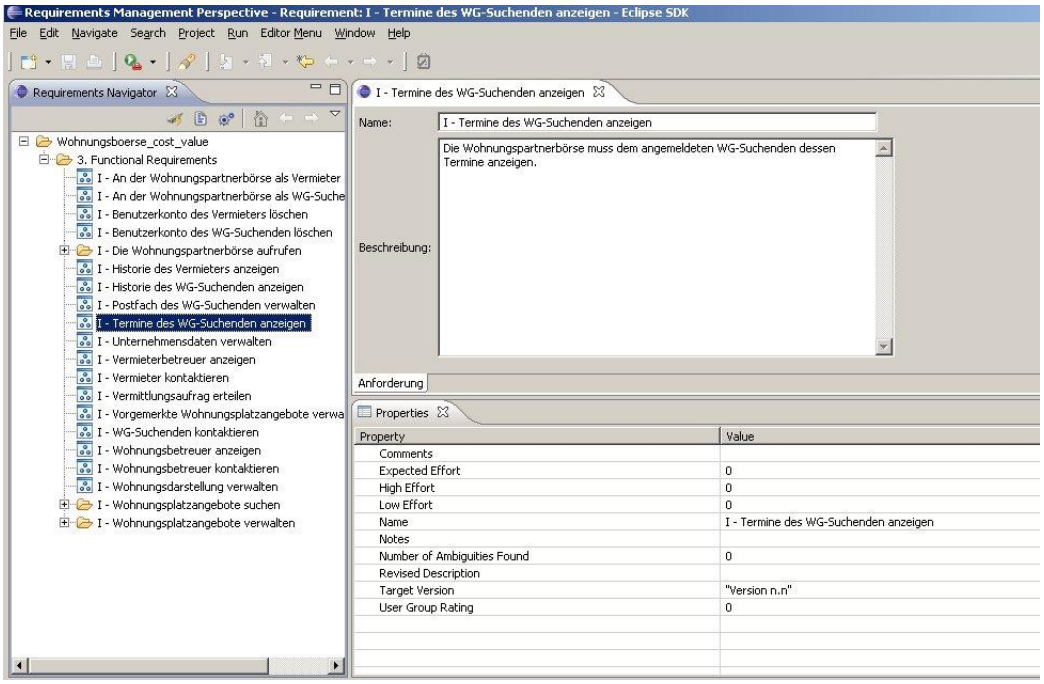


Abbildung 23: Client des Requirement Traceability Werkzeugs

12.1.4 Analyse-Assistent

Der Analyse-Assistent unterstützt den Benutzer bei der Ausführung der Analyse-Algorithmen. Er ermöglicht die Auswahl des Algorithmus und zeigt den Status mit einer Fortschrittsanzeige an.

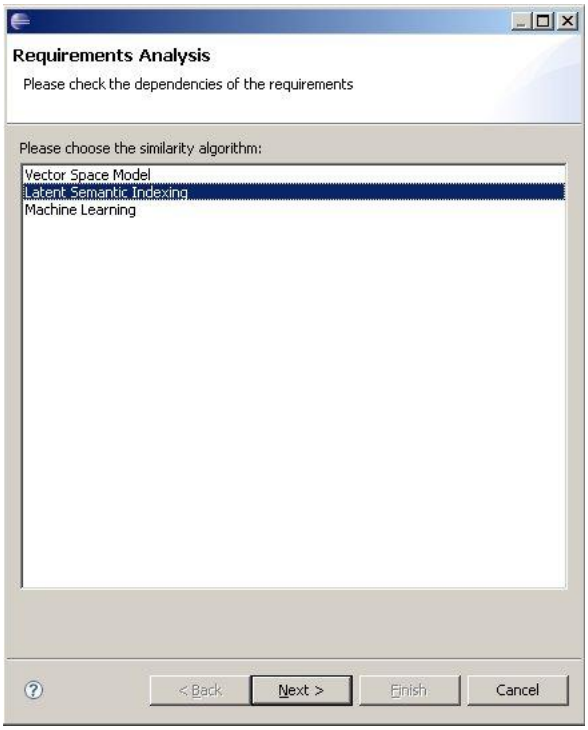


Abbildung 24: Analyse-Assistent - Auswahl des Algorithmus

Die erste Seite des Assistenten zeigt alle implementierten Analyse-Algorithmen in einer Auswahlliste dar. Der Benutzer muss einen Algorithmus wählen und kann dann auf die Übersichtsseite navigieren.

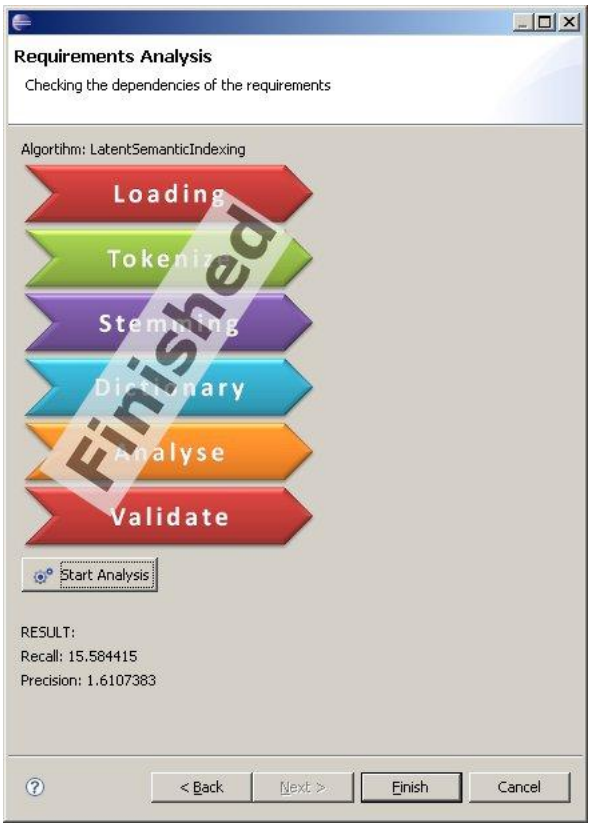


Abbildung 25: Analyse-Assistent - Ausführungsseite des Algorithmus

Der ausgewählte Algorithmus wird am Seitenanfang nochmals zur Kontrolle angezeigt. Die farbigen Balken zeigen den Fortschritt des Algorithmus an. Je nach Phase wird der Balken hervorgehoben. Die Analyse wird über die Schaltfläche „Start Analysis“ gestartet. Nach erfolgreicher Analyse erscheint der Schriftzug „Finished“ über den Balken und unter „Result“ stehen die Werte für Recall und Precision. Der Benutzer kann den Assistenten über die „Finish“ Schaltfläche beenden.

12.1.5 Modi des Werkzeugs

Für die erfolgreiche Auswertung der Algorithmen werden einige Parameter angepasst. Diese Parameter können nicht über die Oberfläche gesteuert werden, sondern müssen in einer eigenen Konfigurationsdatei angepasst werden. Diese wird in den Code compiliert und bei erneutem Ausführen des Werkzeugs berücksichtigt.

Folgende wichtige Parameter können eingestellt werden:

DEBUG	Damit kann das Werkzeug auf Fehler untersucht werden und die Zwischenergebnisse werden auf der Konsole ausgegeben.
WRITE_TO_FILE	Die detaillierten Ergebnisse werden zusätzlich in eine Requirements Traceability Matrix geschrieben.
SUBSTITUE_SIM_WORDS	Mit diesem Parameter kann die Wörtersubstitution dazu geschalten werden.
LSI_DIMENSION_k	Die gewünschte Dimensionsreduktion für den LSI-Algorithmus.
LSI_DEPENDENCY_LIMIT	Der LSI Schwellenwert für eine tatsächliche Abhängigkeit zwischen den Anforderungen. Dieser Wert wird beim LSI-Algorithmus standardmäßig auf 0,07 gestellt.
VSM_DEPENDENCY_LIMIT	Der VSM Schwellenwert für eine tatsächliche Abhängigkeit zwischen den Anforderungen. Dieser Wert wird beim VSM-Algorithmus standardmäßig auf 0.7 gestellt.
TESTSET	Hier wird das Testprojekt mit den bereits eingetragenen Abhängigkeiten konfiguriert.
LANGUAGE	Die Spracheinstellungen für die Fallstudie.

12.1.6 Ausgabe der Informationen

Die wichtigsten Informationen können über die grafische Oberfläche eingestellt und ausgegeben werden. Für die detailierte Analyse werden die Informationen auch über die Konsole ausgegeben. Zusätzlich erfolgt der Export über kommaseparierte Dateien. Beim LSI-Algorithmus können alle Matrizen ins Dateisystem geschrieben werden. Dieser Parameter ist standardmäßig nicht aktiviert, da sich sonst die Laufzeit signifikant erhöhen würde.

Die wichtigste Ausgabedatei ist die Requirements Traceability Matrix. Hier werden die berechneten Wahrscheinlichkeitswerte für die Abhängigkeiten zwischen den Anforderungen ausgegeben.