Entwicklung und Umsetzung eines Entscheidungsunterstützungssystems für das Outsourcing in der komponentenbasierten Softwareentwicklung

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

vorgelegt
von
Tommi Dirk Kramer
aus Heidelberg

Dekan: Prof. Dr. Dieter Truxius
Referent: Prof. Dr. Armin Heinzl
Korreferent: Prof. Dr. Christian Becker

Tag der mündlichen Prüfung: 17. Mai 2016

Danksagung

Unzählige Arbeitsstunden, große Sorgfalt und viel Fleiß waren für den Erstellungsprozess meiner Dissertation notwendig. Ohne einen großen Unterstützerkreis wäre dies während meiner Promotion kaum möglich gewesen. Aus diesem Grund möchte ich mich an an dieser Stelle bei all denjenigen bedanken, die mich auf diesem anstrengenden Weg begleitet und zum Gelingen dieser Arbeit beigetragen haben.

Zuerst möchte ich mich bei meinem Doktorvater und Erstgutachter Professor Dr. Armin Heinzl für sein entgegengebrachtes Vertrauen und die uneingeschränkte Unterstützung während der Zeit meiner Promotion bedanken. Er hat mir an seinem Lehrstuhl sämtliche Freiheiten für meine thematische und persönliche Entfaltung zugestanden. Außerdem hat er es mir ermöglicht, Erfahrungen beim Aufbau von Forschungs- und Entwicklungsinfrastrukturen sowie im Lehrbetrieb zu sammeln. Zudem gestattete er mir die Vorstellung meiner Forschungsergebnisse auf zahlreichen Konferenzen, Workshops und Kolloquien sowie einen Forschungsaufenthalt in Australien, was ich sehr zu schätzen wusste.

Zusätzlich möchte ich mich bei den weiteren Mitgliedern des Prüfungskomitees Professor Dr. Dieter Truxius und Professor Dr. Christian Becker bedanken. Sie haben mein Promotionsvorhaben in ihrer jeweiligen Rolle als Prüfungsausschussvorsitzender und Korreferent wesentlich unterstützt. Der Austausch mit ihnen war stets konstruktiv und vertrauensvoll.

Für die Durchführung meiner Forschung war ein regelmäßiger Austausch mit Industriepartnern erforderlich. Durch sie konnte ich praxisrelevante Erkenntnisse in meine Arbeit einfließen lassen. Für die Validierung der Ergebnisse waren sie zwingend erforderlich. So war es mir nicht nur möglich die theoretische Konzeption aus dem Geschäftsumfeld herauszuarbeiten, sondern auch den Praxisbezug in meiner Doktorarbeit herzustellen. Viele dieser Unternehmen haben mich weiterhin beim Aufbau neuer Lehrinfrastrukturen unterstützt. Für all das möchte ich

mich bei den beteiligten Unternehmen und den jeweiligen Mitarbeitern außerordentlich bedanken.

Darüber hinaus möchte ich mich für eine hervorragende und ereignisreiche Zeit am Lehrstuhl ganz besonders bei meinen aktuellen und ehemaligen Kollegen Jens Arndt, Okan Aydingül, Saskia Bick, Thomas Butter, Jens Dibbern, Jens Förderer, Michael Geisser, Erik Hemmer, Tobias Hildenbrand, der mir den Weg in die Promotion aufgezeigt hat, Simon Hubert, Lars Klimpke, Miroslav Lazic, Thomas Kude, Nele Lüker, Marko Nöhren, Tillmann Neben, der stets ein zuverlässiger Büronachbar und Diskussionspartner war, Alexander Scheerer, Sven Scheibmayr, Anna-Maria Seeger, Jessica Slamka, Kai Spohrer, Christoph Schmidt, Sebastian Stuckenberg und Aliona von der Trenck bedanken. Weiterhin gilt mein Dank Luise Bühler, Ingrid Distelrath, Ronja Ebner, Stefan Eckhardt, Nina Jäger, Alexandra Lang, Melanie Marnet, Lea Offermann, Olga Oster und Kathrin Teupe für die administrativen Angelegenheiten am Lehrstuhl. Außerdem möchte ich mich bei meinen studentischen Hilfskräften Pascal Kunz, Martin Pfannemüller, Viktor Schulz, Maximilian Wich und natürlich Lorenz Törmer, der maßgeblich zur Entstehung dieser Dissertation beigetragen hat, für ihre Unterstützung danken.

Schließlich möchte ich meine Familie, Verwandten und Freunde erwähnen. Ganz besonderer Dank gilt hierbei meinen Eltern Annette und Arno sowie meinen Brüdern Jochen und Fabian. Sie sind mir in allen Belangen bezüglich meiner Dissertation mit Rat und Tat zur Seite gestanden. Außerdem war es nur mit ihrem Rückhalt und Einsatz möglich, während der Promotion unser Haus zu bauen. Weiterhin möchte ich meiner Familie, meinen Verwandten und meinen Freunden für ihren Rückhalt, ihre konstruktiven Gespräche sowie die zahlreichen Unterhaltungsaktivitäten, die im Wesentlichen zum Freizeitausgleich beigetragen haben, danken.

Forst, den 23. Mai 2016

Inhaltsverzeichnis

Abbildungsverzeichnis **xi**

Tabellenverzeichnis **xiii**

Abkürzungsverzeichnis **xv**

1. Einleitung **1**

1.1. Motivation 1

1.2. Forschungsschwerpunkt und Zielsetzung 5

1.3. Methodische Vorgehensweise und Struktur der Arbeit 6

2. Theoretische und konzeptionelle Grundlagen **11**

2.1. Grundlagen der Entscheidungsfindung 11

2.1.1. Die Entscheidung 12

2.1.2. Entscheidungsprobleme 13

2.1.3. Entscheidungsmodelle und deren Logik 17

2.1.3.1. Entscheidungsfeld 18

2.1.3.2. Zielfunktion und Zielsystem 19

2.1.4. Vorgehensweise zur Entscheidungsfindung 20

2.2. Entscheidungsunterstützungssysteme 21

2.2.1. Typen von Entscheidungsunterstützungssystemen 24

2.2.1.1. Datenzentrische Entscheidungsunterstützung . . . 25

2.2.1.2. Kollaborative Entscheidungssysteme 27

2.2.1.3. Entscheidungsunterstützung für das Outsourcing 28

2.2.2. Mobile Entscheidungsunterstützungssysteme 33

2.2.2.1. Endgeräte für den mobilen Einsatz 33

2.2.2.2. Unterschiede zwischen mobilen Endgeräten 36

2.2.2.3. Mobile Geschäftsanwendungen 37

2.2.2.4. Einsatz mobiler Entscheidungsunterstützungssysteme . . . 39
2.3. Komponentenbasierte Entwicklung von Anwendungssoftware . . . 41
2.3.1. Grundlagen der Softwareentwicklung . . . 42
2.3.2. Der Softwarelebenszyklus . . . 43
2.3.2.1. Anforderungserhebung und -analyse . . . 43
2.3.2.2. Design des Softwaresystems . . . 44
2.3.2.3. Implementierung und Komponententests . . . 46
2.3.2.4. Integrationstests . . . 47
2.3.2.5. Betrieb und Wartung . . . 48
2.3.3. Prozessmodelle der Softwareentwicklung . . . 48
2.3.3.1. Wasserfallmodell . . . 49
2.3.3.2. Komponentenbasierte Entwicklung . . . 50
2.3.3.3. Agile Prozessmodelle . . . 51
2.3.4. Verteilte Softwareentwicklung . . . 53
2.3.4.1. Nachvollziehbarkeit und Begründungsmanagement 55
2.3.4.2. Werkzeuge für die Kollaboration . . . 56
2.4. Eigenerstellung oder Fremdbezug von Softwarekomponenten . . . 58
2.4.1. Genese des IT-Outsourcings . . . 58
2.4.1.1. Beweggründe für das Outsourcing . . . 62
2.4.1.2. Herausforderungen und Risiken des Outsourcings 66
2.4.2. Die Outsourcingentscheidung . . . 70
2.4.3. Outsourcingentscheidungen im Softwareentwicklungsprozess 72
2.4.3.1. Integration der Outsourcingentscheidung in den Softwareentwicklungsprozess . . . 75
2.4.3.2. Relevanz technischer Eigenschaften des Softwareprodukts für die Outsourcingentscheidung . . . 78
2.5. Zusammenfassung . . . 80

3. Konzeption und Implementierung 83
3.1. Die zu unterstützende Outsourcingentscheidung und ihre Anforderungen . . . 84
3.2. Konzeption des Entscheidungsmodells . . . 91
3.2.1. Entscheidungsgrößen und Zieldimension . . . 92
3.2.1.1. Strukturelle Eigenschaften . . . 93

3.2.1.2. Prozessspezifische Merkmale der Entwicklung . . 96
3.2.1.3. Wissensspezifische Merkmale 99
3.2.2. Konzeptionelle Überlegungen zur Teilautomation der Entscheidungsfindung bei einer anforderungszentrierten Vorgehensweise 102
3.2.2.1. Relevante Grundlagen der Anforderungserhebung 103
3.2.2.2. Relevante Grundlagen der Graphentheorie 104
3.2.2.3. Vorarbeiten zur Verknüpfung der Anforderungserhebung mit den Möglichkeiten der Graphentheorie 105
3.2.2.4. SODA – eine Entscheidungsunterstützungstechnik 108
3.2.3. Entscheidungslogik 114
3.2.3.1. Entscheidungstabellen als einzusetzendes Verfahren für die Aggregate 118
3.2.3.2. Gewichtungsfunktion der Aggregate 119
3.2.3.3. Amalgamation der Teilnutzenwerte 120
3.2.4. Anwendung des Entscheidungsmodells 121
3.2.4.1. Anwendung der Entscheidungstabellen 121
3.2.4.2. Aggregation der Entscheidungsgrößen und Entscheidungsfindung 125
3.2.4.3. Anwendung des Entscheidungsmodells an einem Beispiel 128
3.3. Implementierung des Modells auf einem Tablet-PC 133
3.3.1. Technologischer Aufbau und Entwicklungsumgebung . . . 134
3.3.1.1. Gesamtarchitektur 134
3.3.1.2. Technologieauswahl 136
3.3.1.3. Entwicklungsumgebung 140
3.3.2. Funktionsbeschreibung 144
3.3.2.1. Projektauswahl 145
3.3.2.2. Bewertungsvorgang 146
3.3.2.3. Ergebnisverarbeitung 149
3.3.2.4. Gewichtung und Berechnung 152
3.3.2.5. Teilautomatisierung der Merkmalsbewertung . . . 154
3.4. Zusammenfassung 156

4. Evaluation des Systems zur Entscheidungsunterstützung 161
4.1. Bedeutung der Evaluation 162
4.2. Evaluationsdesign 166
4.2.1. Theoriezentrisches Bewertungsmodell für das neu entwickelte Artefakt 167
4.2.2. Messgrößen 170
4.2.3. Evaluationsstrategie und Instrumente 174
4.2.3.1. Künstliche ex post Evaluation: Studentenexperiment 177
4.2.3.2. Naturalistische ex post Evaluation: Expertenbefragung 179
4.3. Ergebnisse 181
4.3.1. Auswertung der deskriptiven Statistiken 181
4.3.2. Qualitative Auswertung 189
4.3.2.1. Qualitative Auswertung der Studentenexperimente 189
4.3.2.2. Qualitative Auswertung der Expertenbefragungen 192
4.4. Zusammenfassung 195

5. Zusammenfassung und Ausblick 197
5.1. Forschungsbeitrag 197
5.2. Limitationen und Ausblick 200

A. Entscheidungstabelle für die strukturellen Eigenschaften einer Komponente in Bezug auf das Softwareprodukt 205

B. Entscheidungstabelle für die Prozesssmerkmale der Entwicklung von Softwarekomponenten 207

C. Entscheidungstabelle für die Wissensmerkmale von Softwarekomponenten 213

D. Entscheidungstabelle für das Outsourcingpotenzial 219

E. Beispielcode der mobilen App 221

F. Beispielcode des Proxyservers 227

G. Fragebogen für die künstliche und naturalistische Evaluation des Artefakts **229**

H. Interviewleitfaden für die qualitative Evaluation **233**

I. Ergebnistabellen der deskriptiven Evaluation **237**

Literaturverzeichnis **241**

Abbildungsverzeichnis

1.1. Die Forschungszyklen in der *Design Science* 8

2.1. Bestandteile eines Entscheidungsmodells 18
2.2. Entscheidungsprozess von EUS 23
2.3. *OLAP*-Anaylse 26
2.4. Beispiel eines Entscheidungsbaums 32
2.5. Bauweise mobiler Endgeräte 36
2.6. Wasserfallmodell 49
2.7. V-Modell XT 50
2.8. Agile Vorgehensweise mit *Scrum* 53
2.9. Formen des Outsourcings 60
2.10. Ergebnis einer Literatursynopse über IT-Outsourcing 72
2.11. Outsourcingoptionen im Softwareentwicklungsprozess 73
2.12. Einbettung der Outsourcingentscheidung in den Softwareentwicklungsprozess 77

3.1. Entscheidungsmodell für Softwarekomponenten 93
3.2. Kopplung und Köhasion von Softwarekomponenten 96
3.3. Einordnung von SODA in den Kontext des Softwareentwicklungsprozesses 109
3.4. Vorgehensmodell anforderungsbasierter Ansätze 110
3.5. Struktur einer Entscheidungstabelle 119
3.6. Berechnung der Teilnutzenwerte und des Gesamtnutzens 126
3.7. Beispielhafte Eingabeoberfläche einer Produktbewertung 129
3.8. Komponentenmodell des Beispiels 130
3.9. Gesamtarchitektur 134
3.10. Modularer Aufbau der Architektur 138
3.11. Datenübertragung eines *JSON*-Objekts 139
3.12. Integrierte Entwicklungsumgebung *XCode* 142

3.13. *Storyboard* von *SmartSourcer* 143
3.14. Datenmodell von *SmartSourcer* 144
3.15. Datenaustausch zwischen *SmartSourcer* und *CodeBeamer* 144
3.16. Startbildschirm und Hauptmenü von *SmartSourcer* 146
3.17. Screenshot der Einstellungen 147
3.18. Screenshot der Komponentenbewertung 148
3.19. Einstellung der Gewichte 148
3.20. Übersicht der Outsourcingempfehlungen 150
3.21. Detaillierte Ergebnisübersicht 151
3.22. Exportmöglichkeiten von *SmartSourcer* 151
3.23. Vorschlagsfunktion für Kopplung und Kohäsion 155
3.24. Empfehlungsseite des generierten Reports 158
3.25. Detailseite des generierten Reports 159
3.26. Übersichtsseite der Kategorien im generierten Report 160

4.1. Klassifizierung von Evaluationsmethoden in der *Design Science* . 164
4.2. Evaluationsmodell 169
4.3. Evaluationsmodell mit Ergebnissen 183
4.4. SUS-Auswertung 185
4.5. Evaluationsmodell mit Effektstärken 188

Tabellenverzeichnis

2.1. Methoden der multikriteriellen Entscheidungsverfahren 29
2.2. Prinzipien des Agilen Manifests 52
2.3. Einzelfunktionalitäten einer Kollaborationsplattform für die Softwareentwicklung 57

3.1. Zusammenfassung der Anforderungen 86
3.2. Zusammenfassung der strukturspezifischen Entscheidungskriterien einer Softwarekomponente 94
3.3. Zusammenfassung der prozessspezifischen Entscheidungskriterien einer Sofzwarekomponente 97
3.4. Zusammenfassung der wissensspezifischen Entscheidungskriterien einer Sofzwarekomponente 100
3.5. Vergleich multikriterieller Verfahren 115
3.6. Entscheidungstabelle für die Prozessmerkmale der Entwicklung von Softwarekomponenten 122
3.7. Wertzuweisung zu den Ausprägungen 123
3.8. Intervalle für den spezifischen Einfluss 124
3.9. Intervalle für das Outsourcingpotenzial 127

4.1. Indikatoren der Informationsqualität 171
4.2. Indikatoren der Qualität der Entscheidungslogik 172
4.3. Indikatoren der Qualität der Implementierung 173
4.4. Indikator der Nutzungsbereitschaft 173
4.5. Wertzuweisung zu den Stufen der Likert-Skala 174
4.6. Teilnehmer der experimentellen Evaluation 178
4.7. Teilnehmer der Expertenevaluation 180
4.8. Aggregiertes Ergebnis der deskriptiven Analyse durch Studenten und Experten 181
4.9. Berechnung der *System Usability Scale* (SUS) 184

4.10. Korrelationsmatrix 187

I.1. Gruppenstatistik beim t-Test zum Vergleich der beiden Gruppen (Experten / Studenten) 237
I.2. Ergebnisse des t-Tests 237
I.3. Gesamtergebnis der deskriptiven Analyse 238
I.4. Ergebnis der deskriptiven Analyse des Studentenexperiments . . . 239
I.5. Ergebnis der deskriptiven Analyse der Expertenbefragung 240

Abkürzungsverzeichnis

API	*Application Programming Interface* - Programmierschnittstelle
BWL	Betriebswirtschaftslehre
CASE	*Computer-Aided Software Engineering* - Computergestützte Softwareentwicklung
EUS	Entscheidungsunterstützungssystem(e)
GVSE	Global verteilte Softwareentwicklung
IDE	*Integrated Development Environment* - Integrierte Entwicklungsumgebung
ISO	*International Organization for Standardization* - Internationale Organisation für Normung
IT	Informationstechnologie
JSON	*JavaScript Object Notation* - Kompaktes Datenformat in Textform
KMU	Kleine und mittelständische Unternehmen
MADM	*Multi Attribute Decision Making* - Multikriterielle Entscheidungsmodelle
MODM	*Multi Objective Decision Making* - Multiobjektive Entscheidungsmodelle
OLAP	*Online Analytical Processing* - Analytische Onlineverarbeitung
PDF	*Portable Document Format* - Portables Dokumentenformat
RUP	*Rational Unified Process* (Standardisiertes Softwareprozessmodell der Firma Rational)
SDK	*Software Development Kit*
SE	Softwareentwicklung
SODA	*Software Outsourcing Decision Aid*

SUS	*System Usability Scale*
TAM	Technologieakzeptanzmodell
UML	*Unified Modeling Language* - Vereinheitlichte Modellierungssprache
VWL	Volkswirtschaftslehre
WI	Wirtschaftsinformatik
WiWi	Wirtschaftswissenschaftler

1. Einleitung

Dieses einführende Kapitel dient zunächst zur Erläuterung des Problemfeldes der vorliegenden Arbeit. Weiterhin werden die Zielsetzung und die Forschungsfragen aufgezeigt, bevor die methodische Vorgehensweise und der konkrete Aufbau der Arbeit präsentiert werden.

1.1. Motivation

Im Umfeld von Softwareentwicklungsprojekten ist das Thema Outsourcing längst allgegenwärtig geworden und wird als weit verbreitete, akzeptierte Vorgehensweise zum Umgang mit kompetitiven Aspekten wie Kostendruck oder dem Mangel an talentierten Fachkräften angewendet. Bereits existierende Formen des Outsourcings von Entwicklungsaufgaben in Form von firmeneigenen Offshoring-Zentren (*captive offshoring centers*[1]) oder Kunden-Lieferanten-Beziehungen (*client vendor relationships*) neigen in ihrer bisherigen Anwendungsweise fehleranfällig zu sein. Gründe dafür sind, dass Outsourcingentscheidungen auf der Basis von Softwarekomponenten eines Gesamtsystems meist unstrukturiert und bestenfalls heuristisch, aber nicht systematisch getroffen werden (Kramer et al., 2013).

Die Entscheidungsfindung im Rahmen von Softwareprojekten umfasst dabei hauptsächlich die Identifikation von Komponenten eines Informationssystems, die entweder lokal von eigenen Entwicklern oder vom Outsourcingpartner entwickelt werden. An dieser Stelle setzt diese konstruktionswissenschaftlich orientierte Forschungsarbeit an und soll dazu beitragen, ein Entscheidungsmodell und ein zugehöriges Softwareartefakt zu schaffen und zu evaluieren, um den Entscheidungsprozess für das Outsourcing in der globalen Softwareentwicklung zu verbessern. Dabei soll eine systematische und strukturierte Entscheidungsfindung dazu verhelfen, transparente und nachvollziehbare Entscheidungen zu treffen.

1 Englische Begriffe sowie Eigennamen, wie z.B. Firmennamen oder Namen von Produkten, werden im weiteren Verlauf dieser Arbeit kursiv dargestellt.

Hierfür werden im Verlauf der Arbeit Kalküle aus der Entscheidungstheorie mit Konzepten des Outsourcings von Informationssystemen in der kollaborativen Softwareentwicklung und mobiler Technologien kombiniert. Die Verknüpfung dieser Forschungsgebiete bildet schließlich die Grundlage für das in dieser Arbeit entwickelte Entscheidungsmodell und das zugehörige Informationssystem zur Unterstützung von IT-Projektleitern bei der Entscheidungsfindung zum Outsourcing von Softwarekomponenten. Die Kombination von Modell und Software soll zur Vermeidung von heuristischen Entscheidungen bei der Bewertung von Softwarekomponenten beitragen, indem die Fundierung jeder Einzelentscheidung stets nachvollzogen werden kann.

Unter nationalen und internationalen Gesichtspunkten wurde das Outsourcing bei der Entwicklung von Anwendungssoftware flächendeckend angenommen und hat sich innerhalb der letzten Jahre kontinuierlich weiterentwickelt. Ein beachtlicher Teil der hiesigen Softwareentwicklungsfirmen betreibt heutzutage Softwareoutsourcing und erwartet dadurch betriebliche Vorteile, wie beispielsweise Kosteneinsparungen in der Entwicklung, Rund-um-die-Uhr-Entwicklung von Softwarepaketen oder zusätzliche Arbeitskraft in Form von talentierten Mitarbeitern (Dibbern et al., 2004; Lacity et al., 2010, 2011; Lacity und Willcocks, 1998).

Über die Jahre haben sich eine Menge von Outsourcingarrangements mit unterschiedlichen Akteuren entwickelt. Begonnen haben die großen Softwarefirmen, die es sich zunächst leisten konnten, Offshoring-Zentren in Niedriglohnländern (z.B. Indien) aufzubauen, deren Mitarbeiter eine vergleichsweise gute informationstechnologische Universitätsausbildung besitzen (Carmel und Agarwal, 2001), um dort eigene Entwickler vorzuhalten. In der weiteren Entwicklung verloren diese Zentren immer mehr ihren unternehmenseigenen Status, wurden ausgegliedert und boten schließlich ihre Dienstleistungen auch an dritte Unternehmen an, um einerseits von Skaleneffekten und Verbundvorteilen profitieren zu können und andererseits auch auf dem globalen Markt wettbewerbsfähig zu bleiben (King und Torkzadeh, 2008; Levina und Vaast, 2008).

Weitere Formen des Outsourcings im Bereich der Informationstechnologie waren zunächst geprägt von Radikalität und beinhalteten das Auslagern ganzer Informationstechnologiebereiche eines Unternehmens an externe Dienstleister. Dies hatte auch den Transfer von Mitarbeitern und materiellen Ressourcen (Applegate und Montealegre, 1991; Arnett und Jones, 1994) zum neuen Lieferanten zur Kon-

sequenz, bis sich später der Fokus auf selektives Outsourcing verlagerte (Heinzl, 1993). Bei dieser Vorgehensweise wird der Fokus auf einzelne spezifische Aktivitäten der Softwareentwicklungs- und Bereitstellungsprozesse gelegt oder ist von den jeweiligen untergeordneten Geschäftsprozessen (*business process outsourcing*) geprägt (Gewald, 2010; Lacity et al., 2011, 2009). Erst dadurch wurde Outsourcing auch für kleine und mittelständische Softwareunternehmen interessant und ermöglichte einer Vielzahl dieser Unternehmen am globalen Markt der Softwareentwicklung zu partizipieren, um dessen Vor- und Nachteile zu erfahren (Klimpke et al., 2011).

Das stetig wachsende Angebot von Cloud-Computing Dienstleistungen und die damit verbundene Bereitstellung von Softwarediensten für den direkten Gebrauch ohne notwendige Wartung von Kundenseite, belebt den Outsourcingmarkt mit neuen Möglichkeiten. Die Transformation von der Bereitstellung reiner Hardwareressourcen hin zu umfangreichen Softwarelösungen, die bei Bedarf als online verfügbare Dienstleistungen abgerufen werden können und verbrauchsabhängig abgerechnet werden, machen die Cloud als Basistechnologie für weitere Outsourcing- und Integrationsmöglichkeiten bei der Entwicklung von Software attraktiv (Buxmann und Hess, 2008; Böhm et al., 2009; Lehman et al., 2010; Stuckenberg et al., 2014).

Hinsichtlich dieser Entwicklung im Bereich Softwareoutsourcing wurden parallel dazu die Entwicklungskonzepte für Software auf den Prüfstand gestellt und kontinuierlich angepasst. Von der ursprünglich reinen Eigenherstellung von Softwarelösungen passten sich die Anbieter den Eigenschaften der jeweiligen Outsourcingmodelle an. Im Mittelpunkt standen dabei besonders die Erhebung und Weiterleitung (oder auch Übersetzung) von Anforderungen an ein geplantes Softwaresystem, die Einrichtung von Kollaborationsplattformen zur gemeinsamen Erstellung von Softwarelösungen, die Nachvollziehbarkeit und Verfolgung (*Traceability*) von Änderungen im Entwicklungsprozess oder auch die Einrichtung und Steuerung global verteilter Entwicklungsteams (Geisser, 2008; Herbsleb und Moitra, 2001; Hildenbrand, 2008). Einen wichtigen Stellenwert bei dieser global orientierten Entwicklung nimmt die Outsourcingentscheidung ein, die festlegt, welche Teile einer Software selbst entwickelt und welche fremd vergeben werden. Unter diese zentrale Entscheidung fallen in erster Linie Aspekte wie die Wahl des Outsourcingmodells (gesamte Entwicklungsabteilung, einzelne Entwicklungsphasen,

selektive Entwicklungsaufgaben), die Auswahl des Outsourcingpartners, die Festlegung einer einheitlichen und systematischen Vorgehensweise für die gemeinsame Entwicklung oder, abhängig vom Outsourcingmodell, welche einzelnen Teile des Gesamtsystems selbst entwickelt werden sollen und welche extern bezogen werden.

Die zentrale Bedeutung der Outsourcingentscheidung wird durch die Vielzahl an wissenschaftlichen Studien zu diesem Thema belegt. Zahlreiche Forschungsbeiträge zur Untersuchung der Outsourcingmodelle (Dibbern et al., 2004), zur Wahl von Outsourcingpartnern (Kim, 2009; Qu und Brocklehurst, 2003) sowie zur vertraglichen Gestaltung von Outsourcingbeziehungen (Gopal et al., 2003) wurden bereits veröffentlicht. In Bezug auf das Outsourcing befassen sich die bisherigen Arbeiten also überwiegend mit strategischen und taktischen Aspekten des Lieferanten- oder Beschaffungsmanagements aus Sicht der Unternehmen und vernachlässigen die technischen Aspekte eines geplanten Gesamtsystems, die für die operativen Phasen des Vorhabens entscheidend sind, da diese das Zusammenspiel zwischen interner und externer Entwicklung maßgeblich beeinflussen.

Die operative Seite von Outsourcingbeziehungen, also insbesondere wie sich der Mix gestaltet, welche Komponenten einer Softwarearchitektur spezifisch für die Auslagerung geeignet sind und welche nicht, ist im bisherigen Wissensfundus weitgehend unerforscht. Die Besonderheiten des Softwareentwicklungsprozesses, die komponentenweise Unterteilung des Softwareprodukts und die Merkmale dieser Komponenten werden bisher nur rudimentär als Untersuchungsgegenstand von Outsourcingstudien herangezogen. Doch gerade die Komponenten mit ihren spezifischen Eigenschaften stehen im Mittelpunkt der Softwareentwicklung und müssen zu den wichtigsten Entscheidungskriterien für das Vorgehen bei der Outsourcingentscheidung im operativen Prozess gezählt und dementsprechend einbezogen werden. Erst dadurch und zusammen mit dem entsprechenden Entscheidungsmodell kann deren Eignung für die Eigenerstellung oder Fremdvergabe ermittelt und sie damit zuverlässig dem richtigen Entwicklungsteam zugeordnet werden. Dies ermöglicht eine Zuweisung der einzelnen Komponenten entweder zur eigenen Entwicklungsorganisation oder zu den Entwicklern des Outsourcingpartners, abhängig von ihren spezifischen Eigenschaften im Gesamtsystem. Durch diese Unterstützungsform der Outsourcingentscheidung sollen prominente Probleme beim Outsourcing, die in Form von Lieferverzug, hohen Fehlerraten (Bugs), In-

tegrationsschwierigkeiten, Kommunikationsschwierigkeiten oder Unzufriedenheit beim Kunden auftreten können, gemindert oder bestenfalls vermieden werden.

1.2. Forschungsschwerpunkt und Zielsetzung

Im vorherigen Abschnitt wurden die wissenschaftlichen und praktischen Forschungsdefizite im operativen Bereich des Softwareoutsourcings hervorgehoben. Darin wurde verdeutlicht, welchen Herausforderungen sich Unternehmen bei Outsourcingkooperationen stellen müssen und welche Besonderheiten damit in global verteilten Softwareentwicklungsprojekten auftreten. Auch wenn bisher Handlungsempfehlungen und Leitfäden einige Ratschläge zur Durchführung von Outsourcingaktivitäten in Softwareprojekten liefern, so werden die Entscheidungen, welche Komponenten einer Softwarearchitektur selbst hergestellt und welche extern vergeben werden, unsystematisch bzw. heuristisch getroffen und stellen daher keine hinreichende Entscheidungsgrundlage dar (Kramer et al., 2013). Durch eine strukturierte Herangehensweise, eine systematische Abschätzung von Einflussfaktoren einzelner Softwarekomponenten auf die Entscheidung und einen transparenten Entscheidungsprozess soll schließlich die Qualität der Entscheidungen verbessert werden, um damit zukünftig die im Vorfeld aufgezeigten Outsourcingprobleme zu vermeiden. Hierfür erscheint ein Ansatz zur Unterstützung von Outsourcingentscheidungen in der komponentenbasierten Softwareentwicklung in Form eines Entscheidungsmodells als zielführend, um in diesem Kontext eine Zielgruppe bestehend aus Projektleitern, Outsourcingmanagern und IT-Leitern anzusprechen (Kramer et al., 2011, 2013). Die Mobilitätsanforderungen dieser Personengruppe prädestinieren mobile Technologien zur Umsetzung des Lösungsansatzes.

Um die identifizierten Defizite im Softwareoutsourcing folgerichtig zu adressieren, bedarf es jedoch einer Lösung, die sich nahtlos in bestehende Softwareentwicklungsprozesse einbinden lässt, die Schwachstellen der bisherigen Entscheidungsfindung strukturiert aufgreift und die den verantwortlichen Personenkreis systematisch bei der Entscheidungsfindung unterstützt. Der in dieser Arbeit entwickelte Lösungsansatz umfasst daher ein Entscheidungsmodell sowie dessen Implementierung als mobiles System zur Entscheidungsunterstützung. In dieser Arbeit wird der Fokus auf die Konzeptions- und Designphasen des Softwareentwick-

lungsprozesses gelegt, denn darin müssen die Architektur und die Arbeitspakete für die Erstellung des zukünftigen Softwareprodukts und seiner Komponenten definiert werden. Erst die nachfolgenden Phasen (Entwicklung und Test) werden dann oftmals unter Einbeziehung von Zulieferern durchgeführt (Kramer et al., 2013). Bereits vor Eintritt dieser Phasen muss eine Entscheidung darüber getroffen werden, welche Softwarekomponenten selbst erstellt und welche fremdbezogen werden. Zur gezielten Unterstützung der Entscheidungsträger (Kramer et al., 2013), soll das in dieser Arbeit entwickelte System zur Entscheidungsunterstützung den Einsatz mobiler Kommunikationstechnologien (*Mobile Apps*) ermöglichen, damit trotz der Mobilitätsanforderungen der zuständigen Manager deren Entscheidungen flexibel und zeitnah getroffen werden können. Zusammengefasst lassen sich daher folgende Forschungsziele für diese Arbeit aufstellen:

Z1 Herleitung und Entwicklung eines Entscheidungsmodells für das Outsourcing von Softwarekomponenten im Softwareentwicklungsprozess.

Z2 Design und Implementierung eines prototypischen Werkzeugs zur Instanziierung des Entscheidungsmodells.

Z3 Bewertung des im Softwareprototypen instanziierten Entscheidungsmodells zur Unterstützung von Outsourcingentscheidungen in kollaborativen Softwareentwicklungsprojekten.

1.3. Methodische Vorgehensweise und Struktur der Arbeit

Die methodische Vorgehensweise dieser Arbeit ist am konstruktionswissenschaftlichen Forschungsparadigma (*Design Science*[2]) der Disziplin Wirtschaftsinformatik orientiert, wobei die Überlegungen von Hevner et al. (2004) zugrunde gelegt werden.

Dieser gestaltungsorientierte Forschungsansatz hat seinen Ursprung in den Ingenieurswissenschaften und verfolgt das Ziel, Technologie in Form von innovativen Artefakten zu gestalten und zu erschaffen. Im Gegensatz zum verhaltenswissenschaftlichen Forschungsparadigma (*Behavioral Science*), das die Entwicklung und

2 Wird im Nachfolgenden als Kurzform für den konstruktionswissenschaftlichen Forschungsansatz verwendet

Begründung von Theorien zur Erklärung und Vorhersage von Ursache-Wirkungs-Beziehungen verfolgt, zielt das als komplementär zu betrachtende gestaltungsorientierte Paradigma auf die Erstellung und Evaluierung neuartiger Artefakte zur verbesserten Erfüllung a priori definierter Anforderungen ab (Hevner et al., 2004). Nach der Definition von March und Smith (1995) können Artefakte Konstrukte, Methoden, Modelle oder Instanzen sein. Sie spiegeln die vielfältigen Möglichkeiten zur Schaffung von Produkt- oder Verfahrensverbesserungen wider.

Dieser innovationsgetriebene Forschungsansatz vereint die theoretischen Konzepte der Wirtschaftsinformatik mit den praktischen Problemen, die im Geschäftsumfeld auftreten. Das Forschungsparadigma wird daher an der Schnittstelle zwischen Theorie und Praxis angesiedelt, um einen Gestaltungsrahmen für innovative und nützliche Lösungen (vgl. Artefakte) zu realen Problemen (aus dem betrieblichen Umfeld) bereitzustellen (Gregor und Jones, 2007; March und Smith, 1995; Peffers et al., 2007).

Das konstruktionswissenschaftliche Vorgehen orientiert sich entlang dreier zentraler Forschungszyklen, die eng miteinander verzahnt (vgl. Abbildung 1.1) den Charakter einer gestaltungsorientierten Arbeit prägen (Hevner et al., 2004). Hierzu gehören neben dem zentralen Zyklus, der die Entwicklung eines Artefakts konzipiert (*Design Cycle*), ebenfalls der Zyklus zur Herleitung und Überprüfung der praktischen Relevanz (*Relevance Cylce*) sowie der Zyklus zur Herleitung der theoretischen Fundierung und zur Erweiterung der theoretischen Wissensbasis (*Rigor Cycle*).

Die zentrale Rolle in diesem Forschungsansatz nimmt der *Design Cycle* ein. Er umfasst die iterative Entwicklung innovativer Lösungsansätze und deren anschließende Evaluation. Innovationen im Sinne des konstruktionswissenschaftlichen Paradigmas umfassen nicht ausschließlich die Erstellung neuer Artefakte, sondern auch die Verbesserung bereits bestehender Technologien. Der *Relevance Cylce* und der *Rigor Cycle* sind dabei als informierende und profitierende Zyklen zu betrachten. Während im Rahmen des *Rigor Cycle* auf der einen Seite die theoretischen Grundlagen und Verhaltenstheorien zur Schaffung von Innovationen einbezogen werden, wird dafür auf der Gegenseite die Wissensbasis durch Erkenntnisse aus der neu erschaffenen Technologie erweitert. Durch die Integration des *Relevance Cycle* werden hingegen konkrete praxisrelevante Problemstellungen aus der Geschäftswelt identifiziert, zur Erstellung neuer Innovationen herangezogen und

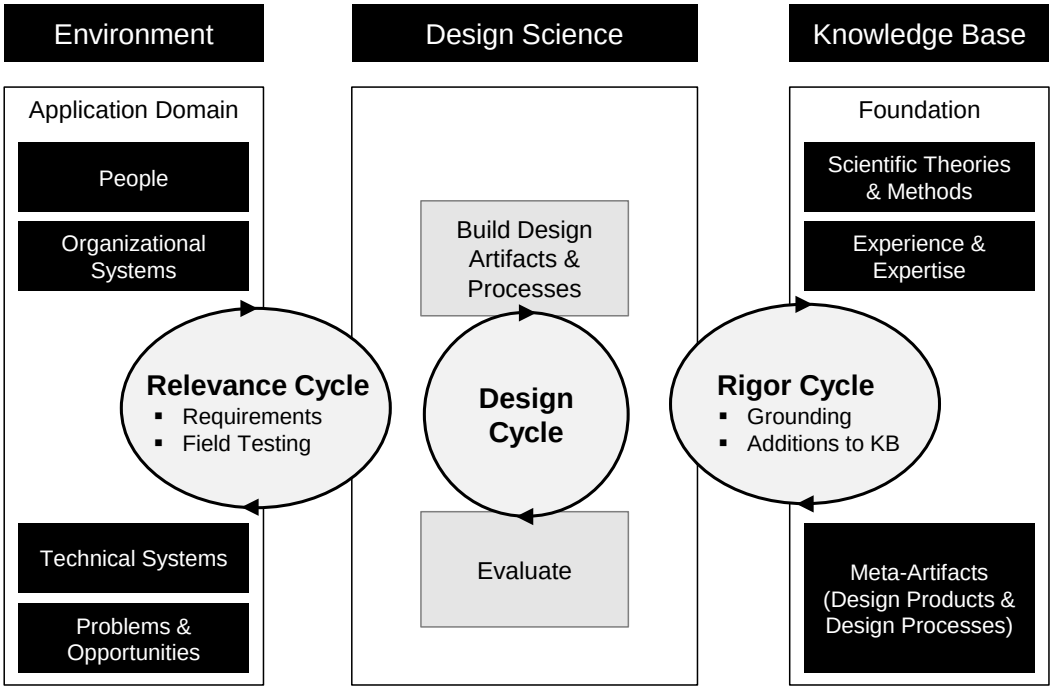

Abbildung 1.1.: Die Forschungszyklen in der *Design Science*, abgeleitet von Hevner et al. (2004)

im Nachgang eine Lösungstechnologie zur praktischen Nutzung zurückgegeben. Die wesentlichen Unterschiede dieses Paradigmas im Vergleich zur professionellen Softwareentwicklung stellen die enge Verzahnung mit der theoretischen Wissensbasis, die rigorose Evaluation von neuen Artefakten und die Notwendigkeit zur Veröffentlichung von erzielten Beiträgen zur Wissenschaft und Forschung dar (Hevner et al., 2004; Peffers et al., 2007).

Das mit dieser Arbeit verfolgte Ziel zur Entwicklung eines Systems zur Entscheidungsunterstützung für das Outsourcing in der komponentenbasierten Softwareentwicklung folgt dem pragmatischen Forschungsparadigma (Heinrich et al., 2011) zur Schaffung und Evaluation neuer Artefakte und kann daher als konstruktionswissenschaftliches Forschungsziel bezeichnet werden (Gregor und Jones, 2007).

In dieser Arbeit werden zur Erreichung der Zielsetzung zunächst die für die Outsourcingentscheidung relevanten Eigenschaften von Softwarekomponenten in einem Entscheidungsmodell zusammengefasst, um deren Potenzial für die Fremd- bzw. Eigenerstellung bestimmen zu können. Dieses Modell wird einen strukturier-

ten, systematischen und transparenten Unterstützungsprozess für die Outsourcingentscheidung beinhalten, der es dem designierten Anwenderkreis von Entscheidungsträgern ermöglicht, Entscheidungen nachvollziehbar zu treffen und zu dokumentieren. Anschließend wird es prototypisch in Form eines mobilen Systems zur Entscheidungsunterstützung (EUS) implementiert und systematisch evaluiert, um die Nützlichkeit des neuen Artefakts zu zeigen.

Zu diesem Zweck orientiert sich der Aufbau dieser Arbeit an den erforderlichen Schritten zur synchronisierten Durchführung der drei Forschungszyklen der *Design Science*. Nach der bereits eingangs aufgezeigten Problemstellung und Zielsetzung der Arbeit werden deshalb im nachfolgenden Kapitel 2 zunächst Begrifflichkeiten und Konzepte zum grundlegenden Verständnis des Problemfeldes erörtert und definiert. Neben den grundlegenden Begrifflichkeiten von Entscheidungsproblemen und Entscheidungsmodellen werden Typen von Entscheidungsunterstützungssystemen eingeführt und erläutert. Im weiteren Verlauf werden die Grundlagen der komponentenbasierten Softwareentwicklung vorgestellt, indem die einzelnen Phasen des Softwareentwicklungsprozesses und bekannte Prozessmodelle vorgestellt werden. Außerdem wird auf die Besonderheiten der verteilten Entwicklung näher eingegangen. Abschließend wird mit den Vorteilen und Herausforderungen der ausgelagerten Softwareentwicklung und existierenden Vorgehensweisen zum Auslagern von Entwicklungsaufgaben Basiswissen über das Outsourcing von Informationssystemen vermittelt.

In Kapitel 3 dieser Arbeit wird die Konzeption eines Entscheidungsmodells zur Unterstützung der Outsourcingentscheidung in der komponentenbasierten Softwareentwicklung und die Implementierung des Modells als mobile Applikation beschrieben. In der Konzeptionierung werden die Bestandteile des Entscheidungsmodells aus der theoretischen Wissensbasis hergeleitet und für die Bewertung von Softwarekomponenten verwendet. Die anschließende Implementierung eines mobilen Systems zur Entscheidungsunterstützung setzt dieses Modell dann um. Dadurch wird Entscheidern eine systematische, strukturierte und teilautomatisierte Möglichkeit zur Bewertung des Outsourcingpotenzials von Softwarekomponenten eines Projekts gegeben.

Im anschließenden Kapitel 4 wird das neu entwickelte Softwarewerkzeug zur Entscheidungsunterstützung in einer Expertenbefragung und in einem Studentenexperiment evaluiert. Damit wird überprüft, ob die oben genannten Ziele

dieser Arbeit zur Verbesserung der Entscheidungsfindung im Outsourcing erreicht werden und damit eine nützliche Lösungstechnologie für die praktische Verwendung vorliegt. Dafür werden zunächst die gängigen Konzepte zur Evaluation innovativer Artefakte bei konstruktionswissenschaftlichen Arbeiten vorgestellt. Anschließend wird ein theoretischer Bezugsrahmen hergeleitet, innerhalb dessen das System zur Entscheidungsunterstützung hinsichtlich seiner Qualität des Entscheidungsmodells und seiner Qualität der Implementierung sowie hinsichtlich der Nutzungsbereitschaft der Anwender evaluiert wird. Die Verwertung der Ergebnisse wird in eine deskriptive und eine qualitative Analyse aufgeteilt. Abschließend folgt in Kapitel 5 die Zusammenfassung des theoretischen und praktischen Beitrags dieser Arbeit, die mit einem Ausblick unter Berücksichtigung der Limitationen abgerundet wird.

2. Theoretische und konzeptionelle Grundlagen

In diesem Kapitel werden die thematischen Grundlagen, deren Kenntnis für das Verständnis dieser Arbeit erforderlich ist, als Wissensbasis im Sinne der konstruktionswissenschaftlichen Forschung aufgezeigt (Hevner et al., 2004). Insgesamt wird ein Überblick über die beiden zentralen Themen Entscheidungsfindung mit seinen zugehörigen Unterstützungssystemen sowie Softwareentwicklung mit Outsourcingbezug gegeben. Neben Begriffsdefinitionen für die Entscheidungsfindung im betrieblichen Kontext, werden im ersten Teil dieses Kapitels auch mobile Systeme zur Entscheidungsunterstützung und deren Einsatzmöglichkeiten als moderne Informationstechnologie vorgestellt. Für das Verständnis des Zusammenhangs zwischen Entscheidungsunterstützungssystemen und Softwareoutsourcing werden die wesentlichen Elemente der Softwareentwicklung, deren Ablaufreihenfolge und die einzelnen Auslagerungsmöglichkeiten im Detail erörtert. Am Ende des Kapitels werden die Grundlagen und bisherigen Forschungsergebnisse der jeweiligen Themengebiete hinsichtlich ihrer Verwendung für die Entwicklung eines neuartigen Entscheidungsmodells und einem zugehörigen System zur Entscheidungsunterstützung zusammengefasst.

2.1. Grundlagen der Entscheidungsfindung

In dieser Arbeit steht die betriebswirtschaftliche Entscheidungsfindung hinsichtlich zu entwickelnder Software im Vordergrund. Deshalb werden zunächst die Grundlagen von Entscheidungen, mögliche Entscheidungsprobleme in Organisationen sowie Entscheidungsmodelle und deren Entscheidungslogik vorgestellt, um damit den Weg zur Entwicklung eines Entscheidungsmodells für das Softwareoutsourcing zu ebnen.

2.1.1. Die Entscheidung

Im täglichen operativen Betrieb sowie auf strategischer Ebene von Unternehmen müssen kontinuierlich Entscheidungen getroffen werden, um den gesetzten Unternehmenszielen schrittweise näher zu kommen. Der strukturelle Aufbau von Unternehmen wird in erster Linie in hierarchischen Organisationsdiagrammen dargestellt und kann für sich betrachtet als Prozessmodell für das Treffen von Entscheidungen angesehen werden (Cyert und March, 1963). Darin werden Autoritäten und Verantwortlichkeitsstrukturen aufgezeigt, die als Indikator für Entscheidungswege und Entscheidungsgewalten dienen und so die Organisation in ihrer Gesamtheit als Entscheidungsfindungsprozess zur Durchführung von Einzelentscheidungen beschreiben.

Unter Einzelentscheidungen versteht man in diesem Kontext die mehr oder weniger bewusste Auswahl einer oder mehrerer Handlungsalternativen, die im Geschäftsumfeld auftreten (Keen und Scott Morton, 1978). Im speziellen Fall dieser Arbeit handelt es sich daher um Einzelentscheidungen, ob eine Softwarekomponente selbst erstellt oder ausgelagert werden soll. Dabei bedient man sich bekannter Entscheidungstheorien, erweitert diese oder erforscht neue. Die theoretischen Modelle dienen einerseits der empirischen Erklärung wie die Entscheidungsfindung in realen Umgebungen durchgeführt wird und zeigen andererseits auf, welche Konsequenzen aus den gegebenen Wahloptionen entstehen können. Die Entscheidungstheorien werden in normative, deskriptive und präskriptive Ansätze unterschieden.

Normative Entscheidungstheorien stehen für den Auswahlprozess, der aus einer möglichen Menge von Handlungsalternativen eine vorteilhafte ermittelt. Bei diesem Vorgang wird unter Berücksichtigung der im Vorfeld festgelegten Zieldefinitionen eine rationale Vorgehensweise als zweckmäßige Entscheidungsmethode zugrunde gelegt. Die menschlichen Fähigkeiten des rationalen Handelns der verantwortlichen Akteure bleiben dabei unberücksichtigt (Schneeweiß, 1991; Sieben und Schildbach, 1994).

Zur Beschreibung und Erklärung der Entscheidungsfindung in realistischen Szenarien werden deskriptive Entscheidungstheorien, die auch als empirisch-realistisch bezeichnet werden, verwendet. Zu diesem Zweck werden empirisch validierte Thesen über das menschliche Entscheidungsverhalten entwickelt, die im Gegen-

satz zu den Realwissenschaften auf den bei Individuen oder Gruppen gemessenen Beobachtungen beruhen und dem prognostizierten Entscheidungsverhalten der Realität widersprechen können (Schneeweiß, 1991; Sieben und Schildbach, 1994).

Die präskriptiven Entscheidungstheorien hingegen integrieren Aspekte der normativen und deskriptiven Theorien. Hierbei werden beobachtete Entscheidungsverhalten mit den normativen Vorgaben situativ in Verbindung gesetzt (Schneeweiß, 1991; Sieben und Schildbach, 1994).

Die Erkenntnisse aus den vorgestellten Forschungsfeldern zur Entscheidungsfindung werden in den nachfolgenden Kapiteln unter Berücksichtigung der Problemstellung dieser Arbeit weiter vorgestellt. Aus diesem Grund wird zunächst das vorherrschende Entscheidungsproblem klassifiziert und anschließend Entscheidungsmodelle sowie entsprechende Logikansätze aufgeführt, die zur finalen Entscheidungsfindung zweckmäßig und vorteilhaft sind.

2.1.2. Entscheidungsprobleme

Die Definition von Entscheidungsproblemen stammt in erster Linie aus der Komplexitätstheorie der Prädikatenlogik und behandelt Fragestellungen mit binärem Ausgang, sog. „ja/nein“-Fragen. Dabei werden bestimmte Mengen auf Entscheidbarkeit untersucht, indem überprüft wird, ob für diese ein Entscheidungsverfahren vorliegt. Sofern dies der Fall ist, gilt eine gegebene Menge als entscheidbar. Liegt ein solches Verfahren nicht vor, so wird die Menge als unentscheidbar klassifiziert. Das Verfahren bestimmt in diesem Fall, welche Elemente der Menge mit *ja* und welche davon mit *nein* beantwortet werden (Hilbert und Ackermann, 1928).

Im betriebswirtschaftlichen Umfeld entstehen Entscheidungsprobleme, sobald Akteure bewusst einen gewünschten Zielzustand erreichen möchten, der sich vom aktuellen Zustand als vorteilhafter unterscheidet. Dafür muss der Abstand der wesentlichen Merkmale, an denen der Zustandsunterschied festgemacht wird, von den Akteuren minimiert werden (Sanders, 1999). Dieser Unterschied zwischen zwei Zuständen macht alleine allerdings noch kein Entscheidungsproblem aus. Ein solches Problem entsteht erst dann, wenn es mehrere Möglichkeiten gibt, die Diskrepanz zwischen den Zuständen zu verringern. Dadurch entstehen für den Entscheider mehrere Handlungsalternativen, die zunächst bewertet werden müssen und anschließend sorgfältig gegeneinander abzuwägen sind. Auch wenn auf

den ersten Blick nur eine Option ersichtlich scheint, ist es Teil der Entscheidungsfindung, systematisch nach weiteren Handlungsoptionen zu suchen und diese zu bewerten, um dadurch die Qualität der schließlich selektierten Lösung zu verbessern (Grünig und Kühn, 2005). Die systematische Vorgehensweise wird durch Entscheidungsmodelle und deren Logiken sichergestellt, welche im nächsten Kapitel vorgestellt werden.

Zur Unterscheidung von Entscheidungsproblemen existieren zahlreiche Kriterien mit unterschiedlichen Ausprägungen, die eine Einteilung in unterschiedliche Typen von Entscheidungsproblemen ermöglichen. Darunter fallen die auszugsweise von Grünig und Kühn (2005) abgeleiteten Kriterien sowie ihre Ausprägungen (im Folgenden Dimensionen genannt):

1. Schwierigkeitsgrad: Handelt es sich um eine einfache oder komplexe Entscheidung?

2. Problemstruktur: Ist das Problem, das zugrunde liegt, gut oder schlecht strukturiert?

3. Grad der Verknüpfung: Ist das Entscheidungsproblem unabhängig oder einhergehend mit anderen Entscheidungsproblemen?

4. Akteure: Handelt es sich um einen oder mehrere Entscheider, d.h. wird die Entscheidung in Einzelverantwortung oder im Kollektiv getroffen?

5. Zielgröße: Handelt es sich um ein Ziel oder mehrere Ziele, die erreicht werden können?

6. Eintrittswahrscheinlichkeit: Wie sicher lassen sich die Konsequenzen der Entscheidung vorhersagen?

Entlang der erstgenannten Dimension zur Typenunterscheidung von Entscheidungsproblemen werden diese in einfache und komplexe Entscheidungsprobleme (Dimension 1) gegliedert. Die komplexe Version wird nach Grünig und Kühn (2005) dadurch kenntlich gemacht, dass mindestens zwei oder mehr der nachfolgenden Bedingungen erfüllt werden. Im Gegensatz dazu spricht man von einem einfachen Entscheidungsproblem, wenn von den genannten Bedingungen nur eine erfüllt ist:

- Durch den Akteur werden mehrere Zielzustände gleichzeitig angestrebt, die teilweise nicht ausführlich definiert sind oder sich sogar möglicherweise widersprechen (Morieux, 2011).
- Es existiert eine große Anzahl an Entscheidungsvariablen, die mitunter selbst eine ebenfalls große Anzahl an möglichen Ausprägungen haben können, zur Minimierung des Abstands zwischen der aktuellen Situation und dem Zielzustand.
- Die zukünftige Entwicklung von mehreren Umgebungsvariablen ist unsicher oder mit Risiko behaftet und erfordert daher eine Bewertung der Handlungsoptionen hinsichtlich verschiedener Umweltszenarien.
- Dem Akteur steht nur eine begrenzte Anzahl an Entscheidungsmodellen zur Verfügung oder er hat zu wenig Erfahrung, um seine zum Ziel führenden Handlungsoptionen zu evaluieren.

Die Klassifizierung in gut und schlecht strukturierte Entscheidungsprobleme (Dimension 2) wird durch die Genauigkeit zur Beschreibung des Problems festgelegt (Simon und Newell, 1958, S. 4). Wenn ein Problem derart genau beschrieben werden kann, dass ein analytisches Entscheidungsverfahren verwendet werden kann, um seine Lösung zu finden, dann liegt ein gut strukturiertes Entscheidungsproblem vor. Andernfalls sind aufwändigere Verfahren zur systematischen Gliederung und Lösung nicht strukturierter Probleme erforderlich (Grünig und Kühn, 2005).

Der Grad der Verknüpfung von Entscheidungsproblemen (Dimension 3) unterscheidet diese in voneinander unabhängige Probleme oder Entscheidungsprobleme, die in Abhängigkeit voneinander gelöst werden müssen. In diesem Fall werden durch die Auswahl einer oder mehrerer Lösungsoptionen weitere Entscheidungen in der Zukunft erforderlich. Bei der unabhängigen Variante wird schlicht die beste Lösungsoption ausgewählt (Grünig und Kühn, 2005).

Bezüglich der Akteure (Dimension 4) wird zwischen Individual- und Kollektiventscheidungen unterschieden. Eine Einzelentscheidung schließt zwar eine Beteiligung weiterer Personen bei der Problemanalyse und dem Auffinden von Handlungsmöglichkeiten nicht aus, die Entscheidung wird aber vom Einzelnen getroffen. Bei Kollektiventscheidungen muss die finale Lösungsoption von mehreren präferiert werden (Grünig und Kühn, 2005).

Die Zielgröße (Dimension 5) von Entscheidungsproblemen ordnet diese nach der Anzahl der betrachteten Ziele. Sofern dies nur ein Ziel ist oder ein arithmetischer Zusammenhang zwischen mehreren Zielen besteht, so wird dies ein univalentes bzw. uni-kriterielles Entscheidungsproblem genannt. Problemstellungen mit mehreren Zielsetzungen ohne arithmetischen Zusammenhang werden hingegen als polyvalente bzw. multi-kriterielle Entscheidungsprobleme klassifiziert (Grünig und Kühn, 2005).

Die Eintrittswahrscheinlichkeit (Dimension 6) der Konsequenzen von Handlungsoptionen können mal besser und mal schlechter vorhergesagt werden. Eine mit Sicherheit vorhersagbare Konsequenz ist äußerst unwahrscheinlich, würde aber eine sichere Entscheidung mit sich bringen. Risikobehaftete Folgen von Entscheidungsoptionen sind da häufiger aufzufinden und stellen die Klasse der Entscheidungsfindung unter Risiko dar. Wenn nicht ausreichend Informationen über die Eintrittswahrscheinlichkeiten von Konsequenzen vorliegen, dann nennt man dies auch Entscheidungsfindung unter Unsicherheit (Bamberg und Coenenberg, 1996; Grünig und Kühn, 2005).

Wie im Verlauf dieses Kapitels gezeigt wurde, lassen sich Entscheidungsprobleme in unterschiedliche Kategorien entlang der aufgezeigten Dimensionen klassifizieren. Das Entscheidungsproblem stellt dabei die Diskrepanz zwischen einem aktuellen Zustand und einem beabsichtigten Zielzustand dar. Zur Erreichung des gewünschten Zielzustandes existiert eine Menge an Aktivitäten, die eine Zielerreichung auf unterschiedliche Weisen ermöglichen. Entscheidungen können nach Grünig und Kühn (2005) folgendermaßen getroffen werden:

- Durch intuitive Auswahl einer Lösung;
- Durch routiniertes Auswählen einer Lösung, die einer vorherigen Lösung ähnlich ist;
- Durch Expertenempfehlung;
- Durch zufälliges Auswählen;
- Auf der Basis eines systematischen und rational verhaltenden Vorgehens.

Für letztere Möglichkeit der Entscheidungsfindung werden systematische Vorgehensweisen benötigt, die ein rationales Handeln ermöglichen. Hierfür existieren Entscheidungsmodelle und zugehörige Logikregeln, die eine Analyse des Problemfeldes, die Festlegung eines Handlungsraumes, die Auswahl einer bevorzugten

Handlungsalternative und die Durchführung der Entscheidungsfindung ermöglichen. Diese Modelle werden im nächsten Kapitel näher erläutert.

2.1.3. Entscheidungsmodelle und deren Logik

Als Entscheidungsmodell wird ein Aussagensystem bezeichnet, das für die Bestimmung von Handlungsmaßnahmen geeignet ist. Die Bestimmung von optimalen Entscheidungen zeichnet solche Modelle aus (Schneeweiß, 1991; Sieben und Schildbach, 1994). Wesentliche Bestandteile eines Optimierungsmodells, wie es überwiegend in der Unternehmensforschung verwendet wird, bilden:

- die Entscheidungsvariablen, auf die ein Entscheider Einfluss hat und die es zu optimieren gilt.
- das Zielsystem, das die Wertigkeit von Lösungen evaluiert und damit die optimale Entscheidung bestimmt.
- die problemkontextspezifischen Restriktionen, die sich aus dem Umfeld der Entscheidungssituation ergeben.
- die natürlichen Restriktionen, welche durch die Definition der Entscheidungsvariablen vorgegeben sind.

Derartige Entscheidungsmodelle und deren zugehörige Entscheidungslogik eignen sich für eine strukturierte und nachvollziehbare Lösungsfindung unter einer Vielzahl von Entscheidungsvariablen. Modelle sind in erster Linie eine vereinfachte Darstellung komplexer Sachverhalte, um so übersichtlicher oder schneller zu Erkenntnissen zu gelangen, die in ihrer ursprünglichen komplexen Umgebung nur schwerlich zu identifizieren wären. Mit Hilfe von Entscheidungsmodellen wird einerseits der Fokus auf Situationen gelegt, welche die wesentlichen Elemente eines Problems und deren Verknüpfungen abbilden, und andererseits ein strukturierter und logischer Entscheidungsweg zur Auflösung der vorherrschenden Konfliktsituation gegeben (Schneeweiß, 1991; Sieben und Schildbach, 1994).

In der normativen Entscheidungstheorie bezeichnet ein Entscheidungsmodell den Versuch, eine bestehende Handlungssituation, die als problematisch erachtet wird, so zu strukturieren, dass sich durch logische Schlussfolgerung die Lösung des Problems aus einer Anzahl an Handlungsalternativen ableiten lässt (Manz et al., 1993). In einem Entscheidungsmodell (vgl. Abbildung 2.1) wird zwischen

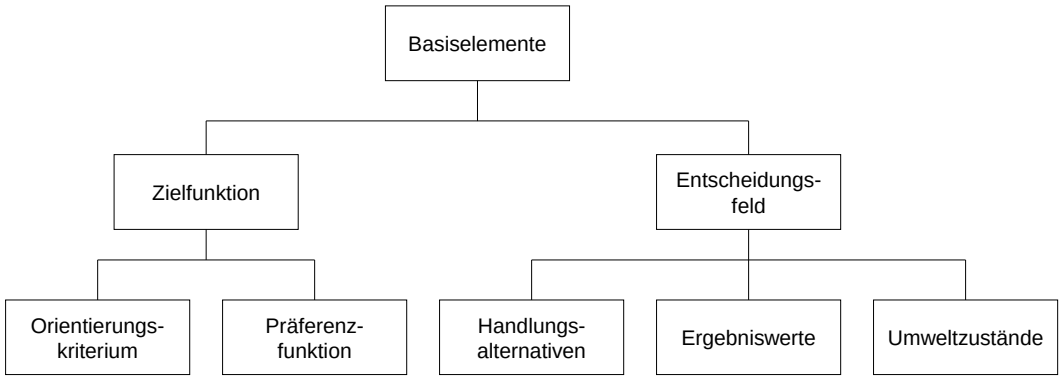

Abbildung 2.1.: Bestandteile eines Entscheidungsmodells (nach Manz et al., 1993, S. 9)

zwei wesentlichen Arten von Daten unterschieden. Das *Entscheidungsfeld* fasst in diesem Zusammenhang relevante Daten über die Entscheidungssituation der Akteure zusammen. Im Gegensatz dazu fließen über die *Zielfunktion* Daten der Akteure über beabsichtigte Ziele in das Entscheidungsmodell mit ein.

2.1.3.1. Entscheidungsfeld

Das Entscheidungsfeld umfasst den Aktionsraum für mögliche Handlungsalternativen, den Zustandsraum mit vorherrschenden Umweltzuständen und die Ergebnisfunktion mit möglichen Ergebniswerten. Als Handlungsalternativen bezeichnet man diejenigen Artefakte, auf deren Anzahl und Zustand der Entscheidungsträger direkt oder indirekt Einfluss nehmen kann. Unter Berücksichtigung der einzelnen Umweltzustände liefert die Ergebnisfunktion dann für jede Handlungsoption die entsprechenden Konsequenzen. Wichtig ist hierbei, dass die Aktionsmöglichkeiten des Entscheidungsfelds alle möglichen Alternativen zu einer Entscheidungsoption umfassen, aber gleichzeitig so gestaltet sind, dass nur eine Möglichkeit davon ausgewählt werden kann. Die Wahl einer Option muss dementsprechend alle anderen Lösungsmöglichkeiten ausschließen (Schneeweiß, 1991; Sieben und Schildbach, 1994). Weiterhin gilt für den Zustandsraum, also die Menge möglicher Zustände der Welt, Umwelt, Natur oder Realität, dass dieser in drei unterschiedliche Situationen zur Bestimmung des Risikos gegliedert ist:

1. *Unsicherheits- bzw. Ungewissheitssituation*: In diesem Fall wird einer der bekannten Zustände eintreten.

2. *Risikosituation*: In diesem Fall werden die bekannten Wahrscheinlichkeit mit einer gewissen Wahrscheinlichkeit eintreten, die entweder subjektiv oder objektiv ermittelt wurde.
3. *Sicherheitssituation*: In diesem Extremfall ist der eintretende Umweltzustand bereits bekannt.

2.1.3.2. Zielfunktion und Zielsystem

Die Zielfunktion, auch Zielsystem genannt, übernimmt die Aufgabe der logischen Komponente in Entscheidungsmodellen. Abhängig von den Präferenzrelationen des Entscheiders hinsichtlich verschiedener Ergebnismerkmale wird mit ihr eine spezifische Handlungsentscheidung festgelegt, die unmittelbar in eine Aktivität umgesetzt werden kann. Die Präferenzrelation drückt dabei die subjektiv empfundenen Unterschiede der Wichtigkeit einzelner Entscheidungsmerkmale aus und ist zusammen mit der Zielgröße ein essentieller Bestandteil der Zielfunktion. Die Zielgröße gibt an, welche direkten Handlungen aus der Bewertung der einzelnen Ergebnisaktionen abzuleiten sind (Manz et al., 1993; Schneeweiß, 1991; Sieben und Schildbach, 1994).

Ein Zielsystem muss in erster Linie vollständig sein, d.h. es müssen sämtliche Möglichkeiten der Ergebnisfunktion bewertbar und eine eindeutige Wertrangfolge der Präferenzrelationen festgelegt sein. Weiterhin müssen sämtliche Ziele operational sein, so dass stets deren Erreichung gemessen und überprüft werden kann. Schließlich muss ein Zielsystem koordinationsgerecht sein, um es in sachlich oder zeitlich gruppierte Teilentscheidungen, die isoliert voneinander bewertet werden können, zerlegen zu können. So wird dem Entscheider ermöglicht, seine persönlichen Wertvorstellungen zu den einzelnen Teilbereichen klar zu dokumentieren und kommunizieren. Die hier beschriebenen Anforderungen an ein Zielsystem resultieren aus der betriebswirtschaftlichen Entscheidungslehre und sind für die Entscheidungsfindung essentiell (Manz et al., 1993; Schneeweiß, 1991; Sieben und Schildbach, 1994).

Um abschließend die Überlegenheit einer Entscheidung gewährleisten zu können, ist es erforderlich, dass sich sämtliche Informationen über die Zielerreichung und Auswahl von Handlungsoptionen verknüpfen lassen. Nur so kann jeder Aktion eine Zahl zugeordnet werden und eine Wertrangfolge aller Alternativen aus

Sicht des Entscheidungsträgers erreicht werden. Aussagen, welche Aktionen gegenüber anderen präferiert werden (sowohl schwach als auch stark) und zwischen welchen der Entscheider indifferent ist, können damit getätigt werden (Pfohl und Braun, 1981). Zusätzlich ermöglicht die Nutzenmessung neben der Aktionenbewertung auch die Evaluation der Nutzenstiftung von Ergebnissen:

- *Ordinale Nutzenfunktion*: Die Reihenfolge der Präferenz von Ergebnissen ist bestimmbar, nicht aber das Maß ihres Abstands.
- *Kardinale Nutzenfunktion*: Neben der Reihenfolge der Präferenz von Ergebnissen ist auch ihr Abstandsmaß bestimmbar.

2.1.4. Vorgehensweise zur Entscheidungsfindung

Aus Prozesssicht wird die Entscheidungsfindung, genauer gesagt das Zusammenspiel zwischen Entscheidungsmodell und dessen Logik, von dem amerikanischen Sozialwissenschaftler und Nobelpreisträger Herbert A. Simon zunächst in drei aufeinanderfolgende Phasen eingeteilt (Simon, 1960, S. 40):

- Intelligenz
- Design
- Auswahl

In der *Intelligenzphase* geht es in erster Linie um die Identifikation von Problemen oder Möglichkeiten. Hierfür sind alle möglichen Methoden zur Gewinnung relevanter Informationen zugelassen. Neben aktivem Zuhören, der Befragung von internen und externen Datenbanken und dem Durchführen von Interviews mit dem relevanten Personenkreis für eine Entscheidung (z.B. Kunden, Lieferanten, Mitarbeitern etc.) sind Umgebungsanalysen oder die Erarbeitung von Lücken zwischen einer bestehenden zu einer angestrebten Situation (vgl. *SWOT*-Analyse zur Identifikation von Stärken, Schwächen, Möglichkeiten und Risiken) adäquate Mittel zur Intelligenzbildung (Simon, 1960).

In der *Designphase* werden Lösungsalternativen hinsichtlich der Entscheidungsvorlage erarbeitet. Aus prozessgetriebener Sicht sind hierfür Methoden wie Brainstorming, Literaturanalyse, Forschungsarbeit oder auch Benchmarkanalysen valide Instrumente der Informationsgewinnung. Zusätzlich ist es in dieser Phase

erforderlich, die beabsichtigten Ziele der Entscheidung klar zu definieren. Das Ergebnis dieser Phase beinhaltet damit die Entwicklung und Finalisierung des Entscheidungsmodells (vgl. Abschnitt 2.1.3).

Die *Auswahlphase* rundet den Entscheidungsfindungsprozess dahingehend ab, indem die Logik des Entscheidungsmodells Anwendung findet und damit sämtliche Handlungsalternativen hinsichtlich der Zielsetzung gegeneinander abgewogen werden. Die dominierende Lösung des Entscheidungsproblems gilt dann als das Resultat der Entscheidungsfindung und findet Anwendung in der praktischen Ausführung oder Umsetzung.

Der Prozess von Simon (1960) wurde im Laufe der Anwendung um zwei weitere Phasen, nämlich *Implementierung* und *Überwachung*, ergänzt (Simon, 1976). In diesen beiden Phasen wird die herausgearbeitete Lösungsalternative zunächst in ein handlungsfähiges Konzept übertragen und schließlich umgesetzt, bevor in der Überwachungsphase kontinuierlich die angestrebte Zielsetzung erreicht wird (Simon, 1976). Neben den vorgestellten Prozessschritten der Entscheidungsfindung wurden in der Vergangenheit Bestrebungen zur Erweiterung des Prozessmodells angestellt. So erwähnen Mintzberg et al. (1976) und Mintzberg (1979) die *Autorisierungsphase*, in der ein ausgewählter Lösungsansatz auf Managementebene zunächst geprüft wird und erst dann möglicherweise für die Implementierung freigegeben wird. Derartige Erweiterungen stellen jedoch nur kleine Prozessergänzungen dar und minimieren daher nicht die Bedeutung der vorgestellten Phasen in ihrer Rolle als die wesentlichen Elemente der Entscheidungsfindung.

2.2. Entscheidungsunterstützungssysteme

Informationssysteme zur Unterstützung von semi-strukturierten Entscheidungen und Problemlösungen werden Entscheidungsunterstützungssysteme (EUS) genannt (Keen und Scott Morton, 1978; Power, 2008; Shim et al., 2002). Sie haben sich aus den Teildisziplinen der organisatorischen Entscheidungsfindung und der Informatik herausgebildet, um strukturierte, halb strukturierte oder unstrukturierte Probleme von Managementaktivitäten, wie beispielsweise strategische Planung, Steuerung des Managements oder die operative Steuerung, zu bearbeiten (Gorry und Scott Morton, 1971; Simon, 1960). Die theoretischen Organisationsstudien wurden zwischen 1955 und 1965 am *Carnegie Institute of Technology*

durchgeführt. Die technologischen Studien hingegen wurden hauptsächlich im Bereich der interaktiven Computersysteme am *Massachusetts Institute of Technology* um 1960 erforscht. Das Konzept der EUS ging dann in der Mitte der Siebzigerjahre als eigenes Forschungsgebiet aus diesen beiden Forschungsrichtungen hervor (Keen und Scott Morton, 1978). Mitte der Achtzigerjahre entstanden aus den Einzellösungen an EUS die Führungsinformationssysteme sowie gruppenbasierte und organisatorische EUS. Mit Beginn der Neunzigerjahre kamen dann zusätzliche Systeme wie Datenlager oder direkt auswertbare Prozesssysteme hinzu, welche in den letzten Jahren durch webbasierte Analysesysteme ergänzt wurden (Power, 2008).

Grundlegende Definitionen für EUS unterscheiden sich zunächst hinsichtlich der jeweiligen Begrifflichkeiten. So ist es nach Keen und Scott Morton (1978) unmöglich, von EUS zu sprechen, da bei diesem Begriff drei weitgehend unterschiedliche Verwendungen von Aufwand zu betrachten sind: *Entscheidungen* stehen in diesem Zusammenhang für die nicht technischen aber funktionalen und analytischen Aspekte der EUS sowie für die Auswahlkriterien von Handlungsalternativen. *Unterstützung* bezieht sich hierbei auf die Implementierung und das Verständnis dafür, wie Menschen handeln und wie diese dabei unterstützt werden können. *Systeme* meinen in diesem Zusammenhang, die Fähigkeiten von Design, Implementierung und Technologie maximal auszuschöpfen (Keen und Scott Morton, 1978).

Eine weitere Definition von Sprague (1980) klassifiziert EUS nach ihren Eigenschaften: Mit derartigen Systemen werden in erster Linie wenig strukturierte und unspezifizierte Probleme höherer Managementebenen adressiert. Die EUS kombinieren die Verwendung von Entscheidungsmodellen und Analysetechniken mit traditionellen Funktionen des Datenzugriffs und -abrufs. Dabei wird besonders Wert auf die Einfachheit der interaktiven Nutzung, auch für Computerlaien, gelegt. Zusätzlich zeichnen sich solche Systeme durch ihre Flexibilität bei der Anpassung an sich schnell verändernde Umgebungsvariablen sowie Änderungen am Entscheidungsansatz der Anwenders aus (Sprague, 1980; Turban, 1990). Sie werden weiterhin mit einer verbesserten Darstellung und Verarbeitung von Informationen in Verbindung gebracht, so dass die Entscheidungsfindung produktiver, agiler, innovativer und seriöser bzw. objektiver abläuft (Burstein und Holsapple, 2008). Arnott und Pervan (2008) bezeichnen EUS als die Entwicklung und Aus-

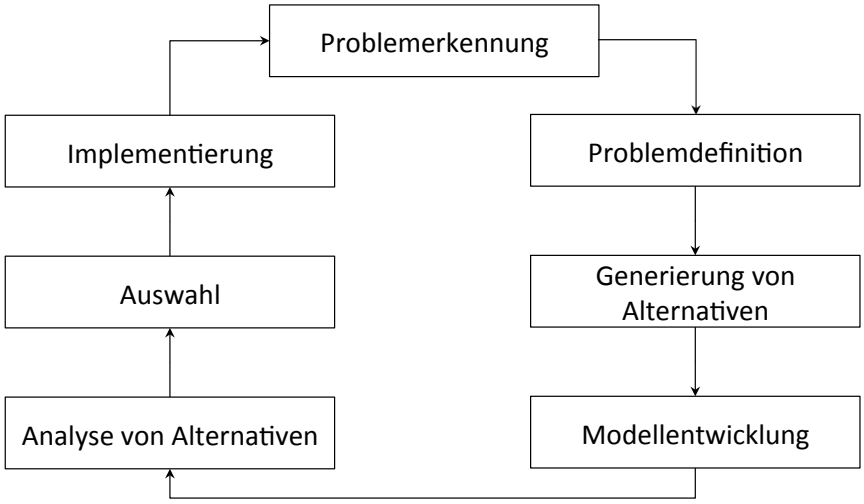

Abbildung 2.2.: Entscheidungsprozess von EUS nach Shim et al. (2002)

lieferung von Informationssystemen, welche von menschlichen Akteuren dazu benutzt werden, bessere Entscheidungen zu treffen. Genauer gesagt verhelfen EUS den Anwendern zur optimierten Entscheidungsfindung (Arnott und Pervan, 2008; Shim et al., 2002).

Sämtliche Umsetzungen von EUS umfassen neben dem Aufspüren von Problemen, der Erarbeitung von Entscheidungsalternativen und der Analyse bzw. Auswahl einer geeigneten Lösung schließlich auch deren Implementierung (vgl. Abbildung 2.2). Dafür gliedern Shim et al. (2002) EUS in drei Kernkomponenten:

1. Datenbankverwaltung mit Zugang zu internen und externen Daten sowie codiertem Wissen;

2. Umfangreiche Modellierungsfunktion und Verwaltung der Modelle;

3. Einfache, aber ausgereifte Anwenderschnittstelle für interaktive Abfragemöglichkeiten, Berichte und grafische Auswertungen.

Als besondere Herausforderung für EUS gilt es, die unstrukturierte Datengrundlage für ein Computersystem verwendbar und analysierbar zu machen, um sowohl die Effizienz, mit der die Anwender eine Entscheidung treffen, als auch die Effektivität dieser Entscheidungen zu steigern (Shim et al., 2002). Dadurch

lassen sich spezifische Entscheidungsprobleme im Unternehmenskontext gezielt aufbereiten und schneller entscheiden (Gorry und Scott Morton, 1971).

Die kontinuierliche Weiterentwicklung von Computersystemen hinsichtlich ihrer Leistungsfähigkeit, ihrer Vernetzung, den optimierten Eingabemöglichkeiten und ihrer Bauweise hat dazu beigetragen, dass diese sich immer besser für EUS eignen. Mit gesteigerten Rechenkapazitäten können schnellere und umfangreichere Lösungsmöglichkeiten angeboten werden. Neue Interaktionsmöglichkeiten mit dem Anwender liefern weiterhin effektivere Bedienmasken zur schnelleren Bearbeitung von Entscheidungsproblemen. Zusätzlich ermöglicht das Internet und die kabellose Vernetzung von Computern den unbeschränkten Zugang zu Analysetools für Entscheider. Die ortsunabhängige Abarbeitung von Aufgaben trägt dabei zur schnelleren Entscheidungsfindung bei (Earle und Keen, 2000).

Weiterentwicklungen von klassischen EUS sind gruppenbasierte Entscheidungssysteme, die durch die starke Vernetzung über das Internet und die Bildung virtueller Teams entstanden sind. Durch diese Form der Entscheidungsunterstützung werden hilfreiche Techniken, wie z.B. gemeinsames Brainstorming oder kollaborative Ideenbewertung ermöglicht, die zur Entscheidungsfindung in der Gruppe beitragen. Eine besondere Form dieser Systeme, die aber nur einer kleinen Personengruppe im Unternehmen zur Verfügung steht, sind Führungsinformationssysteme, welche unter Einwirkung von Techniken der künstlichen Intelligenz besseren Entscheidungssupport für die Unternehmenslenker bereitstellen (Bonczek et al., 1981; Courtney und Paradice, 1993).

2.2.1. Typen von Entscheidungsunterstützungssystemen

Die Vielfalt an Definitionen für EUS deutet darauf hin, dass es unterschiedliche Taxonomien zur Einordnung dieser Systeme gibt. So können EUS z.B. hinsichtlich ihres Anwendungsbereichs (Power, 1997), ihrer Nutzer (Hättenschwiler, 2001) oder ihres Unterstützungsgrads (Power, 2002) klassifiziert werden. Auf der Anwenderebene wird zwischen aktiven, passiven und kooperativen EUS unterschieden (Hättenschwiler, 2001).

Die passive Form ist ein System, das den Prozess der Entscheidungsfindung unterstützt, aber keine expliziten Lösungsvorschläge hervorbringt. Das aktive System hingegen schlägt spezifische Lösungsmöglichkeiten für das Entscheidungs-

problem vor. Kooperative Systeme hingegen erlauben die Interaktion mit dem Anwender, der zusätzliche Veränderungen an einer vorgeschlagenen Lösung vornehmen kann und diese anschließend zur Validierung wieder an das System zurückgibt. Dies wird iteriert bis eine konsolidierte Lösung vorliegt.

Auf der Anwendungsebene wird zwischen unternehmensweiten EUS und Einzelplatzlösungen unterschieden. Im Gegensatz zur Einzellösung, die unabhängig vom Firmennetzwerk auf dem eigenen Computer eines Managers gespeichert und genutzt werden kann, greifen unternehmensweite Lösungen auf sämtliche Datenbanken der Firmensysteme zurück, um eine Vielzahl von Informationen in die Entscheidungsfindung mit einfließen zu lassen (Power, 1997).

Hinsichtlich ihres Unterstützungsgrads wird zwischen kommunikations-, daten-, dokumenten-, wissens- und modellzentrischen EUS unterschieden (Power, 2002). Für die Entscheidungsunterstützung im Outsourcingkontext werden gut vernetzte EUS benötigt, um die Vielzahl an relevanten Informationen aus Systemen und von Wissensträgern beschaffen und in die Entscheidung mit einfließen lassen zu können. Daher werden nun diejenigen Typen von EUS näher erläutert, die charakteristisch für die Vernetzung von Systemen sind und Kollaboration zwischen Entscheidungsbeteiligten ermöglichen.

2.2.1.1. Datenzentrische Entscheidungsunterstützung

Bei der datenzentrischen Entscheidungsunterstützung steht die Analyse und Verarbeitung von großen Datenmengen zur Problem- und Entscheidungsfindung im Vordergrund. Dabei werden Schlüsseltechnologien wie analytische Datenverarbeitung (*Online Analytical Processing – OLAP*) oder Datenlager (*Data Warehouses*), Mustererkennungsverfahren (*Data Mining*) und vernetzte EUS eingesetzt, auf die im weiteren Verlauf kurz eingegangen wird.

Datenlager basieren auf verfügbarer Datenbanktechnologie und vereinigen relevante Transaktionsdaten aus operativen Systemen eines Unternehmens in einer zentralen Datenbank für Managemententscheidungen. Dabei werden nicht die gesamten Daten aus den operativen Bereichen kopiert, sondern die vorhandenen Daten themenbezogen (Produkt, Lieferant, Kunde usw.) in der Managementdatenbank abgelegt, so dass diese integriert betrachtet und damit einheitlich über die Systeme hinweg nachvollzogen werden können (Totok, 2001). Der zentrale

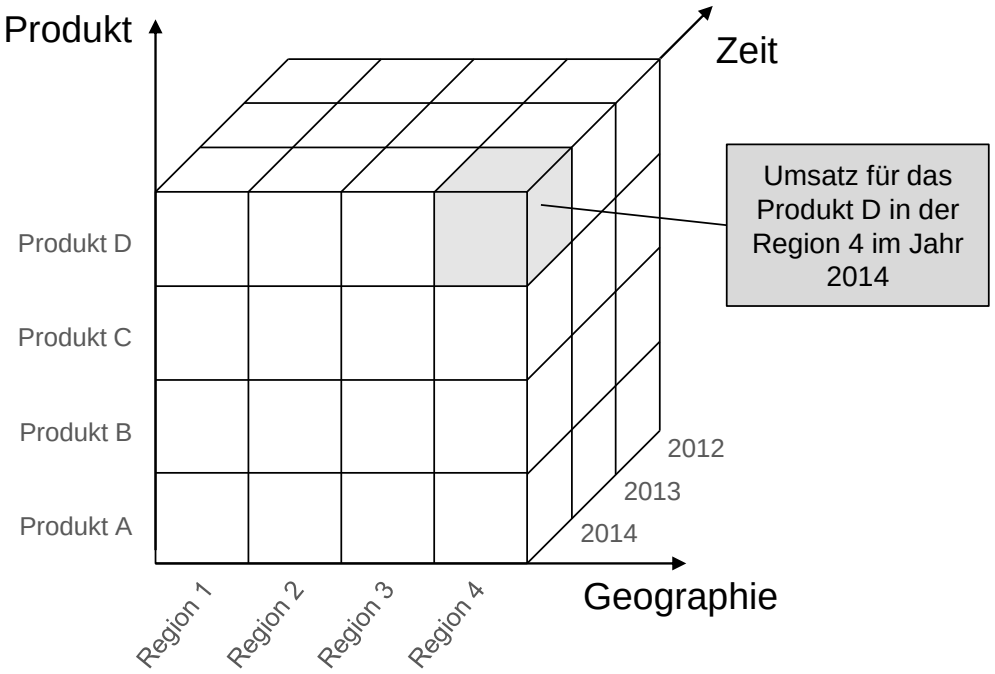

Abbildung 2.3.: *OLAP*-Anaylse

Datenspeicher ist nicht volatil und versieht alle Einträge mit einem Zeitstempel, so dass eine zeitlich differenzierte Analyse der Datensätze möglich ist (vgl. Abbildung 2.3).

Als Auswertungsverfahren für die großen eingelagerten Datenmengen kann OLAP, ein Analyseverfahren zur subjektiven Auswertung zeitlich geordneter Daten aus Datenlagern, in Betracht gezogen werden. Das analytische Onlineverfahren wird auf leistungsfähigen Servern bereitgestellt und profitiert nicht nur von Leistung besonders performanter Computersysteme, sondern glänzt auch durch einen schnellen, konsistenten und interaktiven Zugang zu großen Datenbergen. Dies wird durch eine mehrdimensionale Sicht auf im Vorfeld definierte Datenpunkte garantiert (Inmon, 2005).

Die zunehmende Digitalisierung von Unternehmen und deren starke Vernetzung durch das Internet lassen die zu analysierenden Datenmengen überproportional anwachsen. Um dieser Entwicklung gerecht zu werden, werden Verfahren aus dem *Data Mining* angewendet, die Methoden der künstlichen Intelligenz mit statistischen Werkzeugen verknüpfen. Hierbei wird mit intelligenten Algorithmen

versucht, Muster zur Entscheidungsunterstützung in Echtzeit aus Datenströmen herauszulesen und die Datenflut bereits bei der Entstehung zu filtern (Shaw et al., 2001).

Die vernetzten EUS sind entwickelt worden, um die technischen Barrieren zur Bereitstellung von entscheidungsrelevanten Informationen für Manager und Entscheider in verteilten Lokationen zu überwinden. Der besondere Vorteil liegt in der Möglichkeit, Manager eines Unternehmens in einem eigenen Intranet unabhängig von ihrem Aufenthaltsort mit Entscheidungsinformationen in Echtzeit auszustatten (Shim et al., 2002).

2.2.1.2. Kollaborative Entscheidungssysteme

Die Entscheidungsfindung in Unternehmen findet nicht nur durch Einzelpersonen statt, sondern vermehrt durch Gruppenentscheidungen, die von selbstregulierenden und eigenständig arbeitenden Teams als Konsens getroffen werden (Eisenführ et al., 2010). Dies trifft besonders bei Aufgaben der Softwareentwicklung zu, da hier in gegenseitiger Zusammenarbeit im Team an einem gemeinsamen Ziel mit vorgegebenen Methoden gearbeitet wird. Dabei sind die Ziele der Individuen nicht einheitlich, aber überlappen zu einem bestimmten Grad. Aus diesem Grund ist es für das Treffen von Entscheidungen in der Gruppe erforderlich, dass die individuellen Ziele koordiniert und harmonisiert werden, um so die gemeinsamen Aufgaben erledigen zu können (DeSanctis und Gallup, 1987; Malone und Crowston, 1994).

Die geografische Streuung von Teams und deren Teammitgliedern macht die Verwendung von computergestützten Kollaborationssystemen notwendig, um einen gemeinsamen Konsens bei Entscheidungen eines Teams zu ermöglichen. Hierfür bedient man sich Informationssystemen zur synchronen (kurz und unstrukturiert) und asynchronen (ausführlicher und strukturiert) Kommunikation, die ebenfalls bei der verteilten Softwareentwicklung zum Einsatz kommen (vgl. Kapitel 2.3.4). Die so erzielte Steigerung der Kommunikation innerhalb der Gruppe führt damit zu einer besseren Effektivität bei der gruppenbasierten Entscheidungsfindung (Warkentin et al., 1997).

Zur verbesserten Entscheidungsunterstützung virtueller Teams müssen daher Kollaborationssysteme eingesetzt werden, die eine optimale Kommunikations-

struktur zwischen den Teammitgliedern erzeugen, so dass persönliche Abstimmungen durch intermediäre computergestützte Kommunikationstools ersetzt werden. Durch sie werden akzeptable und zufriedenstellende Kommunikationswege für Gruppen geschaffen, die einerseits ihr Aufgabenfeld (wie Softwareentwicklung) optimal abdecken und andererseits eine solide Grundlage für das kollaborative Treffen von Entscheidungen bieten (Walther und Burgoon, 1992; Warkentin et al., 1997; Zack, 1993).

2.2.1.3. Entscheidungsunterstützung für das Outsourcing

Zur Entscheidungsunterstützung beim IT-Outsourcing bedient man sich Methoden, die teilweise in Informationssystemen umgesetzt sind. Da Outsourcingentscheidungen nicht nur auf einem Auswahlkriterium beruhen, werden hierfür Mehrzielentscheidungsmodelle (als konkrete Instanz der in Kapitel 2.1.3 eingeführten Entscheidungsmodelle) angewendet. Dabei unterscheidet man zwischen multikriteriellen Entscheidungsmodellen (MADM - *Multi Attribute Decision Making*) und multiobjektiven Entscheidungsmodellen bei unterschiedlichen Zielsystemen (MODM - *Multi Objective Decision Making*). MADM dienen zur Auswahl einer bestimmten Handlungsoption durch Bewertung mehrerer im Vorfeld gegebener Kriterien, während MODM zur Berechnung einer bevorzugten Handlungsoption aus einem Lösungsraum unbekannter Größe herangezogen werden (Figueira et al., 2005a).

Multiobjektive Verfahren Im Vergleich zu den auf Attributen basierenden Entscheidungsmodellen gibt es bei den multiobjektiven Verfahren keine finale Menge an Handlungsalternativen für die gegebene Problemstellung. Aus diesem Grund werden mathematische Verfahren benötigt, um sich mögliche und optimale Handlungsalternativen berechnen zu können. Für jede dieser Alternativen wird der Grad berechnet, inwieweit die Lösung die Anforderungen an eine Handlungsalternative abdeckt. Die Alternative mit dem höchsten Zufriedenheitsgrad gilt schließlich als zu wählende Handlungsalternative (Jahn, 2004). Da im Falle der komponentenbasierten Outsourcingentscheidung die Handlungsalternativen (endliche Menge an Softwarekomponenten, die jeweils entweder selbst entwickelt

oder ausgelagert werden) bereits im Vorfeld bekannt sind, wird auf dieses Entscheidungsverfahren nicht weiter eingegangen.

Multikriterielle Verfahren Bei den multikriteriellen Entscheidungsverfahren werden mehrere Eigenschaften, die auf die Entscheidung einwirken, in Betracht gezogen. Dabei können die Eigenschaften sogar teilweise im Widerspruch zueinander stehen, charakterisieren aber dennoch die möglichen Alternativen des Entscheidungsproblems. Daher werden bei MADM die Alternativen hinsichtlich ihrer Wichtigkeit sortiert und weniger priorisierte Lösungsmöglichkeiten eliminiert. Der Entscheider selbst übernimmt hierbei die Aufgabe des Bewerters von subjektiven Eigenschaften und von Gewichtungen für die Sortierung (Chen et al., 1992; Zangemeister, 1976).

Bekannte Verfahren der multikriteriellen Entscheidungsfindung lassen sich bezüglich der Art bekannter Informationen über das Entscheidungsproblem klassifizieren. So wird zwischen Methoden differenziert, bei denen Details über die möglichen Lösungsalternativen oder über die Attribute für die Entscheidungsfindung bekannt sind. Die Einzigartigkeit von Outsourcingproblemen in Softwareentwicklungsprojekten macht eine Einschätzung und den Vergleich von Lösungsalternativen unmöglich, da der Erfolg der Alternativen erst im Nachgang an das Projekt bewertet werden kann. Daher werden Verfahren eingesetzt, bei denen die ordinalen Präferenzen der Attribute von Entscheidern vorliegen oder bei denen über kardinale Informationen Präferenzen berechnet werden können. In Tabelle 2.1 werden die für Outsourcingentscheidungen verwendeten multikriteriellen Verfahren, gruppiert nach der Informationsqualität ihrer Attribute, aufgezeigt.

Tabelle 2.1.: Methoden der multikriteriellen Entscheidungsverfahren

Qualität der Attributinformationen	**Verfahren**
Ordinale Information	Aspektweise Eliminierung
	Entscheidungsbaum
	Entscheidungsmatrix/-tabelle
	Lexikografische Methode
Kardinale Information	Analytischer Hierarchieprozess
	ELECTRE
	PROMETHEE
	SMART, SMARTS, SMARTER

Aspektweise Eliminierung Die aspektweise Eliminierung von Handlungsalternativen kombiniert die lexikografische Methode mit verbindenden Entscheidungsregeln. Dabei werden die Alternativen einerseits wie bei der lexikografischen Methode entsprechend der wichtigsten Attribute sortiert, aber andererseits auch mit den Alternativen vermischt, die gewisse Mindestanforderungen erfüllen (Kahraman, 2008; Ranyard, 1976; Tversky, 1972).

Entscheidungsbaum Diese Methode ermöglicht das Aufteilen eines komplexen Systems in eine Vielzahl einfacherer Teilprobleme. Sie wird besonders bei Expertensystemen und neuronalen Netzen verwendet. Die ebenfalls visuelle Methode startet mit einem Wurzelknoten und dem ersten Entscheidungskriterium (vgl. Abbildung 2.4). Anhand spezifischer Bedingungen gelangt man zum nächsten Entscheidungsknoten, bis man schließlich am Ende eines Astes angekommen ist, welches die Handlungsalternative repräsentiert (Dietterich, 2000; Eisenführ et al., 2010; Huysmans et al., 2011; Lee, 2010).

Entscheidungsmatrix/-tabelle Hierbei werden die Handlungsalternativen basierend auf der gewichteten Bewertung von outsourcingrelevanten Attributen miteinander verglichen. Die visuelle Methode hilft dabei, die Handlungsalternative mit der besten Bewertung zu identifizieren (Eisenführ et al., 2010; Mullur et al., 2003; Ranyard et al., 1997).

Lexikografische Methode Bei dieser Methode werden zunächst die Attribute hinsichtlich ihrer Wichtigkeit sortiert, bevor ihre Handlungsalternativen verglichen werden. Ein Vergleich der möglichen Alternativen findet zunächst auf Basis des wichtigsten Attributs statt. Sofern hier mehrere Alternativen gleich gut abschneiden, wird das zweitwichtigste Attribut zum Vergleich der übriggebliebenen Alternativen herangezogen. Dies wird so oft wiederholt, bis nur eine Alternative übrig bleibt (Blume et al., 1991; Fishburn, 1974, 1980; Kohli und Jedidi, 2007; Svenson, 1979; Tversky, 1969).

Analytischer Hierarchieprozess (AHP) Die strukturierte Entscheidungsmethode basiert auf paarweisen Vergleichen von Handlungsalternativen und liefert eine sortierte Liste hinsichtlich ihrer Eignung für das Entscheidungsproblem. Dabei werden die Alternativen hinsichtlich ihrer Attribute paarweise eingeschätzt (Saaty, 1986, 2003, 2005; Wind und Saaty, 1980).

ELECTRE Diese Methode belegt Handlungsalternativen mit zwei Werten, einen für die Übereinstimmung (Konkordanz) und einen weiteren für die Gegenseitigkeit (Diskordanz), beim paarweisen Vergleich. Es wird schließlich die Handlungsalternative ausgewählt, deren Werte im Grenzbereich für Konkordanz und Diskordanz liegen (Figueira et al., 2005b). Sie ist aus dem Französischen „***EL****imination* ***E****t* ***C****hoix* ***T****raduisant la* ***RE****alité*“ abgeleitet.

PROMETHEE Mit der Methode PROMETHEE, die ihren Namen aus dem Englischen „***P****reference* ***R****anking* ***O****rganization* ***METH****od for* ***E****nrichment of* ***E****valuations*“ ableitet, werden klare mathematische Regeln für den paarweisen Vergleich von Alternativen verwendet. Eine saubere Berechnung der besten Alternative wird damit gewährleistet und als Vorteil gegenüber den anderen Methoden angeführt (Brans und Vincke, 1985; Brans et al., 1986).

SMART Dies ist eine einfache multikriterielle Entscheidungsmethode für Entscheidungen unter Risiko. Hierbei werden Kompromisse bei mehreren Attributen, die sich in ihrer Gewichtung nur wenig unterscheiden, abgewogen. Zusätzliche Entscheidungsregeln erlauben eine wertbasierte Einschätzung von Attributen, wenn eine Sortierung sonst nicht möglich ist (Edwards, 1977; Edwards und Barron, 1994). Der Name leitet sich aus englischen Adjektiven zur Beschreibung von Unternehmenszielen ab: „***S****mart,* ***M****easurable,* ***A****ssignable,* ***R****ealistic,* ***T****ime-related*“.

Alle vorgestellten Verfahren haben haben die Gemeinsamkeit, dass für alle Handlungsalternativen Nutzwerte berechnet werden. Sie bauen damit auf der Nutzwertanalyse (Zangemeister, 1976) auf und ermöglichen ein Ranking der subjektiven Güte von Lösungen. Bei den ordinalen Verfahren oder dem analytischen Hierarchieprozess sind dem Entscheider die Attribute und deren Gewichtungen bestens bekannt. Er kann so die Handlungsalternativen entsprechend dieser Kriterien bewerten. Bei den verbleibenden kardinalen Methoden sind dem Entscheider die Charakteristiken nicht zu Beginn des Entscheidungsprozesses offensichtlich, daher werden Handlungsalternativen sukzessive ausgeschlossen (Brans et al., 1986).

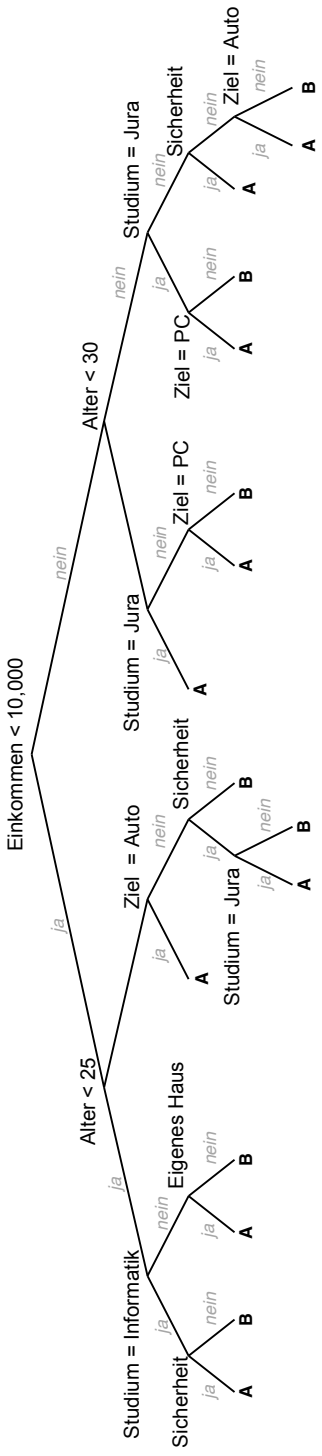

Abbildung 2.4.: Beispiel eines Entscheidungsbaums

2.2.2. Mobile Entscheidungsunterstützungssysteme

Die Auswahl und Einbeziehung multipler Entscheidungskriterien zeichnen EUS aus und verhelfen zu optimalen Entscheidungen im Unternehmenskontext (vgl. Kapitel 2.2). Neben der Berücksichtigung mehrerer Entscheidungsträger im Lösungsprozess werden auch zahlreiche Informationen über die Rahmenbedingungen des Problems mit einbezogen, die meist durch Informationssysteme verarbeitet und bereitgestellt werden. Informationen lassen sich durch mobile Technologien heutzutage von nahezu überall abrufen, verarbeiten und bereitstellen (Pérez et al., 2010). Die Vorteile einer derartigen Technologie liegen auf der Hand: Mit mobilen Endgeräten ist die Informationsbereitstellung überall und jederzeit möglich, so dass Entscheidungen unabhängig von Ort und Zeit informiert oder sogar getroffen werden können. Gruppenentscheidungen werden erleichtert, weil sich die Konsensfindung zwischen den Beteiligten durch den schnelleren Austausch über mobile Geräte vereinfacht. Insgesamt wird dadurch der Entscheidungsprozess beschleunigt, weil kontinuierlich die aktuellsten Informationen vorliegen und untereinander ausgetauscht werden (Pérez et al., 2010; Shim et al., 2002).

Für die Verwendbarkeit von mobilen Endgeräten im Zusammenhang mit EUS werden im Verlauf dieses Kapitels zunächst die Eigenschaften mobiler Technologien vorgestellt, bevor die Unterschiede mobiler Endgeräte und deren Einsatzmöglichkeiten näher erläutert werden. Abschließend werden deren Tauglichkeit und Einsatzgebiete für die Entscheidungsunterstützung näher erläutert.

2.2.2.1. Endgeräte für den mobilen Einsatz

Obwohl mobile Endgeräte heutzutage zahlreich in privaten und geschäftlichen Umfeld in Verwendung sind, gibt es keine eindeutige Definition für mobile Technologien. Dennoch wird darunter weitläufig der Einsatz von Endgeräten verstanden, die auf mobile Dienste zugreifen (Tarasewich et al., 2002, S. 42). Darunter fallen Endgeräte wie Laptops, *Personal Digital Assistants* (PDAs), Smartphones, Tablets sowie Mobilfunkgeräte (Gorlenko und Merrick, 2003). Deren zentrale Unterscheidungsmerkmale, die sie auch von anderen Computersystemen abgrenzen, lassen sich in den nachfolgenden sechs Kategorien zusammenfassen.

Performanz Mobile Endgeräte besitzen im Vergleich zu schwer tragbaren stationären Geräten nur begrenzte Hardwareressourcen und eine geringere Leistungsfähigkeit. Aufgrund ihrer kompakten Bauweise sowie der eingeschränkten Batteriekapazität, müssen Einschränkungen bei der Performanz hinsichtlich der Rechen- und Speicherkapazität hingenommen werden (Adipat und Zhang, 2005; Satyanarayanan, 1996).

Dateneingabe Durch das Aussehen und die Bauweise von mobilen Endgeräten wird nicht nur die Leistung derartiger Geräte eingeschränkt, sondern auch die Möglichkeit der Dateneingabe. Auf dem geringen verfügbaren Platz kann keine Tastatur in Standardformat untergebracht werden, die sonst als gewohntes Eingabegerät bei Computern zur Verfügung steht. Trotz intelligenter Lösungsansätze, wie beispielsweise virtuelle Tastaturen, Autokorrektur und automatische Textergänzungen, Spracheingabe oder Gesten, gestaltet sich die Dateneingabe bei längeren Texten dennoch schwierig. Eingaben sind daher nur langsam und fehlerbehaftet im Vergleich zur gewohnten Dateneingabe am Computer möglich. Gleichzeitig stellt der Kontext, in dem mobile Technologien verwendet werden, eine weitere Herausforderung bei der Dateneingabe dar. So interagieren die Nutzer mit den Geräten meist beim Gehen oder mit nur einer Hand (Adipat und Zhang, 2005). Neue Entwicklungen wie berührungsempfindliche Anzeigen (*Touchscreens*) offenbaren zusätzliche und bessere Möglichkeiten zur Interaktion. Der Anwender kann direkt mit der Eingabeschnittstelle interagieren. Die Anwendungen müssen dafür jedoch auf die neuen Eingabemöglichkeiten ausgerichtet sein, um den Eingabeaufwand des Anwenders auf ein Minimum zu reduzieren und gleichzeitig die Nutzungsgeschwindigkeit zu maximieren (Balagtas-Fernandez et al., 2009).

Konnektivität Mobile Endgeräte sind in erster Linie in ihrer Funktionsweise abhängig von Mobilfunknetzwerken, die als Verbindungsträger zur vernetzten Welt dienen. Trotz ständiger Weiterentwicklung der drahtlosen Kommunikationsnetze, sind schlechte Verbindungen zum Netzwerk für mobile Endgeräte keine Seltenheit. Für eine gute Anwenderzufriedenheit müssen daher die Latenzzeiten, die Bandbreite des Netzwerks sowie die Zuverlässigkeit bei der Kommunikation beachtet werden (Adipat und Zhang, 2005; Satyanarayanan, 1996).

Mobilität Wie der Name bereits vermuten lässt, stellt die Eignung zur portablen Nutzung eine wichtige Eigenschaft mobiler Endgeräte dar. Um dies zu gewährleisten, müssen mobile Geräte leicht sein, eine zufriedenstellende Batteriekapazität aufweisen und somit ein komfortables Nutzungserlebnis erzeugen (Abrahamsson et al., 2004; Satyanarayanan, 1996). Die Mobilitätseigenschaft wirkt sich allerdings negativ auf die Performanz aus, so dass ein leistungsfähiges mobiles Endgerät nur eine geringe Laufzeit pro Akkuladung besitzt oder ein höheres Gewicht infolge einer größeren Batterie zur Konsequenz hat. Mobilität hat nicht nur direkte Auswirkungen auf die anderen Eigenschaften mobiler Endgeräte, sondern stellt auch Anforderungen an das Umfeld des Nutzers. Die stark verbreitete Nutzung dieser Geräte macht es daher erforderlich, Störfaktoren wie blitzende oder blinkende Lichter sowie ständige Geräusche und Töne, die das Gerät von sich gibt, zu vermeiden und diese nur mit Bedacht einzusetzen (Adipat und Zhang, 2005). Weiterhin stellt die kompakte Bauweise eine Herausforderung an den Diebstahlschutz und somit an die Datensicherheit dar, was besonders beim Einsatz im geschäftlichen Umfeld als kritisch zu bewerten ist (Satyanarayanan, 1996).

Kontexterkennung Der Einsatz zahlreicher Sensoren in mobilen Endgeräten ermöglicht die Bereitstellung und Verwendung von kontextsensitiven Daten in Echtzeit. Diese Daten können für Applikationen, die auf dem Endgerät bereitgestellt werden, verwendet werden. Sie reduzieren und erleichtern damit den Eingabeaufwand des Anwenders, wenn Kontextdaten (z.B. Aufenthaltsort) benötigt werden. Zusätzlich zu den Kontextdaten der Umwelt nehmen Kontextdaten des Anwenders eine wichtige Rolle bei der Kontexterkennung ein. Die meist eindeutige Zuordnung eines mobilen Endgeräts zu einer Person erlaubt es daher, Kontextdaten zur Nutzungsdauer von Applikationen sowie dem Nutzerverhalten allgemein zu erheben und diese kontextspezifisch auszuwerten (Butter, 2009; Hofer et al., 2003).

Benutzerschnittstelle Die Darstellung von Inhalten und Interaktion mit dem Nutzer geschieht über einen Bildschirm, der bei mobilen Endgeräten aufgrund ihrer Bauweise (vgl. Abbildung 2.5) in der Größe beschränkt ist (Hofer et al., 2003). Bei Smartphones beträgt die Bildschirmdiagonale zwischen drei und fünf Zoll (ca. 7,6 - 12,7 cm) und bei Tablets zwischen zwischen sieben und elf Zoll (ca. 17,7 -

Abbildung 2.5.: Bauweise mobiler Endgeräte

28 cm). Dies hat zur Konsequenz, dass die Endgeräte mit unterschiedlichen Bildschirmauflösungen betrieben werden. Dadurch steht zur Anzeige und Interaktion nur begrenzt Platz zur Verfügung, der je nach Gerätetyp sogar variieren kann (Adipat und Zhang, 2005).

2.2.2.2. Unterschiede zwischen mobilen Endgeräten

Als Konsequenz des Fortschritts im Bereich der mobilen Technologien haben sich die Fähigkeiten von Smartphones und Tablets in den letzten Jahren enorm erweitert. Diese Endgeräte sind mittlerweile in der Lage, immer mehr Funktionen, die zuvor nur auf stationären Computern oder Laptops verfügbar waren, bereitzustellen. Zusätzlich kombinieren sie die klassischen Telefonfunktionen eines Mobilfunkgerätes mit einer intuitiven Bedienung (Goetz et al., 2012). Obwohl Laptops für den mobilen Einsatz konzipiert wurden, gelingt es ihnen nicht den Grad an Mobilität aufzuweisen, wie das bei Tablets und Smartphones der Fall ist, welche wesentlich kleiner, handlicher und ständig betriebsbereit sind. Dennoch sind die klassischen Mobiltelefone Spitzenreiter in dieser Kategorie, da sie mit kompakter Bauweise und langer Batteriedauer die Kriterien der Mobilität am besten abdecken. Bei der Performanz verhält es sich genau umgekehrt, da Laptops bedingt durch ihre Bauart mehr Speicher und Rechenkapazität bereitstellen können als Smartphones und Tablets und letztere wiederum mehr als Mobiltelefone. Das gleiche Bild zeichnet sich auch bei den Kategorien Dateneingabe und Displaygröße ab. Hier wirkt sich die immer kompaktere Bauweise aus. Je kleiner die Geräte werden, umso weniger Platz ist für Prozessoren, Bildschirme und Eingabegeräte. Hinsichtlich der Konnektivität sind die Mobiltelefone, Smartphones und Tablets den Laptops deutlich überlegen. Einerseits bieten sie mehrere Ver-

bindungsmöglichkeiten zu kabellosen Netzwerken (WLAN und Mobilfunknetze) und andererseits sind sie ununterbrochen angeschaltet und mit dem Datennetzwerk verbunden. Ein Laptop hingegen trennt sich vom Netzwerk, wenn er in den Ruhemodus geschaltet wird. In der Kategorie Kontexterkennung sind die Smartphones und Tablets führend, da sie mit ihren zahlreich eingebauten Sensoren am besten Kontextinformationen bereitstellen können und gegenüber Mobiltelefonen aufgrund ihrer besseren Performanz überlegen sind (Székely et al., 2013; Wich und Kramer, 2015).

Zusammenfassend kann festgehalten werden, dass sich die neuste Generation mobiler Endgeräte (Smartphones und Tablets) besonders bei der Kontexterkennung von Laptops oder Mobiltelefonen unterscheidet, während ihre Funktionalitäten in den anderen Kategorien oftmals ähnlich sind. Besondere Stärken der Kontexterkennung sowie die überdurchschnittlichen Eigenschaften in den restlichen Kategorien können Smartphones und Tablets bei der Benutzerfreundlichkeit ihrer Anwendungen ausspielen. Daher wird bereits beim Entwicklungsprozess von Anwendungen für diese mobilen Geräte verstärkt auf das Nutzererlebnis geachtet (Wich und Kramer, 2015).

2.2.2.3. Mobile Geschäftsanwendungen

In den beiden vorherigen Unterkapiteln wurden die Eigenschaften mobiler Endgeräte und die Unterschiede der neusten Generation zu Laptops oder Mobiltelefonen erörtert. In diesem Unterkapitel hingegen wird nun der Fokus auf die Software, insbesondere Geschäftsanwendungen, gelegt, welche für die Nutzung mit Smartphones oder Tablets bestimmt ist. Hierfür wird zunächst ihr Einsatzkontext erläutert.

Mobiler Geschäftsverkehr (*Mobile Business*) ist in der Literatur nicht eindeutig definiert, dennoch werden geschäftliche Interaktionen anhand spezieller Eigenschaften als solcher klassifiziert. Zu dieser Kategorie von Geschäften zählen daher die Dienstleistungen, die einerseits überall verfügbar sind wo ein Anwender Zugriff auf sein mobiles Endgerät hat. Andererseits müssen die angebotenen Dienste Geschäftsprozesse abbilden, an denen alle Marktteilnehmer (Kunden, Unternehmen, Regierungen) beteiligt sein können (Königstorfer, 2008).

Mobile Geschäftsanwendungen nehmen im Kontext des mobilen Geschäftsverkehrs eine besondere Rolle ein, da diese erst den Zugriff auf mobile Dienste ermöglichen. Mobile Geschäftsanwendungen zeichnen sich durch zahlreiche Charakteristiken aus, die in die nachfolgenden Kategorien gruppiert werden können (Wich und Kramer, 2015).

Ubiquität. Die mittlerweile allgegenwärtige Verbreitung von mobilen Endgeräten sorgt dafür, dass mobile Geschäftsanwendungen omnipräsent und nutzbar sind. Der Zugang zu geschäftlichen Informationen und die Kommunikation mit bereitgestellten Diensten ist damit in Echtzeit und unabhängig vom Aufenthaltsort möglich (Alahuhta et al., 2005; Xiaojun et al., 2004).

Erreichbarkeit. Ein Anwender ist immer erreichbar unabhängig von Ort und Zeit. Der Nutzer hat jedoch die Möglichkeit, die Erreichbarkeit abhängig von Personen oder Uhrzeiten zu definieren (Alahuhta et al., 2005; Nah et al., 2005; Xiaojun et al., 2004).

Komfort. Die Fähigkeit von mobilen Endgeräten Daten zu speichern, erlauben es, diese Daten auch überall und jederzeit abzurufen. Somit wird ein bequemer und einfacher Zugang zu mobilen Geschäftsanwendungen ermöglicht (Xiaojun et al., 2004).

Personalisierung. Üblicherweise wird ein mobiles Endgerät von einer einzigen Person genutzt. Daher zeichnen sich mobile Geschäftsanwendungen durch einen hohen Grad an Personalisierungsmöglichkeiten aus. Dadurch wird die Interaktion des Nutzers mit der Anwendung erleichtert, weil er sich gewünschte Ansichten selbst konfigurieren und beibehalten oder auch persönliche Daten für den Zugang langfristig speichern kann (Alahuhta et al., 2005; Xiaojun et al., 2004).

Integration. Die Anbindung mobiler Geschäftsanwendungen an die Kommunikationsstruktur oder die externen Dienste eines Unternehmens schafft zusätzlichen Wert für den Anwender und damit auch für das Unternehmen (Nah et al., 2005).

Sicherheit. Die kompakte Bauweise und der mobile Charakter tragen zu einem großen Verlustrisiko sowie einer erhöhten Diebstahlgefahr von Smartphones und Tablets bei. Aus diesem Grund benötigen mobile Anwendungen effektive Schutzmechanismen beim Umgang mit sensiblen Daten, vor allem aber mit sensiblen Unternehmensdaten (Verclas und Linnhoff-Popien, 2012). Gleichzeitig bieten mobile Technologien aber auch eine Verbesserung der Sicherheit von Unternehmensanwendungen durch neue Authentifizierungsmöglichkeiten für den Anwender (Alahuhta et al., 2005). So können mobile Endgeräte für ein doppelt gesichertes Authentifizierungssystem bei Geldautomaten eingesetzt werden, welches die Verifizierung des berechtigten Nutzers sicherstellen soll (Aloul et al., 2009).

Zusammenfassend können mobile Geschäftsanwendungen anhand der vorgestellten Merkmale, welche neben den Eigenschaften mobiler Technologien die Besonderheiten des geschäftlichen Umfelds beinhalten, identifiziert werden. Diese Merkmale gelten nicht speziell als Restriktionen für den mobilen Geschäftsverkehr, sondern sie bieten gleichzeitig neue Möglichkeiten, wenn sie sorgsam und vorteilhaft angewendet werden (Alahuhta et al., 2005).

2.2.2.4. Einsatz mobiler Entscheidungsunterstützungssysteme

Die besonderen Eigenschaften von mobilen Geräten und Applikationen erlauben eine zeitliche und örtliche Ungebundenheit der Akteure, die in die Entscheidungsfindung involviert sind (vgl. Kapitel 2.2.2.3 und 2.2.2.1). Parallelen zu dieser Vorgehensweise können aus der global verteilten Softwareentwicklung (GVSE) abgeleitet werden, welche maßgeblich durch zielgerichtete Informationskoordination geprägt ist (Herbsleb, 2007), die auch beim Treffen von datenzentrischen Einzel- oder Gruppenentscheidungen erforderlich ist.

Die GVSE ist eine Entwicklungsmethode, die sich mit der zunehmend besseren Bereitstellung von Infrastruktur auf weltweiter Ebene, besonders aber in den Entwicklungs- und Schwellenländern, stark verbreitet hat. Dadurch lassen sich wesentlich mehr Ressourcen in den Entwicklungsprozess einbinden und gleichzeitig eine höhere Geschwindigkeit für die Produkterstellung erzielen (vgl. Kapitel 2.3.4). Als die wichtigsten Erfolgsfaktoren für ein derartiges Vorgehen nennt Herbsleb (2007) die folgenden:

- Verwendung verfügbarer Ressourcen unabhängig von ihrem Aufenthaltsort

- Angemessener Einsatz von Praktiken und Technologie für die Koordination der Aufgaben zwischen den Arbeitsstätten
- Schaffung eines gemeinsamen Verständnis für die Anforderungen
- Effizientes Änderungsmanagement

Diese Erfolgsfaktoren lassen sich direkt mit EUS in ihrer jeweiligen Ausprägung in Verbindung bringen. So fließen in Entscheidungen je nach Situation und Zielsetzung sowohl unterschiedliche Meinungen der beteiligten Akteure als auch Informationen aus verschiedenen Quellen mit ein. Da sich bei Datenquellen noch mit höherer Wahrscheinlichkeit ein Ursprungsort festlegen lässt, so können die Aufenthaltsorte des Entscheiders sowie der einbezogenen Personen grundsätzlich variieren. Oftmals sind Entscheidungsträger gleichzeitig auch Führungspersonen im Unternehmen, deren Alltag von zahlreichen Meetings an unterschiedlichen Lokationen geprägt ist. Mobile Endgeräte eignen sich wegen ihrer hohen Rechenleistung und ihrer zuverlässigen Konnektivität besonders gut, um Anwendungen zur Entscheidungsunterstützung zu betreiben, und zwar unabhängig vom Aufenthaltsort (vgl. Kapitel 2.2.2.1).

Der angemessene Einsatz von Praktiken und Technologie zur Informationskoordination bei der GVSE (Vlaar et al., 2008) trifft ebenfalls gänzlich für den Prozess der Entscheidungsfindung zu. So erlauben es die Vorteile von mobilen Geschäftsanwendungen, dass diese durch ihre Personalsierungsmöglichkeit eine Umgebung für den Entscheider schaffen, in der die für ihn wichtigen Informationen auf einen Blick verfügbar und verständlich aufbereitet sind. Dadurch wird den Anwendern eine schnellere Entscheidungsfindung ermöglicht. Die Integrationsmöglichkeiten mobiler Geschäftsanwendungen in bestehende Systemlandschaften sorgen dafür, dass notwendige Informationen nicht erst mühselig über die erschwerten Eingabemöglichkeiten mobiler Endgeräte gesucht werden müssen, sondern diese Informationen nahtlos aus bestehenden Unternehmenssystemen in die mobile Anwendung integriert werden können.

Die unbegrenzte zeitliche und örtliche Erreichbarkeit durch mobile Geschäftsanwendungen erlaubt schließlich auch die Schaffung eines gemeinsamen Verständnisses des Entscheidungsproblems. Dies kann dadurch herbeigeführt werden, dass ein einfacher Austausch zwischen den Entscheidungsträgern in der mobilen Anwendung etabliert wird. Damit lassen sich Nachfragen und Meinungen direkt

anzeigen und gegenseitig beantworten. Mit dieser Funktionalität lässt sich auch das effiziente Änderungsmanagement für die Entscheidungsfindung aus der GVSE übertragen. Der schnelle Informationsaustausch lässt sich somit auch für die unmittelbare Benachrichtigung von Änderungen in der Datenbasis nutzen, sofern die Informationen relevant für die zu treffende Entscheidung sind.

Bislang gibt es allerdings nur wenige Studien, die sich mit der Verwendung von mobilen Technologien für die Entscheidungsunterstützung befassen und von den beschriebenen Vorteilen Gebrauch machen. Dennoch werden mobile EUS als die nächste Entwicklungsstufe in diesem Bereich erwartet (Shim et al., 2002). Bisherige Arbeiten führen bereits folgende zentrale Eigenschaften von mobilen Endgeräten als entscheidende Vorteile gegenüber etablierten EUS an:

- Verbesserte Leistungsfähigkeit mobiler Endgeräte (Heijden, 2006; Kwon et al., 2005),
- weit verbreitete Akzeptanz und Verwendung durch Verbraucher (Heijden, 2006; Pérez et al., 2010) sowie
- verbesserter Informationszugang (Heijden, 2006; Kwon et al., 2005; Pérez et al., 2010).

Um diesen eingeschränkten Wissensfundus der mobilen EUS zu erweitern, soll diese Arbeit dazu beitragen, ein mobiles EUS zu entwerfen, umzusetzen und dessen Nützlichkeit und Nutzbarkeit zu bewerten. Dabei wird ein besonderer Fokus auf die Möglichkeiten mobiler Endgeräte und Anwendungen gelegt, wie diese gewinnbringend in Systemen zur Entscheidungsunterstützung im Softwareoutsourcing eingesetzt werden können.

2.3. Komponentenbasierte Entwicklung von Anwendungssoftware

In diesem Unterkapitel werden die Grundlagen der Softwareentwicklung für Anwendungssoftware bis hin zur komponentenbasierten Vorgehensweise vorgestellt. Nach einer grundlegenden Einführung in das Thema selbst, werden aktuelle Entwicklungsmethoden und Tools für die global verteilte Softwareentwicklung, deren komponentenbasierte Ausrichtung sowie deren Besonderheiten beim Outsourcing erörtert.

2.3.1. Grundlagen der Softwareentwicklung

Unter dem Begriff der Softwareentwicklung (SE) fasst man den systematischen Ansatz zur Entwicklung, zum Betrieb und zur Pflege von Software zusammen (Sommerville, 2007). Dabei lassen sich die verwendeten Vorgehensweisen mit der Konstruktion und Entwicklung von Gebäuden in der Architektur oder von Fahrzeugen im Ingenieurswesen vergleichen (Marciniak, 2002). Der Begriff umfasst daher neben der eigentlichen Tätigkeit, nämlich der Programmierung von Softwarecode, auch sämtliche begleitenden Prozesse der Softwareproduktion, beginnend mit der Systemspezifikation bis hin zur Pflege und Wartung eines Systems im Betrieb (Sommerville, 2007). Damit wird zusammenfassend der Softwareentwicklungsprozess und die zugehörigen Methoden zur Herstellung eines gewünschten Softwareprodukts beschrieben.

Ein Softwareprodukt ist die Kopie einer Software, welche die gewünschten Anforderungen eines Kunden erfüllt, kundenseitig installierbar und mit geringem Pflegeaufwand ausführbar ist (Balzert, 2009; Dumke, 2003; Heinrich et al., 2004). Alternativ kann ein Softwareprodukt auch als Dienstleistung über das Internet bereitgestellt werden (Choudhary, 2007; Lizhe et al., 2008; Stuckenberg et al., 2014; Xin und Levina, 2008). Ein Softwareprodukt besteht aus einer zugehörigen Anleitung, einer Demoversion, einer Installationsroutine, dem eigentlichen Programmcode sowie aus einer ausführlichen Entwicklungsdokumentation (Dumke, 2003). Wenn das Produkt als Dienstleistung angeboten wird, dann reduziert sich der Umfang auf die Softwaredokumentation, eine Testversion sowie die Bereitstellung eines Zugangs zur Benutzung der Software als Dienstleistung. Der Programmcode selbst und die Entwicklungsdokumentation verbleiben beim Hersteller, der über die vereinbarte Nutzungsdauer ein Dienstleistungsverhältnis mit dem Kunden eingeht (Choudhary, 2007; Stuckenberg, 2014).

Zur Erstellung der Software, egal ob als Endprodukt oder als Produkt für den Dienstleistungsbezug, orientieren sich die Hersteller am Softwarelebenszyklus, der die grundlegenden Phasen und Aktivitäten zur Durchführung des Entwicklungsprozesses beschreibt. Dieser wird im nachfolgenden Unterkapitel beschrieben.

2.3.2. Der Softwarelebenszyklus

Für die im Rahmen der Softwareentwicklung stattfinde Erstellung eines Softwareprodukts hat man sich auf die Definition von Phasen und zugehörigen Aktivitäten festgelegt, die in ihrer Gesamtheit den Softwarelebenszyklus eines Produkts beschreiben. Für die ingenieurmäßige Entwicklung von Software ist es daher erforderlich, dass all diese Phasen entsprechend des gewählten Prozessmodells (vgl. Kapitel 2.3.3) durchlaufen werden und parallel dazu das Lebenszyklusmodell sicherstellt, dass die jeweiligen Aktivitäten auch phasenübergreifend koordiniert werden (Schwaber et al., 2006). Die unterschiedlichen Stufen des Lebenszyklus, die im weiteren Verlauf des Unterkapitels näher erläutert werden, werden von Sommerville (2007) unterteilt in:

- Anforderungserhebung und -analyse,
- Design des Softwaresystems,
- Implementierung und Komponententests,
- Integrationstests,
- Betrieb und Wartung.

2.3.2.1. Anforderungserhebung und -analyse

Bei der Anforderungserhebung wird zunächst das Problemfeld des Kunden analysiert bevor anknüpfend daran seine Anforderungen an ein zu entwickelndes Softwaresystem aufgenommen werden. Es wird dabei zwischen funktionalen Anforderungen, Anforderungen an die Performanz, Akzeptanzkriterien, Anforderungen an die Dokumentation sowie Sicherheitsaspekten unterschieden (Kotonya und Sommerville, 2004; Pfleeger, 2001). Für die nachfolgenden Softwareentwicklungsphasen ist es notwendig, eine Trennung zwischen funktionalen und nicht funktionalen Anforderungen vorzunehmen, um die Architektur des neuen Softwaresystems entsprechend entwerfen zu können. Funktionale Anforderungen stellen dabei die Vorschriften für ein System dar, wie es eingegebene Daten verarbeiten soll oder wie es auf die Interaktion mit dem Anwender reagieren soll (Rupp, 2009). Im Gegensatz dazu werden durch nicht funktionale Anforderungen die Bedingungen definiert, die von außen auf das zu entwickelnde System einwirken, wie

z.B. einzuhaltende Standards, vorgeschriebene Entwicklungsprozesse oder auch Schnittstellen zu anderen Systemen (Rupp, 2009).

Zu dieser Phase zählt ebenfalls der Prozess zur Erhebung von Anforderungen beim Kunden. Hierbei werden systematisch alle Anforderungen an das neue Softwaresystem eingesammelt, um sich damit als Hersteller Einblicke in das Problemfeld des Kunden zu verschaffen und um herauszufinden, welche Funktionen die Softwarelösung schließlich für die unterschiedlichen Nutzergruppen beherrschen muss (Pan und Flynn, 2003; Robertson und Robertson, 2006; Sommerville, 2007). Für die Erhebung selbst werden neben Fragenkatalogen und Interviews mit den Kunden zusätzliche Methoden verwendet, um die Schwierigkeiten der Kommunikation zwischen Experten der Geschäftsdomäne und technisch versierten Systemexperten (unterschiedliche Fachtermini, technisch unrealistische Anforderungen usw.) zu überwinden und dadurch einen klar formulierten Anforderungskatalog zu erhalten (Robertson und Robertson, 2006; Rupp, 2009; Versteegen, 2004).

Bei der Analyse von Anforderungen stehen die Klassifizierung (funktional oder nicht funktional) sowie das Priorisieren von Anforderungen im Fokus (Rupp, 2009). Die Aktivität der Analyse ist allerdings eng mit der Erhebung verzahnt, um die oben genannten auftretenden Schwierigkeiten bei der Ermittlung bestenfalls zu beseitigen. Dadurch ist ein iteratives Vorgehen zwischen den Aktivitäten der Identifikation, Klassifizierung, Priorisierung und Dokumentation notwendig (Sommerville, 2007). Aktuelle ganzheitliche Ansätze zur Erhebung und Definition von Anforderungen greifen dieses Vorgehen auf und verknüpfen es mit analytischen Verfahren zur Bewertung und damit zur Prisorisierung von Anforderungen (Karlsson et al., 1998; Saaty und Vargas, 2001). Dabei wird in unterschiedlichen Methoden der Nutzen einer Anforderung dem Aufwand für die Umsetzung gegenübergestellt und von den unterschiedlichen Interessengruppen bewertet, bis schließlich eine vollständige Liste der umzusetzenden Anforderungen ermittelt werden kann (Geisser, 2008; Karlsson und Ryan, 1997; Ruhe et al., 2002).

2.3.2.2. Design des Softwaresystems

Zur Designphase einer Softwarelösung gehören die Modellierung der im Vorfeld aufgenommenen Anforderungen und der Entwurf der Systemarchitektur für die nachfolgende Implementierung (Booch et al., 2007; Sommerville, 2007).

Das Modellieren bei der Softwareentwicklung kennzeichnet die strukturelle, operative sowie formlose Transformation von Anforderungen in eine geeignete Interpretation der zukünftigen Softwarelösung, damit sie gleichermaßen von Kunden und Entwicklern verstanden werden kann (Boehm, 1999). Dabei werden die in natürlicher Sprache verfassten Anforderungen in unterschiedliche grafische Modelle übertragen, die beispielsweise das Interaktionsverhalten der Software, die Datenflüsse, die Zustände des Systems usw. darstellen, um damit die unterschiedlichen Perspektiven auf ein Softwaresystem (externe Sicht, strukturelle Sicht oder die Verhaltensperspektive) zu bedienen. Hierfür bedient man sich bei der objektorientierten Softwareentwicklung sowohl der Objektmodelle (Booch et al., 2007; Lahres und Rayman, 2009) zur Abbildung und Interpretation realer Objekte und deren Interaktionsverhalten als auch der Datenmodelle (Atkinson et al., 1983; Chen, 1976) zur Repräsentation der Datenhaltung in Softwarelösungen.

Das Design der Systemarchitektur findet auf Basis der zwischen Kunde und Entwickler abgestimmten Modelle statt. Dabei wird die abstrakt definierte Lösung des komplexen Systems in unterschiedliche Teilsysteme sowie in ein Kernsystem, das die Teilsysteme integriert, zerlegt (Dumke, 2003). Entsprechend der gestellten Anforderungen werden einzelne Komponenten in den Subsystemen entwickelt, welche die zuvor abgestimmten Objekt- und Datenmodelle umsetzen (Bass et al., 2003). Dadurch haben die Entwickler die Möglichkeit, die zentralen Abstraktionen mit dem Kunden frühzeitig abzustimmen und bei Bedarf zu korrigieren, ohne dabei ins Detail des komplexen Systems gehen zu müssen (Hofmeister et al., 2000). Die Analyse des abstrakten Systems ermöglicht es ebenfalls, die Umsetzung systemkritischer Anforderungen zu gewährleisten (Bosch, 2000).

Das Ziel dieser Phase ist die Erstellung einer formalen Spezifikation für die erwartete Softwarelösung, welche mit entsprechenden Hilfsprogrammen bereits in dieser frühen Phase in beispielhaften Softwarecode überführt werden kann, um damit frühzeitig systemische Fehler zu erkennen und auszubessern (Hall, 1990; Sommerville, 2007).

Zur grafischen Repräsentation der Anforderungen in unterschiedlichen Modellen, die zur gemeinsamen Abstimmung verwendet werden, bedient man sich Softwaretools für den Entwurf der Diagramme. Diese unterstützenden Programme werden CASE-Tools (*Computer-Aided Software Engineering Tools*) genannt (Dumke, 2003). Sie unterstützen die Arbeit der Softwarefirma insbesondere in der

Designphase, um Diagramme zu entwerfen, die standardisierten Regeln entsprechen und damit die Kommunikation zwischen Systemarchitekten, Entwicklern und Kunden erleichtern und gleichermaßen ermöglichen. Die CASE-Tools orientieren sich daher überwiegend an der *Unified Modeling Language*[1] (UML), einer vereinheitlichten Modellierungssprache zur Darstellung von Objekt-, Kommunikations- und Datenmodellen im Softwareentwicklungsprozess (Jacobson et al., 2003).

2.3.2.3. Implementierung und Komponententests

In dieser Phase werden die definierten abstrakten Modelle und Komponenten des Systems von Softwareentwicklern in spezifischen Softwarecode umgesetzt. Die Softwarelösung wird in diesem Schritt real entwickelt. Anschließend werden die implementierten Codefragmente hinsichtlich ihrer Funktionalität geprüft und finalisiert (Dumke, 2003).

Bei der Entwicklung unterscheidet man zwischen vier unterschiedlichen aber kombinierbaren Möglichkeiten:

- Der Entwickler schreibt und bearbeitet Softwarecode in einer bestimmten Programmiersprache im dazugehörigen Editor und hält sich bei der Implementierung einer Funktion an die entsprechende Syntax.
- Softwarecode wird automatisch durch ein CASE-Tool erzeugt, das abhängig von der Granularität der vorher definierten abstrakten Modelle bereits Codefragmente generieren kann. Diese dienen dann als Einstiegspunkt für den Entwickler, der den Code weiterentwickeln und verfeinern kann.
- Bereits vorher verwendete Codeschnipsel, welche die erforderliche Funktionalität abbilden, werden als Implementierung herangezogen und angepasst.
- Bereits entwickelter Softwarecode kann direkt wiederverwendet werden, um die gleiche Funktionalität wie in einem anderen Projekt abzudecken.

Die konkreten Aufgaben für einen Softwareentwickler hängen in dieser Phase besonders vom gewählten Prozessmodell ab, welche im nächsten Unterkapitel

1 http://www.omg.org/spec/UML/2.2/ (abgerufen am 31.10.2014)

beschrieben werden. Die Kernaufgabe besteht aber darin, die abstrakt definierten Komponenten Schritt für Schritt in eine Softwarelösung umzusetzen. Jeder Entwickler ist dabei dafür verantwortlich, seine fertiggestellten Komponenten auf die gewünschte Funktionalität zu überprüfen und bei Änderungen sicherzustellen, dass die Funktionsweise nicht beeinträchtigt wurde (Dumke, 2003; Heinrich et al., 2011, 2004; Pomberger und Pree, 2004; Sommerville, 2007).

Der hierbei entwickelte Code wird üblicherweise fortlaufend in zentralen Codeverwaltungssystemen abgelegt und in seinen verschiedenen Versionierungsstufen gespeichert. Dadurch wird allen Entwicklern der Zugriff auf eine gemeinsame Codebasis ermöglicht, deren Aufgabe es ist, einen ordnungsgemäßen Zusammenhang der eigenen Codefragmente im Verbund mit anderen Komponenten sicherzustellen (Loelinger und McCullough, 2012).

2.3.2.4. Integrationstests

Im Rahmen von Integrationstests muss die entwickelte Software schließlich ganzheitlich auf eigene Fehler der Funktionalität und Integrität getestet werden. Hierfür werden Testfälle definiert, die im neu entwickelten System teilweise durch Testpersonen oder automatisiert durchgespielt werden. Die Resultate und Verhaltensweisen der neu entwickelten Software werden anschließend mit den definierten Akzeptanzkriterien abgeglichen und auf Fehler überprüft. Erst dann wird das Produkt für den Einsatz in der gewünschten Umgebung freigegeben (Sommerville, 2007).

Auch für diese Phase des Entwicklungsprozesses gibt es unterstützende Programme, die eine kontinuierliche Integration von neuem Softwarecode in die gewünschte Zielumgebung sicherstellen. Solche Programme laden automatisch den letzten Stand aus dem Codeverwaltungssystem, erstellen daraus das funktionsfähige Softwareprodukt und lassen automatisiert alle zuvor definierten Tests ablaufen. Dieser periodisch stattfindende Vorgang soll die Entdeckung von Fehlern verbessern und (bei täglicher Durchführung) hohe Nachbesserungskosten vermeiden (Duvall et al., 2007).

2.3.2.5. Betrieb und Wartung

In dieser Phase des Softwarelebenszyklus befindet sich das Softwareprodukt im produktiven Einsatz beim Kunden und wird für die angeforderten Anwendungszwecke eingesetzt. Hierfür müssen die designierten Anwender, je nach Komplexität der Software, zunächst geschult werden, bevor sie mit dem Produkt arbeiten können. Zusätzlich können beim produktiven Einsatz Fehler auftauchen, die in keinem der Tests zuvor entdeckt wurden. Diese werden dann dem Entwicklungsteam zur Korrektur zurückgespielt (Dumke, 2003).

Zur Wartung eines Systems gehören neben der Gewährleistung eines fehlerfreien Betriebs der Software auch das Sammeln von Änderungsanforderungen an das Produktivsystem. Hierbei wird zwischen kritischen Systemreparaturen oder Erweiterungen der Funktionalität unterschieden. Bei kritischen Reparaturen sollte das Entwicklungsteam unmittelbar aktiv werden, wohingegen bei Funktionserweiterungen ein neuer Softwareentwicklungslebenszyklus beginnen kann (Heinrich et al., 2011; Pomberger und Pree, 2004; Sommerville, 2007).

Die kombinierte Anwendung der in diesem Unterkapitel vorgestellten Phasen und Aktivitäten wird durch Prozessmodelle zur systematischen Entwicklung von Software definiert. Vielfach verwendete Prozessmodelle werden daher im nachfolgenden Unterkapitel vorgestellt und ihre jeweiligen Ablaufweisen beschrieben.

2.3.3. Prozessmodelle der Softwareentwicklung

Im Softwareentwicklungsprozess ist die Durchführung zahlreicher Aktivitäten aus dem Softwarelebenszyklus erforderlich (vgl. Kapitel 2.3.2), um ein Softwareprodukt von der initialen Idee bis zu einem auslieferbaren und nutzbaren Programm zu entwickeln, oder bestehende Produkte zu modifizieren bzw. zu erweitern. Zur Erreichung dieser Ziele werden Prozessmodelle als abstrakte Anleitungen zum Vorgehen angewandt, welche die Phasen des Softwarelebenszyklus planmäßig aneinanderreihen (Sommerville, 2007). Abhängig von den etablierten Standards einer Softwareentwicklungsfirma, den vertraglichen Vereinbarungen zwischen Hersteller und Kunde oder den Selbstbestimmungsmöglichkeiten von Entwicklungsteams werden entweder langfristig bewährte (auch als schwerfällig bezeichnete) Prozessmodelle wie das Wasserfallmodell und Abwandlungen davon oder moder-

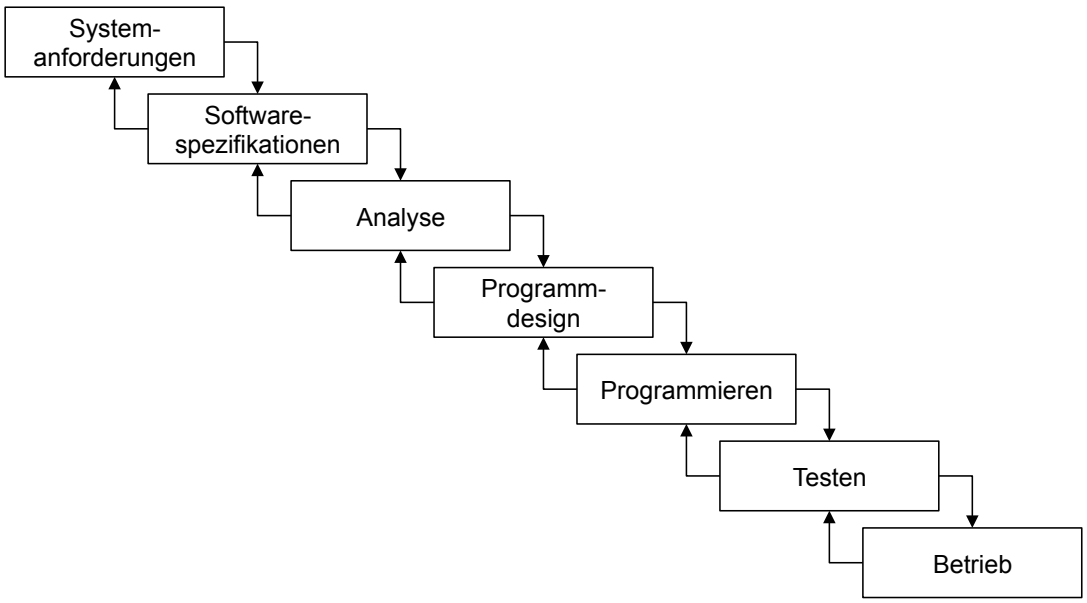

Abbildung 2.6.: Wasserfallmodell, abgeleitet von Royce (1970)

ne leichtgewichtige Modelle wie die Entwicklungsmethode *Scrum* eingesetzt (Lee und Xia, 2010).

2.3.3.1. Wasserfallmodell

Das Wasserfallmodell ist das am weitesten verbreitete Entwicklungsmodell der Softwareentwicklung, da es bereits seit vielen Jahren von kleinen und großen Softwarefirmen für die Entwicklung favorisiert wird. Es sieht die serielle Abfolge der Phasen aus dem Softwarelebenszyklus wie in Abbildung 2.6 vor. Dabei wird jede Phase einmalig durchlaufen und liefert nach Abschluss aller zugehörigen Aktivitäten eine Zusammenstellung von Dokumenten oder Daten, welche als Startbedingung für die nachfolgende Phase benötigt werden. Bei dieser Vorgehensweise werden Fehler in der Entwicklung per Definition erst spät in der Testphase entdeckt und sind daher meist kostspielig. Dennoch ist das Entwicklungsmodell ausgiebig erprobt und gut dokumentiert und daher auch weit verbreitet (Royce, 1970; Sommerville, 2007).

Zur Ausbesserung der Schwachstellen einer späten Fehlererkennung wurde das V-Modell XT (vgl. Abbildung 2.7) als Abwandlung des Wasserfallmodells definiert. Dieses Prozessmodell beinhaltet zahlreiche Feedbackschleifen zu vorherigen

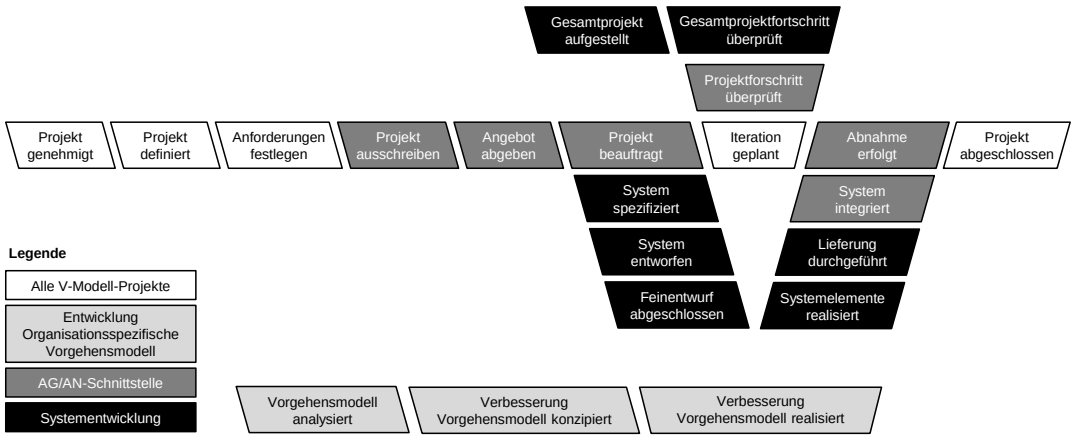

Abbildung 2.7.: V-Modell XT[2]

Phasen und sieht einen iterativen Verlauf des Softwarelebenszyklus vor. Seine Bekanntheit, besonders in Deutschland, verdankt das Modell der deutschen Bundesregierung, welche das Prozessmodell für staatliche Softwareentwicklungsprojekte verpflichtend vorschreibt (Friedrich et al., 2008).

2.3.3.2. Komponentenbasierte Entwicklung

Die Wiederverwendung von mehrmals erforderlicher Funktionalität in verschiedenen Softwareprojekten führte zum komponentenbasierten Entwicklungsmodell. Dabei wird ein Gesamtsystem in einzelne Komponenten zerlegt, so dass diese in sich abgeschlossen sind, sich aber durch klar definierte Schnittstellen in weiteren Softwareprojekten wiederverwenden lassen. Somit lassen sich Komponenten auch an andere Hersteller verkaufen. Umgekehrt können selbst benötigte Komponenten vom externen Markt bezogen werden (Booch et al., 2007; Sommerville, 2007).

Hieraus entwickelte sich das standardisierte Prozessmodell der Firma *Rational* mit dem Namen *Rational Unified Process* (RUP). Der RUP legt besonderen Wert auf Visualisierung der komplexen Zusammenhänge von Systemen zur erleichterten Kommunikation zwischen den Beteiligten. Besonders in der Designphase ist daher die Verwendung von UML-Diagrammen vorgeschrieben (Jacobson et al.,

2 Vorlage von http://www.infforum.de/themen/anwendungsentwicklung/se-v-modell.htm (abgerufen am 01.12.2014)

2003). Mit dem RUP wird ein multiperspektivischer Ansatz verfolgt, der in einer dynamischen Perspektive die iterativen Ansätze bereits etablierter Prozessmodelle forciert, in einer statischen Perspektive die Durchführung der festgelegten Aktivitäten des Softwarelebenszyklus erfordert und in einer praktischen Perspektive Hilfestellungen zur vollständigen Umsetzung eines Softwareprojekts gibt (Jacobson et al., 2003). Die Besonderheit dieses Prozessmodells ist die gezielte Verwendung objektorientierter Modelle und Programmiersprachen zur Umsetzung der komponentenbasierten Softwareentwicklung (Heinrich et al., 2011; Kruchten, 2000).

2.3.3.3. Agile Prozessmodelle

Die Anwendung agiler Prinzipien in der Softwareentwicklung hat zur Entwicklung neuer Prozessmodelle geführt, die hinsichtlich ihrer Eigenschaften allgemein als leichtgewichtig anerkannt sind. Das Ziel agiler Prozessmodelle ist die schnelle und kundenorientierte Entwicklung von Softwarelösungen ohne dabei präzise die bis ins Detail definierten Prozessphasen des Softwarelebenszyklus zu durchlaufen. Der Grundgedanke der agilen Modelle ist daher an der Entwicklungstechnik Extremprogrammierung (*Extreme Programming*) orientiert, welche die Bereitstellung von funktionierenden Softwarelösungen den bürokratischen Prozessen wie dem akribischen Design der Gesamtarchitektur oder dem Schreiben von Dokumentationen vorzieht (Beck, 1999). Dabei liegen den agilen Modellen die Prinzipien des agilen Manifests[3] zugrunde, die in Tabelle 2.2 aufgelistet sind.

Eine konkrete Umsetzung des agilen Prozessmodells ist die Entwicklungsmethode *Scrum*. Die leichtgewichtige Entwicklungsmethode wurde aus der Produktentwicklung übernommen und für die Softwareentwicklung angepasst (Takeuchi und Nonaka, 1986). Hierbei steht die Entwicklung der persönlichen Reife von Entwicklern und Kunden hinsichtlich des angeforderten Softwaresystems im Mittelpunkt der Vorgehensweise. Die kontinuierliche Verbesserung pro Entwicklungszyklus garantiert optimale Ablaufprozesse und schließlich bessere Qualität der entwickelten Software mit minimalem Ressourcenaufwand. Um dies zu erreichen, werden den Teams maximale Freiheiten (mit selbst definierten Aktivitäten und Ablaufprozessen) innerhalb von lose definierten Grenzen (Ressourceneinsatz,

3 http://www.agilemanifesto.org/ (abgerufen am 31.10.2014)

Tabelle 2.2.: Prinzipien des Agilen Manifests

Prinzipien	**Beschreibung**
Einbindung der Kunden	Der Fokus liegt auf der kontinuierlichen Einbeziehung der Kunden während des gesamten Prozesses. Ihre Aufgabe ist nicht nur die Definition von Anforderungen und deren Priorisierung, sondern auch die Überprüfung der korrekten Implementierung.
Inkrementelle Auslieferung	Ständig wiederholende Zyklen im Entwicklungsprozess sorgen für eine inkrementelle Programmierung der Software. Die Anforderungen werden für jeden Zyklus neu definiert.
Menschen über Prozesse	Die Kreativität der Entwickler wird gegenüber vorgeschrieben Prozessen bevorzugt.
Änderungen aufgreifen	Die inkrementelle Entwicklung von Software führt zu ständig ändernden Anforderungen. Dafür soll die Software ausgelegt sein.
Einfachheit beibehalten	Sowohl der Entwicklungsprozess als auch die Software selbst sollen von Einfachheit geprägt sein. Komplexität soll bei jeder Iteration vermieden bzw. eliminiert werden.

Technologie usw.) gesetzt, um die unberechenbare Komplexität eines Zielsystems iterativ zu minimieren und ein vom Kunden akzeptiertes Softwareprodukt zu erschaffen (Padberg und Tichy, 2007). Zusätzlich gibt es unter Einbeziehung des Kunden eine klare Rollenverteilung mit verpflichtenden Aufgaben, um eine agile Vorgehensweise zu gewährleisten (Schwaber und Beedle, 2002).

Die wesentlichen Änderungen gegenüber den schwerfälligen Prozessmodellen sind dabei die in kleine Arbeitspakete geschnürten Entwicklungsaufgaben (*Backlog*), die kurzen Entwicklungszyklen (Sprints mit einer Dauer zwischen 2 und 4 Wochen) mit anschließenden Akzeptanztests durch den Kunden sowie die nach jedem Sprint neu priorisierten Anforderungen (vgl. Abbildung 2.8). Die Besonderheit dabei ist die starke Einbeziehung des Kunden, der damit stets über die aktuelle Entwicklung informiert ist und frühzeitig die Möglichkeit hat, im Falle von Fehlern oder ungewollten Anforderungen einzugreifen, anstatt lange Zeit auf ein fertig implementiertes Produkt zu warten, das nicht den Erwartungen entspricht (Lee und Xia, 2010; Schwaber und Beedle, 2002). *Scrum* lässt sich nicht nur in kleinen Entwicklungsteams anwenden, sondern auch auf große Soft-

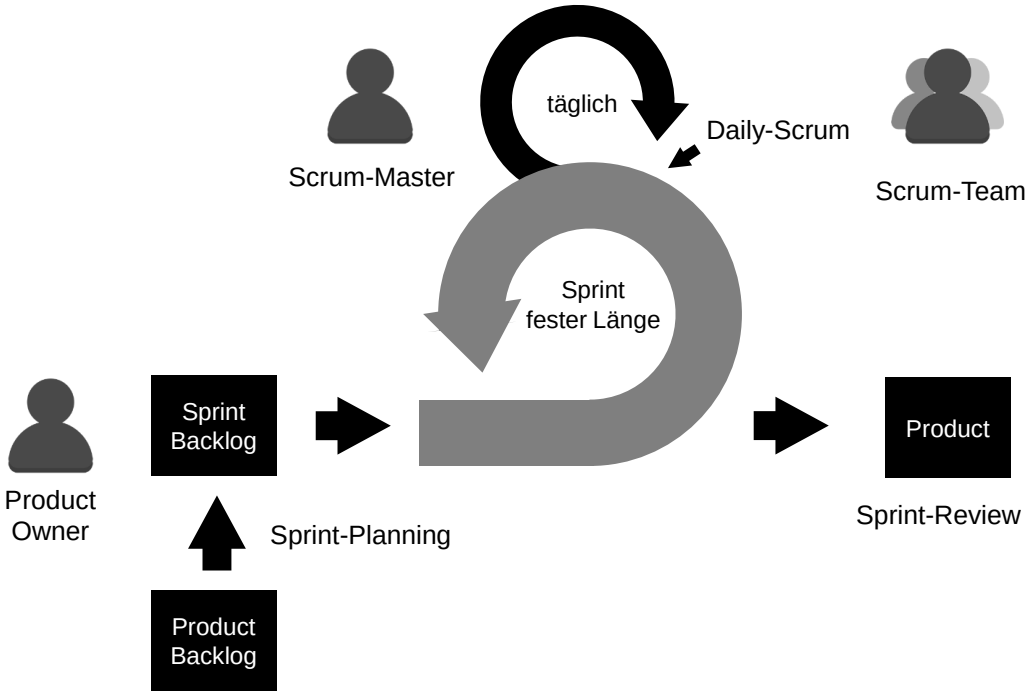

Abbildung 2.8.: Agile Vorgehensweise mit *Scrum*

wareprojekte mit einer Vielzahl von Entwicklern übertragen (Larman und Vodde, 2008; Reinertson, 2009).

2.3.4. Verteilte Softwareentwicklung

Die Entwicklung von Software über Ländergrenzen hinweg hängt einerseits mit der Outsourcingnachfrage von Softwareunternehmen (vgl. Kapitel 2.4) und andererseits mit dem Trend der Globalisierung zusammen. Ambitionierte Ziele wie Kostensenkung, neue qualifizierte Arbeitskräfte, schnellere Durchdringung ausländischer Märkte und die schnelle Formierung von virtuellen Teams treiben Unternehmen an, weltweite Entwicklungsteams zu formen (Herbsleb und Moitra, 2001). Mit der Idee, Software über verschiedene Lokationen hinweg zu entwickeln, sehen Unternehmen die Chance, ihre Entwicklungszeiten zu reduzieren, indem sie Software rund um die Uhr auf verschiedenen Kontinenten entwickeln. Dafür müssen jedoch die traditionellen Prozesse und Konzepte auf den globalen Betrieb umgestellt werden. Bei Softwarefirmen gilt dies insbesondere für die Entwicklungsprozesse, die eingesetzten Werkzeuge aber auch die Verhaltenskultur

der Organisation (Bode und Mertens, 2006; Lacity und Willcocks, 2000; Sahay et al., 2007).

Besondere Herausforderungen bei der GVSE stellen die strategischen Aspekte der Arbeitsteilung dar, wenn Aufgaben über eine Vielzahl von Lokationen mit begrenzten Ressourcen zu verteilen sind (Grinter et al., 1999). Zusätzlich spielen kulturelle Aspekte neben technischen Herausforderungen eine entscheidende Rolle, wenn Nachrichten von Menschen anderer Kulturen unterschiedlich interpretiert werden oder wenn die Projektziele schriftlich nicht eindeutig definiert werden können und ein entsprechendes Kommunikationsmedium gewählt werden muss. Aus technischer Sicht muss gewährleistet sein, dass das Kommunikationsnetzwerk hochverfügbar und ausreichend schnell ist. Selbst das Management muss den Herausforderungen der Softwareentwicklung an mehreren Standorten gewachsen sein und sich eines erhöhten Reiseaufwands und weniger persönlicher Abstimmungen bewusst sein (Herbsleb und Moitra, 2001; Hildenbrand et al., 2007b; Kraut und Streeter, 1995).

Diese besonderen Herausforderungen von verteilten Softwareprojekten sind auf die physische Trennung der Teammitglieder zurückzuführen, welche sich auf die Kollaboration untereinander negativ auswirkt. Die dadurch gehemmte Kommunikation zwischen Teammitgliedern mit oftmals unterschiedlichem kulturellen Hintergrund kann daher zu ungewollten gegenseitigen Missverständnissen oder fehlendem Vertrauen führen und schließlich den Entwicklungsprozess der Software verlangsamen. Persönliche Treffen für den Wissensaustausch und zum Aufbau von gegenseitigem Vertrauen im verteilten Umfeld sind mit hohen Reisekosten verbunden. Somit wären die Vorteile der globalen Entwicklung wieder egalisiert (Herbsleb und Moitra, 2001).

Um den Nachteilen der GVSE, die durch eine räumliche und zeitliche Trennung der Teammitglieder entstehen, entgegenzuwirken, wurden pragmatische Lösungen für die formalen Prozesse und etablierten Werkzeuge der Softwareentwicklung erforscht und den Gegebenheiten der GVSE angepasst. Ablaufprozesse des Softwarelebenszyklus wurden für die Verwendung von Workflowmanagementsystemen modifiziert und lassen sich nun ungehindert über räumliche Grenzen hinweg einsetzen (Schwaber et al., 2006). Weiterhin wurden die vorhandenen technischen Möglichkeiten zur Kommunikation, wie E-Mail, Foren oder Chatsysteme, entsprechend an die globalen Anforderungen angepasst. Dadurch sind neue Software-

systeme, sog. kollaborative Softwareentwicklungsplattformen, entstanden, welche die zuvor beschrieben Herausforderungen der GVSE mit einer Kombination aus Prozessunterstützung und Werkzeugsupport, vereint in einer Plattform für alle beteiligten Entwickler, bewältigen (Heindl und Biffl, 2006; Hildenbrand et al., 2007b; Redmiles et al., 2007).

2.3.4.1. Nachvollziehbarkeit und Begründungsmanagement

Bei der GVSE findet Kommunikation wegen der räumlichen und zeitlichen Distanz vermehrt asynchron statt. Daher erfordert die Übertragung von Softwareprozessen in der GVSE in praktische Anwendungssysteme zusätzliche Maßnahmen, die es ermöglichen, Entscheidungen zu treffen, die eine direkte Abstimmung zwischen Projektbeteiligten voraussetzen. Hierunter fallen z.B. Design- oder Entwicklungsentscheidungen oder auch Implementierungsentscheidungen abhängig von den Kundenanforderungen, also aus allen Phasen des Entwicklungsprozesses. Es handelt sich dabei um Entscheidungen, die nachvollziehbar dokumentiert werden sollten. Zur Reduzierung des synchronen Kommunikationsbedarfs werden daher im kollaborativen Softwareentwicklungsprozess den erschaffenen Artefakten (Anforderungen, Designdokumente, Codeabschnitte, Tests usw.) Details zur Nachvollziehbarkeit beigefügt (Abran et al., 1999; Gotel und Finkelstein, 1994; Hildenbrand, 2008; Lindvall und Sandahl, 1996).

Neben der Entscheidungsfindung ist im weiteren Verlauf des Projekts auch die Begründung dafür eine relevante Information, um zukünftige Entscheidungen treffen und bisherige nachvollziehen zu können. Beim Begründungsmanagement werden die Prozessschritte und Artefakte der Entwicklung mit Begründungen annotiert, sobald ein Problem auftaucht oder Entwicklungsentscheidungen zu treffen sind (Dutoit et al., 2006). Dadurch wird in den Projekten eine zentrale Sammlung mit explizitem Wissen über Herangehensweisen und Entscheidungsansätze bei Problemen geschaffen, welche nicht an zentrale Personen des Projekts gebunden ist, zu schnelleren Entscheidungen verhilft und beim Einlernen neuer Projektmitarbeiter unterstützend wirkt. Schließlich trägt dies zu einer Qualitätssteigerung im Prozessablauf bei (Heindl und Biffl, 2006; Hildenbrand, 2008).

2.3.4.2. Werkzeuge für die Kollaboration

Zur Zusammenarbeit von verteilten Entwicklungsteams über räumliche und zeitliche Grenzen hinweg, leisten Softwareplattformen zur Kollaboration im Entwicklungsprozess einen wichtigen Beitrag. Kollaborationsplattformen integrieren einerseits die Prozesssicht der Softwareentwicklung mit den Möglichkeiten existierender Werkzeuge, um die Probleme asynchroner Kommunikation zu überwinden, den Prozessablauf systematisch zu koordinieren und die Zusammenarbeit von Interessengruppen zur Problemlösung innerhalb des Projekts zu forcieren (Bair, 1989). Der Zugang zu Kollaborationsplattformen erfolgt überwiegend über den Browser eines Nutzers oder über die Oberfläche eines Programms, das für die Erfüllung spezieller Aufgaben benötigt wird (z.B. Code-Editor, Software zur Anforderungspriorisierung usw.) und in die Plattform eingebunden ist. Die Software wird von verschieden Anbietern kommerziell oder frei verfügbar mit unterschiedlichem Funktionsumfang angeboten und kann als Cloud-Angebot oder durch eigenes Bereitstellen auf einem Server genutzt werden (Hildenbrand et al., 2007a; Mühlbauer und Versteegen, 2007).

Auch wenn der Funktionsumfang variiert, bieten die Kollaborationsplattformen Grundfunktionalitäten an, die über Trackersysteme (eine einfache Art von Workflowmanagementsystemen) die einzelnen Prozessschritte der Softwareentwicklung im System abbilden, so dass sich aktuelle Aktivitäten jederzeit einsehen, nachvollziehen und überwachen lassen können. Zusätzlich werden Funktionalitäten zur Kommunikation, Nachvollziehbarkeit und für das Begründungsmanagement angeboten, welche sich als Kombination von einzelnen Informationssystemen, die in Tabelle 2.3 zusammengefasst sind, erzielen lassen (Hildenbrand et al., 2007a; Mühlbauer, 2007). Diese Einzelfunktionalitäten werden durch den integrierten Ansatz mit dem Trackersystem ergänzt und erlauben dadurch die Annotation von zusätzlichen Informationen zu jedem einzelnen Artefakt.

Die Notwendigkeit von Nachvollziehbarkeit und Begründungsmanagement in der global verteilten Softwareentwicklung, wie im vorherigen Abschnitt beschrieben, erfordert zusätzlich zu den Grundfunktionalitäten einer Kollaborationsplattform erweiterte Möglichkeiten zur Visualisierung und grafischen Bearbeitung von Verknüpfungen zwischen den Artefakten des verteilten Entwicklungsprozesses (Hildenbrand, 2008). Durch die visuelle Bereitstellung und die Möglichkeit der

Tabelle 2.3.: Einzelfunktionalitäten einer Kollaborationsplattform für die Softwareentwicklung

Funktion	**Verwendung**
Chatsystem	Synchrone Kommunikation zwischen Teammitgliedern mit Speicherfunktion des Verlaufs. Videounterstützung möglich.
Codeverwaltung und Versionierung	Ablageort für die gemeinsame Codebasis der Zielsoftware. Historische Versionierung sowie Verzweigungen zum Experimentieren möglich.
Dokumentenverwaltungssystem	Zentraler Ablageort für Projektdokumente mit Ordnerverwaltung, Zugriffsregelung und Versionierung der Dateien, um Informationsverlust zu vermeiden und Datensicherheit zu gewährleisten.
Forum	System zur Informationssammlung und zum Meinungsaustausch, gruppiert nach Themen.
Konfigurationsmanagement	Pflege und Einstellung der Zusammenhänge von Artefakten des Entwicklungsprozesses in Abhängigkeit zu den geplanten und auslieferbaren Versionen des Softwareprodukts.
Trackersystem	Verwaltung von Prozessen und Artefakten des Entwicklungsprozesses. Ein Tracker kann von unterschiedlicher Ausprägung sein (Änderungsanfrage, Anforderung, Aufgabe, Fehler, Softwarekomponente, Testfall), mit anderen Trackern verknüpft sein, Statusinformation sowie Informationen zur Nachvollziehbarkeit und für das Begründungsmanagement beinhalten.
Wiki	Nutzung zu Dokumentationszwecken für den Entwicklungsprozess und für die Dokumentation der Software. Möglichkeit der linkbasierten Navigation durch das Projekt.

Interaktion mit dem System wird ein effektiveres Management der Nachvollziehbarkeit sowie Begründungsmanagement erreicht. Verantwortlich dafür sind eine Vielzahl von Darstellungsmöglichkeiten komplexer Zusammenhänge je nach Analyseabsicht, schnellere Entscheidungsmöglichkeiten und eine schnellere Rückführung der Ergebnisse ins Kollaborationssystem (Hildenbrand, 2008; Klimpke und Hildenbrand, 2009). Hinzu kommt die Möglichkeit soziale Netzwerke innerhalb von verteilten Softwareprojekten zu analysieren, indem die Autoreninformationen aller Artefakte zur Evaluation herangezogen werden und sich dadurch zentrale

Wissensträger, Gruppen mit starker Interaktion sowie fehlende soziale Verbindungen zum Wissensaustausch identifiziert werden können (Kramer et al., 2009). Ebenso ist die Einbindung und Vernetzung von CASE-Tools (vgl. Abschnitt 2.3.2) ein wichtiger Bestandteil der Kollaboration in der verteilten Softwareentwicklung. So trägt der Einsatz sozialer Netzwerktechnologien in der Softwareentwicklungsumgebung (Code-Editor) zum besseren Austausch unter den Entwicklern bei (Klimpke, 2013).

2.4. Eigenerstellung oder Fremdbezug von Softwarekomponenten

In diesem Abschnitt werden zunächst die historische Entwicklung des betrieblichen Outsourcings von Informationssystemen (IT-Outsourcing) sowie die Forschungsrichtungen, die sich daraus entwickelten, aufgezeigt. Darin werden die Beweggründe für Unternehmen erörtert, die die Durchführung von Outsourcingprojekten motivieren, und ein Überblick über die besonderen Herausforderungen und die speziellen Risiken dieser Vorgehensweise gegeben. Anschließend wird mit der Beschreibung des Zusammenspiels unterschiedlicher Einflussfaktoren und ihrem Einfluss auf die Fremdvergabe oder Eigenerstellung von Softwareentwicklungsaufgaben der Begriff der Outsourcingentscheidung näher erläutert. Bekannte Vorgehensweisen und Ansätze zum Treffen solcher Entscheidungen werden danach vorgestellt und abschließend auf ihre Verwendbarkeit zur komponentenbasierten Entscheidungsfindung auf der operativen Ebene von Unternehmen überprüft.

2.4.1. Genese des IT-Outsourcings

Das Thema Outsourcing hat sich im Bereich der Softwareindustrie erst in den 90er Jahren entwickelt und hat seinen Ursprung in der Fertigungsindustrie. Mit Outsourcing wird grundsätzlich der Aufbau von Arrangements mit externen Entitäten zur Bereitstellung von Waren oder Dienstleistungen bezeichnet, um interne Arbeitsaufwände zu ergänzen oder zu ersetzen (Dibbern et al., 2004; Heinrich, 1996; Heinzl, 1993).

Übertragen auf die Softwareindustrie spricht man bei Outsourcing daher von externem Leistungsbezug sämtlicher Funktionen eines Unternehmens, welche sich

auf Informationssysteme beziehen. Schaut man in der Geschichte zurück, so hat die amerikanische Firma *Eastman Kodak* als erste große Firma im Jahr 1989 einen Outsourcingvertrag mit *IBM*, *DEC* und *Businessland* geschlossen, welcher alle bis dahin bekannten Outsourcingdeals nicht nur wegen seines Millionenvolumens in den Schatten stellte, sondern auch die Neuigkeit besaß, dass erstmals ein als wichtig erachteter Geschäftsbereich komplett ausgelagert wurde (Applegate und Montealegre, 1991; Dibbern et al., 2004; Loh und Venkatraman, 1992).

Daraufhin haben große Softwarefirmen schnell das Potenzial von günstiger Arbeitskraft in Ländern mit günstigen Lohnklassen im asiatischen Raum entdeckt und vermehrt betriebseigene Zentren (sog. *Offshoring Centers*) geschaffen, um Aufgaben der IT dorthin auszulagern (Arnett und Jones, 1994; Carmel und Tjia, 2006; Young, 2000). Typischerweise wurden in diesen Zentren meist einfach zu bewältigende bzw. repetitive IT-Aufgaben durchgeführt, um teure Arbeitskräfte im Ursprungsland zu entlasten und Kapazitäten für andere Angelegenheiten zu schaffen. Die so entstandene Bündelung von Aufgaben schaffte Synergieeffekte in den jeweiligen Niederlassungen, so dass die spezifisch bereitgestellten Dienstleistungen auch für andere Firmen angeboten werden konnten (in sog. *Shared Service Centers*). Somit wurden vorzugsweise Aufgaben des Rechenzentrumsbetriebs, weltweiter Telefonsupport für IT oder auch die Durchführung von Softwaretests an das jeweilige *Shared Service Center* ausgelagert (Dressler, 2007; Heinzl, 1993; Klimpke et al., 2011). Betriebe konnten dadurch einzelne Funktionen ihrer IT-Abteilung selektiv auf den Prüfstand stellen und bei Bedarf und technischer Machbarkeit durch kosteneffizienten Fremdbezug ersetzen. Die Software blieb als Ganzes aber im Besitz der Firma. Man spricht deshalb von selektivem Outsourcing (Heinzl, 1993; Lacity und Willcocks, 1998; Lacity et al., 1996). Bisherige Formen des Outsourcings können wie in Abbildung 2.9 dargestellt zusammengefasst werden.

Dadurch wurde auch für kleine und mittelständische Softwareunternehmen (KMU) die Möglichkeit geschaffen, von den zentral bereitgestellten Dienstleistungen zu profitieren und am globalen Trend des IT-Outsourcings teilzunehmen. Auch diese konnten damit Teile ihres Softwareentwicklungsprozesses selektiv auslagern, um auf dem Markt wettbewerbsfähig zu bleiben. Die Outsourcingaktivitäten von deutschen KMU konzentrieren sich in dieser Konstellation vermehrt auf osteuropäische Outsourcinganbieter (Klimpke et al., 2011).

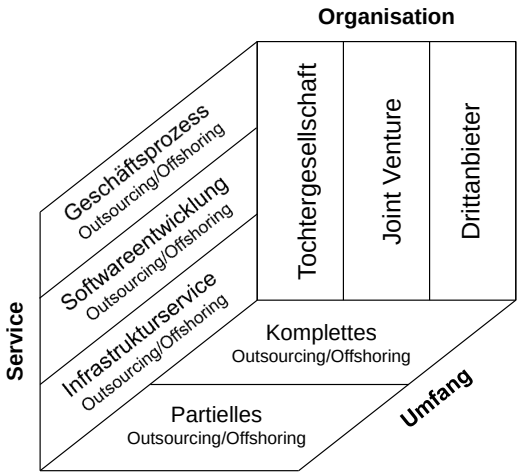

Abbildung 2.9.: Formen des Outsourcings (abgeleitet von Wiener, 2006, S. 37)

Ein interessanter Aspekt beim IT-Outsourcing ist der Teilbereich der Softwareentwicklung, der den gesamten Lebenszyklus eines Informationssystems umfasst. Dieser wird meist in zwei Bereiche unterteilt, nämlich die Anwendungsentwicklung (*Application [Development] Outsourcing*) und die Wartung der Software (*Maintenance Outsourcing*). Diesen beiden Abschnitten eines Softwareprojekts liegen jeweils komplexe Abläufe zugrunde (vgl. Abschnitt 2.3), die eine Fremdvergabe an Drittanbieter erheblich erschweren (Grover et al., 1996).

Auch aus wissenschaftlicher Sicht wurde das Thema Outsourcing von Informationssystemen zahlreich und aus unterschiedlichen Blickwinkeln erforscht. So gliedern Dibbern et al. (2004) in einer umfassenden Synopse der Outsourcingliteratur die bestehenden Forschungsarbeiten entlang ihrer Untersuchungsobjekte und identifizieren dabei zwei Kernbereiche. Bis dato verfasste wissenschaftliche Arbeiten lassen sich einerseits nach der Frage der Outsourcingentscheidung von Unternehmen gliedern, bei der es darum geht, warum ausgelagert wird, was ausgelagert wird und welcher Entscheidungsfindungsprozess verfolgt wird. Andererseits wurde untersucht, wie die Ergebnisse der Outsourcingentscheidungen umgesetzt wurden (Dibbern et al., 2004; Lacity et al., 2010). Die Zusammenfassung legt offen, dass bisherige Studien die Outsourcingentscheidung auf rein strategischer Ebene von Unternehmen erforschen. Parallel dazu geben Dibbern et al. (2004) einen Überblick über die den Beiträgen zugrundeliegenden wissenschaftlichen Theorien, von denen die Transaktionskostentheorie am häufigsten

zur Herleitung und Erklärung von Zusammenhängen verwendet wird (Ang und Slaughter, 1998; Chalos und Sung, 1998; Lacity und Willcocks, 1995).

Die Outsourcingentscheidung steht bei weiteren Studien ebenfalls im Mittelpunkt. So entwickelten Fjermestad und Saitta (2005) ein Rahmenwerk mit kritischen Erfolgsfaktoren, das von Wiener (2006) aufgegriffen und erweitert wurde. Unter Berücksichtigung der Dimensionen wie Vertragsgestaltung, Ausrichtung der Unternehmensstrategie, kulturelle Hintergründe, Infrastruktur und Technologie oder auch Managementsupport und vielen weiteren Faktoren, entstanden damit erste Ratgeber für Praktiker, die sich mit Auslagerungsentscheidungen konfrontiert sehen.

Die praktische Sichtweise auf das vorliegende Thema greifen auch Lacity et al. (2009) in ihrer Studie über die praxisrelevanten Themen des IT-Outsourcings auf. Sie legen in ihrer Analyse von 191 Forschungsbeiträgen neben den Eigenschaften von Firmen, die bevorzugt Teile ihrer IT-Abteilungen auslagern, ebenfalls die strategischen Zielsetzungen sowie Voraussetzungen aus Kunden- und Anbietersicht offen, die bis dahin zu erfolgreichen Outsourcingkooperationen geführt haben (Lacity et al., 2009). So werden vor allem Firmen, deren IT-Abteilung nur mäßige Performanz bei der Entwicklung von neuen Informationssystemen aufzeigen, als besonders affin für Outsourcing identifiziert (Hall und Liedtka, 2005; Mojsilovic et al., 2007). Die Firmengröße von Unternehmen, die Outsourcingbeziehungen pflegen, wird hingegen als nicht relevanter Faktor für die Entscheidung gewertet, weil große, mittelständische und kleine Firmen gleichermaßen am Outsourcing teilnehmen (Grover et al., 1994). Aus strategischer Sicht kommen Lacity et al. (2009) zu dem Schluss, dass der Erfolg des Outsourcings maßgeblich aus der Kombination einer guten Entscheidungsfindung mit einer guten vertraglichen sowie kooperativen Steuerung des Vorhabens zusammenhängt. Ebenfalls werden komplementäre Fähigkeiten von Lieferanten und Kunden als Voraussetzung für bessere Ergebnissen beim Outosurcing gesehen (Lacity et al., 2010).

Komplementär zu den bisherigen Betrachtungen von Outsourcingentscheidungen auf strategischer Organisationsebene, liefert der Beitrag von Dibbern et al. (2012a) wertvolle Erkenntnisse im Outsourcingkontext über die systemischen Verknüpfungen einzelner Informationssysteme zu den übrigen betrieblich verwendeten Systemen eines Unternehmens (Ariav und Ginzberg, 1985; Davis und Olson, 1985). Im Gegensatz zur bisherigen Annahme, dass Entscheidungen zur Auslage-

rung einzelner Informationssysteme unabhängig von weiteren bereits implementierten Systemen getroffen werden können, konnten landesspezifische Unterschiede der Verflechtung von Informationssystemen auf die Entscheidung aufgedeckt werden. So ergibt sich für das moderat individualistische Deutschland, dass sich die erzielbaren Vorteile durch eine optimale Verflechtung von Informationssystemen auf die Auslagerungsentscheidung auswirken. Stark integrierte System werden daher bevorzugt nicht ausgelagert (Dibbern et al., 2012a). Technische und systemische Eigenschaften von Informationssystemen rücken damit verstärkt in der Vordergrund der Betrachtung von Outsourcingentscheidungen.

Aktuelle Forschungsarbeiten zum Thema IT-Outsourcing beschäftigen sich mit der Verteilung der Kontrolle über den Outsourcingprozess zwischen Anbieter und Kunde (Gregory et al., 2013; Huber et al., 2014). Für eine erfolgreiche Projektdurchführung in einem derart dynamischen und unsicheren Umfeld, wie das bei Outsourcingprojekten der Fall ist, werden daher formelle Kontrollmechanismen vorgeschlagen, die exakt auf den spezifischen Kontext jedes einzelnen Projekts zugeschnitten sind. Sie behandeln zwar einen operativen Teilaspekt von Outsourcingbeziehungen, befinden sich aber in einer nachgelagerten Stufe zur Entscheidungsfindung und grenzen sich damit vom vorliegenden Thema ab.

2.4.1.1. Beweggründe für das Outsourcing

Zahlreiche Beweggründe für kleine, mittelständische und große Unternehmen haben die facettenreichen Entwicklungen im IT-Outsourcing motiviert und dazu geführt, dass immer mehr Outsourcingprojekte zur Auslagerung von einzelnen Aufgaben der IT-Abteilung sowie von Aufgaben aus dem Softwareentwicklungsprozess durchgeführt werden. Einflussfaktoren, die die Auslagerung motivieren, lassen sich in drei wesentliche Kategorien einteilen: Globalisierung, Kosteneinsparungen und strategische Vorteile. Sie werden im weiteren Verlauf dieses Kapitels näher erläutert.

Globalisierung in der Softwarebranche Aus historischer Sicht ist der technologische Aufschwung von Entwicklungsländern weitgehend auf den Erfolg von Firmen aus Industrienationen zurückzuführen, da zunächst nur große Unternehmen umfangreiche Outsourcingaufträge zur Entwicklung und Auslagerung von

standardisierten Prozessen vergeben haben und damit versucht haben, den eigenen Ertrag zu maximieren. Die hohen Kosten eines solchen Vorhabens ermöglichte es zunächst nur großen und zahlungswilligen Unternehmen, über nationale Grenzen hinweg derart umfangreiche Vorhaben durchzuführen. Solche Firmen werden von Bartlett und Ghoshal (2000) als international, multinational, transnational oder global klassifiziert, die Kooperationszwecke von der Versorgung mit Rohmaterialien bis hin zur vollständigen Integration zur Erzielung größtmöglicher Synergieeffekte verfolgen (Bartlett und Ghoshal, 2000). Neuartige Kommunikationsmöglichkeiten sowie die Verbesserung der Infrastruktur in Entwicklungsländern ermöglichten es schließlich auch kleinen und mittleren Softwareunternehmen, sogar auch Neugründungen (*Start-Ups*), Partnerschaften mit Firmen in den aufstrebenden Ländern aufzubauen. Die Zusammenarbeit über Grenzen und Zeitzonen hinweg wird durch virtuelle Konferenzräume und durch viele weitere innovative Möglichkeiten zur Kollaboration in dieser „*New Economy*“ stark vereinfacht. Unternehmen profitieren dabei besonders von externer Expertise, die oftmals günstiger zu erwerben ist, als im Herkunftsland selbst (Saxenian, 2002). Die weltweit gespannten Netzwerke zwischen IT-Dienstleistungsanbietern und deren Abnehmern ermöglichen es nun Unternehmen nicht nur als lokaler, sondern auch auch als globaler Marktteilnehmer zu agieren und damit in zahlreichen Märkten Fuß zu fassen (Herbsleb und Moitra, 2001).

Diese Entwicklung wird als Globalisierung der IT-Industrie bezeichnet, da Firmen mittlerweile in langfristige Partnerschaften zu externen Dienstleistern aus Entwicklungs- und Schwellenländern investieren, um Projekte über große Entfernungen, mehrere Zeitzonen und verschiedene Kulturen hinweg in einer Atmosphäre des gegenseitigen Vertrauens durchzuführen. Das ursprüngliche *Body-Shopping*, die Beschaffung ausländischer Arbeitskräfte, die zum Arbeiten ins Heimatland des Unternehmens geholt werden, nimmt eine Vorreiterrolle für die jetzige globale Entwicklung ein. In der neuen Konstellation profitieren Unternehmen von der zusätzlichen Arbeitskraft in Form einer partnerschaftlichen Beziehung oder als Gemeinschaftsunternehmen (*Joint Venture*) von globalen Vorteilen sowie einem globalen Image (Heinzl und Sinß, 1993; Sahay et al., 2003).

Vom Globalisierungsgedanken geleitet und vom Konkurrenzdruck getrieben, bietet sich Outsourcing für KMU der Softwarebranche als hilfreiche Vorgehensweise an. Das Netzwerk von Dienstleistungsanbietern für IT-Outsourcing ist weltweit

gespannt und lockt besonders im asiatischen Raum mit einer großen Kapazität an menschlichen Ressourcen zu vergleichsweise günstigen Preisen im Gegensatz zu den Personalkosten in Industrienationen (Qu und Brocklehurst, 2003). Deutsche KMU bevorzugen bei der Standortwahl ihrer Outsourcingpartner jedoch eine geringe räumliche, zeitliche sowie kulturelle Distanz und wählen daher bevorzugt Unternehmen aus dem osteuropäischen Ausland, um sich den besonderen Herausforderungen (wie im Kapitel 2.4.1.2 dargelegt) solcher Outsourcingkonstellationen besser stellen zu können (Klimpke et al., 2011).

Kosteneinsparungen Die Reduktion von Kosten stellt neben den Globalisierungsaspekten einen weiteren wesentlichen Anreiz für Unternehmen dar, Outsourcingbeziehungen aufzubauen. In erster Linie können Unternehmen durch Auslagerung von IT-Funktionen an Länder mit geringerem Lohnniveau deutliche Einsparungen realisieren und damit Software kostengünstiger entwickeln. Der Trend hin zu Budgeteinsparungen in IT-Abteilungen, der in wirtschaftlich unsicheren Zeiten verstärkt zu beobachten ist, wird dadurch optimal unterstützt und führt dazu, dass IT-Projekte kosteneffizient und als Outsourcingkooperationen mit vorhersagbaren Kosten durchgeführt werden können (Rifkin, 1991).

Dennoch besteht die Gefahr von impliziten Risiken, wie ausufernden Kosten oder zeitlichen Verzögerungen bei der Auslieferung von Software, die sich im Falle eines Eintritts ebenfalls negativ auf den Erfolg von Softwareunternehmen niederschlagen. Zur Vermeidung solcher unvorhergesehener Kosten werden schließlich Outsourcingkonstellationen bei Softwareentwicklungsprojekten bevorzugt, um die möglichen Risiken durch vertragliche Strafzahlungen bei Nichterfüllung der Softwarespezifikation oder bei zeitlichem Verzug zu minimieren. Das Risiko wird damit teilweise oder ganz auf den Zulieferer abgewälzt (Apte, 1990; Clermont, 1991).

Ein zusätzliches Einsparpotenzial für Unternehmen, die Outsourcingbeziehungen eingehen, stellt die Möglichkeit dar, eigenes Entwicklungspersonal abzubauen, das aufgrund der ausgelagerten Tätigkeiten nicht mehr im Betrieb selbst vorgehalten werden muss (Apte, 1990). Obwohl dies zunächst als Vorteil erscheint, kann sich ein zu starker Fokus auf die Kostenreduzierung auch nachteilig auswirken, weil dadurch wertvolle und talentierte Mitarbeiter entlassen werden, für die bei erneutem Bedarf kaum Ersatz auf dem umkämpften Arbeitsmarkt zu finden sein

wird. Diese und weitere Herausforderungen werden detailliert im Kapitel 2.4.1.2 beleuchtet.

Kosteneinsparungen stehen auch bei KMU in der Softwarebranche neben Flexibilität an erster Stelle der Motivationsfaktoren (Klimpke et al., 2011). Für sie steht dabei im Vordergrund, sowohl konkurrenzfähige Preise für ihre Softwareprodukte anbieten zu können, als auch situativ auf zusätzliche Kapazitäten für den Entwicklungsprozess entsprechend der Nachfrage zurückgreifen zu können. Die Motivation der Globalisierung reiht sich bei KMU mit weiteren Beweggründen, wie z.B. das Auffinden neuer Talente, nachgelagert ein (Klimpke et al., 2011).

Strategische Vorteile Neben dem Erfolg in Verbindung mit dem globalen Image, also der primären Zielsetzung von Unternehmen (Schierenbeck, 2003), spielt die strategische Ausrichtung international agierender Unternehmen im Vergleich zu ihren Wettbewerbern eine entscheidende Rolle. Dies beinhaltet auch die Kombination von Aktivitäten, die in Eigenerstellung oder durch Fremdbezug erfolgen (Carmel und Agarwal, 2001). Besonders der direkte Vergleich der Strategie eines Unternehmens zu einem Wettbewerber und dabei entdeckte Vorteile rufen einen gegenseitigen Wettbewerb untereinander hervor, bei dem sich immer schnellere Anpassungszeiten auf die Strategiewechsel konkurrierender Firmen ergeben. Um dem entgegenzuwirken, bedienen sich Unternehmen einer dynamischen Produktionsweise, die sie hinsichtlich der Kosten wettbewerbsfähig hält und gleichzeitig eine dem Outsourcing aufgeschlossene Mentalität hervorruft (Carmel und Agarwal, 2001; Grinter et al., 1999).

Softwareunternehmen profitieren bei Anwendung solcher Mechanismen vor allem von der Beschleunigung ihrer Produktionszeiten sowie der Agilität und Flexibilität ihres Produktionsprozesses. Dadurch können Projekte in einem kürzeren Zeitraum und unter geringerem Personalaufwand durchgeführt werden. Besonders die feine Granularität der Aufgaben in ausgelagerten Softwareentwicklungsprojekten, die in Form von einzelnen Support- und Änderungsanfragen oder Implementierungsaufgaben weltweit umgesetzt werden können, ermöglicht Zeiteinsparungen bei entsprechend geplanter Einbeziehung von Outsourcingpartnern. So kann z.B. in einem über drei Kontinente verteilten Outsourcingprojekt der bearbeitete Softwarecode am Abend vom Softwareanbieter in Deutschland an den Zulieferer in Kanada übergeben werden. Nach dessen Bearbeitung wird der Code an den

Partner in Indien übergeben und landet schließlich wieder zur weiteren Bearbeitung in Deutschland. Durch Anwendung des *Follow-the-Sun*-Prinzips werden in Softwareprojekten enorme Entwicklungszeiten erzielt (Carmel und Agarwal, 2001). Unter der Prämisse einer kostenneutralen Transition des Softwarecodes von einer Lokation zur anderen könnte damit eine dreifach so schnelle Lieferzeit für die gewünschte Software erreicht werden. Dieser optimistische Fall kann sich auch ins Negative umkehren und bei mangelnder Kommunikation zwischen den Entwicklern der jeweiligen Standorte die Arbeit eines ganzen Arbeitstages (nun bestehend aus drei Tagesschichten) zunichte machen (Carmel und Nicholson, 2005; Carmel und Tjia, 2006).

Ein weiterer strategischer Vorteil des Outsourcings stellt der Erwerb von zusätzlichem Humankapital dar (Carmel und Tjia, 2006). Hierbei spielt die Gewinnung von stark spezialisierten Outsourcingpartnern eine wichtige Rolle. Nicht selten werden dadurch unentdeckte Talente mit besonderen Fähigkeiten in bestimmten Domänen entdeckt. Dies spielt vor allem Softwareunternehmen in die Karten, die stets auf frische Ideen und besondere Kreativität angewiesen sind, wenn es um das Design von neuen Softwarelösungen in der Entwicklungsphase geht. Daher sind Firmen mit Outsourcingkonstellationen bei der Entdeckung und Auswahl neuer Talente besonders bevorteilt (Carmel und Tjia, 2006; Herbsleb und Moitra, 2001).

2.4.1.2. Herausforderungen und Risiken des Outsourcings

Obwohl Unternehmen, ungeachtet ihrer Größe, aus eigenen strategischen Gründen oder durch Auflagen von Endkunden oftmals zum IT-Outsourcing gedrängt werden (Dibbern et al., 2004; Klimpke et al., 2011), ist der Anteil der kleinen, mittleren und großen Firmen, die Outsourcing meiden, nach wie vor hoch (Carmel und Tjia, 2006). Dafür ist eine Vielzahl von Herausforderungen verantwortlich, die eine Outsourcingkonstellation in der Softwarebranche erschweren und damit möglicherweise die erhofften Outsourcingeffekte neutralisieren oder ins Negative umwandeln würden. Diese Risiken werden in den nachfolgend beschriebenen Rubriken mit besonderem Fokus auf das Thema Softwareentwicklung detailliert erläutert.

Kulturbarrieren. Das Sprichwort *„Andere Länder, andere Sitten!“* gilt auch beim Outsourcing in der Softwareindustrie. In der Evolution der Menschheit liegt die unterschiedliche Herausbildung von vorherrschenden Normen, Werten und Vorstellungen, die tief und fest im Geist eines Individuums verankert sind, begründet. Dementsprechend unterscheiden sich auch die Normen und Verhaltensweisen bei der zwischenmenschlichen Kommunikation von einem Kulturkreis zum anderen. Dies erschwert die Zusammenarbeit bei Softwareprojekten erheblich, da bei diesem kommunikationsintensiven Prozess falsch interpretierte Nachrichten und Ideen zunächst zu Missverständnissen und damit zu fundamentalen Problemen in der Softwarelösung führen (Carmel und Tjia, 2006; Dibbern et al., 2008; Heeks et al., 2001; Herbsleb und Moitra, 2001). Besonders fehleranfällig wird die Kommunikation zwischen Kulturkreisen, die sich sehr stark voneinander unterscheiden. Aus diesem Grund entscheiden sich deutsche KMU bei der Wahl ihrer Outosurcingpartner vorwiegend für Unternehmen aus dem osteuropäischen Ausland, um diesen Unsicherheitsfaktor und die damit einhergehende Fehleranfälligkeit der gemeinsamen Kommunikation zu minimieren (Dibbern et al., 2012b; Klimpke et al., 2011; Winkler et al., 2008).

Soziale Verbundenheit. Die Zusammenarbeit in einem Team an einem gemeinsamen Ort schweißt die Teammitglieder zusammen und schafft dadurch ein Gefühl der sozialen Verbundenheit unter den Beteiligten (Carmel und Tjia, 2006). Ähnliche Interessen, der tägliche Austausch über Privates und Berufliches sowie gemeinsam verbrachte Pausen verstärken das Gefühl der Verbundenheit und schaffen eine Umgebung gegenseitigen Vertrauens und der Zusammengehörigkeit. Dies hat zur Konsequenz, dass sich die Mitglieder einer derart eingeschworenen Gemeinschaft gegenseitig aushelfen und auch bereit sind, über das normale Maß hinaus zu arbeiten, um ein gemeinsames Ziel zu erreichen (Carmel und Tjia, 2006; Heeks et al., 2001). In einem verteilten Szenario können sich eine solche Gemeinschaft und solch enge Bindungen deshalb nur sehr schwer bis gar nicht entwickeln (Herbsleb und Moitra, 2001).

Koordinationskomplexität. Die Entwicklung von Anwendungssoftware ist ein komplexer Prozess, der aus zahlreichen Einzelschritten besteht. Der Prozess ist stark von reziproken Interdependenzen geprägt, da die Ergebnisse eines Teams

als Ausgangsbasis für ein anderes Team dienen (Thompson, 1967). Hierfür ist ein erheblicher Aufwand an Koordination notwendig, damit sich am Ende die einzelnen Ergebnisse zu einem Gesamtsystem zusammenfügen lassen. Die Vermutung, dass sich die Restlaufzeit von verspäteten Softwareprojekten durch Erhöhung der Entwicklerzahl verkürzen lässt, wird im Brooksschen Gesetz widerlegt (Brooks, 2003). Es zeigt, dass die Softwareentwicklung von komplexen Koordinationsprozessen geprägt ist, die einen erheblichen Aufwand zur Koordination von Mitarbeitern und Aufgaben erfordern.

Beispielsweise muss im Outsourcingkontext bei Rückfragen eines Entwicklers, wenn schnelle oder unkomplizierte Lösungsansätze benötigt werden, oder bei Anfragen und Änderungswünschen des Kunden die Koordination über mehrere Stellen (lokal und entfernt) erfolgen. Dies bringt einen erhöhten Kommunikationsaufwand für die Koordination der Entwicklungsteams mit sich, der sich bei sprachlichen Unterschieden (vgl. Sprachbarrieren) unterschiedlich stark auswirken kann. Sogenannte *Water-Cooler-Talks*, frei übersetzt als Gespräche in der Kaffeeecke bezeichnet, bei denen sich die Entwickler informell über ihre Implementierungsaufgaben austauschen können und dadurch intrinsisches Wissen ansammeln, sind in verteilten Entwicklungsumgebungen nicht durchführbar. Dies hat zur Folge, dass deshalb kleinere Anpassungen bei der Entwicklung von Software über einen komplexen Änderungsprozess gesteuert werden müssen, die vorher auf informeller Ebene zwischen den Entwicklern abgesprochen und unmittelbar umgesetzt wurden (Carmel und Tjia, 2006; Heeks et al., 2001).

Steuerungsaufwand. Eine der Aufgaben des Managements ist die Überwachung und Steuerung sowie, bei Bedarf, die Eskalation laufender Softwareprojekte. Dafür verschaffen sich Manager in persönlichen Arbeitstreffen einen Überblick über den tagesaktuellen Projektstatus. Sie bekommen dadurch ein Gefühl, wie sie die Beschwerden, die Sorgen, aber auch den Fortschritt von Mitarbeitern interpretieren müssen, um die aktuelle Projektlage richtig einschätzen zu können. Ein erhöhter Steuerungsaufwand ergibt sich im Falle von Outsourcingbeziehungen einerseits durch die räumliche Distanz der einzelnen Teams und des Managements, durch die erhöhte Koordinationskomplexität im Entwicklungsprozess und andererseits durch die Schwierigkeit der der Entwicklungsaufgabe selbst (Carmel und Tjia, 2006; Dibbern et al., 2008; Herbsleb und Moitra, 2001).

Risikofaktoren. Unterschiedliche Risikoarten beeinflussen zusätzlich zu den Herausforderungen aus den vorherigen Abschnitten die Outsourcingentscheidung und müssen im Vorfeld eines solchen Vorhabens berücksichtigt werden. So können sich die zuvor beschriebenen Herausforderungen bei unsachgemäßer Planung ebenfalls in Risiken umwandeln und sich gefährdend auf den Gesamterfolg des Projekts auswirken.

Als Kostenrisiko werden beim Outsourcing schwer antizipierbar Kosten sowie für die Kooperation notwendige Extrakosten identifiziert. So sollten möglicherweise auftretende Kosten stets einkalkuliert werden, wie z.B. unerwartet hoher Reiseaufwand im Falle der Notwendigkeit von persönlichen Arbeitstreffen bei einer Eskalation oder unerwartet auftretende Lizenzkosten, die für die Entwicklung notwendig sind aber vertraglich nicht geregelt wurden. Außerdem existiert das Risiko der Extrakosten, die bei der Transition eines Projekts zum Outsourcingpartner entstehen. Diese Kosten dürfen das durch Outsourcing entstandene Einsparpotenzial nicht in vollem Umfang aufbrauchen (Dibbern et al., 2008).

Infrastrukturrisiken durch instabile Kommunikationsinfrastrukturen im Zielland des Outsourcings sind aus technischer Sicht besonders kritisch, und damit für IT-Projekte relevant. Für den Entwicklungsprozess von Software ist jedoch ein kontinuierlicher Austausch von Daten und Nachrichten erforderlich. Unzuverlässige Verbindungen zwischen den Partnern oder sogar Ausfälle würden sich negativ auf die Produktivität der Kooperation auswirken und sollten daher im Vorfeld bei der Standortwahl berücksichtigt werden (Carmel und Tjia, 2006).

Rechtliche Risiken bedingt durch gesetzliche Vorgaben hinsichtlich des Datenschutzes erweisen sich beim Outsourcing ebenfalls als kritisch. Unter diesen Gesichtspunkten muss der Auftraggeber sicherstellen, dass die neu erstellten sowie betrieblichen Daten, die untereinander ausgetauscht werden, entsprechend geschützt werden und nicht in falsche Hände geraten (Carmel und Tjia, 2006).

All diese risikobehafteten Faktoren sind bei einer Outsourcingentscheidung gründlich mit den Beweggründen (vgl. Unterkapitel 2.4.1.1) abzuwägen. Der Entscheidungsfindungsprozess gestaltet sich damit als ein komplexes Unterfangen, das aus mehreren Prozessschritten besteht, die im nächsten Kapitel beschrieben werden.

2.4.2. Die Outsourcingentscheidung

Die Outsourcingentscheidung bei der Entwicklung von Informationssystemen beschreibt das Zusammenspiel der unterschiedlichen Einflussfaktoren wie z.B. Chancen und Risiken beim Outsourcing (vgl. Unterkapitel 2.4.1.1 und Unterkapitel 2.4.1.2) mit den erzielbaren Ergebnissen einer solchen Kunden-Lieferanten-Beziehung. Diese Entscheidung steht damit unausweichlich am Beginn von IT-Auslagerungsprozessen und dient als Hilfestellung für oder gegen Outsourcingkooperationen. Im Detail wird damit abgewogen, mit welchem Lieferanten eine Kooperation eingegangen werden soll, welche Teile eines Informationssystems fremdvergeben werden sollen und welche Vereinbarungen vertraglich geregelt werden müssen (Grover et al., 1996; Koh et al., 2004; Lee et al., 2003; Smith und McKeen, 2004; Straub et al., 2008).

Unternehmen, die Outsourcing betreiben möchten, sind zunächst mit einer Vielzahl von Entscheidungsfeldern konfrontiert, die sie sorgsam gegeneinander abwägen müssen, bevor sie mit der eigentlichen Auslagerung der Arbeit und der verteilten Entwicklung beginnen können. Dazu zählen z.B. Entscheidungen über den Umfang des Outsourcings, die Möglichkeit der Einbindung entfernt gelegener Betriebsstandorte, die Planung und das Management eines solchen Vorhabens oder auch den Aufbau eines Kommunikationsnetzwerks und entsprechender Infrastruktur zu treffen (Lacity et al., 2010). Ist schließlich ein Lieferant ausgewählt, so empfehlen Lee et al. (2003) eine Liste von „optimalen“ Vorgehensweisen zur Handhabung des Kooperationsverhältnisses, um offensichtliche Fehlerquellen von Beginn an zu vermeiden:

- *Gemeinsames Verständnis aufbauen.* Eine gemeinsame Wissensbasis sollte für die angestrebten Ziele geschaffen werden.
- *Kurzfristige und langfristige Ziele festlegen.* Gemeinsame Ziele sollten priorisiert werden, um den langfristigen Fokus nicht zu verlieren.
- *Klare und realistische Ziele.* Unter Berücksichtigung einer zu erwartenden Lernkurve sollten erreichbare Ziele klar und deutlich vereinbart werden.
- *Einklang zwischen Nutzen und Risiken.* Über mögliche Risiken und zu erwartenden Nutzen sollten explizite Vereinbarungen getroffen werden.

- *Festlegung der Leistungsmessung.* Klare Standards zur Leistungsmessung und Überprüfung sollten definiert und kommuniziert werden.

- *Veränderungen und Nachbesserungen am Endprodukt.* Verbesserungen lassen sich durch die Überarbeitung und Verfeinerung erzielen und sollten daher stets mit einkalkuliert werden.

- *Das Unerwartete vorhersehen.* Das vorläufige Testen von 'was passiert, wenn...' Szenarien und die Erörterung von Handlungsoptionen sollte unerwartete Risiken abfedern.

- *Die Geschäftsbeziehung hegen.* Eine erfolgreiche Partnerschaft benötigt kontinuierliche Pflege zur gemeinsamen Wertsteigerung.

Aus wissenschaftlicher Sicht nehmen neben den Motivatoren und Risiken auch die Fähigkeiten des Zulieferers, Möglichkeiten der Nachahmung sowie Transaktionsaspekte eine beeinflussende Rolle bei der Outsourcingfrage ein, wie Lacity et al. (2010) auf S. 408 zeigen. Die dort aufgezeigten Determinanten sind die am zahlreichsten in der IT-Outsourcingliteratur untersuchten Faktoren, die sich auf die Entscheidung auswirken und deren Zusammenhänge vielfach gezeigt werden konnten (Lacity et al., 2010, 2009). Die Outsourcingentscheidung selbst hat ebenfalls zusammen mit weiteren Determinanten einen wichtigen Einfluss auf das Resultat und damit den Erfolg des IT-Ousourcings (vgl. Abbildung 2.10).

Anhand der Abbildung 2.10 wird deutlich, dass technische Eigenschaften der Softwareprodukte und prozessuale Eigenschaften des Entwicklungsprozesses in der Literatur kaum berücksichtigt wurden (Grover et al., 1996; Lacity et al., 2010).

Zusammenfassend wird unter der IT-Outsourcingentscheidung also die Entscheidung des Top-Managements eines Unternehmens zusammengefasst, ob ein Vorhaben als Outsourcingprojekt (totale Auslagerung, selektive Auslagerung, mit mehreren Lieferanten oder in einer Outsourcingallianz) durchgeführt werden soll oder nicht (vgl. Abbildung 2.9). Daran knüpft schließlich die Gestaltung der vertraglichen Steuerung der Geschäftsbeziehung sowie die Durchführung und Steuerung der Kooperation selbst an (Kern und Willcocks, 2002; Lacity et al., 2010, 2009; Poppo und Zenger, 2002).

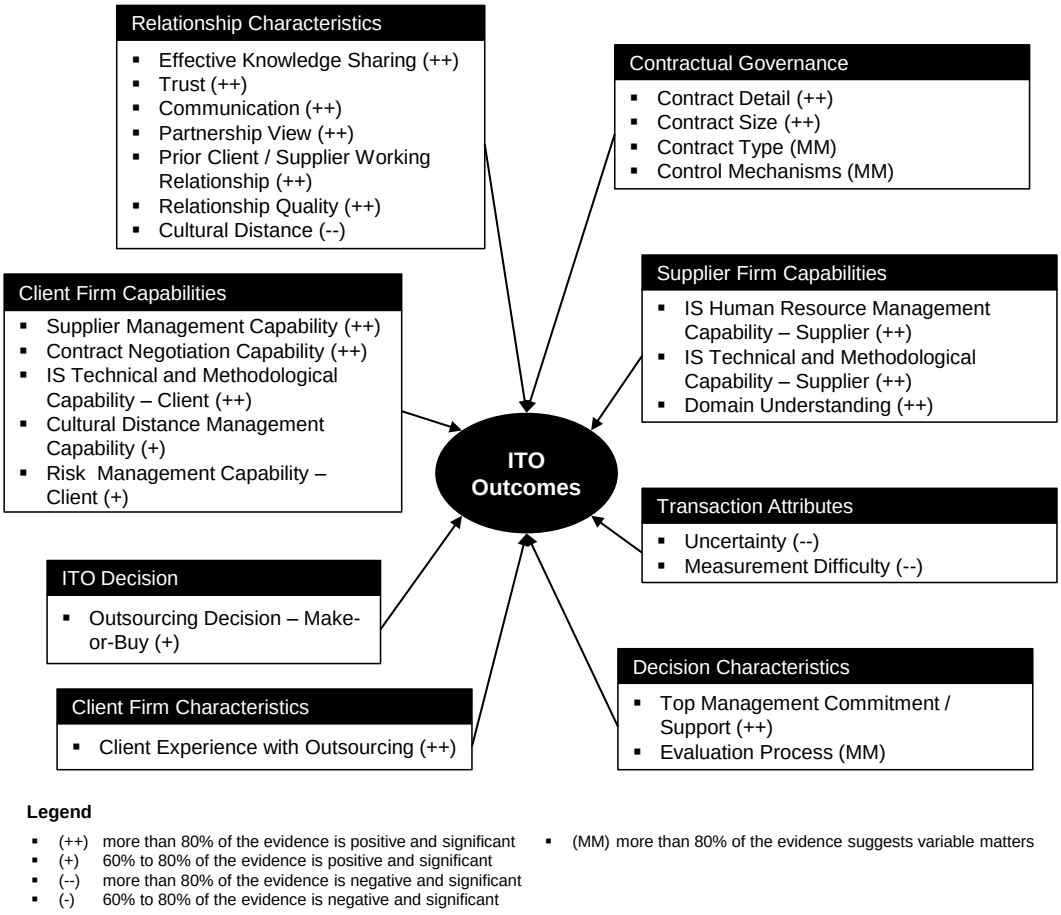

Abbildung 2.10.: Ergebnis einer Literatursynopse über IT-Outsourcing nach Lacity et al. (2010)

2.4.3. Outsourcingentscheidungen im Softwareentwicklungsprozess

Im Rahmen dieser Arbeit werden technische Eigenschaften des Outsourcings in der Softwareentwicklung näher beleuchtet. Es stehen dabei die Softwareprodukte und ihr Herstellungsprozess mit den jeweiligen Ausprägungen und Gestaltungsmöglichkeiten für die Fremdvergabe im Vordergrund. Daher muss die Entwicklung von Software entsprechend existierender Produktkategorien und Durchführungsoptionen differenziert werden (vgl. Abbildung 2.11). Es werden Klassifizierungen aus der Literatur zur globalen Softwareentwicklung (vgl. Kapitel 2.3.4) herangezogen und für ihre Verwendung im Rahmen des IT-Outsourcings diskutiert.

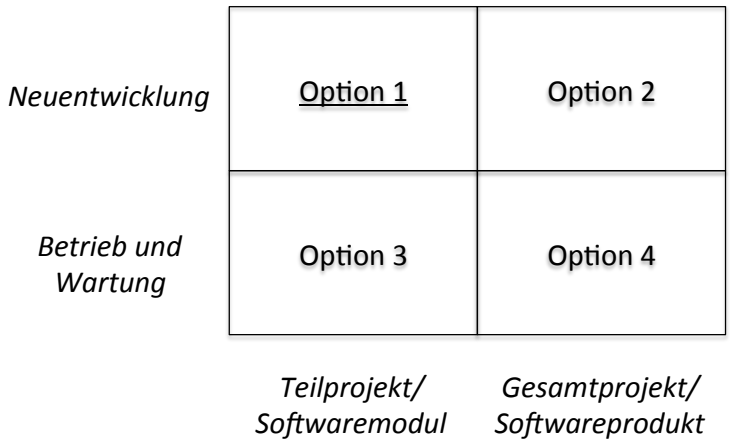

Abbildung 2.11.: Outsourcingoptionen im Softwareentwicklungsprozess

Bei der globalen Softwareentwicklung wird in erster Linie zwischen der *Neuentwicklung* und dem *Betrieb (inkl. Wartung)* von Software unterschieden (King, 2005; Smith et al., 1996). Durch die Outsourcingentscheidung wird dann festgelegt, was davon selbst durchgeführt und was zu einem externen Lieferanten oder einem entfernt gelegenen Firmenstandort ausgelagert werden soll (Prikladnicki et al., 2007).

Bei der Neuentwicklung von Software handelt es sich um die Entwicklung und Umsetzung von Softwaresystemen, die in der beabsichtigten Form so noch nicht existieren (Optionen 1 und 2 in Abbildung 2.11). Dies umfasst alle Phasen des Softwarelebenszyklus, in denen das Endprodukt noch nicht im produktiven Einsatz ist (vgl. Kapitel 2.3.2). Eine Besonderheit stellt die Anforderungserhebung dar, die stets in enger Abstimmung mit dem Auftraggeber erfolgen sollte. Dieser legt die Funktionalität und den Umfang des erforderlichen Endprodukts fest. Der gewählte Outsourcingansatz bestimmt dann den Grad der Inanspruchnahme von Fremdleistungen für die Durchführung der Neuentwicklung (Buxmann et al., 2011; Patane und Jurison, 1994). Entsprechend einer Studie mit 498 befragten Softwareunternehmen haben 79,1% derartige Fremdleistungen bei der Neuentwicklung eines Softwareprodukts in Anspruch genommen. 33,2% der Unternehmen ließen das Softwareprodukt sogar komplett von Zulieferern entwickeln (Buxmann et al., 2011).

Beim Outsourcing der Betriebs- und Wartungsphase des Softwarelebenszyklus wird die Betreuung eines sich im laufenden Betrieb befindlichen Softwaresystems in den Aufgabenbereich eines Zulieferers verlagert (Optionen 3 und 4 in Abbildung 2.11). Dieser Service beinhaltet z.B. die Einspielung von Softwareupdates, die Korrektur kleiner Softwarefehler oder auch die Anwenderbetreuung in Form einer Hotline (King, 2005). Laut Buxmann et al. (2011) betreibt diese Phase nur ein kleiner Teil der befragten Unternehmen selbst. Über 85% der Unternehmen lagern solche Dienstleistungen an Zulieferer aus (Buxmann et al., 2011).

Weiterhin werden Outsourcingentscheidungen auf der Basis von *Projekt- oder Produktstrukturen* im Softwareentwicklungsprozess getroffen (Aspray et al., 2006; Herbsleb, 2007; Noll et al., 2010). Aus projektspezifischer Sicht sind Outsourcingentscheidungen darüber zu treffen, ob Teilprojekte oder das gesamte Entwicklungsprojekt an einen Zulieferer vergeben werden. Alternativ werden derartige Entscheidungen auch auf Basis der Produktstruktur getroffen, indem einzelne Komponenten des Gesamtprodukts oder des gesamte System durch Fremdleistungen erstellt werden (Buxmann et al., 2011).

Wenn für die Outsourcingentscheidung Projekteigenschaften herangezogen werden, steht die Ressourcenverfügbarkeit für die Durchführung des Softwareprojekts im Vordergrund (Kramer et al., 2013; Noll et al., 2010). Unter Berücksichtigung der verfügbaren Arbeitskräfte und deren Qualifikation für die Umsetzung der einzelnen Phasen des Softwareentwicklungsprozesses wird dann entschieden, ob Teilprojekte (einzelne Aufgaben oder Abschnitte eines Projekts) durch Fremdleistung erstellt werden oder ob ausreichend eigene Ressourcen zur Verfügung stehen, die ausgelastet werden müssen (Optionen 1 und 3 in Abbildung 2.11). Situationsbedingt kann auch eine Entscheidung zur Auslagerung des Gesamtprojekts (Optionen 2 und 4 in Abbildung 2.11) getroffen werden (Kramer et al., 2013).

Alternativ dazu werden Outsourcingentscheidungen auch basierend auf der Struktur des Softwareprodukts getroffen. Die Produktarchitektur spielt dabei die entscheidende Rolle (Herbsleb, 2007; Herbsleb und Moitra, 2001; Noll et al., 2010). Ein komponentenbasierter Aufbau des Softwaresystems ermöglicht die Untergliederung des Gesamtprodukts in weitgehend eigenständige Module oder Komponenten (Cai et al., 2000). Dieser Aufbau erlaubt es dann, Entscheidungen darüber zu treffen, welche dieser Einzelteile im eigenen Unternehmen entwickelt werden sollen oder durch Fremdleistungen zu beziehen sind. Die einzelnen

Entwicklungsteams übernehmen dabei die Verantwortung für ihre entwickelten Komponenten und sind auch Ansprechpartner, wenn Änderungen durchzuführen sind. Durch die Vermeidung von instabilen Architekturen wird bei Änderungen am Produkt das Risiko unklarer und überflüssiger Kommunikation reduziert, da die Anzahl der an der Änderung beteiligten Teams so gering wie möglich gehalten wird (Herbsleb, 2007; Noll et al., 2010). Ebenso kann auch bei dieser Vorgehensweise die Auslagerung des gesamten Softwareprodukts an einen Zulieferer entschieden werden.

Zusammenfassend sind für Outsourcingentscheidungen im Softwareentwicklungsprozess die Phasen des Softwarelebenszyklus maßgeblich. Es wird einerseits unterschieden, ob es sich um eine Neuentwicklung oder die Weiterentwicklung von Software im Rahmen der Betriebsphase handelt. Andererseits ist die Projekt- bzw. Produktstruktur eines Softwareentwicklungsprojekts relevant für die Entscheidungsfindung. Der Bezug eines Softwareprodukts ausschließlich durch Fremdleistung, also das totale Outsourcing, wird im Rahmen dieser Arbeit nicht weiter verfolgt, da hierbei keine Entscheidungen zur Differenzierung von einzelnen Softwarekomponenten auf der Basis softwaretechnischer Eigenschaften des Produkts zu treffen sind. Ebenso sind Entscheidungen, ob der Betrieb und die Wartung eines Softwareprodukts ausgelagert werden sollen, für diese Forschungsarbeit nicht relevant, da es sich dabei nur um kleinere Anpassungen oder Einstellungen einer Software und nicht um die Durchführung eines Softwareentwicklungsprozesses im eigentlichen Sinne handelt. Infolgedessen stehen die Neuentwicklung von Softwareprodukten und die Outsourcingmöglichkeiten von Teilaufgaben (also Komponenten) im Rahmen des Softwareentwicklungsprozesses im Mittelpunkt dieser Forschung (vgl. Option 1 in Abbildung 2.11).

2.4.3.1. Integration der Outsourcingentscheidung in den Softwareentwicklungsprozess

Wenn Outsourcingentscheidungen nicht als binäre Entscheidung (Auslagerung des gesamten Projekts oder Produkts), sondern im Sinne der Option 1 (vgl. Abbildung 2.11) durchgeführt werden sollen, muss dieser Vorgang in den Softwareentwicklungsprozess eingebettet werden, um die Teilprojekte und Softwarekomponenten bestimmen zu können, die durch den Bezug von Fremdleistung erstellt

werden sollen. Die einzelnen Phasen des Softwarelebenszyklus (vgl. Kapitel 2.3.2) vor Inbetriebnahme der Software müssen daher einzeln danach analysiert werden, ob die gesamte Phase oder einzelne Aufgaben daraus durch Zulieferer fremdbezogen werden können.

In einer Studie mit kleinen und mittelständischen Softwareherstellern wurde der Softwareentwicklungsprozesses dahingehend untersucht, welche der einzelnen Phasen davon beim Outsourcing teilweise oder ganz durch Fremdleistung erbracht werden können (Kramer et al., 2013). Die Ergebnisse legen offen, dass die frühen Phasen der Softwareentwicklung, bestehend aus Analyse und Erhebung der Anforderungen sowie dem Architekturentwurf des Softwaresystems, gar nicht oder nur sehr begrenzt an externe Dienstleister vergeben werden. Die Implementierung des Softwarecodes mit ihren zugehörigen Komponententests wird als zentrale Phase des Softwareentwicklungsprozesses hingegen von allen befragten Unternehmen mit der Hilfe externer Dienstleister durchgeführt. Nachgelagerte Phasen der Entwicklung wie Integrationstests und Qualitätssicherung werden einstimmig im eigenen Unternehmen durchgeführt (Kramer et al., 2013).

Die schlechte Eignung früher Phasen des Softwareentwicklungsprozesses für das Outsourcing sehen die teilnehmenden Unternehmen darin, dass gerade dort die Interaktion mit zukünftigen Anwendern von besonderer Bedeutung ist, weil sie die Anforderungen an das finale Softwareprodukt vorgeben. Hierbei ist den Firmen die Qualität der Anforderungen sehr wichtig, damit sich nicht schon sehr früh Fehler in das Produkt einschleichen, die im späteren Verlauf nur sehr kostspielig behoben werden können. Gleiches gilt für die Softwarearchitektur, welche als maßgeblicher Bauplan für die Entwicklung gilt (Kramer et al., 2013; Mikkola, 2003).

Die Umsetzung von Anforderungen und Architekturspezifikationen in Softwarecode wird von allen Unternehmen als die Phase des Softwarelebenszyklus eingeschätzt, die sich am besten für die Auslagerung oder den Bezug von Fremdleistungen eignet. Im Sinne der komponentenbasierten Softwareentwicklung lassen sich so einzelne Komponenten an Zulieferer auslagern, da diese eine nach außen hin abgeschlossene Funktionsbeschreibung und Schnittstelle besitzen (Cai et al., 2000). Eine Übermittlung zum externen Dienstleister ist in diesem Fall ohne großen Aufwand möglich, weil weder die Gesamtarchitektur des Endprodukts noch Wissen über andere Komponenten ausgetauscht werden muss. Eng damit

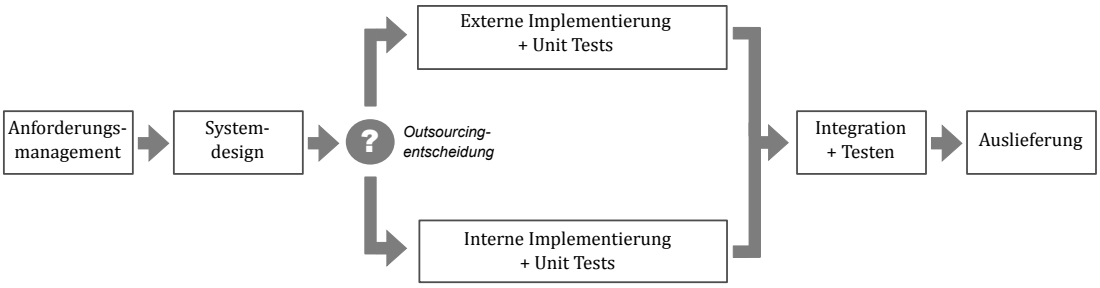

Abbildung 2.12.: Einbettung der Outsourcingentscheidung in den Softwareentwicklungsprozess

verbunden sind die Komponententests, die zur Überprüfung der Funktionalität und Konformität einzelner Softwaremodule zu ihren Schnittstellen dienen. Sie gehören zur Entwicklung einzelner Komponenten unmittelbar dazu und werden gleichermaßen wie ihre zugehörige Komponente vergeben (Baldwin und Clark, 2000; Kramer et al., 2013).

Die nachgelagerten Phasen der Entwicklung, also die Durchführung von Integrationstest aller Komponenten sowie die Qualitätssicherung des Gesamtprodukts, sollen nicht betrachtet werden. Die Auslagerung dieser Phasen ist aufgrund der anteiligen Verwendung von Fremdleistung bei der Implementierungsphase nicht möglich, da ein externer Dienstleister nicht im Besitz der gesamten Software ist, die als Ganzes getestet und geprüft werden soll (Kramer et al., 2013; Murray und Crandall, 2006).

In Abbildung 2.12 ist zusammenfassend in einem Prozessmodell dargestellt, wie sich Outsourcingentscheidungen in den Softwareentwicklungsprozess integrieren. Die entscheidende Stelle ist unmittelbar beim Phasenwechsel zwischen Systemdesign und Implementierung. Spätestens hier muss entschieden werden, welche Softwarekomponenten selbst entwickelt und welche extern vergeben werden.

Allgemeingültige Phasenmodelle für das IT-Outsourcing, in denen die Verwaltung und Durchführung eines kompletten Outsourcingprojekts angeleitet wird, um die Chancen eines erfolgreichen Projektabschlusses zu erhöhen und die Risiken zu minimieren, sind bereits etabliert (Cullen et al., 2005; Hodel et al., 2006). Sie geben Hilfestellungen und Handlungsempfehlungen zum Ablauf der einzelnen Phasen eines Outsourcingprojekts. Konkrete Unterstützung bei der Entscheidungsfindung, welche Komponenten ausgelagert werden sollen und welche

nicht, ist in diesen Modellen allerdings nicht zu finden. In der bisher verfügbaren Literatur konnte nun die Stelle im Softwareentwicklungsprozess identifiziert werden, an der notwendigerweise eine Entscheidung über die Verteilung der Entwicklungsarbeit getroffen werden muss, allerdings liefert die Literatur keine Hinweise darüber, welche Kriterien bei einer derartigen Outsourcingentscheidung herangezogen werden sollten und welche Methode sich zur Entscheidungsfindung als geeignet erweist. Ein derartiges Entscheidungsmodell wird im Rahmen dieser Forschungsarbeit erarbeitet.

2.4.3.2. Relevanz technischer Eigenschaften des Softwareprodukts für die Outsourcingentscheidung

Für die Entwicklung eines Entscheidungsmodells, das die Outsourcingentscheidung im Softwareentwicklungsprozess unterstützen soll, ist es erforderlich, die bis dahin verfügbaren softwaretechnischen Eigenschaften eines finalen Gesamtsystems und seiner Komponenten zu identifizieren. Diese Faktoren müssen dann mit in den Entscheidungsansatz einfließen, um die Frage nach der Verteilung der Softwarekomponenten auf das eigene Entwicklungsteam oder den Zulieferer besser zu informieren.

In Abbildung 2.12 wird deutlich, dass bis zur Bestimmung der Outsourcingentscheidung die Phasen des Softwaredesigns und Architekturentwurfs bereits durchlaufen sind. Darin müssen die Grundlagen für einen möglichst reibungslosen Ablauf der Entwicklungsphase mit anteiligem Fremdleistungsbezug geschaffen werden. Unabhängig vom Outsourcing verbessert eine auf Komponenten basierende Architektur des Softwaresystems die Wartbarkeit, Zuverlässigkeit und Gesamtqualität des Systems (Cai et al., 2000). Gleichzeitig wirkt sich die entworfene Architektur auf die Koordination eines Softwareentwicklungsprojekts aus (Conway, 1968; Herbsleb, 2007). Durch sie wird eine Kommunikationsstruktur vorgegeben und der Informationsfluss der Entwickler so gesteuert, dass die einzelnen Aufgaben innerhalb des Entwicklungsprozesses meist autonom durchgeführt werden können. Softwarekomponenten mit genau spezifizierten Schnittstellen können deshalb weitgehend unabhängig voneinander entwickelt werden (Baldwin und Clark, 2000; Herbsleb, 2007). Sie stellen damit ein Alleinstellungsmerkmal für die Outsourcingentscheidung dar. Die Interaktion zwischen Softwareteams, denen jeweils

eigene Komponenten zugeteilt werden, kann damit gering gehalten werden. Auch der Austausch mit externen Zulieferern kann so auf ein notwendiges Minimum reduziert werden (Herbsleb, 2007). Für die Outsourcingentscheidung sollte daher berücksichtigt werden, dass eng miteinander verbundene Softwarekomponenten am gleichen Standort oder sogar vom gleichen Team entwickelt werden, und dass weitgehend unabhängige Komponenten besser für das Outsourcing als die Eigenentwicklung geeignet sind, weil kaum Interaktion mit zentralen Schnittstellen oder anderen Komponenten notwendig ist.

Geht man im Softwareentwicklungsprozess eine weitere Phase nach vorne zur Anforderungserhebung, so existieren bereits Ansätze, die sich mit der Zusammengehörigkeit von Anforderungen beschäftigen. Dadurch wird ermöglicht, dass eng miteinander in Verbindung stehende Anforderungen gruppiert und analysiert werden können, um im Falle eines Architekturdesigns der Software bereits erste Anhaltspunkte für eine mögliche Modularisierung des Softwareprodukts zu liefern (Li et al., 2009; Myers, 1978; Troy und Zweben, 1981). Sofern in kleinen Softwareunternehmen auf die ausführliche Definition einer Architektur verzichtet wird, können diese anforderungsbasierten Ansätze herangezogen werden, um den Aufbau des Softwaresystems zu skizzieren.

Für die Gruppierung von Anforderungen bedient man sich der Konzepte der Nachvollziehbarkeit in Softwareentwicklungsprojekten (Hildenbrand, 2008) sowie der auf Graphen basierten Darstellungsweise von Anforderungen und deren Verknüpfungsinformationen (Heim et al., 2008). Einen pragmatischen Ansatz stellt Yaung (1992) bereit. Er stellt die Anforderungen eines Softwareentwicklungsprojekts als Knoten eines Graphen und bekannte Verbindungen zwischen diesen Anforderungen als gewichtete Kanten dar. Die Gewichte drücken dabei den Grad der Kohäsion zweier Anforderungen aus (Yaung, 1992). Diese Darstellungsweise kann anschließend als Vorlage für die Entwicklung des technischen Aufbaus des Softwaresystems dienen und die Definition der Softwarekomponenten anleiten. Sie fließt in bisherige Outsourcingentscheidungen nur indirekt als Ideengeber für den Architekturentwurf ein. Konkret lassen sich daraus jedoch technische Eigenschaften für das Softwareprodukt ableiten. Anhand des Grads der Verknüpfung von Anforderungen können Rückschlüsse auf die Komponentenstruktur und die in einer Softwarekomponente enthaltenen Anforderungen gezogen werden. Betrachtet man also die Anforderungen einer Komponente und deren weiteren Verknüp-

fungen zu Anforderungen anderer Komponenten, so lässt sich daraus ein Grad für den Zusammenhang zweier Komponenten ableiten. Dies stellt eine wichtige Information für die Outsourcingentscheidung dar und sollte in einem Entscheidungsmodell entsprechend berücksichtigt werden.

Ein Entscheidungsmodell, das die hier aufgeführten technischen Eigenschaften von Softwareprodukten und die Besonderheiten des Entwicklungsprozesses berücksichtigt sowie weitere Aspekte mit in die Entscheidungsfindung einbezieht, existiert derzeit nicht. Unter Berücksichtigung der hier vorgestellten Erkenntnisse wird deshalb im weiteren Verlauf dieser Arbeit ein Entscheidungsmodell für das Outsourcing in Softwareentwicklungsprozessen erarbeitet.

2.5. Zusammenfassung

In diesem Kapitel wurden die begrifflichen Grundlagen vermittelt, die für das Verständnis der Zielsetzung, also die Konzeption eines Entscheidungsmodells und Entwicklung eines mobilen EUS für das komponentenbasierte IT-Outsourcing, notwendig sind. Dabei wurde ein Einblick in die Themengebiete der Entscheidungsfindung und (mobiler) EUS, der Softwareentwicklung sowie des IT-Outsourcings gegeben.

Da die Entscheidungsunterstützung im Softwareoutsourcing das Thema der vorliegenden Arbeit ist, wurde im Grundlagenkapitel zunächst eine Übersicht über das Treffen von Entscheidungen sowie der Definition von Entscheidungsmodellen im Unternehmenskontext vorgestellt (vgl. Kapitel 2.1), bevor auf die Verwendung von EUS und die jeweiligen Typen von EUS näher eingegangen wurde. Komplexe Entscheidungsprobleme mit multiplen Einflusskriterien und vernetzten Anwendern zu unterstützen ist deren Kernaufgabe. Moderne EUS unterstützen diesen Aufgabenbereich durch ihre mobile Verwendung besser.

Während im weiteren Verlauf des Kapitels in den Grundlagen der Softwareentwicklung der Fokus auf die Vorgehensweise von verteilten Entwicklungsteams und deren Unterstützungsmöglichkeiten durch Kollaborationsplattformen gelegt wurde (vgl. Kapitel 2.3.4), standen im Grundlagenkapitel zum Thema Softwareoutsourcing (vgl. Kapitel 2.4) neben den Vor- und Nachteilen auch die bisherigen Ansätze zur Entscheidungsfindung im Vordergrund. Die bisherigen Ansätze zur

Beantwortung der Outsourcingfrage vernachlässigen jedoch die technischen Eigenschaften des zu entwickelnden Softwareprodukts und die damit verbundenen Merkmale des Herstellungsprozesses. Deshalb lassen sich im Rahmen von Phasenmodellen des IT-Outsourcings zwar Entscheidungen z.B. darüber treffen, welche Partner für die Zusammenarbeit ausgewählt werden oder welche Unternehmensfunktion sich für die Auslagerung eignet. Für die Entscheidungsfindung, welche Komponenten eines Softwareprodukts tatsächlich vom externen Zulieferer programmiert und welche Komponenten intern entwickelt werden sollen, existiert jedoch kein Entscheidungsmodell, das technische Eigenschaften eines Softwaresystems berücksichtigt, sondern lediglich erste Ansätze in den Phasen der Anforderungserhebung und des Architekturentwurfs (vgl. Kapitel 2.4.3).

Entsprechend der vorgestellten Grundlagen eignet sich das in dieser Arbeit vorliegende Problem, also die Klassifikation von Komponenten im Softwareentwicklungsprozess basierend auf ihren technischen Eigenschaften für die Eigenentwicklung oder Fremdherstellung, bestens für den Entwurf eines Entscheidungsmodells für Softwarekomponenten und die Umsetzung eines zugehörigen EUS. Für die Implementierung der Entscheidungsunterstützung als innovatives EUS im IT-Outsourcing erscheinen mobile Technologien aufgrund ihrer kompakten Bauweise und ihrer guten Vernetzung als optimal geeignet (vgl. Kapitel 2.2.2), da mit mobilen Endgeräten die Zielgruppe der Entscheidungsträger hervorragend adressiert werden kann. Diese setzt sich aus Managern oder Entwicklungsleitern zusammen, die auf mobile Lösungen angewiesen sind (Klimpke et al., 2011).

3. Konzeption und Implementierung

In diesem Kapitel wird das zentrale Artefakt dieser konstruktionswissenschaftlichen Arbeit konzipiert und prototypisch umgesetzt. Es handelt sich dabei um eine Technologie zur Unterstützung von Outsourcingentscheidungen in der komponentenbasierten Softwareentwicklung, die einerseits aus einem theoretisch basierten Entscheidungsmodell und andererseits aus dessen Implementierung in ein mobiles EUS besteht. Diese Vorgehensweise entspricht der zugrundeliegenden Forschungsmethode dieser Arbeit (vgl. Kapitel 1.3) und initiiert gleichzeitig den *Design Cycle* dieser Methode. Dadurch wird, wie von Hevner et al. (2004) gefordert, die erste Richtlinie designorientierter Forschungsarbeiten umgesetzt:

> *„The result of design-science research in IS is, by definition, a purposeful IT artifact created to address an important organizational problem.“* (Hevner et al., 2004, S. 82)

Dementsprechend sollen das in diesem Kapitel konzipierte Entscheidungsmodell und dessen Umsetzung die Anforderungen von Unternehmen an ein nutzenstiftendes Artefakt erfüllen. Zunächst wird daher die zu unterstützende Outsourcingentscheidung genau spezifiziert und besonders auf die daraus resultierenden Anforderungen an das zu entwerfende Entscheidungsmodell sowie dessen prototypische Umsetzung eingegangen. Im weiteren Verlauf dieses Kapitels werden dann ein theoretisch fundiertes Entscheidungsmodell sowie eine zugehörige Entscheidungslogik hergeleitet, die eine Klassifizierung von Komponenten eines Softwareentwicklungsprojekts ermöglichen. Für jede einzelne Softwarekomponente lässt sich damit entscheiden, ob sie vom eigenen Softwareentwicklungsteam entwickelt oder ausgelagert werden soll. Das Entscheidungsmodell berücksichtigt dabei sowohl strukturelle Eigenschaften einer Komponente in Bezug auf das Softwareprodukt, Prozessmerkmale der Entwicklung von Komponenten als auch Wissensmerkmale von Softwarekomponenten. Die Logik verknüpft alle Entscheidungskriterien des Modells in einem zweistufigen gewichteten Verfahren. In den genannten Merkmalskategorien werden Teilnutzenwerte ermittelt, aus denen

sich unter Berücksichtigung der Gewichtung ein Gesamtnutzen ergibt. Mit einem Praxisbeispiel wird die Konzeptionierung des Entscheidungsmodells komplettiert. Es folgt eine prototypische Umsetzung des konzipierten Entscheidungsmodell in ein Softwarewerkzeug für Outsourcingmanager. Es wird das Design und die Implementierung eines mobilen EUS beschrieben, das sich nahtlos in bestehende Softwareentwicklungsumgebungen integrieren lässt, die Weiterverwendung existierender Kollaborationssysteme sicherstellt und die Durchführung der Outsourcingentscheidung für jede einzelne Komponente gleichermaßen unterstützt sowie begründet. Das Kapitel endet mit einer Zusammenfassung der Ergebnisse.

3.1. Die zu unterstützende Outsourcingentscheidung und ihre Anforderungen

Eine Entscheidung zu treffen bedeutet im betrieblichen Kontext, eine oder mehrere vorteilhafte Handlungsalternativen zur Lösung eines vorliegenden Problems auszuwählen, um den Unternehmenszielen schrittweise näher zu kommen (vgl. Kapitel 2.1.1). Der betriebliche Kontext, der in dieser Arbeit betrachtet wird, ist die Festlegung, welche Teile einer Softwarearchitektur für die interne und welche für die externe Entwicklung bestimmt sind. Im Fokus stehen daher Softwareunternehmen, die sich aus unternehmerischen Gründen dafür entschieden haben, Teile ihrer Softwareentwicklung im jeweiligen Projekt auszulagern.

Im Rahmen von Softwareentwicklungsprojekten eignen sich Softwarekomponenten (vgl. Kapitel 2.3.3) als Entscheidungsgegenstand einer Systemarchitektur optimal. Eine Komponente wird in dieser Arbeit holistisch betrachtet und pragmatisch als logisches Konzept aufgefasst (Cheesman und Daniels, 2001). Im Unterschied zu der überwiegend technischen Bedeutung des Begriffs der Komponente (Atkinson et al., 2002) mit ihren konkreten Abstraktionsebenen (Binär-, Bytecode- oder Codeebene) wird eine Komponente in dieser Arbeit als abgeschlossene und eigenständige logische Einheit von Software mit einer definierten Funktionalität verstanden. Sie ist unabhängig verteilbar und kommuniziert ausschließlich über wohldefinierte Schnittstellen mit anderen Komponenten. Sie wird als die elementare Einheit der Gesamtarchitektur betrachtet.

Aus der Sicht des Entwicklungsprozesses muss die Outsourcingentscheidung nach der konzeptionellen Designphase durchgeführt werden (vgl. Abb. 2.12). Erst dann lassen sich Aussagen über die Struktur der Software sowie deren Teilsysteme und Funktionseinheiten und damit über die Gesamtarchitektur treffen. Die zu unterstützende Entscheidung befindet sich also an der Schnittstelle zwischen Design und Implementierung, bevor die jeweiligen Softwarekomponenten in Code geschrieben werden.

In diesem beschriebenen Kontext ist es nun erforderlich, eine *konkrete Entscheidung über die Verteilung aller zum Projekt zugehörigen Softwarekomponenten* auf das *interne* Entwicklungsteam oder den *externen* Zulieferer zu treffen, da für dieses Vorgehen noch keine geeigneten Verfahren existieren (vgl. Kapitel 2.4.3). Deshalb wird im weiteren Verlauf dieses Kapitels ein Entscheidungsmodell konzipiert, welches die spezifische Outsourcingentscheidung umfänglich unterstützt und sich in einem mobilen EUS umsetzen lässt. Hierfür wird eine *Zielfunktion* definiert, deren Ergebnis jeder Softwarekomponente des Projekts eine Handlungsempfehlung (intern entwickeln, extern entwickeln, beides möglich) zuweist. Die Zielfunktion bedient sich dabei speziell auf Softwarekomponenten zugeschnittener *Entscheidungskriterien* (strukturspezifisch, prozessspezifisch, wissensspezifisch), um das *Entscheidungsfeld* hinsichtlich der Zielsetzung (optimale Aufgabenverteilung bei der Entwicklung) auszuwerten.

Die Anforderungen an das Entscheidungsmodell und die zugehörige Implementierung (siehe Tabelle 3.1) lassen sich aus bestehenden Theorien und Konzepten der Wirtschaftsinformatik bzw. Betriebswirtschaftslehre ableiten. Zusätzliche Anforderungen ergeben sich aus dem Stand der Technik, bisherigen Forschungsarbeiten im Bereich des IT-Outsourcings sowie der identifizierten Forschungslücke (vgl. Kapitel 2.4.3).

Als geeignet für das vorherrschende Entscheidungsproblem und dessen komponentenbasierte Sichtweise erweisen sich in erster Linie erforschte Konzepte, welche zum Treffen einer Entscheidung die für die Organisation spezifischen sowie technischen Eigenschaften einer einzelnen Komponente einbeziehen. Im Outsourcingkontext werden häufig die Transaktionskostentheorie, die ressourcenbasierte Sichtweise von Unternehmen sowie die Systemtheorie mit Entscheidungskriterien für die Auslagerung in Verbindung gebracht (Dibbern et al., 2012a, 2004). Ihre Verwendungsmöglichkeiten auf operativer Ebene motivieren die Definition von

Tabelle 3.1.: Zusammenfassung der Anforderungen

Nr.	Anforderung
1	Das Entscheidungsmodell zur Unterstützung von Outsourcingentscheidungen in der komponentenbasierten Softwareentwicklung muss sowohl strukturelle Eigenschaften einer Komponente in Bezug auf das Softwareprodukt (strukturspezifische Merkmale), Prozessmerkmale der Entwicklung von Softwarekomponenten (prozessspezifische Merkmale) als auch Wissensmerkmale von Softwarekomponenten (wissensspezifische Merkmale) berücksichtigen.
2	Die Selektion und Definition der Merkmale von Softwarekomponenten, die sich auf die Outsourcingentscheidung auswirken, müssen im Entscheidungsmodell und im EUS flexibel gestaltet werden. Merkmale müssen sich hinzufügen oder entfernen lassen. Die Gewichtungen der Aggregationsebenen der Merkmalsgruppen müssen individuell definierbar sein.
3	Das EUS für die komponentenbasierte Outsourcingentscheidung muss sich zur Verwendung entscheidungsrelevanter Daten in bestehende Kollaborationsplattformen für die Softwareentwicklung integrieren lassen. Eine universelle Schnittstelle für die Integration in unterschiedliche Plattformen ist dabei vorteilhaft.
4	Das EUS für die komponentenbasierte Outsourcingentscheidung muss in Form einer mobilen Softwarelösung umgesetzt werden und der relevanten Zielgruppe als Anwendung für das Smartphone oder Tablet verfügbar gemacht werden.

Anforderungen an das zu entwickelnde Entscheidungsmodell und dessen technische Umsetzung.

In der **Transaktionskostentheorie** werden alle Kosten von Transaktionen betrachtet, die für die Erreichung und Durchführung einer Vereinbarung aufgewendet werden müssen. Jede Transaktion erfordert angemessene Vereinbarungen zwischen zwei Parteien, um den Austausch von Produkten und Dienstleistungen zu ermöglichen (Williamson, 1990). Unter Transaktionskosten werden daher Anbahnungskosten (zur Suche von und Informationsbereitstellung über mögliche Austauschpartner), Verhandlungskosten (für die Zeit und Ressourcen zur Vertragsausgestaltung), Überwachungskosten (zur Überprüfung abgesprochener Preise oder Bedingungen) sowie Anpassungskosten (für die nachträgliche An-

gleichung von Preisen, Terminen, Qualitäten oder Mengen) verstanden. Solche Kosten treten kontinuierlich auf und sind aber nur schwierig zu messen (Dibbern et al., 2004; Williamson, 1990). Der Existenz von Transaktionskosten liegen zwei fundamentale Annahmen zugrunde. Einerseits sorgt die *begrenzte Rationalität* von Individuen dafür, dass die Zukunft nicht vollständig vorhergesehen werden kann und dass Verträge nicht vollständig entwickelt und abgeschlossen werden können, sondern diese immer nur Teilaspekte festhalten, die im Vorfeld absehbar sind. Andererseits führt *opportunistisches Verhalten* der involvierten Parteien dazu, dass sie sich auf Kosten des anderen Partners Vorteile verschaffen, sofern sich die Gelegenheit ergibt.

Unter Berücksichtigung dieser Prämissen besagt die Theorie, dass die Transaktionskosten jeweils von spezifischen Assets, also besonderen Fähigkeiten, Fertigkeiten und Expertisen, sowie von Unsicherheit beeinflusst werden. Dabei Die spezifischen Assets werden dabei in drei Kategorien untergliedert:

- Ortsspezifisch: Z.B. Natürliche Ressourcen sind nur am Ort ihres Vorkommens verfügbar und lassen sich nur unter hohen Kosten bewegen.
- Güterspezifisch: Z.B. Komplexe Systeme oder Geräte, die speziell für einen bestimmten Kunden oder Zweck gebaut wurden.
- Humanspezifisch: Z.B. Hochspezialisierte Arbeitskräfte, die ihre Fertigkeiten über Jahre erlernt haben.

Bei Unsicherheit unterscheidet man zwischen umgebungsbedingter und verhaltensbezogener Unsicherheit. Die Umgebungsunsicherheiten sind sich schnell ändernde Technologien oder geschäftsbedingte Unsicherheiten, wenn sich der Geschäftszweck des Verhandlungspartners während der Vertragslaufzeit ändert. Die verhaltensbezogene Unsicherheit beinhaltet die Wahrscheinlichkeit mit der sich ein Geschäftspartner opportunistisch verhält. Weiterhin erklärt die Transaktionskostentheorie, dass sich Unsicherheit bei hoher Spezifität der Assets stärker auf die Transaktionskosten auswirkt.

Bei Outsourcingentscheidungen setzen sich die Beschaffungskosten aus den Produktionskosten sowie den Transaktionskosten zusammen. Beide müssen stets zusammen betrachtet werden, damit die Teile der Software im Unternehmen selbst hergestellt werden, für die die Beschaffungskosten auf dem Markt höher sind als die eigenen, und umgekehrt (Dibbern et al., 2008). Dies gilt damit auch für

die komponentenweise Betrachtung der Outsourcingentscheidung. Es muss also stets berücksichtigt werden, welchen Einfluss spezifische Assets, wie z.B. besondere Programmiererfahrungen oder lang erlerntes Prozesswissen, in Kombination mit Unsicherheiten auf die jeweiligen Transaktionskosten haben. So sollten z.B. Softwarekomponenten, die dem Entwickler besondere Fähigkeiten abverlangen, die nur im eigenen Unternehmen verfügbar sind, in erster Linie nicht an einen Lieferanten vergeben werden.

Bei der **ressourcenbasierten Sichtweise** wird argumentiert, dass der Markt schneller in der Lage ist, sich über die Zeit hinweg weiterzuentwickeln und von der Vergangenheit zu lernen. Dadurch können auf dem Marktumfeld Fähigkeiten und Routinen unabhängig voneinander entwickelt und bereitgestellt werden, die auf den vorhandenen Ressourcen basieren (Langlois, 1995). Sofern ein Unternehmen nicht in der Lage ist, die notwendigen Ressourcen für die Erstellung eines Produkts oder eines Services selbst intern bereitzustellen, müssen die Ressourcen auf dem Markt die vorhandenen Fähigkeiten eines Unternehmens komplettieren und werden so begehrlich für das Unternehmen (Grant, 1991; Teng et al., 1995). Bei mangelnden Ressourcen ist deshalb eine Auslagerung unausweichlich. Hochspezifische Wettbewerbsvorteile, deren Fehlen sich existenzbedrohend auf ein Unternehmen auswirken, sind deshalb für das Outsourcing ungeeignet. Dementsprechend sind leicht zu imitierende Fertigkeiten und Ressourcen mit geringer Spezifität besser dafür geeignet (Dibbern et al., 2005, 2008; Stratman, 2008).

Die **Systemtheorie** wird zur Beschreibung und Strukturierung komplexer Systeme herangezogen und macht sich die Eigenschaften der Modularität zunutze (Picot und Baumann, 2007; Simon, 1962). Weil komplexe Systeme aus einer großen Anzahl einzelner interagierender Fragmente (elementare Einheiten eines Systems) bestehen, ist es bei der Erstellung von Softwaresystemen die Aufgabe der Entwickler das effektive Zusammenspiel aller eingebetteten Fragmente (Komponenten) sicherzustellen (Simon, 1962). Die bevorzugte Aufteilung eines komplexen Softwaresystems in seine Teilsysteme und die zugehörigen Komponenten wird dadurch erreicht, dass diese einen minimalen Grad an Abhängigkeit zueinander aufweisen aber eine starke interne Bindung besitzen und nur über wohldefinierte Schnittstellen interagieren (Baldwin und Clark, 2000, 2006; Endres und Rombach, 2003; Gagsch, 1971, 1980; Michlmayr und Hill, 2003). Aus diesem Grund sind strukturelle Merkmale von Softwarekomponenten ein wesentliches Entschei-

dungskriterium beim Outsourcing von Softwareprojekten. Zusammen mit den Erkenntnissen aus der ressourcenbasierten Sichtweise sowie der Transaktionskostentheorie ergibt sich dadurch folgende Anforderung:

> **Anforderung 1:** Das Entscheidungsmodell zur Unterstützung von Outsourcingentscheidungen in der komponentenbasierten Softwareentwicklung muss sowohl strukturelle Eigenschaften einer Komponente in Bezug auf das Softwareprodukt (strukturspezifische Merkmale), Prozessmerkmale der Entwicklung von Softwarekomponenten (prozessspezifische Merkmale) als auch Wissensmerkmale von Softwarekomponenten (wissensspezifische Merkmale) berücksichtigen.

Das zu entwickelnde Entscheidungsmodell muss neben der theoretisch fundierten Anforderung ebenfalls als Aussagensystem für die stets zuverlässige Bestimmung von vorteilhaften Handlungsmaßnahmen gültig sein (vgl. Kapitel 2.1.3). Die zahlreichen Studien und Forschungsarbeiten im Bereich des IT-Outsourcings zeigen, dass Outsourcingentscheidungen, unabhängig davon, auf welcher Ebene diese getroffen werden, unter Berücksichtigung sämtlicher zur Verfügung stehender Aspekte stets subjektiv betrachtet werden (vgl. Kapitel 2.4.3). Es wäre vermessen, den Anspruch auf Vollständigkeit der Entscheidungskriterien zu erheben, wenn zukünftige Forschungsarbeiten möglicherweise in diesem Bereich weitere Merkmale mit Einfluss auf die Entscheidung identifizieren. Die Vollständigkeit eines Entscheidungsmodells hängt nach Laux et al. (2004) von den den Kriterien ab, die vom Entscheider als bedeutsam für eine originalgetreue Abbildung des Problems wahrgenommen werden. Allerdings ist diese Vollständigkeit subjektiv und kann von Entscheider zu Entscheider variieren (Laux et al., 2004). Um in diesen Fällen eine unvollständig informierte Entscheidung zu vermeiden, wird im Rahmen dieser Arbeit eine initiale Liste mit zugehörigen Kategorien angeboten wie bei Kramer et al. (2013), die sich aber individuell um weitere Entscheidungskriterien oder auch um zusätzliche Kriterienkategorien erweitern und abändern lassen muss. So wird der Forderung nach Vollständigkeit des Entscheidungsmodells Genüge geleistet. Daher wird die folgende Anforderung notwendig:

> **Anforderung 2:** Die Selektion und Definition der Merkmale von Softwarekomponenten, die sich auf die Outsourcingentscheidung auswirken, müssen im Entscheidungsmodell und im EUS flexibel gestaltet

werden. Merkmale müssen sich hinzufügen oder entfernen lassen. Die Gewichtungen der Aggregationsebenen der Merkmalsgruppen müssen individuell definierbar sein.

Die besondere Eigenschaft von EUS im Kontext von Softwareoutsourcing ist in erster Linie deren Vernetzung aller entscheidungsrelevanten Personen und Informationen über Systemgrenzen hinweg (vgl. Kapitel 2.2). Sie ermöglichen es, alle systemseitig erfassten Informationen für eine Entscheidung zusammenzutragen und bei Bedarf die Einschätzungen der einzelnen Entscheider zu erheben, zu aggregieren und zusammenzufassen. Im vorliegenden Fall der zu unterstützenden Outsourcingentscheidung ist es daher besonders wichtig, dass die systemseitig verfügbaren Informationen über die Architektur einer Softwarelösung verwendbar gemacht werden und für die Entscheidungsfindung herangezogen werden können. Diese Daten werden bei Softwareunternehmen mit verteilter Entwicklung üblicherweise in Kollaborationsplattformen (vgl. Kapitel 2.3.4) gepflegt und lassen sich darüber auch abrufen. Es ist daher für die Implementierung des prototypischen EUS notwendig, dass sie sich in die existierenden Plattformen einfügen lässt, um daraus die strukturiert vorliegenden Daten für die Entscheidungsfindung extrahieren zu können. Da sich bei den Kollaborationsplattformen für Softwareprojekte bisher kein einheitlicher Standard entwickelt hat, bietet sich eine universale Schnittstelle für den Datenaustausch mit diesen Plattformen an.

Anforderung 3: Das EUS für die komponentenbasierte Outsourcingentscheidung muss sich zur Verwendung entscheidungsrelevanter Daten in bestehende Kollaborationsplattformen für die Softwareentwicklung integrieren lassen. Eine universelle Schnittstelle für die Integration in unterschiedliche Plattformen ist dabei vorteilhaft.

Fortschrittliche EUS zeichnen sich besonders durch ihre mobilen Charaktereigenschaften aus, welche durch die ununterbrochene Konnektivität von Smartphones und Tablets die Nutzung dieser Systeme nahezu an jedem Ort möglich machen und die bei Bedarf kontextabhängig reagieren können (vgl. Kapitel 2.2.2). Diese Eigenschaften mobiler Technologien lassen sich hervorragend verwenden, um den Nutzerkreis des in dieser Arbeit entwickelten EUS zu adressieren und diese auch zu motivieren, den Prototyp zu verwenden. Die Zielgruppe der entwickelten Applikation ist das mittlere Management mit Projektleitungsaufgaben und

Entscheidungsbefugnissen, aber auch das Topmanagement von Softwareunternehmen. Dieser Personenkreis ist auf die Nutzung von mobilen Lösungen angewiesen, da der Arbeitsalltag durch kontinuierliche Treffen und durch Abstimmungen mit Mitarbeitern, Kollegen und anderen Managern geprägt ist. Entscheidungen müssen daher oft unterwegs oder im Meeting getroffen werden. Daher ist folgende Anforderung besonders wichtig:

> **Anforderung 4:** Das EUS für die komponentenbasierte Outsourcingentscheidung muss in Form einer mobilen Softwarelösung umgesetzt werden und der relevanten Zielgruppe als Anwendung für das Smartphone oder Tablet verfügbar gemacht werden.

Die Umsetzung der erhobenen Anforderungen in ein entsprechendes Entscheidungsmodell sowie die Entwicklung des zugehörigen EUS werden in den anschließenden Unterkapiteln detailliert erläutert.

3.2. Konzeption des Entscheidungsmodells

Entsprechend der ersten Anforderung (siehe Tabelle 3.1) wird ein Entscheidungsmodell benötigt, welches die Entscheidungsfindung beim IT-Outsourcing auf der Basis von Komponenten ermöglicht. Das Modell dient anschließend als Grundlage für die Implementierung eines mobilen EUS, um in Form einer prototypischen Umsetzung die Nützlichkeit und Anwendbarkeit einer solchen Entscheidungsunterstützung demonstrieren zu können.

Für die Konzeption des Entscheidungsmodells werden neben der Entscheidungslogik auch Zieldimensionen und Entscheidungsgrößen benötigt, die eine Klassifizierung jeder Softwarekomponente letztendlich erst ermöglichen (vgl. Kapitel 2.1.3). Zunächst wird daher der Aufbau des Modells schrittweise erläutert, indem die Zieldimensionen vorgestellt und die Entscheidungsgrößen einzeln aus der Literatur hergeleitet werden. Im Anschluss werden die logischen Verknüpfungen zwischen den Entscheidungsgrößen vorgestellt, die die Komponenteneigenschaften mit den Zieldimensionen vereinen und als Aggregationsmethode der drei Merkmalsklassen für die finale Outsourcingentscheidung dienen. Ein praktisches Beispiel rundet die Konzeption ab.

3.2.1. Entscheidungsgrößen und Zieldimension

Das in dieser Arbeit entworfene Entscheidungsmodell greift die Kategorien aus der ersten Anforderung in der Konzeption auf und gliedert die *Entscheidungsgrößen* in strukturelle Eigenschaften, Prozessmerkmale der Entwicklung und Wissensmerkmale einer Softwarekomponente. **Strukturelle Eigenschaften** eines Softwareprodukts beschreiben den Aufbau eines komplexen Softwaresystems und dessen Aufteilung in überschaubare Komponenten mit handhabbarer Komplexität und weitgehend reduzierten Interaktionskanälen zwischen den Komponenten, die von klar definierten Schnittstellen und vorgegebenen Designregeln geprägt sind (Baldwin und Clark, 2000; Subramanyam et al., 2012). Mit den **prozessspezifischen Merkmalen** einer Softwarekomponente werden die Eigenschaften einer Komponente bezeichnet, die für die Entwicklung komplexer Softwaresysteme im Rahmen des Softwareentwicklungsprozesses relevant sind und die ineinandergreifende Abfolge von Aktivitäten betreffen (Heinrich et al., 2011; Sommerville, 2007). Mit den **wissensspezifischen Merkmalen** einer Softwarekomponente werden Eigenschaften einer Komponente bezeichnet, die spezifisches Wissen, Kenntnisse, Fähigkeiten oder Fertigkeiten erfordern, um das betreffende Teilsystem umsetzen zu können (Heinrich et al., 2011).

Die *Zieldimension* im Entscheidungsmodell dieser Arbeit umfasst die konkrete Aussage über das Outsourcingpotenzial jeder einzelnen Softwarekomponente. Diese Dimension kann lediglich drei mögliche Ausprägungen besitzen, die gleichzeitig Bestandteil des Entscheidungsmodells sind. Ihre Ausprägungen sind definiert als:

- *HOCH* – die betrachtete Komponente sollte ausgelagert werden,
- *MITTEL* – die betrachtete Komponente kann sowohl ausgelagert werden als auch selbst hergestellt werden,
- *NIEDRIG* – die betrachtete Komponente sollte selbst entwickelt werden.

Aus struktureller Sicht gliedert sich das Entscheidungsmodell nun in zwei Ebenen. Auf der ersten Stufe werden die drei Spezifitäten von Softwarekomponenten mit ihren jeweiligen Eigenschaften gleichermaßen repräsentiert und nehmen jeweils eigenständig Einfluss auf die zweite Ebene mit der Zieldimension des Outsourcingpotenzials. Dadurch ergibt sich ein Entscheidungsmodell zur Beantwor-

tung der Outsourcingfrage auf operativer Ebene wie in Abbildung 3.1 dargestellt. Die einzelnen Entscheidungsgrößen werden nachfolgend genauer spezifiziert.

3.2.1.1. Strukturelle Eigenschaften

Als erste der drei Entscheidungsgrößen hat die Klasse der Strukturmerkmale von Komponenten in Bezug auf das gesamte Softwareprodukt einen direkten Einfluss auf die Entscheidungsdimension. Hierbei werden Charakteristiken betrachtet, die den strukturellen Aufbau der Komponente selbst sowie hinsichtlich ihrer Einflechtung ins Gesamtsystem betrachtet. Das abstrakte Modell der Softwarekomponente eignet sich unter den Gesichtspunkten des *RUP* optimal als Untersuchungsgegenstand dieses Modells.

Die Merkmale dieser Kategorie lassen sich primär aus der Systemtheorie (vgl. Kapitel 3.1) herleiten. Diese empfiehlt einen modularen Aufbau bei der Entwicklung komplexer Systeme (Simon, 1962). Um diesen Aufbau zu erreichen, muss eine Gesamtarchitektur für das System entworfen werden, welche einzelne Funktionalitäten zum größten Teil in einer einzelnen Komponente abbildet und gleich-

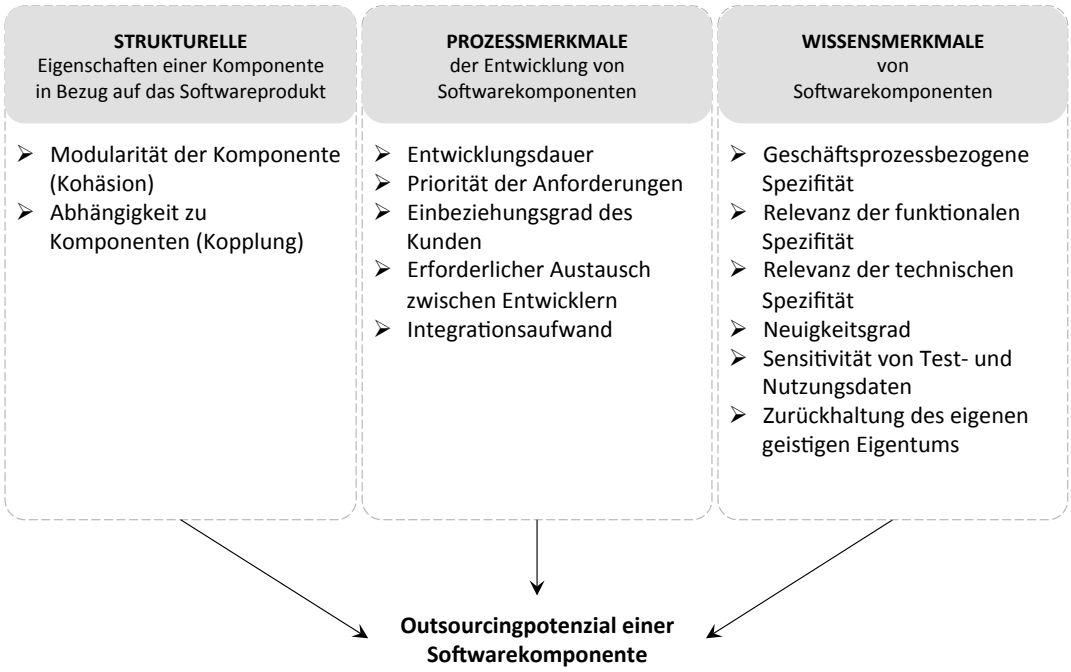

Abbildung 3.1.: Entscheidungsmodell für Softwarekomponenten

Tabelle 3.2.: Zusammenfassung der strukturspezifischen Entscheidungskriterien einer Softwarekomponente

Eigenschaft	**Kurzbeschreibung**	**Kennzeichen**
Modularität (Kohäsion)	Eigenständigkeit und innere Abgeschlossenheit der betrachteten Komponente	E_1
Kopplung	Grad des Zusammenhangs der betrachteten Komponente mit anderen Komponenten des Softwaresystems (über Anforderungen definiert)	E_2

zeitig weitgehend unabhängig von anderen Komponenten ist. Die Konzepte der Kohäsion und Kopplung finden hierbei Verwendung, um die Modularität und die Abhängigkeit einer Komponente zu anderen zu beschreiben (Baldwin und Clark, 2000). Deshalb werden im Entscheidungsmodell dieser Arbeit die beiden Strukturmerkmale

- Modularität der Komponente (Kohäsion) und
- Abhängigkeit zu Komponenten (Kopplung)

zur Einschätzung der Verwendung der betrachteten Komponente im Gesamtsystem verwendet. Eine Übersicht, zusammen mit einer Kurzbeschreibung und der Zuweisung eines Kennzeichens für die spätere Beschreibung der Entscheidungslogik, wird in Tabelle 3.2 gegeben.

Die *Modularität (Kohäsion)* einer zu bewertenden Komponente beschreibt in diesem Modell den Grad der Eigenständigkeit und inneren Abgeschlossenheit der betrachteten Softwarekomponente. Eine Komponente ist dann modular, wenn sie geforderte Funktionalitäten weitgehend selbst ausführen kann, weil alle benötigten Funktionen in der Komponente verfügbar sind. Dadurch sind die Funktionsaufrufe innerhalb einer Komponente meist zahlreich. Funktionsaufrufe zu externen Komponenten, die zur Abarbeitung der Funktionalität erforderlich sind, sind bei stark modularer Bauweise nur sehr gering (Baldwin und Clark, 2000; MacCormack et al., 2006; Picot und Baumann, 2007). Bei hoher Modularität eignet sich eine Softwarekomponente aus technischer Sicht gut für die Auslagerung zu einem Lieferanten, weil dadurch abgeschlossen definierte Einheiten übergeben werden können, wodurch die Anzahl an Rückfragen gering gehalten werden können und

direkte Abstimmungen obsolet sind. Gleichzeitig werden auch die Transaktionskosten gering gehalten (Dibbern et al., 2008; Williamson, 1990).

Die *Kopplung* ist ein Maß für die Verwobenheit von Softwarekomponenten. Sie spezifiziert den Verknüpfungsumfang von Komponenten, sowie das Wissen über und die Abhängigkeit von anderen Komponenten (Baldwin und Clark, 2006; Endres und Rombach, 2003; Larman, 2002; Michlmayr und Hill, 2003). Der Grad der Zusammengehörigkeit von Komponenten lässt sich über die Anforderungen eines Softwareprojekts ermitteln, indem die untereinander definierten Beziehungen zwischen sämtlichen Projektanforderungen ausgewertet werden (vgl. Ansätze für das Outsourcing bei der Anforderungserhebung im Softwareentwicklungsprozess in Kapitel 2.4.3.2). Wenn eine Komponente stark mit anderen Komponenten gekoppelt ist, muss ihr eine tragenden Rolle im Gesamtsystem zugesprochen werden, da sie entweder als Kommunikationsschnittstelle zwischen allen Komponenten des Systems fungiert oder die Komplexität eines kommunikationsintensiven Geschäftsprozesses abdeckt und damit unverzichtbar ist. Die Auslagerung einer solchen Komponente hätte bei Lieferverzug oder schlechter Qualität der Umsetzung gravierende Konsequenzen auf das Gesamtsystem, indem dieses als Ganzes nicht funktionsfähig wäre. Komponenten mit starker Kopplung sollten daher selbst entwickelt werden.

Da sich bereits wissenschaftliche Arbeiten mit der Analyse und Bewertung der Zusammenhänge von Softwareanforderungen untereinander beschäftigt haben (Kramer und Eschweiler, 2013; Kramer et al., 2014), lassen sich Aussagen über die Kopplung und Kohäsion von Softwarekomponenten automatisiert berechnen. Dies ist möglich, weil Information über Anforderungen von Komponenten bereits in Entwicklungsumgebungen erfasst und ermittelbar gemacht werden. Die Berechnungsmöglichkeiten der beiden Strukturmerkmale wird bei der Implementierung des Entscheidungsmodells in Kapitel 3.3.2 detailliert vorgestellt. Das Zusammenspiel zwischen Kopplung und Kohäsion sowie deren Ausprägungen und die Konsequenzen für die Komponenten einer Softwarearchitektur sind in Abbildung 3.2 abgebildet. Bei starker Kohäsion und schwacher Kopplung lässt sich gut erkennen, dass nur wenige Verbindungen zwischen den Komponenten existieren. Dieser anzustrebende Zustand erleichtert das Outsourcing von Komponenten.

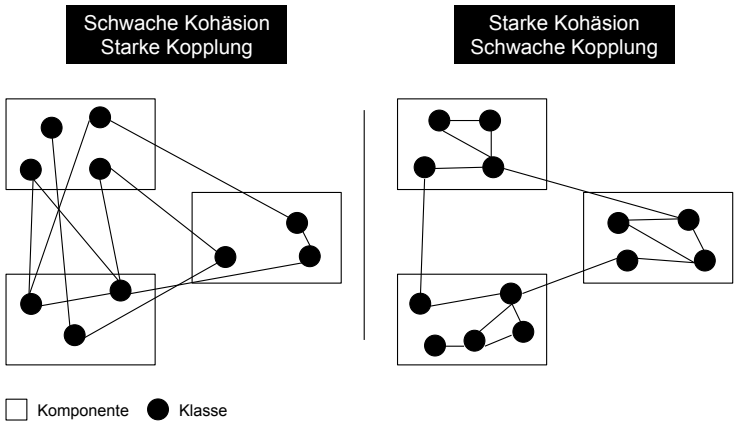

Abbildung 3.2.: Kopplung und Köhasion von Softwarekomponenten

3.2.1.2. Prozessspezifische Merkmale der Entwicklung

Als weitere Entscheidungsgröße werden im Entscheidungsmodell Charakteristiken verwendet, die prozessspezifische Merkmale des Entwicklungsprozesses von Softwarekomponenten beinhalten. Damit werden alle Aktivitäten und Abläufe bezeichnet, die für die Herstellung eines komplexen Softwaresystems erforderlich sind (vgl. Kapitel 2.3). Basierend auf der Transaktionskostentheorie gelten einige Merkmale dieses Prozesses beim IT-Outsourcing als kritisch für die Entscheidungsfindung (vgl. Kapitel 3.1). Besonders diejenigen, die bei der Übergabe von Softwarekomponenten an einen Zulieferer und bei der anschließenden Integration ins Gesamtsystem hohe Kosten im Sinne von Transferleistungen (direkte Kommunikation, persönliche Treffen, Absprachen bei Schnittstellen, zeitliche Verzögerungen usw.) verursachen, fließen als weitere Dimension in das Outsourcingpotenzial mit ein. Eine Übersicht aller prozessspezifischen Eigenschaften von Softwarekomponenten inklusive Kurzbeschreibung und Kennzeichnung wird in Tabelle 3.3 gegeben.

Die *Entwicklungsdauer* einer Komponente nimmt im Entstehungsprozess eines Gesamtsystems eine wichtige Rolle bei der Bewertung der Auslagerungsmöglichkeit ein. Entscheidet man sich dafür, eine Softwarekomponente auszulagern, die eine lange Entwicklungsdauer in Relation zum Gesamtprojekt aufweist, so riskiert man, den kritischen Pfad zu verletzen, der aus der Netzplantechnik bekannt ist. Diese Technik basiert auf der Graphentheorie. Ihr kritischer Pfad gibt an, welche Aktivitäten zum zeitlichen Verzug bis zur Fertigstellung führen, sofern eine dieser

Tabelle 3.3.: Zusammenfassung der prozessspezifischen Entscheidungskriterien einer Sofzwarekomponente

Eigenschaft	**Kurzbeschreibung**	**Kennzeichen**
Entwicklungsdauer	Zeitlicher Entwicklungsaufwand einer Komponente in Relation zum Gesamtsystem	E_3
Priorität der Anforderungen	Prioritätseinstufung der Anforderungen, die mit der betrachteten Komponente verknüpft sind	E_4
Einbeziehungsgrad des Kunden	Kommunikationsintensität für die Abstimmung mit dem Auftraggeber der Software und Wahrscheinlichkeit der Rückfragen	E_5
Austausch zwischen Entwicklern	Erwarteter Abstimmungsaufwand der Entwickler der betrachteten Komponente mit Entwicklern anderer Komponenten des Softwaresystems (Limitation: Gesetz von Conway)	E_6
Integrationsaufwand	Aufwand zur Einbindung der Komponente nach ihrer Fertigstellung in das Gesamtsystem	E_7

Aktivitäten ungeplant verlängert wird und sich damit nachfolgende Aktivitäten verzögern, da sie auf dem verspäteten Vorgänger aufbauen (Heinrich et al., 2004; Homburg, 2000). Ausgelagerte Komponenten mit langer Entwicklungsdauer sind jedoch mit hoher Wahrscheinlichkeit auf dem kritischen Pfad angesiedelt, weil sie vor Auslieferung an den Endkunden in das bestehende System integriert werden müssen (vgl. Kapitel 2.3.2). Weiterhin deutet eine lange Entwicklungsdauer auf die Größe bzw. Komplexität von Systemen hin. Sie sind deshalb mit hohen Entwicklungsrisiken verknüpft (McFarlan, 1981). Es gilt also, Komponenten mit langer Entwicklungsdauer in Relation zum Gesamtprojekt selbst herzustellen.

Einen weiteren entscheidungsrelevanten Aspekt für das Outsourcing stellen die *Prioritäten der Anforderungen* dar, die mit der jeweiligen Komponente verknüpft sind (vgl. Kapitel 2.3.2). Hoch priorisierte Anforderungen bringen die Wichtigkeit der Umsetzung einer Funktionalität für einzelne oder alle Interessengruppen

zum Ausdruck (Karlsson et al., 1998; Saaty und Vargas, 2001). Neben der unbedingt benötigten Funktionalität aus systemischer Sicht des Softwarearchitekten kann eine solch hoch priorisierte Anforderung auch die Funktionspräferenz des Endkunden ausdrücken. Eine Fremdvergabe von Komponenten mit derart favorisierten Anforderungen ist wegen hohen Transaktionskosten zur technischen Abstimmung, als auch wegen Aspekten der Endkundenzufriedenheit zu vermeiden. Letzteres erfordert eine enge Abstimmung mit dem Endkunden und leitet direkt zum nächsten Merkmal über.

Der *Einbeziehungsgrad des Kunden* drückt die Kommunikationsintensität für die Abstimmung mit dem Auftraggeber der Software und die Wahrscheinlichkeit von Rückfragen aus. Abhängig von der Komplexität des Umfeldes, in das die Softwarelösung eingebettet werden soll, oder vom Detaillierungsgrad der Spezifizierung von Anforderungen ist es erforderlich, den Kunden während der Implementierungsphase zu kontaktieren bzw. mit ihm in regem Austausch zu stehen. Um eine größtmögliche Kundenzufriedenheit zu erlangen, sollten Softwarekomponenten, die eine derart hohe Einbeziehung des Endkunden erfordern, aus organisatorischen und sprachlichen Gründen nicht ausgelagert werden (vgl. Kapitel 2.4.1.2).

Ein zusätzlich wichtiger Aspekt der prozessspezifischen Entscheidungsmerkmale stellt der *erforderliche Austausch zwischen Entwicklern* der zu evaluierenden Komponente mit Entwicklern anderer Komponenten des Softwaresystems dar. Wenn eine Komponente viele Schnittstellen zu anderen Komponenten aufweist und daher eine hohe Kopplung besitzt, ist ein hoher Kommunikationsaufwand oftmals unausweichlich (vgl. Kapitel 3.2.1.1). In diesem Fall kommen die besonderen Herausforderungen der verteilten Entwicklung zum Tragen und erschweren die Zusammenarbeit aufgrund sprachlicher und zeitlicher Herausforderungen (vgl. Kapitel 2.4.1.2). Durch diese Barrieren wird die Koordinationskomplexität unweigerlich gesteigert (Carmel und Tjia, 2006; Heeks et al., 2001). Zusätzlich ist der Entwicklungsprozess ein teamgetriebener und kommunikationsintensiver Prozess, der persönliche Abstimmungen der Entwickler notwendig macht, wenn es um Implementierungen von Klassen und Funktionen geht, die möglicherweise mehrmals im Gesamtsystem verwendet werden (Layman et al., 2006). Wenn Softwarekomponenten also von derartigen Softwareklassen geprägt sind, dann nehmen diese eine entscheidende Rolle für die Robustheit des Gesamtsystems ein. Sie sollten

daher nicht fremd vergeben werden. Eine Limitation für dieses Merkmal stellt das Gesetz von Conway dar, welches den Zusammenhang zwischen den Kommunikationsstrukturen einer Organisation und dem Entwurf von Systemen durch diese Unternehmen beschreibt (Conway, 1968). Bestehende Kommunikationsstrukturen sind daher richtungsweisend für den Entwurf des Softwaremodells und damit auch dessen Architektur. Im besten Fall werden die benötigten Softwarekomponenten bereits so definiert, dass ein erhöhter Kommunikationsaufwand bereits von Beginn an vermieden wird.

Abschließend fließt der *Integrationsaufwand*, den eine Komponente aufweist, als Entscheidungsmerkmal in das Modell mit ein. Im Softwarelebenszyklus gelten Integrationstests als eigenständige Phase vor der Auslieferung der Software in den regulären Betrieb (vgl. Kapitel 2.3.2). Dafür müssen zuvor alle Komponenten in das Gesamtsystem integriert werden. Für Komponenten mit zahlreichen Schnittstellen, also hoher Kopplung, ist dieser Aufwand bedeutend höher als für klar abgegrenzte Module mit wenigen Verbindungen zum Gesamtsystem (siehe Abbildung 3.2). Ist somit absehbar, dass der Integrationsaufwand einer Komponente von einer Vielzahl an Verbindungen zum Gesamtsystem geprägt ist, so sollte diese Komponente von den eigenen Entwicklern umgesetzt werden, weil dabei eine kontinuierliche Integration in das Gesamtsystem während der Entwicklung gewährleistet werden kann.

3.2.1.3. Wissensspezifische Merkmale

Den dritten Teil des Entscheidungsmodells stellen die wissensspezifischen Eigenschaften von Softwarekomponenten dar. Sie werden im Entscheidungsmodell für die Bewertung verwendet, in welchem Ausmaß spezifisches Wissen, besondere Fähigkeiten und Fertigkeiten oder auch spezielle Kenntnisse erforderlich sind, um eine Softwarekomponente implementieren zu können. Diese Charakteristiken lassen sich auf die ressourcenbasierte Sichtweise, also die Ressourcenorientierung von Unternehmen, sowie die Transaktionskostentheorie zurückführen (vgl. Kapitel 3.1). Das Konzept dieser Arbeit umfasst insgesamt fünf wissensspezifische Merkmale, die in Tabelle 3.4 zusammen mit einer Kurzbeschreibung und ihrem Kennzeichen aufgelistet sind.

Tabelle 3.4.: Zusammenfassung der wissensspezifischen Entscheidungskriterien einer Sofzwarekomponente

Eigenschaft	**Kurzbeschreibung**	**Kennzeichen**
Ausmaß der geschäftsprozessbezogenen Spezifität	Das Ausmaß in dem eine Softwarekomponente einen hochspezifischen Geschäftsprozess abdeckt	E_8
Ausmaß der funktionalen Spezifität	Das Ausmaß in dem eine Softwarekomponente eine hochspezifische Geschäftsfunktion in der zugehörigen Industrie abdeckt	E_9
Ausmaß der technischen Spezifität	Der Grad an Einbindung hochspezifischer Technologien einer Softwarekomponente	E_{10}
Neuigkeitsgrad	Grad der Neuigkeit der zu entwickelnden Komponente und der dafür erforderliche Erfindergeist von Entwicklern aus Sicht des Entscheiders	E_{11}
Sensitivität von Test- und Nutzungsdaten	Vertraulichkeit von Test- und Nutzungsdaten für die Entwicklung der Komponente aus juristischer oder wettbewerbsrelevanter Sicht	E_{12}

Bevor auf die einzelnen Aspekte der wissensspezifischen Merkmale näher eingegangen wird, muss zunächst geklärt werden, was in dieser Arbeit genau darunter verstanden wird. Der Begriff der *Spezifität* wird aus der Verwendung der *Faktorspezifität* durch Williamson (1985) in der Transaktionskostentheorie abgeleitet. Anhand der Faktorspezifität, wie diese bei der Transaktionskostentheorie verwendet wird, kann Spezifität als eine Besonderheit verstanden werden, die auf spezifischen Merkmalen beruht. Beispielweise zählen dazu die Spezifitäten physischer Güter oder Fertigkeiten (z.B. produktspezifische Fertigungsanlagen oder unternehmensspezifische Qualifikationen), die einen Spezialisierungsvorteil und damit einen Kostenvorteil gegenüber der Konkurrenz ermöglichen. Als hochspezifisch werden in Anlehnung daran Güter oder Fertigkeiten verstanden, die sich anderweitig nur mit sehr großem Aufwand wiederverwenden lassen (angelehnt an Williamson, 1985).

Das Ausmaß der *geschäftsprozessbezogenen Spezifität* ist in diesem Zusammenhang das Ausmaß in dem eine Softwarekomponente einen hochspezifischen Geschäftsprozess im Gegensatz zu nicht-spezifischen (z.B. standardisierten) Geschäftsprozessen abdeckt und damit alternativ kaum verwendbar ist (mögliche Ausprägungen sind dabei: zeitlich, organisatorisch, räumlich). Wenig strukturierte Geschäftsprozesse (*low structure*) ohne klar definierte Ergebnisse repräsentieren diese Klasse am besten (angelehnt an Applegate et al., 2003; Nelson und Winter, 1982; Subramani und Venkatraman, 2003; Winkler et al., 2009).

Das Ausmaß der *funktionalen Spezifität* ist das Ausmaß, in dem eine Softwarekomponente eine hochspezifische Geschäftsfunktion im Gegensatz zu nichtspezifischen (z.B. standardisierten) Geschäftsfunktionen in der zugehörigen Industrie abdeckt und damit alternativ kaum verwendbar ist. Die Geschäftsfunktion einer Softwarekomponente bezeichnet dabei die in die zugehörige Geschäftsdomäne eingebettete Aufgabe, die durch die betrachtete Softwarekomponente abgebildet wird (angelehnt an Heinrich et al., 2011; Winkler et al., 2009).

Das Ausmaß der *technischen Spezifität* ist der Grad an Einbindung hochspezifischer Technologien im Gegensatz zu nicht-spezifischen Technologien einer Softwarekomponente. Derartige proprietäre Technologien lassen sich folglich kaum anderweitig einsetzen. Dies tritt primär bei hochtechnologisierten Projekten auf, bei denen eine starke Technologieführerschaft sowie starke interne Projektintegration erforderlich ist (angelehnt an Applegate et al., 2003; Winkler et al., 2009). In diese Klasse fallen Softwarekomponenten von proprietären Technologien, z.B. die Kernsysteme von Banken oder Fluggesellschaften (Mata et al., 1995).

Sofern die zu bewertende Komponente hochspezifische Eigenschaften der im Vorfeld genannten Ausprägungen besitzt, ist es ratsam, diese kaum zu imitierenden Fähigkeiten und Fertigkeiten, die einen entscheidenden Wettbewerbsvorteil darstellen, so zu deuten, dass die Erstellung einer solchen Softwarekomponente zum Kerngeschäft des Unternehmens zählt und daher auf keinen Fall extern bezogen werden sollte.

Der *Neuigkeitsgrad* bezeichnet den Grad der Neuigkeit der zu entwickelnden Komponente sowie den dafür erforderlichen Grad an Komplexitätsbewältigung (Erfindergeist) von Entwicklern aus Sicht des Entscheiders (angelehnt an Reine-

cke und Tomczak, 2010). Auch dadurch werden schwer zu imitierende Vorteile ausgenutzt, um sich gegenüber dem Markt zu behaupten (vgl. Kapitel 3.1).

Die *Sensitivität von Test- und Nutzungsdaten* umfasst die internen und externen regulatorischen Anforderungen an die Datensicherheit sowie den Datenschutz. Im Rahmen dieses Entscheidungsmodell wird damit die Vertraulichkeit von Test- und Nutzungsdaten für die Entwicklung der zu bewertenden Komponente aus juristischer oder wettbewerbsrelevanter Sicht bezeichnet. Sofern also für die Erstellung einer Softwarekomponente oder für den Test ihrer Funktionsweise sensible Daten erforderlich sind, sollte vom Outsourcing dieser Komponente abgesehen werden.

Das nun vorliegende Entscheidungsmodell ist jetzt vollständig spezifiziert und zeigt alle hergeleiteten Entscheidungsgrößen für das Outsourcingpotenzial einer Softwarekomponente auf. Es muss im nächsten Schritt mit einer passenden Entscheidungslogik vervollständigt werden, um schließlich die betrachtete Komponente einer Ausprägung der Zieldimension zuordnen zu können. Dieses Regelwerk wird im nächsten Kapitel vorgestellt.

3.2.2. Konzeptionelle Überlegungen zur Teilautomation der Entscheidungsfindung bei einer anforderungszentrierten Vorgehensweise

Im nachfolgenden Teilabschnitt zum konzeptionellen Teil des Entscheidungsmodells wird auf ein Verfahren näher eingegangen, das eine teilautomatisierte Berechnungsweise des Entscheidungsmodells erlaubt. So können dem Entscheider bei der Festlegung aller Merkmalsausprägungen einer Komponente im Entscheidungsmodell (vgl. Kap. 3.2.4) bereits automatisiert Vorschläge zur Klassifizierung unterbreitet werden. Besonders im Bereich der strukturellen Eigenschaften einer Softwarekomponente in Bezug auf das Softwareprodukt lassen sich im vorliegenden Entscheidungsmodell automatisierte Klassifizierungen berechnen, da bei einer anforderungsbasierten Vorgehensweise mit existierenden Techniken (vgl. Kap. 3.2.2.3) bereits Rückschlüsse auf die Köhasion und Kopplung einer Softwarekomponente getroffen werden können. Diese Vorgehensweise ermöglicht es auch, dass zu diesem Zeitpunkt noch keine Softwarearchitektur vorliegen muss, um die beiden Entscheidungskriterien des Modells systematisch zu berechnen. Dafür

werden im weiteren Verlauf dieses Kapitels die zugrundeliegenden Konzepte der Anforderungserhebung und der Graphentheorie vorgestellt sowie die Verknüpfung der beiden Bereiche hin zu einer Bewertungstechnik für die Kohäsion und die Kopplung von Softwarekomponenten erläutert.

3.2.2.1. Relevante Grundlagen der Anforderungserhebung

Bei der Anforderungserhebung wird zwischen funktionalen sowie nicht-funktionalen Anforderungen unterschieden. Funktionale Anforderungen umfassen Aussagen zu Diensten, die das System ausführen kann. Weiterhin werden damit Reaktionen des Systems auf bestimmte Ereignisse oder genau vorgegebene Verhaltensweisen in besonderen Situationen umschrieben. Die nicht-funktionalen Anforderungen hingegen beinhalten Bedingungen des Systems hinsichtlich des zeitlichen Ablaufs, des darunterliegenden Prozesses oder bestimmter Standards. Sie beziehen sich meistens auf das Gesamtsystem und nicht auf einzelnen Funktionen oder Dienste. Weil die nicht-funktionalen Anforderungen nichts mit der semantischen Funktionalität des Systems sondern vielmehr die Rahmenbedingungen beschreiben, sind sie für die weitere Betrachtung in diesem Kontext obsolet. Die funktionalen Anforderungen hingegeben definieren den Lösungsraum, der mit dem angestrebten Softwareprodukt erschlossen werden soll. Die zugehörigen semantischen Verknüpfungen werden durch Verbindungen zwischen den funktionalen Anforderungen zum Ausdruck gebracht und erweitern damit den Einblick in die Problemdomäne (Moreira und Araujo, 2011; Rupp, 2009). Verknüpfungen zwischen funktionalen Anforderungen drücken meist Funktionalitäten aus, die für das gesamte System notwendig sind, wie z.B. Persistenz, Kollaboration oder Synchronisation (Kramer und Eschweiler, 2013).

Weil sich dieser Automatisierungsansatz ausschließlich auf Softwareanforderungen fokussiert, ist es notwendig, das System semantisch so zu beschreiben, dass eine Gruppierung der Anforderungen in kohärente Gruppen und darauf aufbauend der Entwurf der Softwarearchitektur ermöglicht wird. Zu diesem Zweck soll ein Modell mit sieben fundamentalen Abhängigkeitstypen zwischen Softwareanforderung herangezogen werden (Dahlstedt und Persson, 2005), von denen zwei als relevant für diese Arbeit erachtet werden. Der erste Abhängigkeitstypen wird mit *similar_to* (ähnlich zu) bezeichnet und bedeutet semantische Übereinstim-

mung mit unterschiedlich starker Ausprägung zwischen zwei Anforderungen. Dies ist bei der Strukturierung von Anforderungen ein Indikator für die Kohäsion zwischen zwei Anforderungen (Dahlstedt und Persson, 2005). Der andere Abhängigkeitstyp wird mit *requires* (erfordert) bezeichnet. Er stellt die Bedingung dar, dass eine Anforderung erfüllt sein muss, damit die verknüpfte Anforderung erfüllt werden kann. Dies stellt nicht nur einen konditionalen, sondern auch einen temporalen Zusammenhang mit unterschiedlicher Güte zwischen derart verknüpften Anforderungen dar. Ein starker Zusammenhang zweier Anforderungen vom Typ *requires* drückt so eine enge konditionale und funktionale Abhängigkeit aus. Daher ist auch dieser Abhängigkeitstyp besonders relevant für die Bestimmung der Kohäsion einer Softwarekomponente, die durch die so verknüpften Anforderungen beschrieben wird.

3.2.2.2. Relevante Grundlagen der Graphentheorie

Die Graphentheorie baut grundlegend auf der Existenz paarweiser Relationen zwischen Objekten auf (Gross und Yellen, 2004). Dazu zählen z.B. auch Anforderungen sowie ihre Abhängigkeitsbeziehungen untereinander. Aus diesem Grund eignet sich die Graphentheorie als Hilfsmittel für die formale Darstellung von Anforderungen und ihren Abhängigkeiten. Dies hat den Vorteil, dass die Darstellung als Graph den notwendigen Grad an Struktur für die Berechnung von Anforderungsclustern und zugehörigen Metriken liefert. Für diesen Ansatz werden typisierte und gewichtete Graphen verwendet, bei denen der Typ den Abhängigkeitstyp zwischen zwei Anforderungen beschreibt und das Gewicht den jeweiligen Grad der Kohäsion. Zur pragmatischen Verwendung wird deshalb auch eine gewichtete Adjazenzmatrix verwendet (Diestel, 2010; Gross und Yellen, 2004).

Werden in einem solchen Graphen die Knoten in disjunkte Teilmengen unterteilt, so spricht man in der Graphentheorie von Partitionierung. Eine Eigenschaft der Partitionierung ist neben der Zeit ihre Qualität, die primär von der Schnittgröße geprägt ist. Die Schnittgröße bezeichnet die Summe der Kanten eines Graphen, die bei zwei disjunkten Mengen *V1* und *V2* ein Ende in *V1* und ihr anderes Ende in *V2* haben bzw. bei gewichteten Kanten die Summe der Gewichte. Typische Algorithmen für die Partitionierung versuchen diesen Wert zu minimieren. Dabei wird zwischen geometrischen und koordinatenfreien Algorithmen unter-

schieden (Fjällström, 1998). Da Anforderungen jedoch keine geografischen Koordinaten besitzen, sind die geometrischen Ansätze in diesem Fall obsolet und die nicht-geometrischen Heuristiken rücken in den Vordergrund. Diese Heuristiken konzentrieren sich auf die kombinatorische Struktur des Graphen, die im Falle dieser Arbeit die semantische Kohäsion der Anforderungen widerspiegelt (Fjällström, 1998). Der Fokus wird für diesen Ansatz auf eine Menge von Algorithmen aus der *recursive spectral bisection* gelegt, da diese leistungsfähiger als traditionelle Ansätze, einfach zu implementieren und durch standardisierte Operationen der linearen Algebra effizient zu lösen sind (Luxburg, 2007).

Weiterhin stellt die Graphentheorie die Möglichkeit der strukturierten Analyse von Graphen zur Verfügung. Diese nutzt Metriken, um Schlussfolgerungen über die Charakteristiken des gesamten Graphen, von Teilgraphen oder einzelnen Knoten treffen zu können. Wohingegen Algorithmen die Struktur des Graphen verarbeiten, um ein bestimmtes Problem zu lösen, generiert die Strukturanalyse quantitative Kennzahlen zur Beschreibung des Graphen und des Konstrukts wovon dieser abgeleitet wurde. Dies ist besonders im vorliegenden Problem besonders hilfreich, da es die objektive Identifikation von wichtigen und weniger wichtigen Anforderungen auf der Basis ihrer Beziehungen und Gewichte untereinander erlaubt. Hierbei wird vom Zentralitätskonzept Gebrauch gemacht, das ursprünglich aus der Analyse von Kommunikationsnetzwerken stammt und versucht, jene Knoten zu identifizieren, die für die Kohäsion am wichtigsten sind (Wasserman und Faust, 1994).

3.2.2.3. Vorarbeiten zur Verknüpfung der Anforderungserhebung mit den Möglichkeiten der Graphentheorie

Bisherige Arbeiten, die über Softwareanforderungen die Bestimmung von Kohäsion und Kopplung von Softwarekomponenten im Hinblick auf die Outsourcingentscheidung untersucht haben, sind schwer zu finden. Ganzheitliche Ansätze konnten dafür keine gefunden werden. Deshalb wird die Herangehensweise in drei Teilschritte untergliedert:

Graphenbasierte Darstellung von Anforderungen. Die Ansätze zur auf Graphen basierenden Darstellung von Anforderungen machen hauptsächlich vom

Konzept der Nachvollziehbarkeit und der Repräsentation von nicht-funktionalen Anforderungen als Graphen Gebrauch. Letzteres beschäftigt sich in erster Linie mit der Entscheidungsfindung. Das Forschungsgebiet der auf Graphen basierten Nachvollziehbarkeit versucht, Graphen als Darstellungsmöglichkeit zur Verbesserung der Beschreibung und Analyse von Anforderungen sowie deren Zusammenhänge zu nutzen. Ein Beispiel dafür ist ein auf einem Graphen basierendes Modell, das beschriftete Kanten benutzt, um sämtliche Anforderungen eines Projekts darzustellen. Das Modell beinhaltet ein gewichtetes Maß für die Kanten, das die semantische Zusammengehörigkeit zwischen den verschiedenen Anforderungen ausdrücken soll (Heim et al., 2008; Hildenbrand, 2008).

Schwarz et al. (2010) hingegen präsentieren einen neueren und umfassenderen Ansatz, in dem eine formale Darstellungsweise eines Graphen dafür verwendet wird, um die Nachvollziehbarkeit zu verbessern. Die Autoren benutzten typisierte Kanten, um zwischen verschiedenen Arten von gegenseitigen Abhängigkeiten zu unterscheiden. Li et al. (2008) stellen einen Ansatz vor, der einen Graphen dafür verwendet, um die Auswirkungen von Veränderungen als Teil der Nachvollziehbarkeit von Anforderungen zu analysieren. Sie verwenden unterschiedliche Typen von Verknüpfungen, die aus einer älteren Version eines Modells elementarer Abhängigkeitstypen abgeleitet werden, und sind deshalb für eine einheitliche Darstellung ungeeignet (Dahlstedt und Persson, 2005). Nur der Ansatz von Yaung (1992) stellt sich als zweckmäßig für das Clustern von Anforderungen durch Anwendung eines auf Graphen basierenden Modells heraus. Sein Ansatz bezieht zusätzlich die funktionalen Anforderungen in die Analyse mit ein, ohne dabei zwischen Verknüpfungstypen zu unterscheiden. Eine Verknüpfung in seinem Modell drückt einen bestimmten Grad der Kohäsion aus. Dieser Grad wird als Kantengewicht eingebunden.

Clustern von Anforderungen. Die Mehrzahl der Ansätze zum Clustern von Anforderungen nutzen das Clustern für die Modularisierung von Systemen, welche üblicherweise als Dekomposition in streng kohäsive und locker gekoppelte Gruppen (vgl. Kapitel 3.2.1.1) von Anforderungen betrachtet wird (Myers, 1978; Troy und Zweben, 1981). Die einzelnen Ansätze, um dieses Ziel zu erreichen, unterscheiden sich jedoch substantiell. Li et al. (2009) setzen auf die Verkapselung von Anforderungen, welche im Wesentlichen die Modularisierung von Anforde-

rungen sowie die Definition von Schnittstellen für diese Module darstellt. Sie definieren eine Menge von sieben Anforderungsattributen, welche die Semantik und Struktur der Anforderungen betreffen. Er werden keine expliziten Verknüpfungen zwischen Anforderungen verwendet.

Anforderungen werden auf der Basis der Gemeinsamkeiten all ihrer Attribute gruppiert, so dass ein multidimensionaler Vergleich vorliegt. Dieser Ansatz erfordert eine detaillierte Anforderungsspezifikation und umfangreiche händische Anpassungen, um die Gesamtmenge der Anforderungen und ihre Attribute zu definieren. Der Ansatz von Yaung (1992) ist richtungsweisend für die Verwendung einer Graphenstruktur zum Clustern von Anforderungen als kohäsive Knoten. Yaung (1992) unterscheidet in seinem Ansatz zwar noch nicht zwischen unterschiedlichen Beziehungstypen der Anforderungen untereinander und sein vorgeschlagener Algorithmus erfordert die Festlegung eines Schwellwertes für die Kohäsion durch einen Experten. Wenn der Kohäsionswert zwischen zwei Anforderungen über dem definierten Schwellenwert liegt, so werden diese dem gleichen Cluster zugeordnet. Daher ist das Ergebnis des Algorithmus stark davon abhängig, wie dieser Wert definiert wurde. Gleichwohl ist diese Vorgehensweise ein frühes Beispiel für die Anwendung graphenbasierter Ansätze. Ergänzt wird dieser Ansatz durch die Erforschung einer Ähnlichkeitsmatrix für Anforderungen zur Softwaremodularisierung von Al-Otaiby und AlSharif (2007). Es ist allerdings schwierig, Schlussfolgerungen aus dem Vergleich der Performanz dieser beiden unterschiedlichen Algorithmen zu ziehen.

Strukturelle Analyse von Anforderungen. Für die Umsetzung einer Technik zur Gruppierung von Anforderungen auf der Basis graphenbasierter Ansätze zur Bestimmung von Kohäsion und Kopplung von Softwarekomponenten werden zur strukturellen Analyse der Graphen bekannte Metriken aus der sozialen Netzwerkanalyse herangezogen (Kramer und Eschweiler, 2013; Kramer et al., 2014, 2009). Die soziale Netzwerkanalyse wurde als Analysewerkzeug für Netzwerke, die aus Personen bestehen, in den Sozialwissenschaften bekannt (Scott, 2013; Wasserman und Faust, 1994). Sie wird in der auf Graphen basierenden Nachvollziehbarkeit von Änderungen in Softwareprojekten dafür verwendet, um zentrale Personen im Beziehungsgeflecht des Gesamtprojekts zu identifizieren (Hildenbrand, 2008). Für die Gruppierung von Anforderungen steht die Maßzahl für die

Zentralität aus der sozialen Netzwerkanalyse, welche die Wichtigkeit eines Knotens im Netzwerk ausdrückt, im Vordergrund (Kramer und Eschweiler, 2013).

Zur Hervorhebung der wichtigen Knoten im Netzwerk, also der zentralen Anforderungen, wird der Ansatz von Fitsilis et al. (2010) herangezogen, der die Techniken der sozialen Netzwerkanalyse umsetzt. Mit der dort verwendeten Abhängigkeitsmatrix zwischen Anforderungen können individuelle Maßzahlen für die Zentralität einzelner Anforderungen bestimmt werden. Die Schwächen dieser Matrix und deren unterschiedlich deutbaren Schlussfolgerungen (je nach Wahl der Metrik) werden im Technikansatz von Kramer und Eschweiler (2013) und Kramer et al. (2014) verbessert. Mit Hilfe dieser Technik ist es nun möglich, Anforderungen aus einem Softwareprojekt, die in einer Kollaborationsplattform abgelegt sind, einzulesen, als Graph darzustellen und zu analysieren. Der Anwender muss dabei nur noch den Schwellenwert für die Kohäsion festlegen oder dynamisch anpassen und erhält als Ergebnis die Gruppierung von Anforderungen entsprechend der Maßzahlen aus der sozialen Netzwerkanalyse (Kramer und Eschweiler, 2013; Kramer et al., 2014).

3.2.2.4. SODA – eine Entscheidungsunterstützungstechnik

Mit *Software Outsourcing Decision Aid*) (SODA) wird nun eine Technik vorgestellt, die durch Umwandlung von Anforderungen und deren Abhängigkeiten in eine graphenbasierte Darstellung eine Strukturanalyse zur Bestimmung der Kohäsion und Kopplung von Softwarekomponenten ermöglicht. Eine solche Technik unterstützt den Entscheider bei der Bewertung der strukturellen Eigenschaften einer Komponente in Bezug auf das Softwareprodukt. Die Entscheidungsunterstützungstechnik lässt sich, wie in Abb. 3.3 dargestellt, auf der Ebene der Analysephase in den Softwareentwicklungsprozess (vgl. Kap. 2.3.2) einordnen. SODA operiert auf der Basis von Anforderungen und verwendet diese, um kohärente Anforderungscluster zu erstellen, die dann wieder in die Analysephase übernommen werden oder in das Design des Softwareprodukts als initiales Architekturmodell einfließen können.

In Abb. 3.4 sind die konzeptuellen Schritte, aus denen SODA besteht, skizziert. Der erste Schritt bezieht sich auf die Erstellung eines Anforderungsmodells. Softwareanforderungen werden in ein graphenbasiertes Modell transformiert, das auf

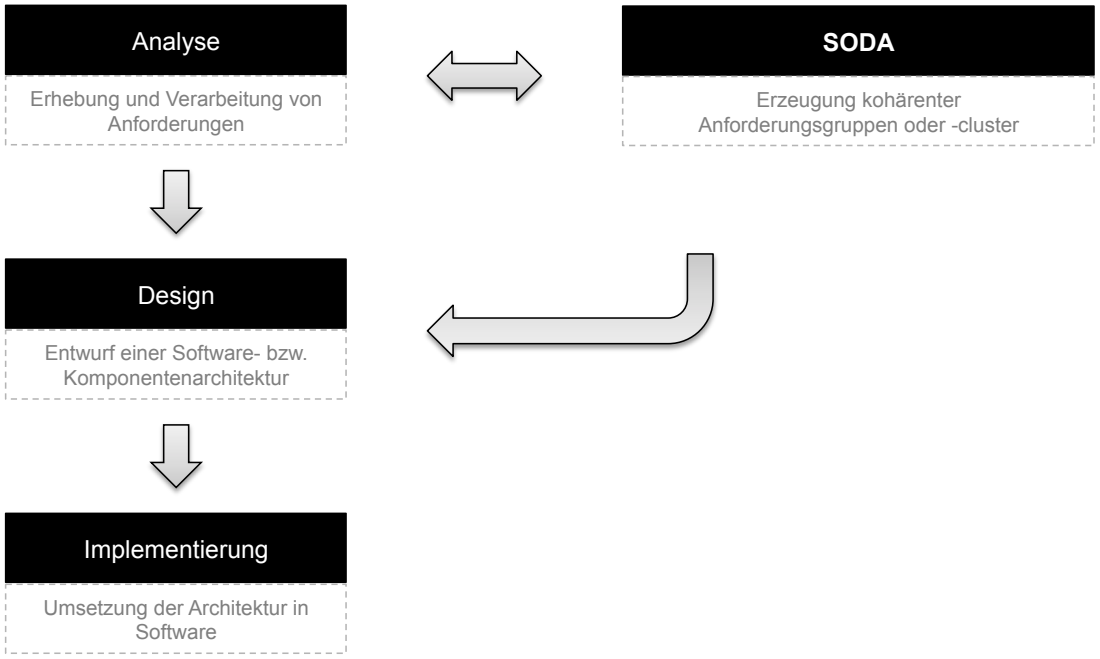

Abbildung 3.3.: Einordnung von SODA in den Kontext des Softwareentwicklungsprozesses

spezifischen syntaktischen Regeln basiert. Das graphenbasierte Modell wird dann an den zweiten Schritt weitergeleitet. Dort wird durch die Anwendung eines Clusteralgorithmus versucht, kohärente Gruppen von Anforderungen zu identifizieren. Der dritte Schritt übernimmt dann die Cluster, die im vorherigen Schritt gefunden wurden, und führt eine Strukturanalyse durch, um vergleichbare Werte für die Kohäsion und Kopplung von einzelnen Elementen dieses Clusters zu erzielen. Zur Weiterverwendung dieser Ergebnisse für die Outsourcingentscheidung wird davon ausgegangen, dass die Cluster, die im zweiten Schritt identifiziert werden, die Architektur des Softwareprodukts darstellen und jedes Cluster damit eine Softwarekomponente ist. Die ermittelten Werte für die Kohäsion und Kopplung können dann direkt für die Berechnung des Entscheidungsmodells übernommen werden.

Darstellung der Anforderungen. Das Anforderungsmodell soll die kombinatorische Struktur aller Anforderungen eines Softwareprojekts widerspiegeln. Dafür wird das Modell von Dahlstedt und Persson (2005) mit den fundamentalen Abhängigkeitstypen auf die beiden Abhängigkeiten *requires* und *similar_to*

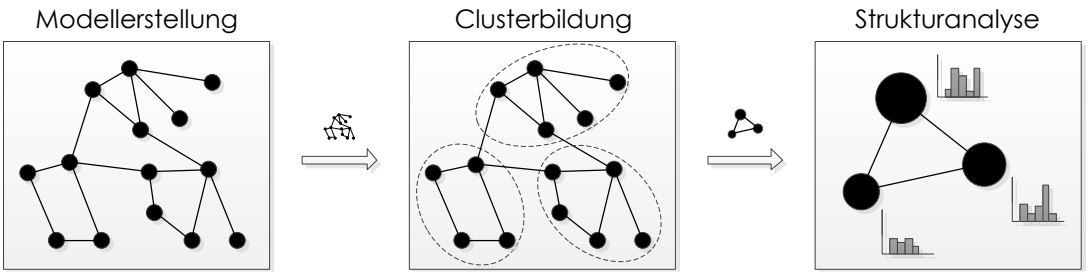

Abbildung 3.4.: Vorgehensmodell anforderungsbasierter Ansätze (Kramer et al., 2014)

reduziert. Das graphenbasierte Modell kann somit definiert werden als

$$G_{Anforderungsmodell} = (V, E, w_E, t_E, W, T)$$

wobei V die Menge der Anforderungen darstellt und E die Multimenge der gerichteten Kanten darstellt, die die Abhängigkeiten zwischen den Anforderungen in V repräsentieren. w_E ist eine Gewichtungsfunktion, die jeder Kante in E ein Gewicht zuordnet, das den Grad der Abhängigkeit reflektiert. Hierfür werden drei verschiedene Grade der Abhängigkeit empfohlen. Wie diese dann in quantitative Werte umgesetzt werden, hängt von der spezifischen Implementierung im Prototypen ab. Die einzige Restriktion in diesem Fall legt fest, dass W kein negatives Element beinhalten darf. $G_{Anforderungsmodell}$ ist ein beschrifteter Graph. Aus diesem Grund stellt t_E eine Abbildung dar, die jeder Kante einen Typ (z.B. eine Beschriftung) aus der Menge der Typen T zuweist. T wird definiert als $T = \{similar\_to, requires\}$. Die Menge kann einfach durch Hinzufügen weiterer Abhängigkeitstypen zu T erweitert werden. Das allgemeine Modell aggregiert schließlich alle Typen in einem einzigen Graphen. Es erlaubt keine Schleifen, da für die gegebene Menge an Abhängigkeitstypen Schleifen keinen informativen Mehrwert bedeuten.

Ein wichtiger Aspekt bei der Erstellung des Anforderungsmodells sind die „*Kosten*“ zur Beschreibung der Laufzeit des Algorithmus aus Sicht der Komplexitätstheorie, die zur Befüllung des Graphen aufgebracht werden müssen (Cormen et al., 2009). Der hauptsächliche Kostentreiber dabei ist die Identifikation von Abhängigkeiten zwischen Anforderungen. Dafür müssen paarweise Vergleiche zu den Kosten von $(n * (n-1))/2$ durchgeführt werden, wobei n die Anzahl der An-

forderungen ist. Eine manuelle Durchführung dieser Vergleiche wäre daher sehr zeitaufwändig. Aus diesem Grund bietet sich ein automatisierter Ansatz zur Unterstützung dieses Vorgangs an. SODA übernimmt diese Aufgabe und ermittelt den erforderlichen Graphen auf der Basis einer einlesbaren Liste von Anforderungen und ihren Verknüpfungen oder direkt aus einem Kollaborationssystem.

Strukturierung der Anforderungen. Die Zielsetzung der Gruppen- oder Clusterbildung ist die Identifikation von kohärenten Gruppen von Anforderungen. Im Falle der Softwareentwicklung ist weder die Zahl der Cluster noch die Größe der einzelnen Cluster im Voraus bekannt. Wichtig dabei ist, dass semantisch kohärente Gruppen von Anforderungen nicht notwendigerweise gleich groß sein müssen. Ebenfalls ist die Anzahl der Cluster unbegrenzt. Eine Beschränkung könnte den Algorithmus fälschlich beeinflussen und ihn daran hindern, eine optimale Partitionierung unter den Gesichtspunkten von SODA zu erzielen. Aus diesem Grund muss der Algorithmus frei von Parametern und exogenen Beschränkungen sein. Der einzige Input ist die Struktur des Anforderungsmodells, das entsprechend in Gruppen aufgeteilt werden soll.

Aus diesem Grund wird ein Algorithmus von Newman (2006) verwendet. Er ist deshalb sehr gut geeignet, da er parameterfrei ist und versucht, die Modularität eines gegebenen Netzwerks (Graphen) zu maximieren. Die Zielsetzung von Newmans Algorithmus ist die Entdeckung isolierter Gemeinschaftsstrukturen. Eine solche Struktur bezeichnet eng miteinander verbundene Gruppen von Knoten mit nur wenigen Verbindungen zu anderen Gruppen (Newman, 2006). Ein Algorithmus, der Gemeinschaftsstrukturen entdeckt, berücksichtigt auch, dass die Anzahl und Größe von Gruppen ausschließlich durch das Netzwerk bestimmt wird und nicht durch eine bestimmende Person. Dieser Algorithmus lässt auch zu, dass eventuell keine Teilung des Netzwerks möglich ist. Ein weiterer Vorteil des Algorithmus von Newman (2006) ist die automatische Erkennung der Anzahl von Clustern durch die Einbindung eines Kontrollmaßes: Modularität. Sein Ansatz ist daher maßgeschneidert für die Identifikation modularer Gemeinschaften, die hinsichtlich ihrer Größe unausgewogen sind. Damit kann schließlich eine Gruppierung im Anforderungsmodell vorgenommen werden, so dass Gruppen entstehen, die eng innerhalb miteinander verbunden sind aber nur lose mit anderen Gruppen

verknüpft sind. Dadurch wird ein initiales Architekturmodell entworfen, das die zukünftigen Softwarekomponenten und ihre zugehörigen Anforderungen abbildet.

Strukturanalyse der Anforderungen. Die Zielsetzung des dritten Schritts von SODA ist die Analyse der Anforderungsarchitektur aus dem vorangestellten Schritt aus einer globalen (gesamter Graph), regionalen (einzelne Cluster) und lokalen Perspektive (einzelne Knoten), um dem Entscheider Unterstützung bei der Outsourcingentscheidung anzubieten. Da sich die Unterstützung von SODA im Rahmen des Entscheidungsmodells dieser Arbeit lediglich auf die Kohäsion und Kopplung von Anforderungen bezieht, wird nun ausschließlich auf die regionale, d.h. auf einzelne Cluster bezogene Perspektive von SODA eingegangen. Die globale und lokale Sichtweise wird von Kramer und Eschweiler (2013) detailliert erläutert.

Kopplung und Kohäsion des Clusters. Die Kopplung und Kohäsion eines Clusters werden durch regionale Metriken oder Gruppenmetriken repräsentiert. Ersteres ist dabei ein Indikator dafür, wie stark eine Teilmenge von Anforderungen, die zu einer Komponente gehören, mit anderen Anforderungsmengen externer Gruppen verknüpft sind. Letzteres drückt aus, wie eng eine Anforderungsmenge einer einzelnen Komponente untereinander verknüpft ist. Es gilt, dass Q_{max} die erreichbare Modularität eines gesamten Anforderungsmodells beschreibt (gloable Perspektive). Im Gegensatz dazu beziehen sich Kopplung und Kohäsion nur auf Gruppen von Anforderungen, genauer gesagt auf die Cluster, die durch den Partitionierungsalgorithmus von SODA generiert wurden. Die beiden Metriken bieten daher einen spezifischen Einblick in die Auslagerungsfähigkeit solcher Cluster, die gleichzeitig Softwarekomponenten repräsentieren. Hierfür definieren Briand et al. (1996) Eigenschaften für Funktionen, die die Kopplung und Kohäsion messen. Daran orientieren sich die im Rahmen von SODA entwickelten Metriken.

Um den Grad der Kopplung eines Clusters zum Ausdruck zu bringen, wird ein einfaches und auf einem Zähler basierendes Maß verwendet, das die Gewichte aller Kanten kumuliert, die aus dem betrachteten Cluster in ein anderes Cluster führen. Das Maß ist absichtlich nicht normalisiert und hat deshalb eine Wertspanne von 0 bis ∞. Die darunterliegende Idee ist, dass die Kopplung unabhängig von der

Größe sowohl des Clusters als auch des Projekts ist und nur von den Gewichten der Verbindungen zu anderen Clustern bestimmt wird.

Im Gegensatz zur Kopplung ist die Kohäsion kein additives Maß. Stattdessen heben Briand et al. (1996) hervor, dass Kohäsion ein normalisiertes Maß ist, das sich stets in einem bestimmten Intervall befindet. Deshalb wird im Rahmen von SODA eine Funktion verwendet, die die Summe aller internen Kanten, die von einem Knoten innerhalb des Clusters zu einem anderen Knoten innerhalb des gleichen Clusters verlaufen, in Relation zu allen internen und externen Kanten gesetzt werden. Mit den externen Kanten sind diejenigen Verbindungen gemeint, die für die Kopplung herangezogen wurden. Der Wertebereich des Maßes ist daher auf das Intervall von 0 bis 1 begrenzt. Ein Kohäsionswert unter 0,5 gibt an, dass ein Cluster mehr externe Abhängigkeiten als interne Abhängigkeiten besitzt. Daher ist es erstrebenswert, einen Kohäsionswert zu erzielen, der signifikant über 0,5 liegt und gegen 1 strebt. Aufgrund der Eigenschaft der nahezu vollständigen Zerlegbarkeit komplexer Systeme ist es unwahrscheinlich, dass für alle Cluster des Anforderungsmodells maximale Kohäsion erreichbar ist (Simon, 1962).

Die globalen und lokalen Perspektiven der Strukturanalyse der Anforderungen sowie die Evaluation der SODA-Technik werden von Kramer und Eschweiler (2013) und Kramer et al. (2014) aufgezeigt und detailliert beschrieben. Mit der Unterstützungstechnik SODA wird einem Entscheider nun die Möglichkeit gegeben, sich die Ausprägungen für Kohäsion und Kopplung einer Softwarekomponente zur Anwendung des Entscheidungsmodells dieser Arbeit zu berechnen. Der Aufwand für eine manuelle Berechnung der beiden Werte wäre jedoch sehr hoch. Aus diesem Grund wird die Funktionsweise von SODA in der prototypischen Implementierung dieser Arbeit berücksichtigt. Dem Entscheider steht damit ein automatisiertes Verfahren zur Bestimmung der beiden Kriterien für die strukturellen Eigenschaften einer Komponente in Bezug auf das Softwareprodukt zur Verfügung. Die Einschätzung der jeweiligen Ausprägungen auf Basis der Entscheidungstabellen, die im nächsten Kapitel eingeführt werden, wird so durch einen systematischen Ansatz unterstützt. Die Details zur Umsetzung werden in Kapitel 3.3.2.5 näher erläutert.

3.2.3. Entscheidungslogik

Zur Vervollständigung des Entscheidungsmodells muss schließlich noch die Zielfunktion definiert werden. Dadurch wird das Modell mit den notwendigen logischen Verknüpfungen zwischen den einzelnen Entscheidungsgrößen versehen, um in der Zielfunktion eine konkrete Aussage über das Outsourcingpotenzial einer Softwarekomponente treffen zu können. Hierfür wird auf die Verfahren von EUS für das Outsourcing zurückgegriffen, wie sie in Kapitel 2.2.1.3 vorgestellt wurden.

Die multiobjektiven und die multikriteriellen Verfahren repräsentieren dabei die beiden Klassen von EUS, welche für die Verwendung von Outsourcingproblemen geeignet sind (vgl. Kapitel 2.2.1.3). Für die Erforschung eines geeigneten Ansatzes für das vorliegende Entscheidungsproblem wird die Suche auf die multikriteriellen Verfahren beschränkt, da es sich hierbei um eine Entscheidungsfindung mit endlichem Lösungsraum handelt und somit die multiobjektiven Verfahren ausscheiden. Weiterhin eignen sich die multikriteriellen Verfahren auch besser, da sie eine Vielzahl an Attributen berücksichtigen können, welche auf die Zieldimension Einfluss nehmen. Genau diese Situation ist beim vorliegenden Entscheidungsmodell gegeben. Im weiteren Verlauf dieses Kapitels werden nun die existierenden Verfahren zur Lösungsfindung hinsichtlich notwendiger Voraussetzungen für den Entscheidungsansatz dieser Arbeit bewertet und die resultierende Methode für ihren Einsatzzweck näher erläutert.

In Tabelle 3.5 ist eine Übersicht der multikriteriellen Verfahren zur Entscheidungsfindung unter Sicherheit (für jede verfügbare Handlungsalternative sind sämtliche Konsequenzen bekannt) dargestellt, in welcher diese nach Kriterien beurteilt werden, die für die Umsetzung des vorliegenden Entscheidungsmodells für komponentenbasierte Outsourcingentscheidungen relevant sind. Hierfür werden Bewertungskriterien wie Erweiterungsmöglichkeit der Entscheidungsattribute, Mehrstufigkeit des Entscheidungsprozesses, Nachvollziehbarkeit, Übersichtlichkeit und Wiederverwendbarkeit herangezogen, die sich aus den Anforderung an das Konzept dieses Entscheidungsmodells ableiten lassen (siehe Tabelle 3.1). Die Erweiterungsmöglichkeit für Attribute ist in diesem Fall von großer Bedeutung. Eine Erweiterung der Liste der Merkmale wäre bei Bedarf möglich. Deshalb müssen auch die Entscheidungsgrößen des Modells sowie dessen Entscheidungslogik erweiterbar sein und den mehrstufigen Charakter des Entscheidungsmodells

abbilden können. Außerdem muss die Entscheidungsfindung transparent für den Entscheider sein. Dies bedeutet, dass das Verfahren mit Entscheidungsinformationen auf mehreren Aggregationsstufen umgehen können muss, bevor die finale Entscheidung bewertet wird. Vor diesem Hintergrund ist ebenfalls die Nachvollziehbarkeit und Übersichtlichkeit ein beachtenswerter Aspekt des auszuwählenden Verfahrens, um die Entscheidungsunterstützung optimal zu gewährleisten. Schließlich ist die Wiederverwendbarkeit eines Verfahrens mit dessen voreingestellten Regeln ein wichtiger Pluspunkt, den das Aggregationsverfahren beherrschen muss, um somit eine unternehmensweite Wissensbasis zu schaffen, die auf zukünftige Outsourcingentscheidungen angewendet werden kann und die Erfahrungen der bisherigen Entscheidungen implizit beinhaltet.

Tabelle 3.5.: Vergleich multikriterieller Verfahren

	Erweiterungsmöglichkeit der Attribute	**Mehrstufiger Entscheidungsprozess**	**Nachvollziehbarkeit**	**Übersichtlichkeit**	**Wiederverwendbarkeit der initialen Konfiguration**
Aspektweise Eliminierung	✓	✓	X	X	X
Entscheidungsbaum	X	✓	✓	✓	✓
Entscheidungsmatrix/-tabelle	✓	✓	✓	✓	✓
Lexikografische Methode	✓	✓	X	X	X
AHP	✓	X	X	X	X
ELECTRE	✓	X	X	X	X
PROMETHEE	✓	X	✓	X	✓
SMART	✓	X	✓	X	✓

✓ ist möglich/gegeben

X ist nicht möglich/gegeben

Die Methode der *aspektweisen Eliminierung* erlaubt einerseits die freie Auswahl an Entscheidungsgrößen, die zur Entscheidungsfindung herangezogen werden, wodurch das Entscheidungsfeld einfach erweitert werden kann. Andererseits lässt sie sich auf unterschiedlichen Ebenen eines Entscheidungsmodells mehrmals anwenden und garantiert damit die durchgängige Anwendung in mehrstufigen Modellen. Durch die Sortierung und Auswahl der Lösungsmöglichkeiten nach

den wichtigsten Entscheidungsgrößen ist jedoch eine Nachvollziehbarkeit der Entscheidung nur bedingt gegeben, obwohl der Algorithmus zur Auswahl klar ist. Die Auswahl einer Outsourcingoption ist bei dieser Methode primär auf der als am wichtigsten eingeschätzten Entscheidungsgröße begründet. Der Einfluss weiterer Merkmale kann daher nicht nachvollzogen werden. Damit ist der Ansatz für Entscheidungsvorlagen ungeeignet. Zahlreiche Sortierungsvorgänge sind notwendig, bis alle ungültigen Lösungsoptionen eliminiert sind. Die Übersichtlichkeit des Ansatzes geht daher schnell verloren. Weiterhin lässt sich die Methode nur dadurch wiederverwenden, indem sie für eine weitere zu bewertende Softwarekomponente erneut komplett durchgeführt wird. Auch hier erweisen sich die unzähligen Sortiervorgänge als unpraktikabel (abgeleitet von Kahraman, 2008; Ranyard, 1976; Tversky, 1972). Die Ähnlichkeit dieses Ansatzes zur *lexikografischen Methode*, die ebenfalls auf ähnlichen Sortiermustern beruht, führt zur gleichen Bewertung der Auswahlkriterien (abgeleitet von Blume et al., 1991; Fishburn, 1980; Kohli und Jedidi, 2007).

Beim *Entscheidungsbaum* lassen sich aufgrund seiner Struktur relativ einfach mehrstufige Entscheidungsmodelle abbilden, da hier zunächst untere Ebenen aggregiert werden bis sie nach oben zu einer finalen Entscheidung zusammengeführt werden (vgl. Abbildung 2.4). Die finale Entscheidung lässt sich daher einfach entlang der Baumstruktur nachvollziehen. Diese Nachvollziehbarkeit hängt eng mit der Übersichtlichkeit dieses Ansatzes zusammen und wird durch die meist grafische Darstellung als Baum wesentlich erleichtert und für Menschen schnell erfassbar gemacht. Ein fertig konfigurierter Entscheidungsbaum lässt sich ohne weitere Anpassungen auf beliebige Softwarekomponenten anwenden. Einziger Nachteil dieser Methode ist die schwierige Erweiterungsmöglichkeit von Entscheidungsgrößen. Auf der untersten Ebene des Entscheidungsbaums lässt sich dies noch recht einfach bewerkstelligen, indem das neue Merkmal einfach als weiterer Zweig angeführt wird. Auf den Ebenen darüber gestaltet sich ein solche Erweiterung als schwierig, da sämtliche Verzweigungen auf der darunterliegenden Ebene eingefügt und an die bereits existierenden Ausprägungen angepasst werden müssen. Außerdem geht bei zunehmender Anzahl an Entscheidungskriterien die Übersichtlichkeit der Baumstruktur verloren (abgeleitet von Eisenführ et al., 2010; Lee, 2010).

Diesen spezifischen Nachteil umgehen Entscheidungsträger mit *Entscheidungstabellen*, indem sie nur die Entscheidungstabelle auf der jeweils betroffenen Ebene und die darin enthaltenen Zieldimensionen anpassen müssen. Die Mehrstufigkeit des Prozesses kann ebenfalls einfach mit einer eigenen Entscheidungstabelle pro Ebene erreicht werden. Die Nachvollziehbarkeit der Entscheidung und die Übersichtlichkeit dieser Vorgehensweise wird durch einfach verständliche Tabellen ermöglicht, die zu jeder Ausprägung der Entscheidungsmerkmale die Ausprägung der Zieldimension vorgeben. Wie beim Entscheidungsbaum auch, lassen sich Entscheidungstabellen nach einmaliger Konfiguration problemlos für die Entscheidungsfindung aller Komponenten wiederverwenden (abgeleitet von Eisenführ et al., 2010; Mullur et al., 2003).

Die multikriteriellen Verfahren mit kardinaler Messung der Zielattributwerte (*AHP*, *ELECTRE*, *PROMETHEE* und *SMART*) eignen sich sehr gut für die Erweiterung von Entscheidungsgrößen, da sie alle Lösungsmöglichkeiten in paarweisen Vergleichen unter Berücksichtigung aller Merkmale evaluieren, auf eine kardinale Skala transformieren und die bestmögliche Lösung ermitteln. Eine weitere Entscheidungsgröße lässt sich bei diesen Verfahren leicht integrieren. Bei mehrstufigen Entscheidungsprozessen sind diese Verfahren jedoch allesamt ungeeignet, weil mehrstufige Zielhierarchien beim paarweisen Vergleich nicht aggregiert werden können. Die Anzahl der paarweisen Vergleiche wird schnell sehr groß, wenn viele Lösungsmöglichkeiten für ein gegebenes Problem vorhanden sind. Dadurch werden diese Verfahren schnell unübersichtlich, erschweren dementsprechend die Nachvollziehbarkeit der vorgeschlagenen Entscheidung und führen zu inkonsistenten Paarvergleichen. Lediglich bei den beiden Verfahren PROMETHEE und SMART, die mathematische Berechnungsweisen für den paarweisen Vergleich der Lösungsalternativen verwenden, lassen sich die Lösungsvorschläge detailliert nachvollziehen. In diesem Fall ist die einfache Wiederverwendung für die Bewertung weiterer Komponenten gegeben (abgeleitet von Brans et al., 1986; Edwards und Barron, 1994; Figueira et al., 2005b; Saaty, 2005).

Unter Berücksichtigung der dargelegten Auswahlkriterien eignen sich Entscheidungsmatrizen bzw. Entscheidungstabellen zur Abbildung der Zielfunktion für das vorliegende Entscheidungsmodell, da die erforderlichen Kriterien von diesem Verfahren erfüllt werden (vgl. Tabelle 3.5). Für den weiteren Verlauf dieser Arbeit

wird der Begriff *Entscheidungstabelle* für das Verfahren benutzt, welches synonym für Entscheidungsmatrix verwendet wird.

3.2.3.1. Entscheidungstabellen als einzusetzendes Verfahren für die Aggregate

Für das Entscheidungsmodell dieser Arbeit werden Entscheidungstabellen als einfach zu verwendende und nachvollziehbare Entscheidungstechnik zur Verknüpfung und Aggregation der Merkmale von Softwarekomponenten eingesetzt, um deren Outsourcingpotenzial zu bewerten. Somit kann jede Komponente eines Softwaresystems von den Entscheidern intendiert auf rationale und intersubjektiv nachvollziehbare Weise evaluiert und eingeordnet werden.

Entscheidungstabellen zählen zu den multikriteriellen Entscheidungsverfahren (vgl. Kapitel 2.2.1.3) und werden verwendet, um Entscheidungsprozesse zu beschreiben (Heinrich et al., 2004; Pooch, 1974). Ihr formaler Charakter kann mit einer Matrix verglichen werden, deren Zeilen im oberen Abschnitt alle Bedingungen einer Situation und im unteren Abschnitt alle möglichen Handlungsalternativen der zu untersuchenden Problemdomäne beschreiben. Die Spalten einer Entscheidungstabelle enthalten die sog. *Regeln*, die alle Kombinationen der Bedingungen einer Situation ausdrücken (vgl. Grundstruktur einer Entscheidungstabelle in Abb. 3.5). Die einzelnen Zellen innerhalb der Matrix im oberen Abschnitt bestimmen die Ausprägungsmöglichkeiten der einzelnen Bedingungen für die jeweilige Regel und im unteren Abschnitt die daraus abzuleitende Handlungsempfehlung. Generell werden dabei die Ausprägungen der Bedingungen, die zu einer Regel gehören (also in der gleichen Spalte stehen) mit dem Operator *UND* verknüpft (Dumke, 2003; Heinrich et al., 2004). In der Entscheidungstheorie werden damit Entscheidungsprobleme in Zusammenhang mit einer Nutzenfunktion betrachtet, die jeder Handlungsalternative einen kardinalen Nutzen zuordnet (Laux et al., 2004). Die Nutzenfunktion wird in dieser Arbeit durch die Kombination von Regeln und Handlungsempfehlungen umgesetzt. Die Anwendung solcher Entscheidungstabellen im Kontext dieser Arbeit wird im Kapitel 3.2.4.1 näher erläutert.

Entscheidungstabelle		$R_1 - R_i - R_p$
C_1 … C_j … C_m	*Bedingungen*	*Alternativen der Bedingungen*
A_1 … A_j … A_m	*Handlungsmöglichkeiten*	*Handlungsempfehlungen*

Abbildung 3.5.: Struktur einer Entscheidungstabelle

3.2.3.2. Gewichtungsfunktion der Aggregate

Die Untergliederung der Entscheidungskriterien in drei unterschiedliche Kategorien (vgl. Kapitel 3.2.1) macht es erforderlich, eine Möglichkeit zur Gewichtung für die jeweiligen Kriteriengruppen einzuführen. Dadurch wird eine Präferenzfunktion für die jeweiligen Entscheider geschaffen, die entsprechend eigener Vorstellungen, situativen Gegebenheiten oder entlang der Ziele des Unternehmens handeln möchten. Zu diesem Zweck bedient man sich den Gewichtungsmöglichkeiten aus der Nutzwertanalyse (Zangemeister, 2003). Deren Gewichtungsfunktion definiert die Aufstellung einer Prioritätenliste für die jeweils vorhandenen Kategorien durch den Anwender. Entsprechend des Rangs können so den einzelnen Kategorien Gewichte für die Nutzwertberechnung zugeordnet werden (Zangemeister, 2003). Diese Möglichkeit wird in dieser Arbeit auf die Entscheidungskategorien übertragen, die je nach Anwender unterschiedlich starke Bedeutungen haben können.

Die Gewichtungsfunktion des Entscheidungsmodells dieser Arbeit soll also die Präferenzstruktur des Entscheiders abbilden. Sie hat damit die Aufgabe, die Bedeutung eines Aggregats von Entscheidungskriterien gegenüber den anderen vorhandenen Aggregaten (vgl. Kapitel 3.2.1) aus Sicht des Entscheiders widerzuspiegeln. Dabei wird davon ausgegangen, dass eine kardinale Bewertung der Kriteriengruppen möglich ist (Weber, 1983). Bekannte Verfahren zur Darstellung der Präferenzen des Entscheiders verwenden einerseits die Reihenfolge der verfügbaren Alternativen, die vom jeweiligen Anwender bestimmt wird, um eine erste Einordnung zu erhalten. Andererseits wird diesen, beginnend mit der am

wenigsten wichtigen Alternative, ein fester Wert zugeordnet, der anschließend in Relation dazu für alle anderen Alternativen entsprechend festgelegt wird. Durch Normierung kann schließlich für jede Alternative ein Gewicht zwischen *0* und *1* erzielt werden (Edwards, 1971; Weber, 1983). Transparente Berechnungsverfahren für die Gewichtung haben den Vorteil, dass der mit der komplexen Situation konfrontierte Entscheider mit leicht verständlichen Mitteln seine Präferenzen zum Ausdruck bringen kann (Weber, 1983). Die entsprechende Gewichtungsfunktion zur Berechnung der Outsourcingentscheidung im Entscheidungsmodell dieser Arbeit wird in Kapitel 3.2.4.2 vorgestellt.

3.2.3.3. Amalgamation der Teilnutzenwerte

Nachdem nun mit Entscheidungstabellen eine Skala zur Bewertung der einzelnen Kriterien und für die Präferenzen des Entscheiders hinsichtlich der Entscheidungskategorien ein Ansatz für eine Gewichtungsfunktion vorgestellt wurde, müssen schließlich alle Alternativen bewertet und gegeneinander abgewogen werden. Um alle Entscheidungskriterien unter Berücksichtigung der Gewichte ihrer jeweiligen Kriterienkategorie in die Berechnung von Handlungsalternativen einfließen zu lassen, müssen alle Teilbewertungen in einen eindimensionalen Wert amalgamiert werden. Erst dadurch kann Vergleichbarkeit zwischen den Alternativen hergestellt werden und eine finale Entscheidung getroffen werden (Zangemeister, 1976).

Damit dieses Ziel erreicht werden kann, müssen zunächst alle einzelnen Entscheidungskriterien mit einer Skala versehen werden. Diese Skala ermöglicht die Festlegung des Entscheiders auf einen zum jeweiligen Kriterium gehörigen Wert, der schließlich in die Amalgamation mit einfließt und den eindimensionalen *Nutzwert* der Alternative bestimmt (Zangemeister, 1976, 2003). Anschließend kann für jede Kriterienkategorie der Teilnutzen einer Alternative berechnet werden, indem alle Einzelwerte aufsummiert werden. Durch Anwendung der im Vorfeld definierten Gewichte pro Kategorie lässt sich so ein synthetisierter Wert erreichen, der den Gesamtnutzen der Alternative ausdrückt. Dafür multipliziert man den Wert der Kategorie mit der zugehörigen Gewichtung und summiert schließlich alle gewichteten Werte zum Nutzwert (Zangemeister, 1976, 2003). Eine angepasste Variante

der Nutzwertanalyse wird für die Amalgamation der Outsourcingentscheidungen dieser Arbeit verwendet und in Kapitel 3.2.4.2 spezifiziert.

3.2.4. Anwendung des Entscheidungsmodells

Zur Bestimmung eines vergleichbaren Werts für die Outsourcingentscheidung einzelner Komponenten wird nach dem Vorbild der Nutzwertanalyse im weiteren Verlauf dieses Kapitels ein angepasstes Verfahren vorgestellt, das für die Verwendung mit dem Entscheidungsmodell dieser Arbeit geeignet ist. Für jede Komponente kann dadurch ein vergleichbarer Wert ermittelt werden, der den Grad der Eignung für ihre Auslagerung ausdrückt. Zunächst wird nun erläutert, wie sich die Teilwerte der einzelnen Kriteriengruppen ermitteln lassen. Anschließend wird die Amalgamation der Entscheidung unter Berücksichtigung der festgelegten Gewichte und schließlich die Interpretation des Ergebniswerts beschrieben.

3.2.4.1. Anwendung der Entscheidungstabellen

Zur Berechnung vergleichbarer Teilwerte für die Kriteriengruppen werden Entscheidungstabellen herangezogen. Als Skala für die Zuordnung eines eindeutigen Werts zu jedem Entscheidungskriterium werden die folgenden drei Ausprägungen definiert: HOCH (*high*), MITTEL (*medium*) und NIEDRIG (*low*). Diese Bezeichner drücken die Einschätzung eines Entscheiders aus, wie stark ein Kriterium bei der vorliegenden Komponente ausgeprägt ist. Mit der Festlegung von Entscheidungstabellen für die jeweiligen Kriterienkategorien sowie einer Entscheidungstabelle für die Zusammenführung der untergeordneten Entscheidungstabellen (vgl. Kapitel 3.2.4.2) wird die Bestimmung des Outsourcingpotenzials einer Softwarekomponente ermöglicht. Dadurch wird die Einschätzung von Outsourcingexperten komponentenweise intersubjektiv nachvollziehbar spezifiziert und einem strukturierten Entscheidungsprozess zur besseren Vergleichbarkeit von Softwarekomponenten zugeführt.

Am Beispiel der Entscheidungstabelle für die Prozessmerkmale der Entwicklung von Softwarekomponenten werden nun der konkrete Aufbau, der Inhalt sowie die strukturierte Berechnungsweise einer solchen Tabelle vorgestellt. Ein Auszug der Tabelle selbst ist in Abb. 3.6 dargestellt. Die komplette Entscheidungstabelle

Tabelle 3.6.: Entscheidungstabelle für die prozessspezifischen Merkmale der Entwicklung von Softwarekomponenten (Auszug)

		R_1 ...	R_{43} ...	R_{132} ...	R_{202} ...	R_{243}
E_3	Entwicklungsdauer	H	H	M	L	L
E_4	Priorität Anf.	H	M	M	M	L
E_5	Einbez. Kunde	H	M	L	M	L
E_6	Erf. Austausch	H	L	M	M	L
E_7	Integrationsaufw.	H	H	L	H	L
L_1	Hoher Einfluss	✓				
L_2	Mittlerer Einfluss		✓		✓	
L_3	Niedriger Einfluss			✓		✓

R_i Regel i als Ausprägungsvektor der Entscheidungsgrößen

E_j Entscheidungsgröße j

L_k Lösungsalternative des Entscheidungsproblems k

H / M / L *hoch / mittel / niedrig*

sowie die Entscheidungstabellen der strukturellen Eigenschaften einer Komponente auf das Softwareprodukt und der Wissensmerkmale von Komponenten sind in den Anhängen A bis C abgebildet.

Im oberen Teil der Matrix werden die Entscheidungskriterien der Prozessmerkmale der Entwicklung von Softwarekomponenten E_3 bis E_7 aufgeführt, die die Ausprägungen *H*, *M* oder *L* annehmen können. Die Regeln R_1 bis R_{243} stellen dabei alle möglichen Kombinationen der Ausprägungen dar. Sie werden in diesem Beispiel nur auszugsweise dargestellt und sind im Anhang B vollständig aufgeführt. Im unteren Teil der Matrix befinden sich die empfohlenen Ausprägungsalternativen der jeweiligen Entscheidungskategorie L_1 bis L_3, von denen jeweils nur eine pro Regel zugeordnet wird und durch einen bestätigenden Haken ausgedrückt wird. Regel R_{43} drückt in diesem Fall aus, dass der Entscheider das Kriterium *Entwicklungsdauer* als *hoch (H)*, die *Priorität der zugehörigen Anforderungen* als *mittel (M)*, den *Einbeziehungsgrad des Kunden* als *mittel (M)*, den *erforderlichen Erfahrungsaustausch* als *mittel (M)* und den *Integrationsaufwand* als *hoch (H)* einschätzt. Durch die Vorgaben der Entscheidungstabelle ergibt sich daraus ein

mittlerer prozessspezifischer Einfluss (L_2) auf die Entscheidungsfindung für das Outsourcing.

Für die Anwendung von Entscheidungstabellen müssen diese a priori vollständig spezifiziert werden. Dies bedeutet, dass für jede Kombination der Ausprägungen der Merkmale (alle Regeln) die entsprechende Handlungsalternative hinterlegt werden muss. Damit sich diese Entscheidungstabellen für die Entwicklung eines Prototyps verwenden lassen, wurde daher eine Berechnungsweise verwendet, die die jeweilige Handlungsalternative für jede einzelne Regel berechnen kann. Im Gegensatz dazu hat die individuelle Spezifikation von Entscheidungstabellen den Vorteil, dass Entscheidungstabellen im Laufe der Zeit ihrer Verwendung durch den Anwender angepasst werden können, wenn sein subjektives Empfinden eine andere Handlungsalternative bevorzugen würde. Damit würde die angepasste Tabelle als Teil des organisatorischen Gedächtnisses bei jeder Entscheidungsfindung mit einfließen (Kramer et al., 2011). Für die Verwendung in dieser Arbeit wird nun die Berechnungsweise der jeweiligen Handlungsalternative für jede Regel vorgestellt.

Zu diesem Zweck muss zunächst die ordinale Skala unter Rückgriff auf eine Hilfsfunktion in eine kardinale Skala umgewandelt werden, ähnlich der Conjoint-Analyse aus dem Marketing (Green und Rao, 1971). Es wird hierbei auf die Berechnung des Medians zurückgegriffen, der den Mittelwert bei ordinalskalierten Daten ermittelt (Cramer und Kamps, 2014). Im Rahmen dieser Hilfsfunktion wird den einzelnen Ausprägungen der Entscheidungsgrößen jeweils ein Wert wie in Tab. 3.7 zugewiesen.

Tabelle 3.7.: Wertzuweisung zu den Ausprägungen der Entscheidungsgrößen

Ausprägung	**Wert**
HOCH (*high*)	3
MITTEL (*medium*)	2
NIEDRIG (*low*)	1

Nach einer solchen Zuweisung lässt sich nun für jede der drei Kriterienkategorien des Entscheidungsmodells der Median ermitteln, der den Teilnutzen der jeweiligen Kategorie darstellt. Am Beispiel von R_{43} lässt sich der Median folgendermaßen aus den Ausprägungen der Entscheidungskriterien dieser Kategorie ermitteln:

- Übersetzung der Ausprägungen in die definierten Werte (vgl. Tab. 3.7):
  {3; 2; 2; 1; 3}
- Sortierung der Menge beginnend mit dem größten Element:
  {3; 3; 2; 2; 1}
- Bestimmung des Medians als der Wert, der die Menge in der Mitte in zwei gleichgroße Untermengen aufspaltet:
  {3; 3; 2; 2; 1}

Alternativ kann dies auch rechnerisch ermittelt werden. Dies erweist sich besonders dann als hilfreich, wenn die Menge aus einer geraden Anzahl an Elementen besteht und sich daher nicht einfach in zwei gleichgroßen Untermengen aufspalten lässt oder wenn man die Hilfsfunktion in der prototypischen Umsetzung des Entscheidungsmodells implementieren möchte. Der Median lässt sich also auch als Mittelwert berechnen und anschließend mit Hilfe der in Tab. 3.8 festgelegten Intervalle als ordinalen Wert bestimmen. Dafür wird der Mittelwert als $(3 + 2 + 2 + 1 + 3)/5 = 2,2$ bestimmt. Im nächsten Schritt wird daraus der Median aus den in Tab. 3.8 aufgezeigten Intervallen abgelesen. Würde ein Entscheider die Ausprägungen der Entscheidungskriterien also wie in R_{43} einschätzen, so würde sich mit dem Mittelwert von *2,2* der Median MITTEL ergeben und daraus ein mittlerer spezifischer Einfluss der Prozessmerkmale der Entwicklung von Softwarekomponenten auf die Gesamtentscheidung abgeleitet werden.

Tabelle 3.8.: Intervalle für den spezifischen Einfluss der jeweiligen Entscheidungskategorie

Intervall	**Klassifizierung**
$1,00 - 1,66$	Geringer spezifischer Einfluss
$1,67 - 2,33$	Mittlerer spezifischer Einfluss
$2,34 - 3,00$	Hoher spezifischer Einfluss

Die Verwendung der Hilfsfunktion ist erforderlich, um eine Amalgamation der Teilnutzenwerte (vgl. Kapitel 3.2.3.3) zu erreichen. Die Besonderheit dieses Transformationsprozesses liegt darin, dass bei ordinalen Werten per Definition nicht automatisch von gleichen Abständen zwischen den Werten ausgegangen werden kann. Dies bedeutet, dass jemand, der ein Kriterium als *hoch* einschätzt, nicht selbstverständlich damit zum Ausdruck bringen möchte, dass dieses Kriterium drei mal so stark ausgeprägt ist, wie wenn es mit *gering* bewertet worden wäre.

Dies kann mit der Verwendung von mehrstufigen Likert-Skalen verglichen werden, bei denen ebenfalls nicht die Abstände der einzelnen Ausprägungen mit gleichem Abstand definiert sind. Dennoch wird für die statistischen Auswertungen solcher Skalen oftmals eine Hilfsfunktion verwendet, die von einem als gleich angenommenen Abstand ausgeht, damit anschließend Rechenoperationen der kardinalen Skalen durchgeführt werden können. Durch eine derartige Hilfsfunktion lassen sich auch deutliche Präferenzen und Abneigungen gegenüber anderer Entscheidungsalternativen ausdrücken und es wird damit eine künstliche Kardinalität erzeugt (Breidert, 2006; Varian, 2003).

Eine Besonderheit bei der Berechnung der Entscheidungstabelle liegt bei den strukturellen Eigenschaften einer Komponente in Bezug auf das Softwareprodukt vor (vgl. Anhang A). In dieser Merkmalskategorie drückt die Kohäsion (E_1) die Modularität einer Komponente aus und beschreibt dessen Eigenständigkeit und innere Abgeschlossenheit. Diese verhält sich im Vergleich zu allen anderen Entscheidungskriterien invers auf den spezifischen Einfluss der entsprechenden Kategorie. Bei der Berechnung muss daher Wert, der für die Ausprägungen H, M und L erhalten wird, von 4 subtrahiert werden, um ein korrektes Ergebnis zu erhalten. Dies lässt sich beispielhaft an R_1 der Entscheidungstabelle der strukturellen Eigenschaften im Anhang A ablesen. Die Hilfsfunktion wird in diesem Fall um den beschriebenen Zwischenschritt ergänzt:

- Übersetzung der Ausprägungen in die definierten Werte (vgl. Tab. 3.7): *{3; 1}*
- **Korrektur des Wertes für Kohäsion:** *{(4-3); 1} = {1; 1}*
- Berechnung des Mittelwerts als $(1 + 1)/2 = 1$
- Ermittlung des zum Mittelwert gehörigen Medians aus Tab. 3.8: Geringer spezifischer Einfluss (LOW)

3.2.4.2. Aggregation der Entscheidungsgrößen und Entscheidungsfindung

Mit den bereits vorgestellten Entscheidungstabellen lassen sich nun die spezifischen Einflüsse (Teilnutzen) der drei Kriterienkategorien auf die Gesamtentscheidung bestimmen. Eine Entscheidungstabelle für die Outsourcingentschei-

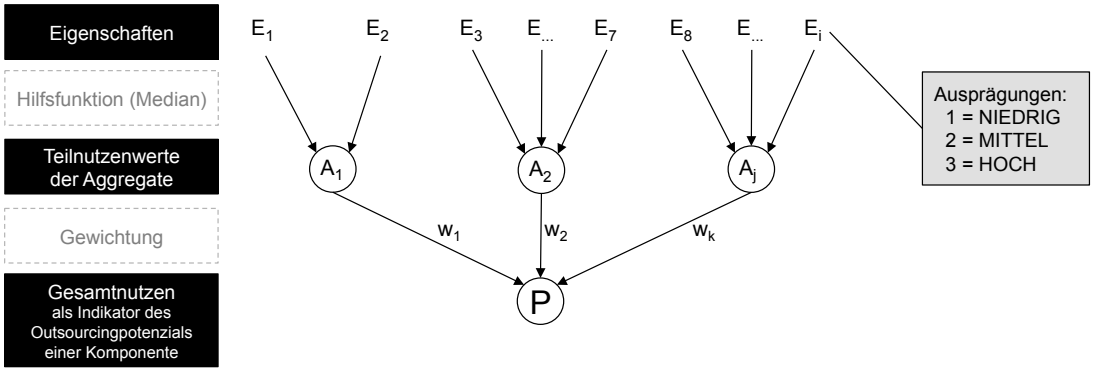

Abbildung 3.6.: Berechnung der Teilnutzenwerte und des Gesamtnutzens

dung lässt sich in diesem Fall analog zu den Tabellen zuvor aufstellen. Dies hätte allerdings den Nachteil, dass dann die Gewichtungsfunktion (wie in Kapitel 3.2.3.2 beschrieben) unberücksichtigt bliebe.

Die Amalgamation der Teilnutzenwerte einer Softwarekomponente folgt deshalb dem Schema wie in Abb. 3.6 dargestellt. Die aggregierten Werte (Teilnutzen) der Entscheidungskategorien (A_1 – A_3) werden jeweils durch das arithmetische Mittel und den zugehörigen Entscheidungstabellen ermittelt. Im nächsten Schritt fließen dann die vom Anwender festgelegten Gewichte (w_k) in die Bestimmung des Gesamtnutzens (P) als Indikator für das Outsourcingpotenzial einer Komponente mit ein. Dazu werden die vorgegebenen Gewichte standardisiert, so dass gilt $\sum_{n=1}^{k} w_n = 1$. Damit wird sichergestellt, dass sich der berechnete Wert des Gesamtnutzens (P) im Intervall von *[1; 3]* befindet. Die Berechnung von P erfolgt ähnlich wie bei den Entscheidungstabellen im vorherigen Kapitel, jedoch werden in diesem Fall die Ausprägungen der drei spezifischen Einflüsse jeweils mit ihrem zugehörigen standardisierten Gewicht multipliziert. Eine Division wird damit obsolet. Der Wert für P wird dann auf der Basis der Hilfsfunktion (vgl. Kap. 3.2.4.1) von kardinalen Werten in ordinale transformiert. Die entsprechenden Intervalle des errechneten Gesamtnutzens sind in Tab. 3.9 abgebildet und repräsentieren die Empfehlung für die Outsourcingentscheidung der betrachteten Komponente.

Ein Beispiel für die Berechnung des Outsourcingpotenzials (P):

- Berechneter Einfluss der strukturellen Eigenschaften einer Komponente in Bezug auf das Softwareprodukt: *HOCH*

- Berechneter Einfluss der Prozessmerkmale der Entwicklung der betrachteten Komponente: *NIEDRIG*
- Berechneter Einfluss der Wissensmerkmale der betrachteten Sofwarekomponente: *HOCH*
- Gewicht w_1 für strukturspezifischen Einfluss: *0,2*
- Gewicht w_2 für prozessspezifischen Einfluss: *0,7*
- Gewicht w_3 für wissensspezifischen Einfluss: *0,1*
- Berechnetes Outsourcingpotenzial P: $0,2 * 3 + 0,7 * 1 + 0,1 * 3 = 1,6$
- **Empfehlung: Auslagerung** (vgl. Tab. 3.9)

Sofern für die Berechnung vom Entscheider die gleichen Gewichte vorgegeben sind, so kann die zugehörige Entscheidungstabelle im Anhang D betrachtet werden. Variieren die Gewichte, dann muss die Tabelle neu berechnet werden. Diese Funktion übernimmt im weiteren Verlauf der Prototyp mit dem implementierten Entscheidungsmodell.

Das Schema zur Amalgamation der Teilnutzenwerte der Outsourcingentscheidung (vgl. Abb. 3.6) wurde so gewählt, dass es variabel ist. Es können zu jeder Kriterienkategorie weitere Kriterien hinzugefügt werden. Das Modell kann ebenso um weitere Kriterienkategorien und zugehörige Gewichte ergänzt werden. Dies ist im abgedruckten Schema durch die Indizes i, j und k gekennzeichnet. Die Berechnungsweisen lassen sich dementsprechend ebenfalls selbsterklärend um die hinzugefügten Elemente erweitern. Das Modell lässt sich auf diese Weise relativ einfach erweitern.

Tabelle 3.9.: Intervalle für das Outsourcingpotenzial der bewerteten Softwarekomponente

Intervall	**Empfehlung**
$1,00 - 1,66$	Auslagerung (hohes Outsourcingpotenzial)
$1,67 - 2,33$	Neutral (mittleres Outsourcingpotenzial)
$2,34 - 3,00$	Eigenentwicklung (niedriges Outsourcingpotenzial)

3.2.4.3. Anwendung des Entscheidungsmodells an einem Beispiel

Um die praktische Verwendung dieses Entscheidungsmodells zu verdeutlichen, wird nun ein Beispiel für ein Softwareentwicklungsprojekt vorgestellt, in dem entschieden werden muss, welche Komponenten selbst erstellt werden sollen und welche ausgelagert werden können.

Ausgangssituation Zunächst müssen einige Vorüberlegungen getätigt werden. In erster Linie müssen die Rahmenbedingungen des Projekts festgelegt werden. Dazu zählt die grundlegende Frage, ob das Softwaresystem entweder durch Eigenerstellung, durch Outsourcing oder unterstützt durch selektives Outsourcing entwickelt werden soll. Die Beantwortung dieser Frage wird meist von der Verfügbarkeit von Ressourcen beeinflusst. Weiterhin umfasst das strategische Management bei Auslagerungsentscheidungen die Auswahl eines oder mehrerer Zulieferer, die den Kriterien und der Strategie des Kunden entsprechen. Die Auswahl und die Evaluation von Outsourcingpartnern sind Bestandteile des Lieferantenmanagements (*vendor management*) und werden als bereits festgelegt angenommen. Das Konzept und die Technologie dieser Arbeit basieren auf einer selektiven Outsourcingstrategie, bei der einzelne Komponenten eines zu entwickelnden Gesamtsystems durch einen oder mehrere Vertragspartner vollständig umgesetzt und für die Integration bereitgestellt werden (vgl. Kapitel 2.4.1).

Eine zusätzliche Voraussetzung für die Anwendung des Entscheidungsmodells stellt die zugrundeliegende Softwareentwicklungsmethode und ihre Ausführung dar (vgl. Kapitel 2.3). Die Anwendung eines komponentenbasierten Prozessmodells (vgl. *RUP*) ist eine unverzichtbare Voraussetzung für die Entwicklung des Konzepts und der Technologie in dieser Arbeit. Damit wird implizit vorgegeben, dass das geplante Gesamtsystem als Komposition einzelner Softwarekomponenten gestaltet ist. Als Entwicklungsmethode können sowohl sequentielle (z.B. Wasserfallmodell) als auch iterative Ansätze (z.B. *Scrum*) zur Anwendung kommen (vgl. Kapitel 2.3.3).

In der operativen Phase des Projekts, bei der schließlich die Anwendung des Entscheidungsmodells zum Tragen kommt, wird eine initiale Liste von Anforderungen an das Softwaresystem zwingend vorausgesetzt. Aus ihr müssen die Abhängigkeiten der Anforderungen untereinander erkenntlich sein. Zusätzlich muss

Abbildung 3.7.: Beispielhafte Eingabeoberfläche einer Produktbewertung

eine Definition der geplanten Softwarearchitektur als Bestandteil des *RUP* vorgehalten werden. Auf dieser Grundlage wird nun ein Beispiel zur Bewertung von Softwarekomponenten beschrieben, um festzulegen, welche sich davon für die Auslagerung eignen und welche innerhalb des Unternehmens entwickelt werden sollten.

Beispielhaftes Softwaresystem Ein Produktbewertungssystem für ein Einkaufsportal dient als praktischer Anwendungsfall für die beispielhafte Anwendung des entwickelten Entscheidungsmodells. Ein solches System ist allgemein bekannt und wird verwendet, um die Kundenmeinungen über ein verkauftes Produkt einzuholen. Diese Informationen werden sowohl für eigene Auswertungen über das Käuferverhalten verwendet als auch für potenzielle Interessenten transparent gemacht (z.B. *Amazon* Kundenrezensionen[1]). Ein Eingabebildschirm für die Produktbewertung kann wie in Abbildung 3.7 dargestellt aussehen.

Für die Entwicklung eines solchen Systems sollen folgende Anforderungen erfüllt werden:

1. Ein Kunde muss für jedes seiner gekauften Produkte, die im Rechnungssystem verbucht sind, eine Bewertung inklusive Kurzbeschreibung abgeben können.
2. Ein Kunde muss alle Bewertungen anderer Kunden zu einem Produkt einsehen können.

1 https://www.amazon.de/gp/help/customer/display.html?nodeId=201470680 (abgerufen am 01.07.2015)

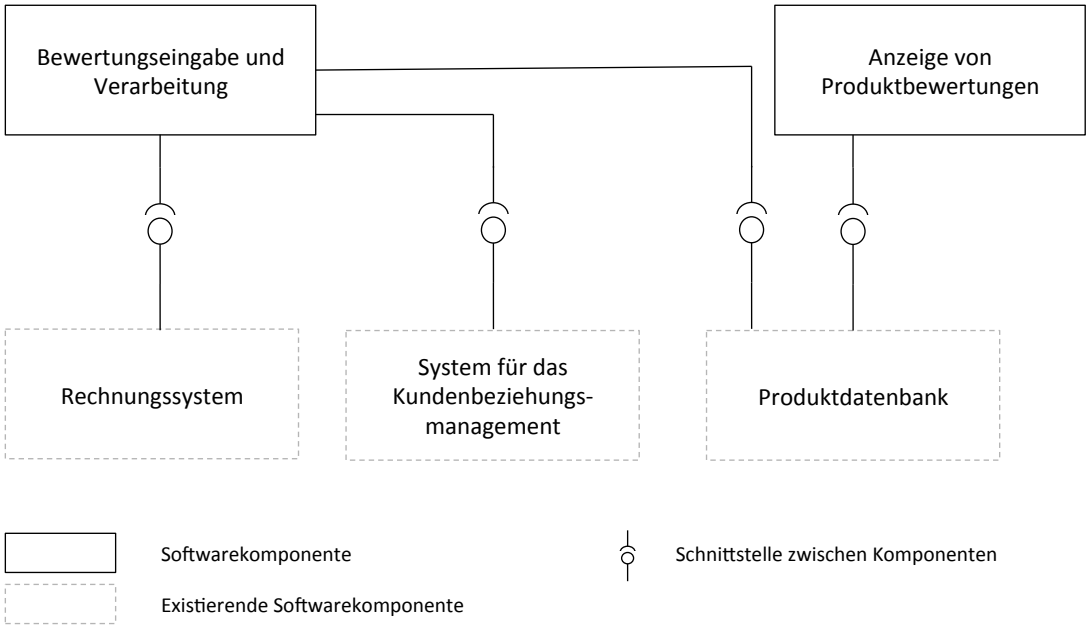

Abbildung 3.8.: Komponentenmodell des Beispiels

3. Alle Produktbewertungen eines Kunden müssen im System für das Kundenbeziehungsmanagement vermerkt werden.

Ein starker Zusammenhang zeichnet sich zwischen der ersten und der dritten Anforderung ab, da beide von der Abgabe einer Produktrezension handeln und in der zeitlichen Abfolge direkt aufeinander folgen. Eine Produktbewertung wird nämlich immer genau dann im Kundenbeziehungsmanagement vermerkt, wenn sie durch den Kunden eingegeben wurde. Aus dieser minimalen Anforderungsdefinition kann sich ein Komponentenmodell wie in Abbildung 3.8 ergeben.

Entscheidungsfindung Die Anwendung des Entscheidungsmodells macht das nachfolgende Vorgehen erforderlich, um eine Empfehlung über die Eigenentwicklung oder Auslagerung für jede Softwarekomponente zu erhalten. Es erfolgt eine komponentenweise Bewertung des Systems.

Komponente „Anzeige von Produktbewertungen“

1. Subjektive Einschätzung der Entscheidungsgrößen E_1 bis E_{12}
   a) Die Modularität (E_1) wird für diese Komponente als *HOCH* eingestuft, da es sich um eine größtenteils abgeschlossene Funktionalität

handelt, die sämtliche Funktionen innerhalb der Komponente (z.B. für das Anzeigen von Information, zur Sortierung usw.) bereitstellen und ausführen kann.

b) Die Kopplung (E_2) wird in diesem Fall mit *NIEDRIG* bewertet, weil lediglich eine Schnittstelle zur Produktdatenbank bereitgestellt werden muss (vgl. Abbildung 3.8), um die erforderlichen Bewertungsinformationen abzurufen.

c) Analog zu den strukturellen Merkmalen müssen alle anderen Entscheidungsgrößen ebenfalls bewertet werden. Zur Verständlichkeit des Entscheidungsmodells ist das exemplarische Vorgehen anhand einer Kategorie ausreichend.

2. Berechnung der Aggregatwerte (Spezifitäten)

a) Für die Bestimmung des Aggregatwerts der strukturellen Eigenschaften bedient man sich der entsprechenden Entscheidungstabelle. In Tabelle 3.6 lässt sich so der Wert ablesen, den das Aggregat bei hoher Modularität und geringer Kopplung einnimmt. Diese Auswahl entspricht dem Ausprägungsvektor R_1. Damit ergibt sich aus der Entscheidungstabelle für die Struktur die Lösungsalternative L_3 und bedeutet, dass dieser Komponente eine geringe strukturelle Spezifität zugeordnet wird.

b) Dieses Vorgehen muss gleichermaßen für die prozessspezifischen Merkmale in der Entscheidungstabelle im Anhang B durchgeführt werden.

c) Gleiches gilt auch für die wissensspezifischen Merkmale, deren Entscheidungstabelle sich in Anhang C befindet.

3. Festlegung der Gewichte (w_1 bis w_k) der einzelnen Kategorien (vgl. Abbildung 3.6). Es bietet sich hierbei an, eine prozentuale Verteilung vorzunehmen, deren Summe 100% ergibt, um sich den Schritt der Normierung zu sparen. Für dieses Beispiel wird folgende Verteilung festgelegt:

a) $w_1 = 40\%$ (Strukturelle Eigenschaften)

b) $w_2 = 30\%$ (Prozessspezifische Merkmale)

c) $w_3 = 30\%$ (Wissensspezifische Merkmale)

4. Bestimmung des Outsourcingpotenzials durch Berechnung

   a) In die Berechnung fließen alle bereits bestimmten Aggregatwerte unter Berücksichtigung der zugehörigen Gewichte mit ein (vgl. Kapitel 3.2.4.2). Das Aggregat der strukturellen Eigenschaften wurde in Schritt 2a bereits bestimmt. Für die Aggregate der prozess- und wissensspezifischen Merkmale werden die Werte in diesem Beispiel geschätzt. Damit ergeben sich folgende Wertzuweisungen für die Berechnung: $A_1 = 1$, $A_2 = 2$, $A_2 = 1$.

   b) Die Berechnung ergibt dann folgendes normiertes Ergebnis für das Outsourcingpotenzial: $\sum_{n=1}^{3} w_n * A_n = 0,4 * 1 + 0,3 * 2 + 0,3 * 1 = 1,3$

   c) Das Ergebnis wird dann in die in Tabelle 3.9 vorgegebenen Intervalle eingeordnet und das Outsourcingpotenzial der Komponente bestimmt. Für die Komponente „Anzeige von Produktbewertungen“ liegt der Ergebniswert im Intervall von *1,00 – 1,66* und führt damit zu der Empfehlung die Komponente auszulagern.

Komponente „Bewertungseingabe und Verarbeitung“ Die Vorgehensweise zur Berechnung des Outsourcingpotenzials dieser Komponente ist identisch mit der aus dem Abschnitt zuvor. Lediglich die subjektiven Bewertungen müssen auf den Inhalt und die Beschaffenheit der Komponente angepasst werden. Die Berechnungen selbst erfolgen ebenfalls analog zu den Vorgaben des Entscheidungsmodells. Für die Bewertung der Strukturspezifität werden bei dieser Komponente die Ausprägungen *HOCH* für die Kopplung und *MITTEL* für die Kohäsion entschieden. Der Wert für die Kopplung ergibt sich aus dem Zusammenspiel der Komponenten mit vielen bestehenden Komponenten (z.B. Rechnungssystem, System für Kundenbeziehungsmanagement und Produktdatenbank), die eine Interaktion für die richtige Funktionsweise notwendig machen (siehe Abbildung 3.8). Weiterhin bedeutet dies auch, dass die Komponente viele Funktionsaufrufe hin zu den verknüpften Komponenten durchführen muss, um sich so die erforderlichen Daten zu beschaffen. Nichtsdestotrotz werden gleichermaßen Funktionen aus der Komponente selbst aufgerufen, um beispielsweise die fehlerfreie Dateneingabe zu überprüfen. Aus diesem Grund wird dieser Komponente eine mittlere Kohäsion zugesprochen. Aus der Entscheidungstabelle 3.6 lässt sich

daher für den Ausprägungsvektor R_6 die Lösungsalternative L_1 ablesen und der Struktur der Komponente eine hohe Spezifität zuweisen.

Auf der aggregierten Ebene (siehe Abbildung 3.6) wird für dieses Beispiel, neben der bereits bestimmten hohen Strukturspezifität (A_1), eine ebenfalls hohe Prozessspezifität (A_2) geschätzt. Diese lässt sich dadurch erklären, dass für die Einbettung dieser stark vernetzten Komponente erhebliche Erfahrung hinsichtlich des Entwicklungsprozesses benötigt wird, um sich mit den jeweiligen Entwicklern über die Schnittstellen abstimmen zu können, und der Integrationsaufwand erheblich ist. Hinsichtlich der Wissensspezifität (A_3) wird die Ausprägung des Aggregats als *MITTEL* eingeschätzt, weil für die richtige Implementierung dieser Komponente spezifisches Wissen über den Ablauf des Verkaufsprozesses benötigt wird oder der Typ und das Format der Daten für das Kundenbeziehungsmanagement bekannt sein muss. Zu schützendes geistiges Eigentum ist dabei allerdings irrelevant.

Abschließend lässt sich das Outsourcingpotenzial in Analogie zur vorherigen Komponente durch folgende Formel berechnen: $\sum_{n=1}^{3} w_n * A_n = 0,4 * 3 + 0,3 * 3 + 0,3 * 2 = 2,7$. Für die Komponente „Bewertungseingabe und Verarbeitung“ liegt der Ergebniswert im Intervall von *2,34 – 3,00* und führt damit zu der Empfehlung die Komponente selbst zu entwickeln, da eine hohe Spezifität gegen das Outsourcing sprechen.

3.3. Implementierung des Modells auf einem Tablet-PC

In diesem Kapitel wird die prototypische Umsetzung des konzipierten Entscheidungsmodells (vgl. Kapitel 3.2) in ein mobiles EUS beschrieben. Das Ziel dabei ist die sinngemäße Implementierung einer Entscheidungstechnik, die das im vorherigen Kapitel entwickelte Entscheidungsmodell in Form einer Anwendungssoftware umsetzt und dadurch die Anforderungen an das Gesamtkonzept dieser Arbeit erfüllt (vgl. Kapitel 3.1). Deshalb wird im weiteren Verlauf zusätzlich auf die guten Integrationsmöglichkeiten und -realisationen des entwickelten Prototyps eingegangen. Um den Stand der Technik (vgl. Kapitel 2.2.2) und die beruflichen Anforderungen der Entscheider abzubilden, wird ein mobiles EUS implementiert, das als Anwendung auf einem Tablet-PC lauffähig ist. Der dafür erforderliche

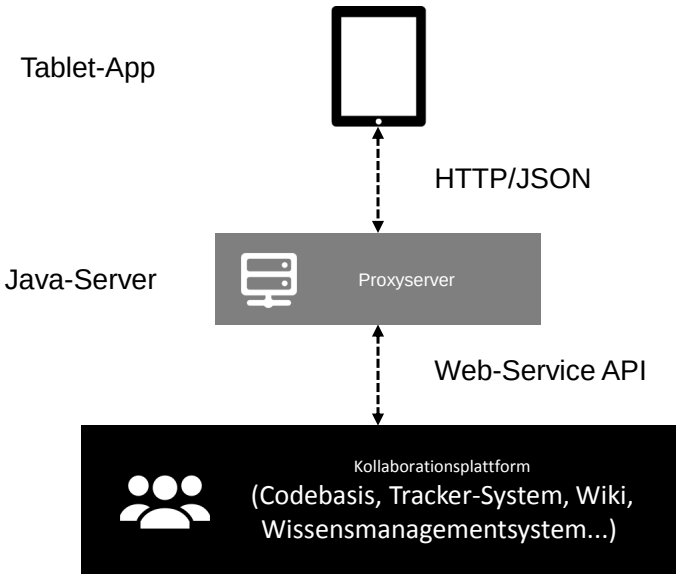

Abbildung 3.9.: Gesamtarchitektur

technologische Aufbau eines solchen EUS sowie dessen Funktionsweise werden in diesem Kapitel detailliert erläutert.

3.3.1. Technologischer Aufbau und Entwicklungsumgebung

Zunächst wird die Architektur des implementierten Ansatzes zur Entscheidungsfindung präsentiert und dessen Bestandteile kurz erläutert. Anschließend wird die Auswahl der zugrundeliegenden Technologie für die einzelnen Architekturbestandteile vorgestellt und begründet. Abschließend werden die Details der Entwicklungsumgebung beschrieben, die für die Entwicklung des Prototyps verwendet wurde. So soll die Nachvollziehbarkeit der Entwicklung dieses mobilen EUS gewährleistet werden.

3.3.1.1. Gesamtarchitektur

Die Architektur des Lösungsansatzes dieser Arbeit hat die Aufgabe, die im Vorfeld definierten Anforderungen in einem integrierten Softwaresystem zu repräsentieren. Aus diesem Grund gliedert sich die Gesamtarchitektur dieser Lösung, wie in Abbildung 3.9 dargestellt, in drei Schichten.

Die oberste Schicht der Architektur stellt die *mobile Applikation* für die Durchführung der Entscheidungsunterstützung und damit die Hauptanwendung der Lösungstechnologie dar. Sie ist eine Tabletanwendung auf Basis des iOS-Betriebssystems[2] und beinhaltet die Funktionalitäten eines fortschrittlichen EUS zur Bewertung aller Komponenten einer Softwarelösung. Dies wird durch die Umsetzung des verfeinerten Entscheidungsmodells in der App sowie durch Anreicherung mit zusätzlichen prozessoptimierenden Funktionen erreicht. Als Name für die App wurde *SmartSourcer* gewählt, um die unterstützende Wirkung beim Treffen von Outsourcingentscheidungen auszudrücken.

Die unterste Schicht des Architekturmodells repräsentiert die Anbindung des Lösungsansatzes an existierende *Kollaborationsplattformen*, die in Softwareunternehmen mit verteilter Entwicklung für diese Arbeit vorausgesetzt werden. Die dazwischenliegende Architekturschicht nimmt die Funktion eines *Proxyservers*[3] im konzipierten System ein und ist in erster Linie für die Kommunikation und den Datenaustausch zwischen der mobilen App und der Kollaborationsplattform des Unternehmens zuständig, bei dem das EUS zum Einsatz kommt. Weiterhin ermöglicht der Proxyserver die flexible Erweiterung der Entscheidungstechnik um weitere Kollaborationsplattformen, welche die notwendigen Schnittstellen anbieten, ohne dabei die App selbst anpassen zu müssen.

Damit sich die vorgeschlagene Lösungstechnologie für den praktischen Einsatz in einem Softwareunternehmen verwenden lässt, müssen firmenseitig die nachfolgenden Voraussetzungen erfüllt sein. In erster Linie muss eine Kollaborationsplattform zur verteilten Entwicklung im Einsatz sein und die frühen Phasen zur Definition der Anforderungen, der Architektur sowie des Designs abdecken. Somit wird sichergestellt, dass die notwendige Datengrundlage (Menge der definierten Softwarekomponenten, deren Verknüpfungen zu den Softwareanforderungen und die Verknüpfungen zwischen den einzelnen Anforderungen) für die Entscheidungsunterstützungstechnologie digital zur Verfügung stehen. Aufgrund der verwendeten Programmiersprache und des iOS-Betriebssystems sind für den Einsatz des entwickelten Prototyps mobile Tablet-PCs der Marke *Apple* erforderlich.

2 Betriebssystem der Firma *Apple* für mobile Endgeräte wie *iPhone* oder *iPad*.

3 Ein Proxyserver übernimmt die Aufgabe eines Vermittlers im Computernetzwerk und ist in diesem Fall für die richtige Weiterleitung der Daten an die zu verwendende Kollaborationsplattform verantwortlich.

3.3.1.2. Technologieauswahl

Für die Umsetzung von *SmartSourcer* wurde auf *ObjectiveC*[4] zurückgegriffen, eine Programmiersprache zur Entwicklung nativer Apps auf der Basis von *Apple iOS*. Native Apps sind Anwendungen, die die Eigenschaften und Funktionen eines zugrundeliegenden Betriebssystems optimal ausnutzen können, da sie in der entsprechenden Programmiersprache entwickelt wurden und deshalb der Maschinencode für die darunterliegende Hardware optimiert ist. Eine alternative Technologie zur Umsetzung von *SmartSourcer* stellt das Betriebssystem *Android* der Firma *Google* dar. Die Erstellung ebenfalls nativer Apps für dieses Betriebssystem erfolgt in der Programmiersprache *Java*[5]. Die beiden vorgestellten Technologien teilen sich die größten Marktanteile für Betriebssysteme auf Tabletgeräten. Aus diesem Grund kamen sie in die engere Auswahl der Zieltechnologie. Die Entwicklung einer hybriden App wurde wegen geringerer Reaktionszeiten bei Nutzereingaben sowie einer oftmals dem Bedienkonzept des jeweiligen Systems widersprechenden Navigationsführung abgelehnt. Hybride Apps werden in einer gemeinsamen Programmiersprache entwickelt. Für sie wird auf jedem Betriebssystem ein Interpreter zur Übersetzung der gewünschten Funktionen in die native Sprache des jeweiligen Systems bereitgestellt, der die genannten Nachteile mit sich bringt. Die finale Wahl zur Umsetzung fiel auf die Technologie von *Apple*, weil die Entwicklungsumgebung zahlreiche Hilfsmittel und grafische Eingabeoberflächen (z.B. *Storyboard*, *Interface Builder* sowie einen *iOS*-Simulator) anbietet, die eine schnelle Umsetzung der gewünschten Software ermöglichen. Eine spätere Portierung dieser Technologie auf andere mobile Plattformen stellt kein Problem dar, weil keine spezifischen Funktionalitäten verwendet wurden, die ausschließlich auf Geräten von *Apple* funktionsfähig sind. Außerdem wird für die Entwicklung von Anwendungen auf der Basis von *Apple*-Technologie eine umfangreiche IDE (integrierte Entwicklungsumgebung) und ein mächtiges SDK (*Software Development Kit*) zur Verfügung gestellt (vgl. Abbildung 3.12).

4 Eine Übersicht über die Programmiersprache *ObjectiveC* ist unter https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html verfügbar (abgerufen am 01.12.2014)

5 Die Programmiersprache *Java* ist eine weitverbreitete Programmiersprache und wird neben mobilen Anwendungen auch für Serveranwendungen verwendet. Eine Referenz zur Spezifikation der *Standard Edition 7* ist unter https://docs.oracle.com/javase/7/docs/api/overview-summary.html verfügbar (abgerufen am 01.12.2014)

Für die Architekturschicht der *Kollaborationsplattformen* wird in dieser Arbeit eine umfassende Plattform zur Kollaboration und für das Lebenszyklusmanagement von Softwareprojekten verwendet. Es handelt sich dabei um das Produkt *CodeBeamer*[6] der Firma *Intland* und ist ein in *Java* programmierter Webserver, der auf dem firmeneigenen Server eingerichtet und über das Internet abgerufen werden kann. Der Funktionsumfang der Plattform umfasst alle in Tabelle 2.3 aufgezählten Funktionalitäten und eignet sich daher für den Einsatz in Kombination mit *SmartSourcer*. Weitere Produkte zur kollaborativen Durchführung von Softwareprojekten sind:

- *CollabNet TeamForge*[7]
- *Google Code*[8]
- *JavaForge CodeBeamer* (Onlineversion von *CodeBeamer*)[9]
- *Microsoft Team Foundation Server*[10]
- *Rally Platform*[11]

Der Funktionsumfang der aufgezählten Plattformen bezüglich der benötigten Projektdaten zur Durchführung des Bewertungsprozesses für alle Softwarekomponenten durch *SmartSourcer* ist bei allen gleichermaßen ausgeprägt. Die Selektion von *CodeBeamer* als Kollaborationsplattform zur Anbindung des Prototyps dieser Arbeit liegt in vorangegangen Forschungsarbeiten auf dem Gebiet der sozialen Netzwerkanalyse und der damit verbundenen Programmierung von Java-Werkzeugen zum Aufruf des Web-Services der *CodeBeamer*-Plattform (Kramer et al., 2009) begründet. Die prozess- und produktorientierte Herangehensweise bei der Nutzung von *CodeBeamer* ermöglicht den differenzierten Zugriff auf Softwarekomponenten, Softwareanforderungen, deren strukturelle Abhängigkeiten, oder auch die Abhängigkeiten der Komponenten von Anforderungen. Diese Informationen werden für die Entscheidungsunterstützung durch *SmartSourcer*

6 http://intland.com/ (abgerufen am 01.12.2014)
7 http://www.collab.net/products/teamforge (abgerufen am 01.12.2014)
8 http://code.google.com (abgerufen am 01.12.2014)
9 http://www.javaforge.com (abgerufen am 01.12.2014)
10 http://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx (abgerufen am 01.12.2014)
11 https://www.rallydev.com/ra/platform-products/rally-platform (abgerufen am 01.12.2014)

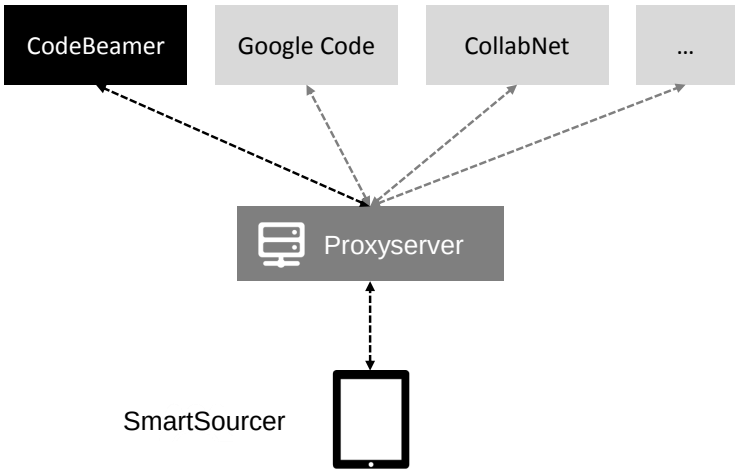

Abbildung 3.10.: Modularer Aufbau der Architektur

benötigt und über den Proxyserver ausgelesen. Die Anbindungsmöglichkeiten des Proxyservers sind in Abbildung 3.10 abgebildet.

Bei der Auswahl der *Schnittstellentechnologie* für die Kommunikation der App mit dem Proxyserver sowie des Proxyservers mit der Kollaborationsplattform wurde auf zwei unterschiedliche Ansätze zurückgegriffen, die aber beide auf dem Internetprotokoll[12] basieren und damit Verbindungen über das Internet untereinander herstellen können. Die Datenkommunikation zwischen der Kollaborationsplattform und dem Proxyserver finden auf Basis einer Web-*API* (webbasierte Programmierschnittstelle) statt. Der Datenaustausch zwischen der Tabletanwendung und dem Proxyserver findet über einen vom Proxyserver bereitgestellten Web-Service statt, welcher mit *SmartSourcer* Daten über das *JSON*-Format austauscht.

Die Web-*API* zwischen Proxyserver und *CodeBeamer* ist eine umfangreiche Schnittstelle zum Auslesen und Manipulieren von Daten, die im *CodeBeamer* gespeichert sind. Zur Datenübermittlung wird das binäre Web-Service-Protokoll *Hessian*[13] benutzt. Das Protokoll eignet sich wegen seines überschaubaren Funktionsumfangs und seiner Eignung für die Übertragung großer Datenmengen und Dateien optimal für den Zugriff auf die Kollaborationsplattform. Ein weiterer Grund für die Auswahl ist die bereits verfügbare Implementation des Web-Ser-

12 *Hypertext Transfer Protocol (http)*
13 Protokollreferenz verfügbar unter http://hessian.caucho.com (abgerufen am 01.12.2014)

```
[
    [
        "114",
        "eCRM",
        "CRM-Software als Beispielprojekt für SmartSourcer-App"
    ],
    [
        "117",
        "eCRM Alpha",
        "Softwareentwicklungsprojekt für das advanced ERP System Alpha"
    ],
    [
        "104",
        "HWS2011 TP SMW Collab Tools",
        "Enhancing the Semantic MediaWiki with\\u0001 Modules for Traceability\
            \u0001 Visualization and Requirements Prioritization"
    ],
    [
        "102",
        "Outsourcing Decision TP",
        "Term project: Decision Making in Software Engineering Outsourcing"
    ],
    [
        "115",
        "SmartSourcer",
        "Software-Prototyp für die Entscheidungsunterstützung beim Outsourcing von
            Softwareentwicklungsaufgaben."
    ]
]
```


Abbildung 3.11.: Datenübertragung eines *JSON*-Objekts

vices in der Programmiersprache Java, die ebenfalls für die Implementierung des Proxyservers verwendet wurde (vgl. Anhang F).

Das *JSON*-Format, das für den webbasierten Datenaustausch zwischen App und Proxyserver verwendet wird, wurde ausgewählt, da es den einfachen und schnellen Austausch von Informationen zwischen mobilen Apps und ihren Web-Services ermöglicht. Für die Datenübertragung müssen keine speziellen Datentypen definiert und mitübertragen werden, sondern die Informationen werden mit Hilfe einer leichten Klammersyntax transportiert. Damit entfällt die Übermittlung zusätzlicher Informationen und die zu übertragende Datenmenge wird dementsprechend reduziert. Dadurch eignet sich das Format optimal für den Datenaustausch mit mobilen Applikationen, bei denen die Datenverbindung nicht immer zuverlässig verfügbar und oftmals durch ihre Bandbreite beschränkt ist. Ein Beispiel für übertragene Daten im *JSON*-Format ist in Abbildung 3.11 dargestellt und zeigt das Ergebnis einer Anfrage von *SmartSourcer* an den Proxyserver nach allen im *CodeBeamer* verfügbaren Softwareprojekten.

3.3.1.3. Entwicklungsumgebung

Für die Entwicklung von *SmartSourcer* wurde die IDE *XCode* in der Version 6.0.1 von *Apple* eingesetzt (vgl. Abbildung 3.12). Diese integrierte Entwicklungsumgebung vereint alle notwendigen Werkzeuge zur Erstellung von Anwendungen für das Betriebssystem *iOS* (optimiert für Version 7). Dazu gehören in erster Linie der Editor zum Erstellen und Editieren von Softwarecode, der gleichzeitig eine automatische Syntaxprüfung für den geschriebenen Code enthält. Weiterhin werden von *XCode* grafische Werkzeuge zur Bearbeitung des *Storyboards* (logische Abfolge der Anzeigeschichten einer App), zum grafischen Entwurf von Eingabebildschirmen sowie zur Erstellung des Datenmodells bereitgestellt. Ein Auszug aus dem *Storyboard* von *SmartSourcer* ist in Abbildung 3.13 dargestellt und zeigt auf der linken Seite die Controller des Quellcodes, die auf der rechten Seite grafisch miteinander verknüpft sind. In Abbildung 3.14 ist das Datenmodell, das bei der Entwicklung der App zugrunde gelegt wurde, dargestellt und zeigt die lokale Datenhaltung auf dem mobilen Endgerät auf. Dabei werden einerseits alle zu einem Projekt gehörenden Metadaten, die für die Anwendung von *SmartSourcer* erforderlich sind, abgespeichert. Andererseits werden alle Softwarekomponenten und Anforderungen sowie deren Beziehungen untereinander und ihre Zugehörigkeit zum Projekt ebenfalls im Datenmodell hinterlegt. Schließlich werden in den Datenklassen *SuperCharacteristic* und *Characteristic* die Aggregationsebenen und Merkmale des Entscheidungsmodells abgebildet. Die zugehörigen Objekte mit dem Zusatz *Available* repräsentieren die Merkmale und Aggregationsebenen von Komponenten, die bereits zur Installation von *SmartSourcer* vorgegeben werden und dann individuell angepasst und erweitert werden können. Weiterhin lassen sich in diese IDE externe Dienste zur Versionierung von Code oder für den kontinuierlichen Auslieferungsprozess einbinden. Für die Entwicklung dieser Arbeit wurde der entworfene Quellcode im über das Internet zugänglichen Versionierungssystem *GitHub*[14] abgelegt und ist dort verfügbar. Die zahlreichen Integrationsmöglichkeiten und Eingabehilfen von *XCode* erleichtern die Entwicklung einer App von der Designphase bis hin zur Auslieferung auf die Geräte.

Die Benutzung von *XCode* erfordert als weitere Maßnahme für eine optimale Ausnutzung der Funktionalitäten der IDE die Verwendung von typischen Soft-

14 http://www.github.com (abgerufen am 01.12.2014)

waremustern (*software patterns*), um von der Entwicklungsumgebung richtig erkannt und somit unterstützt zu werden. Bei der Implementierung von *SmartSourcer* kam daher das *Model-View-Controller-Pattern* zum Einsatz, dessen Anforderungen während der Implementierung kontinuierlich umgesetzt wurden. Das Programmiermuster fordert eine strikte Trennung zwischen Datenhaltung, Präsentationsschicht und Programmablauf der Applikation (Leff und Rayfield, 2001). Datenaustausch zwischen den einzelnen Schichten darf dabei nur auf streng definierten Schnittstellen erfolgen und bietet daher vor allem bei mobilen Anwendungen den Vorteil, dass mit dem gleichen Datenmodell unterschiedliche Präsentationsebenen (z.B. für die Displaygröße eines Smartphones oder Tablets oder für die horizontale und vertikale Darstellung von Benutzerschnittstellen) generiert werden können, ohne dabei Inkonsistenzen im Datenmodell befürchten zu müssen. Die Umsetzung des *Model-View-Controller-Pattern* ist als Codeausschnitt im Anhang E dargestellt. Dieser zeigt, dass das Datenmodell der Anwendung zu Beginn geladen wird, auf welches im weiteren Verlauf des Programmablaufs und für die Darstellung von Informationen zurückgegriffen wird (vgl. Anhang E).

Für die Entwicklung des Proxyservers wurde eine weitere IDE verwendet, da sich dessen Entwicklung in der Programmiersprache *Java* besser für den universellen Einsatz mit weiteren Kollaborationsplattformen (z.B. alle bei der Technologieauswahl genannten) eignet. Aus diesem Grund wurde auf *Eclipse IDE for Java EE Developers*[15] in der Version 4.3 SR2 (Kepler) zurückgegriffen, um den Web-Service für *Apache Tomcat*[16] zu entwickeln. Für die prototypische Umsetzung des EUS wurde ein *Tomcat*-Server in der Version 7.0.41 verwendet, um den Proxyserver zu installieren, der die Kommunikation mit dem *CodeBeamer* (hier in Version 5.6.0-RC3 verwendet) herstellt. Die App tauscht über den Proxy mit dem *CodeBeamer* die Komponentendaten, einen Managementreport und die finalen Bewertungen der Komponenten wie in Abbildung 3.15 abgebildet aus.

15 http://www.eclipse.org/downloads/packages/release/Kepler/SR2 (abgerufen am 01.12.2014)

16 *Tomcat* ist ein Webserver zur Ausführung von Web-Services in Java. Unter http://tomcat.apache.org sind die Dokumentation und der Programmcode abrufbar (aufgerufen am 01.12.2014)

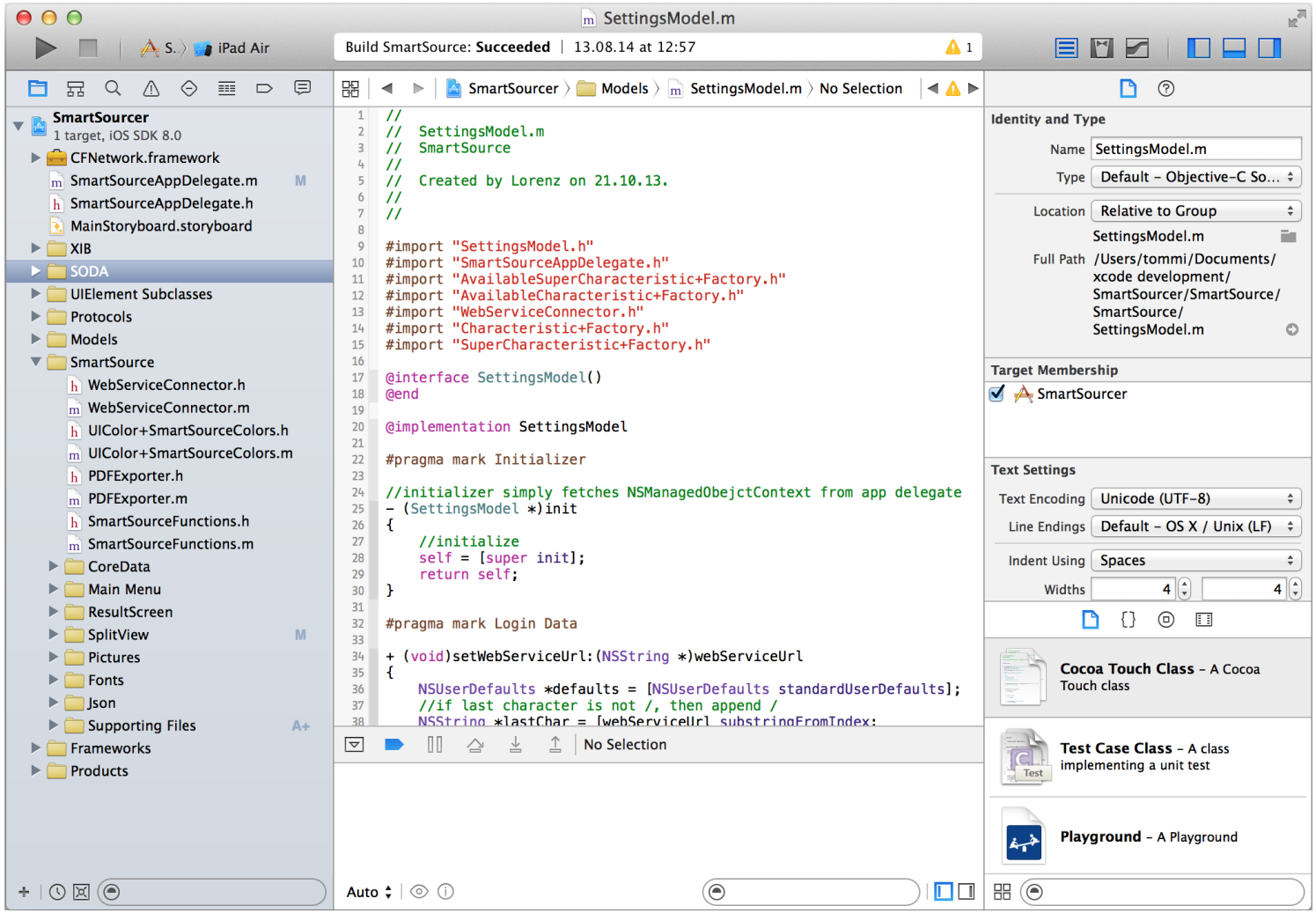

Abbildung 3.12.: Integrierte Entwicklungsumgebung *XCode*

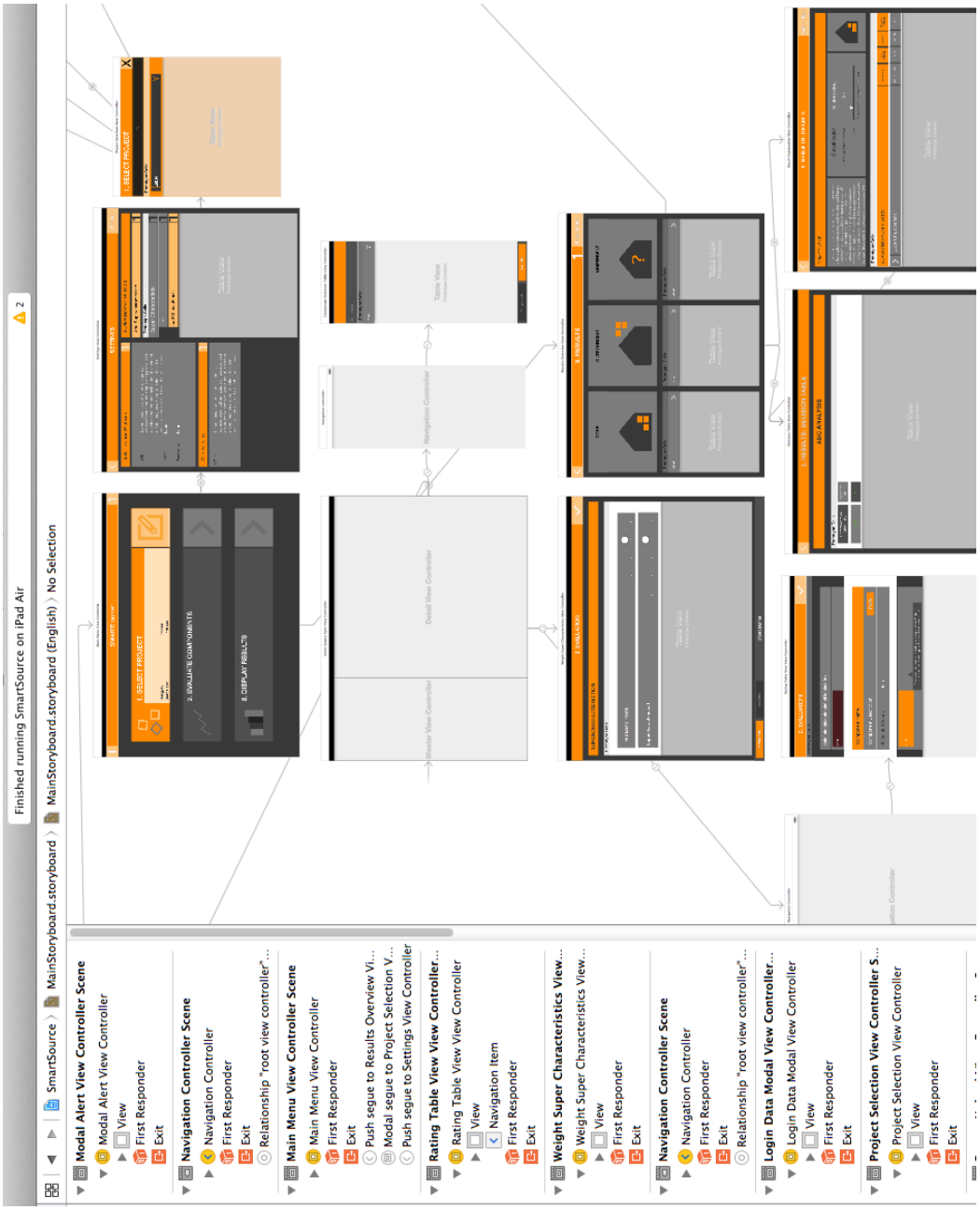

Abbildung 3.13.: *Storyboard* von *SmartSourcer*

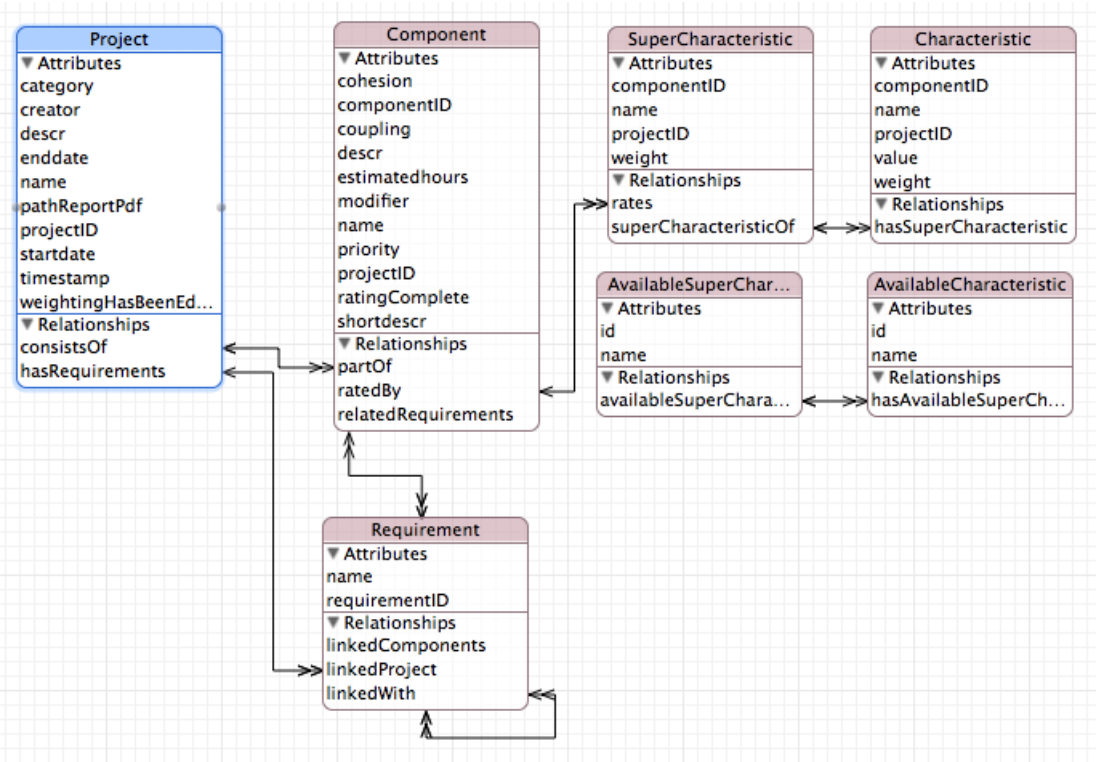

Abbildung 3.14.: Datenmodell von *SmartSourcer*

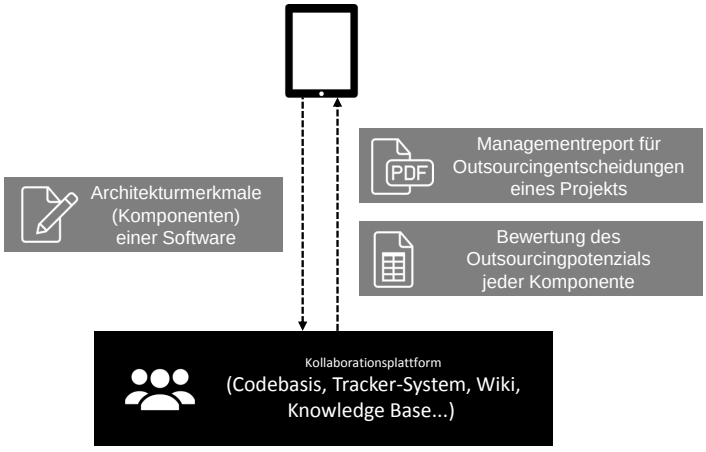

Abbildung 3.15.: Datenaustausch zwischen *SmartSourcer* und *CodeBeamer*

3.3.2. Funktionsbeschreibung

Die Funktionalität der mobilen App *SmartSourcer* steht als Umsetzung der konzeptuellen Definitionen des Entscheidungsmodells im Mittelpunkt dieser Arbeit

und repräsentiert damit den Kern des mobilen EUS für das Outsourcing von Softwarekomponenten. *SmartSourcer* ermöglicht dem entscheidungsbefugten Manager die Bewertung des Outsourcingpotenzials aller Softwarekomponenten eines Projekts in einer auf seinem Tablet-PC installierten App. Diese lässt sich überall dort einsetzen, wo sein *iPad* Zugriff auf die Kollaborationsplattform mit den Projektdaten hat. Der Umfang der App lässt sich in drei Hauptfunktionen aufteilen und beinhaltet die Projektselektion, den Bewertungsvorgang aller zugehörigen Softwarekomponenten sowie die Auswertungsfunktion zur Darstellung und Weiterverarbeitung der Entscheidungsergebnisse.

3.3.2.1. Projektauswahl

Nach dem Aufruf der App kann der Anwender zunächst ein Projekt aus der Kollaborationsplattform auswählen (*Select Project*), für dessen Komponenten noch keine Outsourcinganalyse durchgeführt wurde, oder auf bereits geladene Projekte zurückgreifen, bei denen die Komponentenbewertung bereits abgeschlossen wurde oder gerade im Verlauf ist. Der Startbildschirm sowie das zentrale Navigationsmenü sind in Abbildung 3.16 abgebildet.

Zunächst gibt es die Möglichkeit über den Einstellungsbildschirm der App die Verbindungsadresse und die persönlichen Zugangsdaten zum *CodeBeamer* einzugeben, damit der Proxyserver die Serveranfragen an die Kollaborationsplattform entsprechend weiterleiten kann. Gleichzeitig hat der Anwender in diesem Menü die Möglichkeit, die aus dem Entscheidungsmodell bekannten Entscheidungsgrößen und deren Kategorien (vgl. Abbildung 3.1) zu editieren, zu löschen oder sogar neu anzulegen. Die jeweils aktuelle Konfiguration der verwendeten Entscheidungsgrößen (*characteristics*) und Kategorien (*supercharacteristics*) ist je nach Wahl des persönlich bevorzugten Entscheidungsmodells in diesem Bildschirm strukturiert und übersichtlich dargestellt. Dies ist in Abbildung 3.17 zu erkennen. Hier kann der Anwender sein persönlich bevorzugtes Entscheidungsmodell konfigurieren, sofern ihm die vorgegeben Entscheidungsgrößen zu umfangreich oder nicht ausreichend sind. Beim Hinzufügen neuer Merkmale ist darauf zu achten, dass diese so gewählt werden, dass eine höhere Ausprägung davon zu mehr Spezifität in der jeweiligen Kategorie führt. Nur so kann die Richtigkeit der Berechnung sichergestellt werden, damit Softwarekomponenten mit hoher Gesamtspezifität für

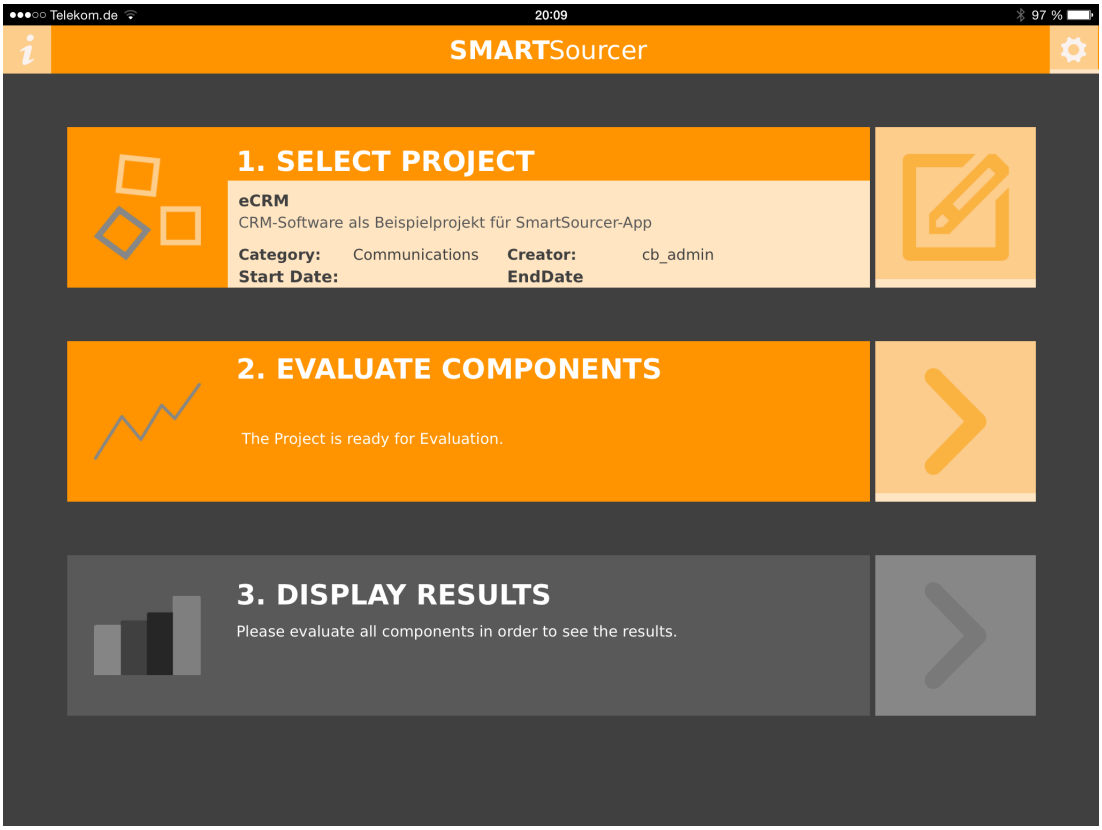

Abbildung 3.16.: Startbildschirm und Hauptmenü von *SmartSourcer*

die Eigenentwicklung und folglich diejenigen mit geringer Gesamtspezifität für das Outsourcing empfohlen werden.

3.3.2.2. Bewertungsvorgang

Im ersten Schritt wurden bereits die Voraussetzungen zum Einlesen und Bewerten der Softwarekomponenten getroffen. Über den Menüpunkt *Evaluate Components* wird der Anwender auf den Bildschirm zur Durchführung der Komponentenbewertung weitergeleitet (vgl. Abbildung 3.18). In diesem Bereich müssen nun alle Softwarekomponenten, die zum Projekt gehören und aus dem *CodeBeamer* ausgelesen wurden, einzeln bezüglich der konfigurierten Merkmale vom Entscheidungsverantwortlichen eingeschätzt werden. Zur besseren Orientierung werden zusätzlich die jeweilige Beschreibung und weitere Metainformationen einer Komponente mit eingeblendet, so dass hier kein Medienbruch stattfindet, wenn dem Benutzer nicht durch den Namen der Komponente ersichtlich wird, welche Funktionalität

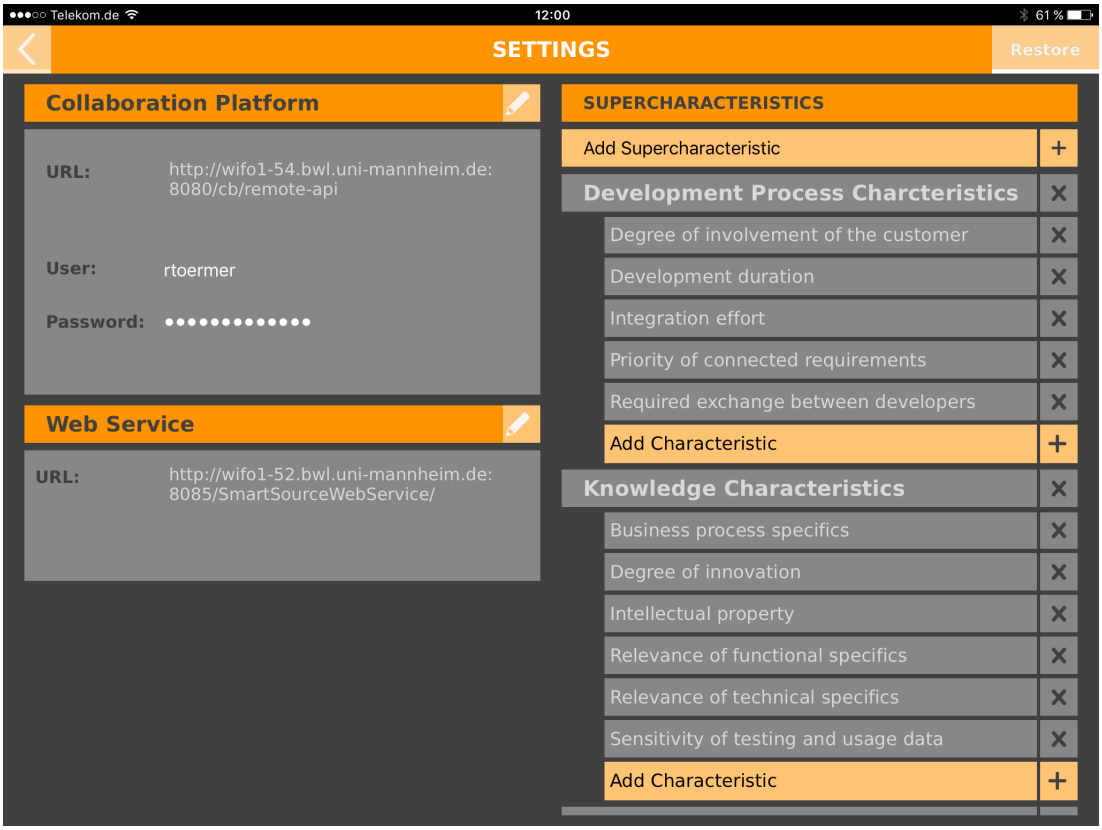

Abbildung 3.17.: Screenshot der Einstellungen

diese beinhaltet und wer dafür verantwortlich ist. Die Bewertung der einzelnen Merkmale folgt den bereits im Konzept des Entscheidungsmodells vorgestellten Ausprägungen HOCH (*high*), MITTEL (*medium*) und NIEDRIG (*low*), die in Kapitel 3.2.4.1 eingeführt wurden. Für jede vollständig bewertete Komponente wird in der Komponentenübersicht neben der Komponente ein kleiner Haken eingeblendet, der symbolisiert, dass die Bewertung bereits abgeschlossen ist. Damit wird automatisch ersichtlich, welche Komponenten noch bewertet werden müssen, bevor eine Ergebnisgenerierung möglich ist.

Ein weiterer essentieller Aspekt für die Auswertung des Outsourcingpotenzials stellt die Einstellung der projektspezifischen Gewichte (vgl. Abbildung 3.6) für die einzelnen Aggregationsebenen dar. Ohne diese ist eine Ergebnisberechnung für den Prototyp nicht möglich. Falls es hier keine Präferenzen des Entscheiders gibt, wird darauf hingewiesen, dass alle Ebenen gleich gewichtet werden. Sollte dies nicht gewünscht sein, kann über Schieberegler die gewünschte Gewichtung eingestellt werden (vgl. Abbildung 3.19). Der Algorithmus der App rechnet die

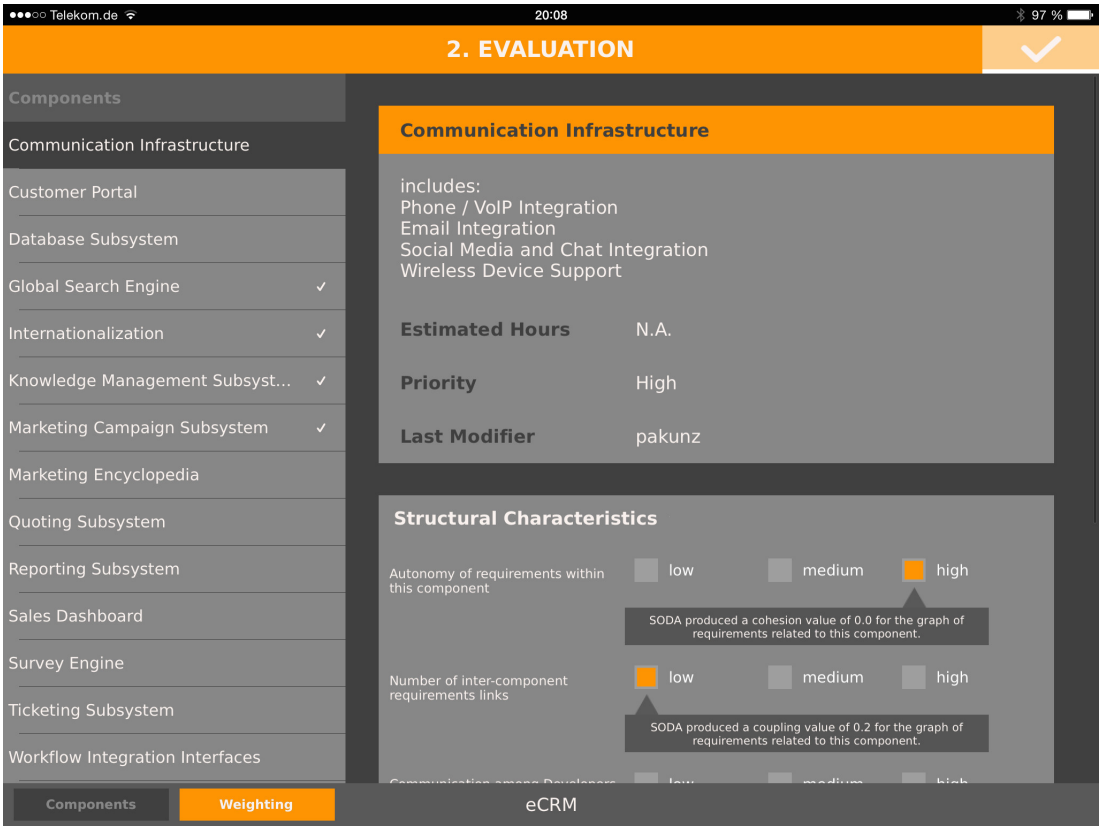

Abbildung 3.18.: Screenshot der Komponentenbewertung

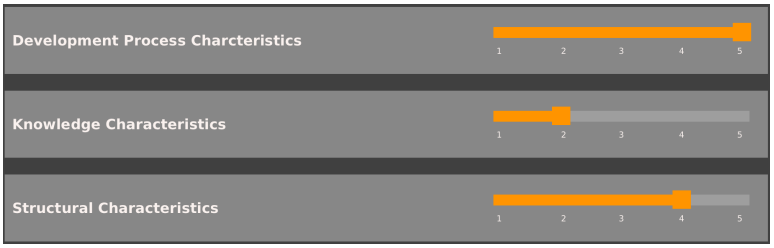

Abbildung 3.19.: Einstellung der Gewichte

Werte der Schieberegler immer so um, dass deren Summe 1 ergibt, so wie das für die Gewichtung der Kriterienkategorien in Kapitel 3.2.4.2 zur Amalgamation der Entscheidung gefordert wird.

Erst nach Bewertung aller Merkmale der zum Projekt zugehörigen Komponenten, was im EUS durch einen Bestätigungshaken neben allen bewerteten Komponenten und in der Menüanzeige links unten (vgl. Abbildung 3.18) angezeigt wird, wird im Hauptmenü der Zugang zur Ergebnisauswertung (*Display Results*) freigeschaltet. Sofern der Anwender beim Bewertungsprozess unterbrochen wird,

ist ein späteres Zurückkehren zur App und den bereits getätigten Bewertungen jederzeit möglich, da die beschiedenen Merkmalsausprägungen zu jeder Komponente automatisch im Hintergrund in der App gespeichert werden.

3.3.2.3. Ergebnisverarbeitung

Die Anzeige der Ergebnisse von *SmartSourcer* enthält die Bewertung des Outsourcingpotenzials für alle im Projekt definierten Komponenten. Die Klassifizierung beruht auf den in den Einstellungen definierten Entscheidungsgrößen und Aggregationsebenen des Entscheidungsmodells. Die Berechnung selbst erfolgt unter Berücksichtigung der vom Outsourcingverantwortlichen festgelegten Ausprägungen der Merkmale und der projektspezifischen Gewichtung der Aggregationsebenen. Als Ergebnis wird von der App daher zunächst eine übersichtliche Ansicht mit den Outsourcingempfehlungen für alle Komponenten des Softwareprojekts angezeigt. Darin wird in Tabellenform ausgegeben, welche der Komponenten selbst erstellt werden sollten, welche sich für die Auslagerung eignen und welche für beides geeignet wären (vgl. Abbildung 3.20).

Für jede einzelne Komponente gibt es nun die Möglichkeit, eine detaillierte Ergebnisübersicht anzeigen zu lassen. Diese Sichtweise fasst im oberen Teil zunächst die Beschreibung und das Gesamtergebnis der Auswertung für diese Komponente zusammen, indem ihr ein absoluter Wert und die zugehörige Bedeutung zugewiesen wird (vgl. Tabelle 3.9). Die genau Berechnung dieses Wertes ist in der Logikfunktion des Entscheidungsmodells (vgl. Kapitel 3.2.3) detailliert beschrieben. Darunter sind in Tabellenform die zusammengefassten Ergebnisse der Bewertung auf den jeweiligen Aggregationsebenen, die zugehörige Einstufung der betrachteten Komponente, die zuvor festgelegte Gewichtung und schließlich das gewichtete Ergebnis aufgeführt. Die Summe aller gewichteten Ergebnisse ergibt die Kennzahl der Gesamteinschätzung der Komponente. Durch ein weiteres Aufklappen der Ansicht auf der Aggregationsebene werden zusätzlich die zugehörigen Merkmale der Komponenten sowie ihre Bewertung durch den Entscheider offengelegt (vgl. Abbildung 3.21), wodurch die Berechnung der Ergebnisse stets nachvollziehbar bleibt.

Weiterhin wurden Exportfunktionalitäten in die App integriert, um die Bewertung der Komponenten unterstützend weiterverwenden zu können (vgl. Abbil-

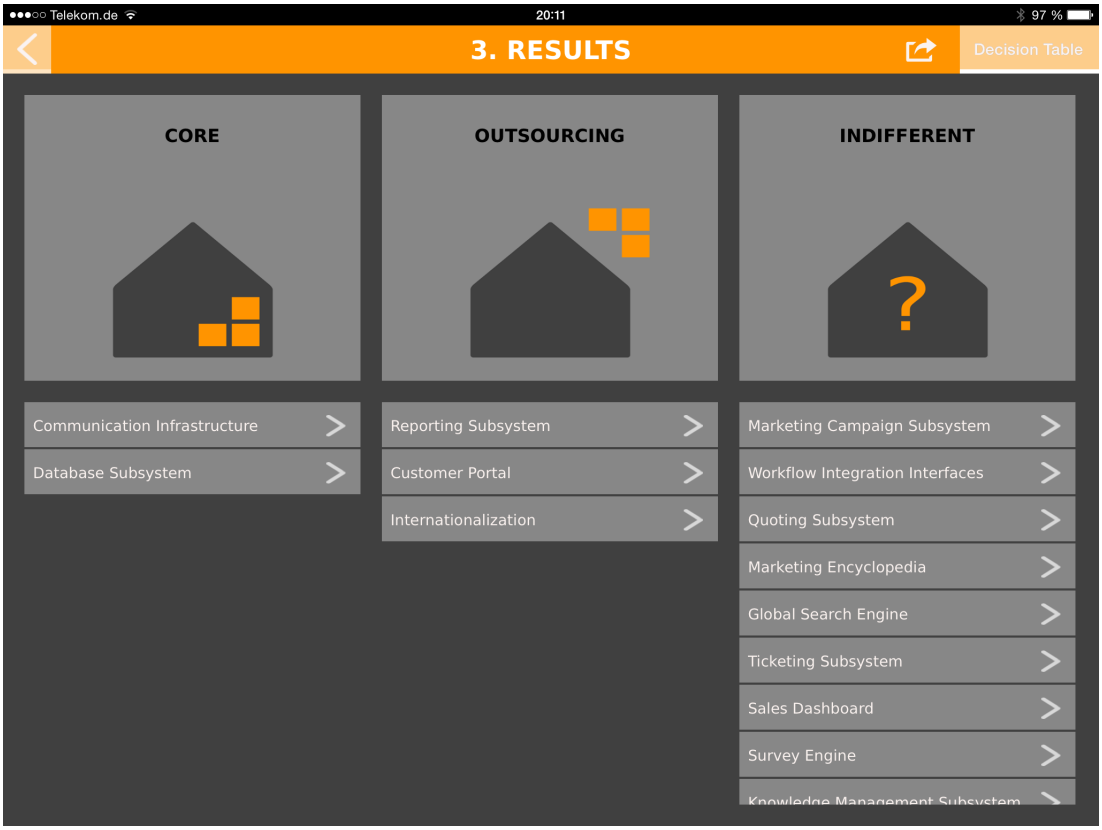

Abbildung 3.20.: Übersicht der Outsourcingempfehlungen

dung 3.22). Eine Exportmöglichkeit des Gesamtergebnisses ist die Generierung eines PDF-Exports (portables Dokumentenformat) zur übersichtlichen und druckfreundlichen Darstellung der komponentenbasierten Entscheidungsunterstützung. Dieses Dokumentenformat eignet sich ebenfalls gut für den Datenaustausch per E-Mail und ist damit ein notwendiges Ausgabeformat von *SmartSourcer* zur optimalen Prozessunterstützung des Managements. Weiterhin lassen sich die Berechnungen der einzelnen Komponenten zurück an den *CodeBeamer* übertragen, um die Entscheidungen in der Kollaborationsplattform für Projektbeteiligte transparent zu gestalten. Entsprechend der Ergebnisse kann der weitere Entwicklungsprozess gesteuert werden.

Der generierte Export umfasst sowohl eine Übersicht über die Auslagerungsempfehlung aller Komponenten (vgl. Abbildung 3.24), eine Detailseite für jede bewertet Komponente, die die Berechnung transparent macht, als auch eine tabellarische Zusammenfassung aller Komponenten auf der aggregierten Ebene. Die Detailseite wird in Abbildung 3.25 beispielhaft für eine Komponente dargestellt.

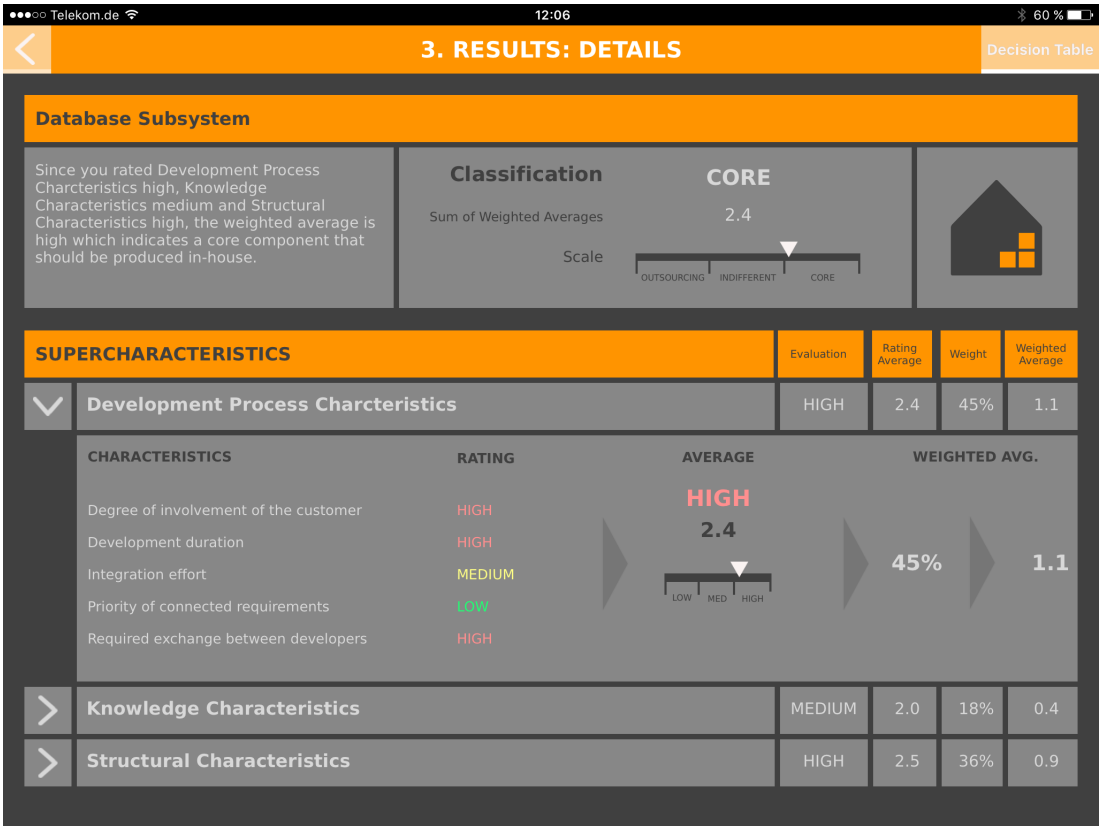

Abbildung 3.21.: Detaillierte Ergebnisübersicht

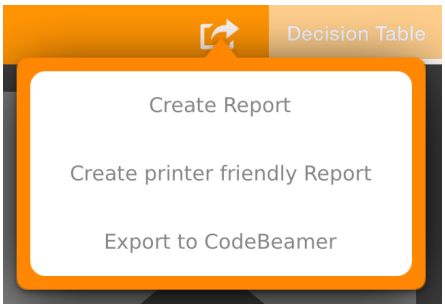

Abbildung 3.22.: Exportmöglichkeiten von *SmartSourcer*

Der Bericht umfasst selbstverständlich die Auswertungen für alle Komponenten, die analog zum abgebildeten Beispiel im Export eingebunden sind. Die tabellarische Zusammenfassung beinhaltet abschließend zu jeder Komponente den berechneten Wert der aggregierten Spezifität und das eingestellte Gewicht jeder Kategorie (vgl. Abbildung 3.26).

3.3.2.4. Gewichtung und Berechnung

Die Berechnungen des Outsourcingpotenzials von Softwarekomponenten finden im EUS analog zu den Berechnungen, die in der Entscheidungslogik des Modells (vgl. Kapitel 3.2.3) vorgegeben sind, statt. Eine Besonderheit dabei stellt die Entscheidungsgröße *Modularität* (E_1) dar. In der Entscheidungslogik des Modells muss dieser Wert zunächst invertiert werden (vgl. Kapitel 3.2.4.1). Dies wird in der Software so umgesetzt, dass anstatt nach der Modularität, nach der Autonomie der mit einer Komponente verknüpften Anforderungen gefragt wird. Denn je isolierter die zu einer Komponente gehörigen Anforderungen sind, desto geringer ist die innere Abhängigkeit (Modularität) einer Komponente. Dadurch kann der Algorithmus, der die Merkmalsausprägungen und die aggregierten Werte für die Kriterienkategorien berechnet in gleicher Weise für alle Kriterien und Kategorien verwendet werden, weil die Ausprägungen *HOCH*, *MITTEL* und *NIEDRIG* immer gleichermaßen in die Outsourcingentscheidung mit einfließen. **Autonomie** wird damit im Kontext dieser Arbeit als die Invertierung von Modularität definiert. Der Softwarecode, der die Berechnung des Outsourcingpotenzials durchführt, ist nachfolgend abgedruckt.

```
1  /*
2   *  Method that calculates the results of the evaluation
3   *  returns array of components based on classification:
4   *  index 0: Core Components, Index 1: Indifferent, Index 2:
       Outsourcing
5   *
6   */
7  - (NSArray *)calculateResults
8  {
9    //iterate through components
10   NSEnumerator *componentEnumerator = [self.project.consistsOf
       objectEnumerator];
11   NSArray *classification = [NSArray arrayWithObjects:[
       NSMutableArray array], [NSMutableArray array], [NSMutableArray
       array], nil];
12   Component *comp;
13   while ((comp = [componentEnumerator nextObject]) != nil) {
14
15       //initiate array for rating values of supercharacteristics
```

```
16      NSMutableArray *ratingValuesSuperChar = [NSMutableArray array
    ];
17
18      //iterate through all SuperCharacteristics to get their
    weighted rating value and add up the total weight of all
    supercharacteristics
19      double totalWeight = 0.0;
20      SuperCharacteristic *superChar;
21      NSEnumerator *superCharEnumerator = [comp.ratedBy
    objectEnumerator];
22      while ((superChar = [superCharEnumerator nextObject]) != nil)
    {
23
24          //initiate rating value for supercharacteristic
25          float valueOfSuperCharacteristic = 0.0;
26          //iterate through all characteristics and add their values
     to the value of supercharacteristic
27          Characteristic *characteristic;
28          NSEnumerator *charEnumerator = [superChar.
    superCharacteristicOf objectEnumerator];
29          while ((characteristic = [charEnumerator nextObject]) !=
    nil) {
30              valueOfSuperCharacteristic =
    valueOfSuperCharacteristic + [characteristic.value doubleValue];
31          }
32          //divide by number of characteristics --> average
33          double numberOfCharacteristics = [[superChar
    superCharacteristicOf] count];
34          valueOfSuperCharacteristic = (valueOfSuperCharacteristic/
    numberOfCharacteristics);
35          //add weight to total weight of all supercharacteristics
36          totalWeight = totalWeight + [superChar.weight doubleValue
    ];
37          //add weighted value to array
38          double weightedValueofSuperCharacteristic =
    valueOfSuperCharacteristic * [superChar.weight doubleValue];
39          [ratingValuesSuperChar addObject:[NSNumber
    numberWithDouble:weightedValueofSuperCharacteristic]];
40      }
41
42      self.totalWeightOfSuperCharacteristics = totalWeight;
```

```
43        //build weighted average of the rating of ONE component
44        NSArray *ratingValues = [ratingValuesSuperChar copy];
45        double sum = 0.0;
46        for (int i=0; i<[ratingValues count]; i++) {
47            sum = sum + [[ratingValues objectAtIndex:i] doubleValue];
48        }
49        //calculate component weighted value
50        double componentWeightedValue = (sum/totalWeight);
51        //put component info into ABC-Classification according to its
      component weighted value
52        // A <=> objectAtIndex 0, B <=> objectAtIndex 1, C <=>
      objectAtIndex 2
53        if (componentWeightedValue < 1.67) {
54            [[classification objectAtIndex:2] addObject:comp];
55        } else if (componentWeightedValue < 2.34) {
56            [[classification objectAtIndex:1] addObject:comp];
57        } else if (componentWeightedValue <= 3.0) {
58            [[classification objectAtIndex:0] addObject:comp];
59        }
60    }
61    self.classificationResult = classification;
62    return self.classificationResult;
63 }
```

3.3.2.5. Teilautomatisierung der Merkmalsbewertung

Die Bewertung der einzelnen Komponenten stellt den manuellen Teil des mobilen EUS dar und erfordert die Einschätzung des bewertenden Outsourcingmanagers für jedes einzelne Merkmal. Ein hilfreiches EUS zeichnet sich jedoch durch seine Unterstützungsmöglichkeiten bei der Entscheidungsfindung aus, also durch die Vernetzung dezentraler Informationen und die Automatisierung von Entscheidungsprozessen (vgl. Kapitel 2.2). Deshalb wurde *SmartSourcer* so entwickelt, dass zusätzlich zu den Komponentendaten alle Informationen über die Anforderungen des Projekts ausgelesen werden, um eine automatisierte Berechnung der strukturellen Entscheidungsgrößen zu ermöglichen.

Im Detail handelt es sich hierbei um die Merkmalsempfehlungen für die beiden Entscheidungsgrößen *Kohäsion* bzw. *Autonomie* (E_1) und *Kopplung* (E_2). Für Ausprägungen dieser beiden Merkmale können Empfehlungen durch *Smart-*

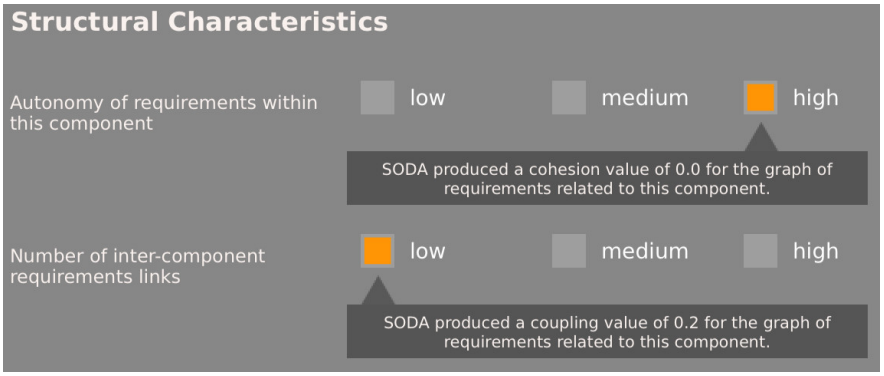

Abbildung 3.23.: Vorschlagsfunktion für Kopplung und Kohäsion (Autonomie)

Sourcer gegeben werden. Entsprechende Hinweise bei der Bewertungseingabe als auch bereits vorselektierte Auswahlboxen im *SmartSourcer*, wie in Abbildung 3.23 dargestellt, erleichtern so den Entscheidungsprozess des Outsourcingmanagers.

Um diese Teilautomatisierung der Merkmalsbewertung zu erreichen, wird auf die *SODA*-Methode, einem anforderungsbasierten Ansatz zur Unterstützung der Outsourcingentscheidung, zurückgegriffen (vgl. Kapitel 3.2.2). Diese Methode beschreibt die Erstellung und Analyse von Clustern in einem Graphen, der aus den strukturiert vorliegenden Anforderungen eines Softwareprojekts und ihren gegenseitigen Beziehungen aufgebaut werden kann. Das Modell enthält die Anforderungen als Knoten und die jeweiligen Beziehungen als Kanten zwischen zwei Knoten (vgl. Kapitel 3.2.2.4).

Für die automatisierte Berechnung der Merkmalsempfehlungen für Kohäsion und Kopplung werden deshalb im EUS *SmartSourcer* über den Proxy alle zu einem Projekt gehörigen Anforderungen eingelesen und mit Hilfe der SODA-Technik zu den bereits vordefinierten Komponenten zugeordnet. Dadurch wird ein Anforderungsmodell im Sinne von SODA erstellt. Mit dem Algorithmus von Newman (2006) sowie den in Kapitel 3.2.2.4 vorgestellten Metriken von Briand et al. (1996) wird dann für jedes Cluster, das stellvertretend für die Komponenten des Projekts steht, ein Wert für die Kopplung und die Kohäsion berechnet.

Eine Anpassung der SODA-Technik ist erforderlich, damit auch für die Kopplung ein normalisiertes Ergebnis pro Cluster erzielt werden kann. Daher wird der Algorithmus in der Implementierung des EUS dieser Arbeit so erweitert, dass die Summe aller Gewichte von Kanten, die ein Cluster verlassen, in Relation zur Gesamtsumme aller Gewichte zwischen Clustern gesetzt werden. Dies ist mög-

lich, weil der Umfang des Projekts bekannt ist und die Clusterbildung bereits abgeschlossen ist. Nur so kann für jede Softwarekomponente Vergleichbarkeit der Kopplungswerte hergestellt werden.

Das Ergebnis der automatisierten Berechnung wird dann für jede Komponente mit einem entsprechenden Verweis auf die Berechnung durch *SODA* in der Bewertungsansicht vorselektiert. Diese Darstellungsmöglichkeit erlaubt aber weiterhin die manuelle Veränderung der Ausprägungen durch den verantwortlichen Entscheider (vgl. Abbildung 3.23). Der Algorithmus zur Berechnung der Werte ist als Auszug aus dem Softwarecode im Anhang E (Algorithmus zur Berechnung von Kopplung und Kohäsion) abgedruckt.

3.4. Zusammenfassung

Im Rahmen dieses Kapitels wurden das Konzept für ein Entscheidungsmodell und dessen Umsetzung in ein mobiles EUS vorgestellt, um das Outsourcing von Softwarekomponenten zu unterstützen. Zusammen bilden diese das Artefakt dieser konstruktionswissenschaftlichen Arbeit. Dafür wurden zunächst die Anforderungen an eine Technologie zur Entscheidungsunterstützung für den operativen Betrieb eines Unternehmens auf Basis von Softwarekomponenten in ein theoretisch hergeleitetes Entscheidungsmodell umgesetzt. Neben den Entscheidungsgrößen, die sich in strukturelle Eigenschaften einer Komponente in Bezug auf das Softwareprodukt, Prozessmerkmale der Entwicklung und Wissensmerkmale einer Softwarekomponente gliedern lassen, wurde eine Entscheidungslogik zur Verknüpfung aller Merkmale des Modells definiert. Dadurch lässt sich für alle Komponenten eines Softwareprojekts systematisch das Outsourcingpotenzial bestimmen, was durch eine beispielhafte Anwendung des Entscheidungsmodells beschrieben wird.

Die Umsetzung des zuvor konzeptionierten Entscheidungsmodells als mobiles EUS, das sich einfach in die Serverlandschaft und Kommunikationsstruktur von Softwareunternehmen integrieren lässt, wurde im weiteren Verlauf des Kapitels detailliert beschrieben. Neben der Vorstellung der Lösungsarchitektur, wurden die Details der Entwicklungsumgebung und die Funktionalität der App *SmartSourcer* ausführlich erläutert. Dabei wurde besonders auf die Einstellungsmög-

lichkeiten und Bewertungsfunktionen des mobilen EUS eingegangen, da diese eine dynamische Erweiterung (oder auch Reduzierung) der Entscheidungskriterien und Merkmalskategorien erlauben während zugleich das Berechnungsverfahren zur Amalgamation der Entscheidung automatisch angepasst wird. Besonderheiten wie die automatische Berechnung des Outsourcingpotenzials einer Komponente, die systemseitige Unterstützung bei der Bewertung der Ausprägungen der strukturellen Merkmale (Kohäsion und Kopplung) sowie die übersichtliche Darstellung und Begründung des Outsourcingmix wurden als Kernfunktionalitäten hervorgehoben und mit Auszügen aus dem Softwarecode belegt.

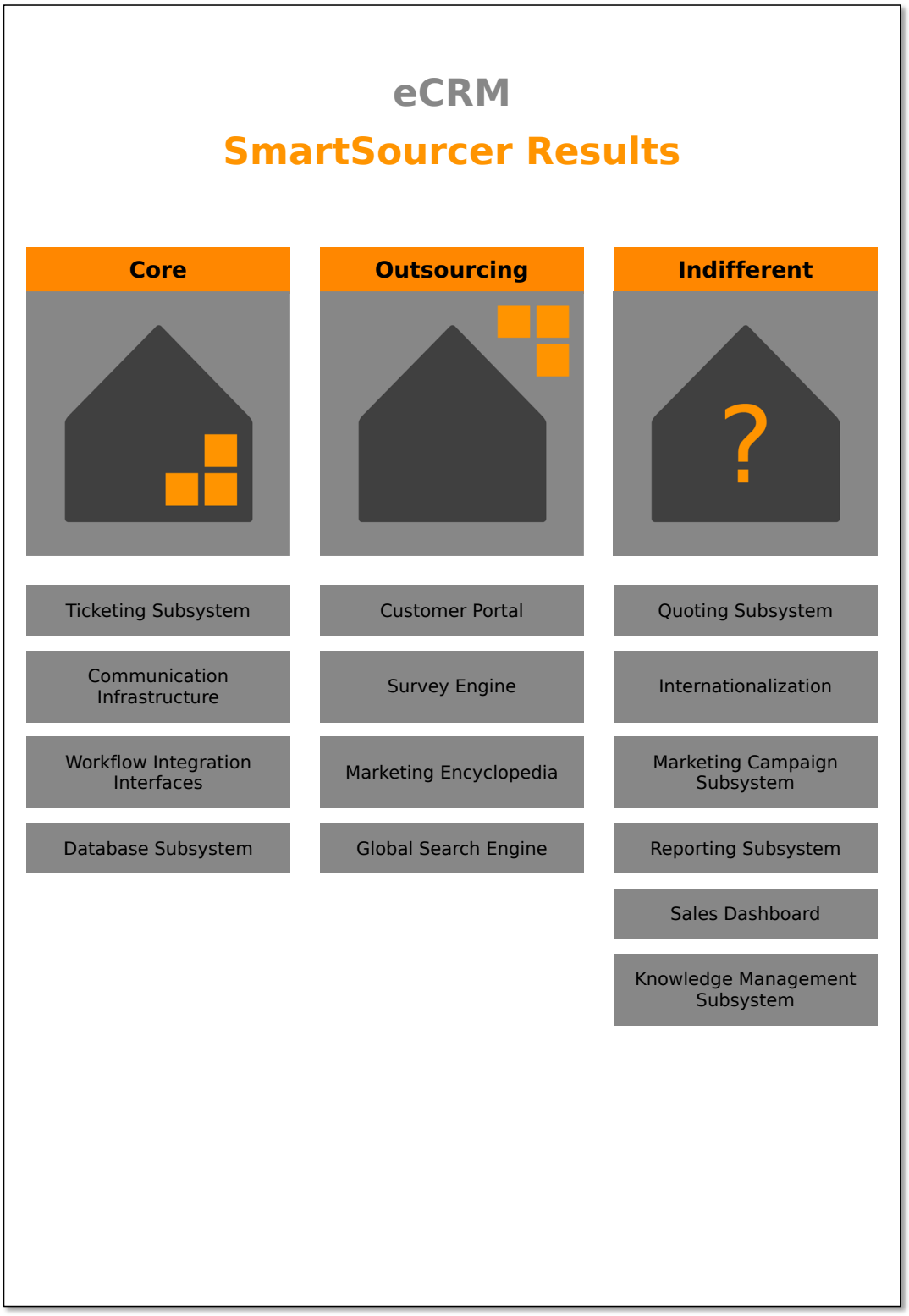

Abbildung 3.24.: Empfehlungsseite des generierten Reports

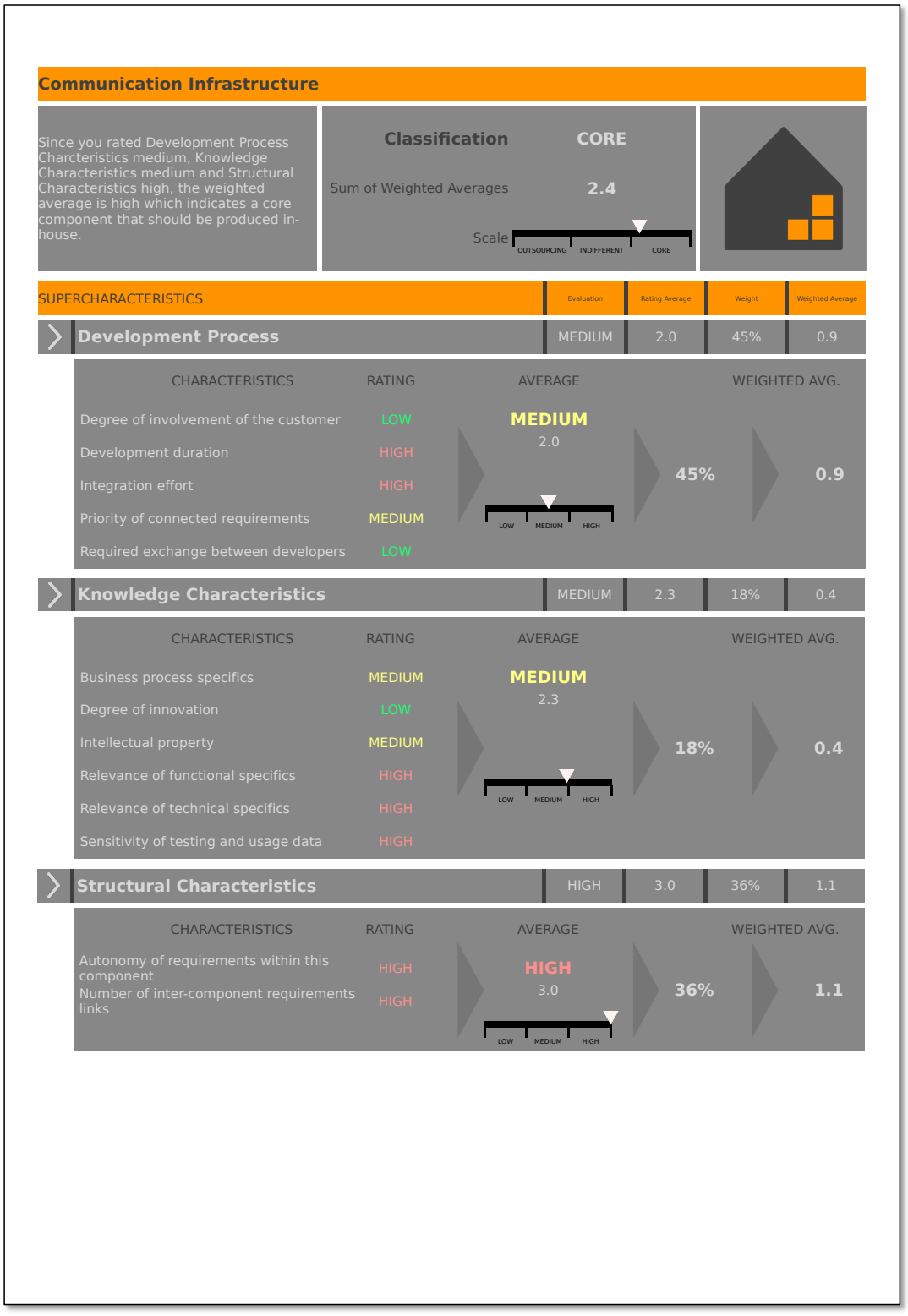

Abbildung 3.25.: Detailseite des generierten Reports

Overview

Component	Result	Development Process Charcteristics (45%)	Knowledge Characteristics (18%)	Structural Characteristics (36%)
Ticketing Subsystem	CORE (2.5)	HIGH (2.4)	HIGH (2.7)	HIGH (2.5)
Communication Infrastructure	CORE (2.4)	MEDIUM (2.0)	MEDIUM (2.3)	HIGH (3.0)
Workflow Integration Interfaces	CORE (2.7)	HIGH (2.8)	HIGH (2.7)	HIGH (2.5)
Database Subsystem	CORE (2.4)	HIGH (2.4)	MEDIUM (2.0)	HIGH (2.5)
Quoting Subsystem	INDIFFERENT (2.3)	HIGH (2.4)	HIGH (2.7)	MEDIUM (2.0)
Internationalization	INDIFFERENT (1.9)	MEDIUM (2.2)	MEDIUM (1.8)	LOW (1.5)
Marketing Campaign Subsystem	INDIFFERENT (2.1)	MEDIUM (2.0)	LOW (1.5)	HIGH (2.5)
Reporting Subsystem	INDIFFERENT (2.2)	MEDIUM (2.2)	MEDIUM (1.8)	HIGH (2.5)
Sales Dashboard	INDIFFERENT (1.8)	MEDIUM (1.8)	MEDIUM (1.7)	MEDIUM (2.0)

Abbildung 3.26.: Übersichtsseite der Kategorien im generierten Report

4. Evaluation des Systems zur Entscheidungsunterstützung

Entsprechend der Literatur zum konstruktionswissenschaftlichen Forschungsparadigma ist der Evaluationsprozess ein integraler Bestandteil bei der Erschaffung von Artefakten im Rahmen eines solchen Forschungsvorhabens. Aus diesem Grund fordern Hevner et al. (2004) bei der Entwicklung neuer Technologie dazu auf, neue Artefakte hinsichtlich ihrer Nützlichkeit, ihrer Qualität und ihrer Leistungsfähigkeit zu evaluieren (Hevner et al., 2004, S. 83). Neuste Studien über Evaluationsmethoden in der *Design Science* betonen, dass die Vorteile einer neuen Technologie sowohl für die theoretische Wissensbasis als auch für den praktischen Einsatz rigoros aufgezeigt werden müssen (Venable et al., 2012). Daran anknüpfend wird in diesem Kapitel das Ziel verfolgt, eine rigorose Evaluation des im vorangegangenen Kapitel entworfenen Systems zur Entscheidungsunterstützung unter Berücksichtigung der bisher erforschten Bewertungsprinzipien aus der Literatur zum konstruktionswissenschaftlichen Forschungsparadigma zu präsentieren. Bei dem zu bewertenden Artefakt handelt es sich um *SmartSourcer*, bestehend aus dem in Kap. 3.2 entworfenen Entscheidungsmodell und zugehöriger Entscheidungslogik sowie der Implementierung des Modells wie in Kap. 3.3 dargelegt.

Eine kurze Literaturübersicht über die Bedeutung der Evaluation wird zunächst vorgestellt. Daraus werden passende Evaluationsmethoden für diese Arbeit abgeleitet. Im anschließenden Evaluationsdesign wird dann das theoretische Bezugssystem als Evaluationsmodell für das System zur Entscheidungsunterstützung definiert. Es folgen eine Herleitung der Messgrößen sowie die Beschreibung der Strategie und der Instrumente zur Durchführung. In einer deskriptiven und qualitativen Analyse der Ergebnisse werden schließlich die Nützlichkeit, die Qualität und die Nutzbarkeit des neu entwickelten Artefakts detailliert beschrieben.

4.1. Bedeutung der Evaluation

Der Evaluation von neu entwickelten Artefakten wird in der *Design Science* eine besondere Bedeutung zugemessen und zählt zu den elementaren Bestandteilen konstruktionsiwssenschaftlichen Arbeitens (Gregor und Jones, 2007; Hevner et al., 2004; March und Smith, 1995; Walls et al., 1992). Sie liefert Hinweise dafür, ob und wie gut ein neu entworfenes Artefakt funktioniert und wie nützlich es ist. Andernfalls wäre die Behauptung, dass das Artefakt seinen Zweck erfüllt, nicht nachgewiesen. Die Evaluation liefert daher den erforderlichen wissenschaftlichen und deterministischen Charakter einer technologiegetriebenen Forschungsarbeit (Venable et al., 2012, S. 425). Sie ist über die eigene Nützlichkeit des Artefakts hinaus für das formalisierte Wissen zuständig, das aus dem Design von Artefakten in Form von Designprinzipien oder technologischen Regeln entsteht. Das schließt ebenfalls den Vergleich des erschaffenen Artefakts mit anderen Artefakten für den selben Zweck und der damit verbundenen Identifikation von Verbesserungsmöglichkeiten oder ungewollten Effekten ein (Venable et al., 2012).

Für diesen Zweck entwickelten Walls et al. (1992) das Konzept der diskreten testbaren Hypothesen für die Evaluation des Designprozesses oder des entworfenen Produkts mit der Absicht abstrakte Anforderungen zu erheben. March und Smith (1995) definieren die beiden Phasen *Erschaffen* und *Bewerten* als die zwei einzigen Phasen der *Design Science*. Dabei spezifizieren sie die Phase des *Bewertens* mit der Entwicklung von Kriterien und der anschließenden Auswertung des Artefakts auf der Basis dieser Kriterien (March und Smith, 1995, S. 258). Die Lauffähigkeit eines Artefakts wird dabei bereits in der ersten Phase gefordert und ist nicht Gegenstand der Bewertungsphase. Letztere ist deshalb ausschließlich für die Überprüfung der Funktionsfähigkeit und Funktionsweise verantwortlich. Im Beitrag von Hevner et al. (2004) wird die Evaluation bereits als wesentlicher Bestandteil des konstruktionswissenschaftlichen Paradigmas bezeichnet und es wird der systematische Nachweis für die Nützlichkeit, Qualität und Leistungsfähigkeit eines Artefakts gefordert. Weiterhin wird dabei großer Wert auf die Verwendung von Evaluationskriterien gelegt, die auf den Anforderungen an das Artefakts basieren, die aus dem Anwendungskontext entstehen. Ebenfalls wird eine erste Klassifizierung von Evaluationsmethoden beschrieben. Jedoch fehlen an

dieser Stelle noch Richtlinien und genaue Prozessbeschreibungen zur eigentlichen Durchführung der Evaluation oder zur Auswahl geeigneter Evaluationsmethoden.

Die Arbeit von Venable (2006) unterscheidet Evaluationsmethoden in zwei Formen. Er nennt sie die *künstliche* und die *naturalistische* Evaluation. Bei der künstlichen Form der Evaluation wird das Artefakt auf stilisierte und kontrollierte Art und Weise getestet, und es werden dabei Methoden wie Laborexperimente, Feldstudien, Simulationen oder mathematische Beweise verwendet. Im Gegensatz dazu erfasst die naturalistische Evaluation die Leistungsfähigkeit eines Artefakts in seiner natürlichen Umgebung mit all den komplexen Vorgängen der menschlichen Vorgehensweisen in wirklichen Organisationen (Venable, 2006). Aufgrund der zahlreichen Störfaktoren in der realen Umgebung kann eine realistische Evaluation unzuverlässig und mit hohen Kosten verbunden sein und damit die Genauigkeit und Reproduzierbarkeit der Ergebnisse beeinträchtigen. Dennoch lassen sich damit die realen technologischen, organisatorischen oder sozialen Auswirkungen eines in Anwendung befindlichen Artefakts aufdecken (Venable, 2006). Die hierfür verwendeten Methoden sind meist Fallstudien, Feldstudien, Umfragen, phänomenologische Studien oder Aktionsforschung.

In Ergänzung dazu beschreiben Sun und Kantor (2006) den naturalistischen Aufbau in Form von realen Anwendern, die reale Systeme benutzen, um reale Probleme zu lösen. Dagegen wird die künstliche Evaluation in der Form von Rollenstellvertretern beschrieben, die das neue Artefakt zur Lösung stilisierter Probleme benutzen. Dafür werden drei unterschiedliche Ebenen eingeführt, auf denen die Ergebnisse eines Informationssystems bewertet werden können: Das Erreichen eines individuellen Teilziels, die vollständige Durchführung einer Aufgabe und die Auswirkungen einer vollständig durchgeführten Aufgabe auf die motivierende Zielstellung des Individuums oder der Organisation.

Pries-Heje et al. (2007) argumentieren, dass die naturalistische Evaluation das soziale System, in dem das Artefakt erschaffen und genutzt wird, näher beleuchtet werden soll. Als Lösungsvorschlag unterbreiten sie die Anwendung von *schwachem Design Science*, das durch Einbindung von Eigenschaften des konstruktionswissenschaftlichen Ansatzes in die *Soft Systems Methodology* erzielt wird. Dies ist eine Methodik für die Analyse und das Design im Rahmen der Systemanalyse, die zwischen systemischem Denken und der realen Welt unterscheidet (Checkland und Scholes, 1990; Venable et al., 2012). Durch diese Trennung wird ein effektiver

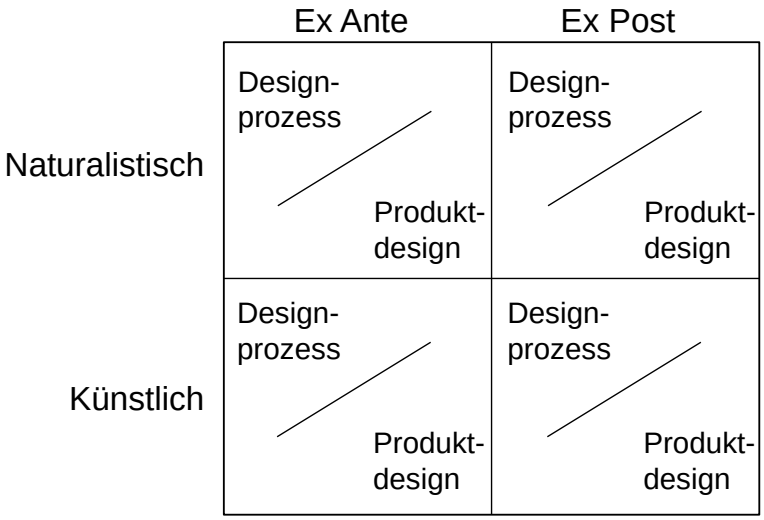

Abbildung 4.1.: Klassifizierung von Evaluationsmethoden in der *Design Science* nach Pries-Heje et al. (2008)

Ansatz zur Entwicklung von Systemen erwartet, die in schwierigen sozialen Organisationsstrukturen besser Rechnung tragen sollen (Baskerville et al., 2009). Das Ergebnis ist eine Methode, die aus sieben Aktivitäten zur Lösung spezifischer Probleme auf allgemeiner Ebene besteht und dabei die Realisierbarkeit einer Lösung innerhalb eines Unternehmens mit einer naturalistischen Evaluation sicherstellt. Die Notwendigkeit, dem Unternehmensumfeld bei solchen Studien mehr Beachtung zu widmen, wird von Sein et al. (2011) in der designorientierten Aktionsforschung (*action design research*) weiter aufgegriffen. Diese Forschungsmethode beschreibt das Erstellen und Bewerten von IT-Artefakten als nicht teilbare und inhärent miteinander verwobene Aktivitäten, die gleichzeitig stattfinden (Sein et al., 2011, S. 37). Demzufolge wird das Designartefakt in dem organisatorischen Umfeld entwickelt und angewendet, in dem es fortlaufend geformt und verfeinert wird.

Pries-Heje et al. (2008) greifen auf die Unterscheidung zwischen künstlichen und naturalistischen Evaluationmethoden zurück, um eine erste Systematik (vgl. Abbildung 4.1) zu entwerfen, die zusätzlich den Zeitpunkt der Evaluation (*ex ante* oder *ex post*) sowie den Gegenstand der Evaluation (Prozess vs. Produkt) beinhaltet. Die Unterscheidung zwischen *ex ante* und *ex post* Evaluationen setzt ein allgemeines Grundverständnis über das Artefakt voraus, da die Evaluation mitten im Designprozess des Artefakts verankert ist. Betrachtet am Beispiel ei-

nes IT-Artefakts (lauffähige Instanziierung eines Systems) bezeichnet *ex ante* in der *Design Science* den Entwurf des Artefakts und *ex post* das entworfene Artefakt selbst. Wenn nun aber die Sichtweise von einem geschaffenen zu einem in der Entwicklung befindlichen Artefakt verschoben wird, dann drückt *ex post* die Evaluation des Entwurfsergebnisses aus und *ex ante* steht für den Suchprozess für den Lösungsentwurf (Pries-Heje et al., 2008). Aus diesem Grund ist erforderlich, für das Artefakt einen Bezugspunkt der Evaluation zu bestimmen, so wie es bei der designorientierten Aktionsforschung mit Alpha- und Betaversionen gehandhabt wird. Weiterhin gelten bei der Systematik von Pries-Heje et al. (2008) die aktuelle Entwicklungsstufe (früh, mittel, spät), in der man sich im Designprozess befindet, die Ziele und Anforderungen an die Forschung (Kontrolle, reale Anwendung) sowie praktische Einflussgrößen (Kosten, Ressourcen, Zeit) als Eingabeparameter für die Auswahl, welcher Quadrant und welche spezifische Strategie des Rahmenwerks dadurch verfolgt werden soll. Dies beeinflusst ebenfalls die Auswahl der Evaluationskriterien, Messinstrumente oder Metriken.

Venable et al. (2012) stellen schließlich die umfassendste Arbeit über die Auswahl von Richtlinien für die Evaluation in der *Design Science* bereit, indem sie ein umfangreiches Klassifikationsschema ausarbeiten, das dabei hilft, die passende Evaluationsstrategie und die zugehörigen Methoden anhand der Kontextmerkmale des betreffenden Artefakts auszuwählen. Zu diesem Zweck wurde das Schema von Pries-Heje et al. (2008) auf zwei unterschiedliche Weisen erweitert. Zum einen werden kontextuelle Aspekte, welche die Kriterien für das Design der Evaluation bilden, mit einer oder mehreren potenziellen Evaluationsstrategien verknüpft. Die Kontextfaktoren beinhalten dabei Aspekte wie Evaluationszweck, Ziele, Artefakttyp und Eigenschaften eines Artefakts. Zum anderen werden eine oder mehrere ausgewählte Evaluationsstrategien (als Quadrant im Rahmenwerk dargestellt) mit empfohlenen Evaluationsmethoden (z.B. Aktionsforschung, Laborexperiment, Umfrage) verknüpft. Zusammen mit diesem doppelten Rahmenwerk stellen Venable et al. (2012) eine umfassende Systematik mit präskriptivem Gehalt für das Evaluationsdesign im konstruktionswissenschaftlichen Paradigma bereit, die üblicherweise aus vier Schritten besteht: Analyse der Anforderungen an die Evaluation, Zuordnung der Anforderungen zu den entsprechenden Quadranten zur Auswahl der Evaluationsstrategie, Auswahl geeigneter Methoden zur identifizierten Strategie und Entwurf einer detaillierten Evaluation.

4.2. Evaluationsdesign

Zur Bewertung des Artefakts dieser Arbeit werden im Verlauf dieses Kapitels ein theoretisches Bezugssystem, entsprechende Messgrößen sowie eine passende Strategie zur Durchführung vorgestellt. Dadurch wird eine systematische Analyse der Qualitäten des Entscheidungsmodells und der zugehörigen Implementierung *SmartSourcer* sichergestellt. Das Ergebnis soll zeigen, in welchem Maße das neu entwickelte Artefakt dieser Arbeit nützlich und nutzbar für Unternehmen ist, die regelmäßig mit Outsourcingentscheidungen auf der Basis von Softwarekomponenten konfrontiert sind.

Gemäß Frank (2000) wird in der Evaluation Objektivität angestrebt. Dies setzt jedoch voraus, dass das Artefakt in einer Umgebung evaluiert wird, die seinem zukünftigen Einsatzszenario entspricht oder diesem ähnlich ist, und dass Entscheidungskriterien verwendet werden, deren Metriken effizient sowie nützlich sind, um vergleichbare Ergebnisse zu erzielen (Hevner et al., 2004; March und Smith, 1995; Venable et al., 2012).

Als Ausgangsbasis für die Festlegung des Bezugsrahmens sowie für die Wahl von Evaluationskriterien wird die Arbeit von Checkland und Scholes (1990) herangezogen, in der vier Kriterien hervorgehoben werden, die für die Qualitätsmessung in der *Soft Systems Methodology* verwendet werden: Effektivität, Effizienz, ethnische Aspekte und Eleganz. Diese Liste wird durch Kriterien von March und Smith (1995) ergänzt, in der individuelle Evaluationskriterien je Artefakttyp aufgezählt werden, darunter auch Effizienz, Effektivität und Einfachheit der Nutzung. Hevner et al. (2004) fordern einen gründlichen Nachweis der Nützlichkeit, Qualität und Leistungsfähigkeit des Artefakts. Als ergänzende Kriterien gelten Funktionalität, Vollständigkeit, Konsistenz, Genauigkeit, Geschwindigkeit, Zuverlässigkeit, Einpassung in den organisatorischen Kontext sowie Aussehen des Artefakts. Pries-Heje et al. (2008) schlagen zusätzlich vor, Qualitätsmodelle für die Evaluation von Artefakten anzuwenden und verweisen dabei auf eine prozessbasierte Qualitätsmessung bei der Evaluation von Prozessen in der *Design Science*. Ein mögliches Qualitätsmodell ist der Standard ISO 9126, ein internationaler Standard für die Evaluation von Software, der zahlreiche externe und interne Maße vorschlägt, anhand derer ein Entwurf gemessen werden kann. Die Forschung von Sonnenberg und vom Brocke (2012) ergänzen die Kriterienliste

um weitere mögliche Kriterien, wie z.B. Übereinstimmung mit dem Phänomen aus der realen Welt, Durchführbarkeit und interne Konsistenz.

Auf dieser soliden Basis gilt es nun einen theoriezentrischen Bezugsrahmen für die Evaluation einer Entscheidungstechnik aufzuspannen, die aus einem Entscheidungsmodell und einem implementierten EUS besteht. Hierfür muss ein Evaluationsmodell verwendet werden, das die Anforderungen der Literatur an die Kriterien zur rigorosen Bewertung von Artefakten erfüllt und gleichzeitig Messgrößen verwendet, die die Nützlichkeit beider Teile des Artefakts ermitteln.

4.2.1. Theoriezentrisches Bewertungsmodell für das neu entwickelte Artefakt

Die generelle Zielsetzung der Evaluation eines Artefakts unter realistischen Bedingungen ist die Abschätzung, welches Potenzial das Artefakt im praktischen Einsatz besitzt, um die Effizienz und Effektivität bestimmter Geschäftsprozesse zu verbessern. Wahrgenommene Nützlichkeit und wahrgenommene Benutzerfreundlichkeit sind anerkannte Instrumente zur Bestimmung von Effizienz und Effektivität (Benbasat und Barki, 2007). Beide sind zentrale Konstrukte des Technologieakzeptanzmodells (TAM), das darauf abzielt, die individuelle Nutzungsbereitschaft vorherzusagen. Das TAM eignet sich für die Durchführung von Bewertungen zu unterschiedlichen Zeitpunkten (Davis et al., 1989; Venkatesh und Bala, 2008). Es wird sowohl für Evaluationen *ex ante* als auch *ex post* eingesetzt (vgl. Kapitel 4.1).

In dieser Arbeit wird das Artefakt nach der Erstellung auf seinen praktischen Nutzen hin bewertet, um die vorteilhafte Verwendung unter realen Bedingungen zu belegen. Für die Bewertung durch tatsächliche Anwender des Entscheidungsmodells und des mobilen EUS steht bereits das real nutzbare Artefakt zur Verfügung. Dieses muss dann nicht abhängig von der Vorstellungskraft der Tester und unter Verwendung von Konjunktiven evaluiert werden, sondern kann als Prototyp in einer Umgebung getestet werden, die dem zukünftigen Einsatzzweck sehr ähnlich oder gleich ist. Weiterhin können so auch optische, haptische und zeitliche Eigenschaften des neuen Artefakts kommentiert werden, um eine bessere Einschätzung der Benutzerfreundlichkeit zu erzielen.

Das Bewertungsmodell, auf das die Evaluation des Artefakts dieser Arbeit ausgerichtet ist, versucht die zentralen Konstrukte des TAM miteinzubeziehen, da es einerseits für die Bewertung des Entscheidungsmodells und andererseits für dessen technische Implementierung (*SmartSourcer*) verwendet werden kann. Die tatsächliche Nutzung des Systems zur Entscheidungsunterstützung stellt dabei die abhängige Variable dar. Die tatsächliche Nutzung des Systems wird von der Nutzungsbereitschaft seiner Anwender beeinflusst. Die Bereitschaft wird analog zum TAM von der wahrgenommenen Nützlichkeit des Entscheidungsmodells und der wahrgenommenen Benutzerfreundlichkeit der technischen Implementierung des mobilen EUS beeinflusst. Die Nützlichkeit des entwickelten Artefakts wird mit Hilfe der Qualität der Implementierung als Indikator überprüft, da ein gut in Software umgesetztes Entscheidungsmodell dessen einfache Verwendung sicherstellt. Eine einfache Verwendung zeichnet sich hierbei durch automatisierte Berechnungen, Bewertungsvorschläge für Entscheidungsmerkmale oder durch unkomplizierte Anpassungsmöglichkeiten des Entscheidungsmodells aus. Eine Übersicht des Bezugsrahmens, der sich am TAM orientiert, ist in Abbildung 4.2 dargestellt. Die Konstrukte werden nachfolgend einzeln erläutert. Ihre Zusammenhänge werden im weiteren Verlauf der Evaluation überprüft.

Die *wahrgenommene Nützlichkeit* wird als das Maß definiert, wie sehr eine Person davon ausgeht, dass die Nutzung des implementierten Entscheidungsmodells seine bzw. ihre Leistung bei der Outsourcingentscheidung durch erhöhte Informationsqualität und einen strukturierten Verknüpfungsprozess der Entscheidungsgrößen verbessert (angelehnt an Davis et al., 1989). Er gibt Auskunft über den konzeptionellen Teil des Artefakts (Entscheidungsmodell) und ermöglicht es, Aussagen über die Qualität der Entscheidung zu treffen, die sich aus der Qualität der Entscheidungsgrößen und der zugehörigen Entscheidungslogik zusammensetzt (vgl. Kapitel 2.1.3).

Die *wahrgenommene Benutzerfreundlichkeit* ist das Maß, wie sehr eine Person davon ausgeht, dass die Nutzung des implementierten Entscheidungsmodells ohne zusätzlichen Aufwand ermöglicht wird (angelehnt an Davis et al., 1989). Sie ist damit ein Indikator für die Qualität der Implementierung und wie sehr dadurch Anwender zur Nutzung motiviert werden. Die Implementierung umfasst die Architektur sowie den Funktionsumfang von *SmartSourcer* (vgl. Kapitel 3.3).

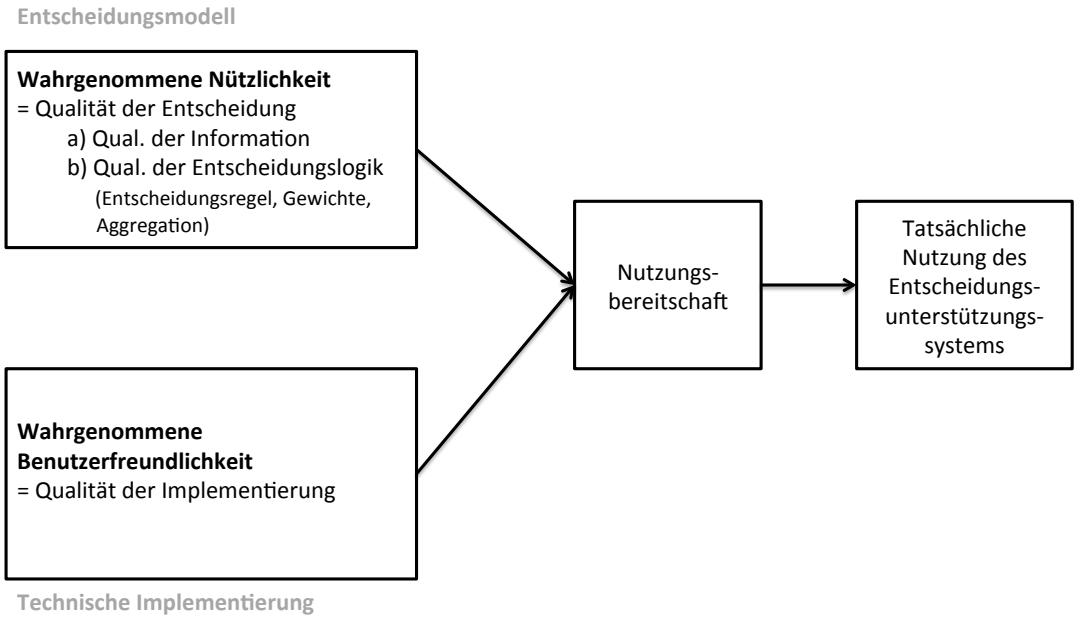

Abbildung 4.2.: Evaluationsmodell (abgeleitet von Davis et al., 1989; Venkatesh et al., 2003)

Die *Nutzungsbereitschaft* ist als das Maß definiert, wie sehr eine Person die konkrete Absicht bekundet, das betreffende Artefakt einzusetzen (angelehnt an Davis et al., 1989; Warshaw und Davis, 1985). Darüber lassen sich Erkenntnisse zur *tatsächlichen Nutzung des Systems zur Entscheidungsunterstützung* direkt ableiten.

Die wahrgenommene Benutzerfreundlichkeit wird mit Hilfe der *System Usability Scale* (SUS) gemessen. Sie wurde entwickelt, um eine Bewertung und Vergleichbarkeit der Benutzerfreundlichkeit von Produkten und digitalen Dienstleistungen zu ermöglichen (Bangor et al., 2008, 2009; Brooke, 1996). Diese Skala umfasst zehn Fragen (siehe Tabelle 4.3 und Tabelle 4.4) zur Benutzerfreundlichkeit eines Artefakts und ist damit ein Verfahren, um effizient eine vergleichbare Einschätzung der Nutzbarkeit einer Software zu erreichen. Die SUS stellt damit ein umfassendes Messinstrument bereit, das innerhalb des Evaluationsmodells dieser Arbeit zur Bewertung der wahrgenommenen Benutzerfreundlichkeit herangezogen wird.

4.2.2. Messgrößen

Um die tatsächliche Nutzungsbereitschaft der neuartigen Technik zur Entscheidungsunterstützung im Rahmen der Evaluation messen zu können, müssen den Konstrukten (vgl. Kapitel 4.2.1) Messgrößen zugeordnet werden. Dadurch können die Zusammenhänge im Modell überprüft und beschrieben werden.

Die wahrgenommene Nützlichkeit des Artefakts ist definiert als die *Qualität der Entscheidung* und ist das Maß für die wahrgenommene Verbesserung durch erhöhte Informationsqualität und einen strukturierten Verknüpfungsprozess der Entscheidungsgrößen. Bereits die Definition orientiert sich an dem zweistufigen Aufbau von Entscheidungsmodellen, der aus dem Entscheidungsfeld und der Entscheidungslogik besteht (vgl. Kapitel 2.1.3). Die Gesamtqualität der Entscheidung ergibt sich deshalb aus der *Qualität der Information* und der *Qualität der Entscheidungslogik*. Während die Informationsqualität die Richtigkeit und Stimmigkeit der selektierten Entscheidungsgrößen misst, gibt die Qualität der Entscheidungslogik an, ob die verwendete Logik verständlich und nachvollziehbar zum gewünschten Ergebnis führt.

Die Informationsqualität kann in drei eng damit zusammenhängenden Dimensionen gemessen werden: Informationsinhalt, Informationsformat und physischer Informationskontext (Auster und Choo, 1993; Culnan, 1985; Zmund, 1978). Wichtige Kriterien zur Messung des Informationsgehalts bestimmen, ob die vorliegende Information aktuell, umfassend, relevant und akkurat ist (Auster und Choo, 1993; Wang und Strong, 1996). Für das Format einer Information sind Kriterien wie Aufbereitung, Durchsuchbarkeit sowie eine verständliche Darstellung maßgeblich (Jeong und Lambert, 2001; Zmund, 1978). Um den physischen Informationskontext zu bewerten, wird die Einfachheit des Zugangs zum System und zu Informationen gemessen. Dies beinhaltet auch die Praktikabilität der Information im Zielkontext und den Zugang zu ergänzenden Informationen (Culnan, 1985). Für die Qualität der Information des Entscheidungsmodells ergeben sich daraus Indikatoren zur Einzelbewertung des Entscheidungsfelds hinsichtlich der Verständlichkeit, Aufbereitung und Darstellung von Entscheidungsgrößen und Aggregaten. Weiterhin beinhaltet die Kontextdimension Indikatoren zur Erreichbarkeit attributiver Informationen in Form von weiteren Experteneinschätzungen bezüg-

Tabelle 4.1.: Indikatoren der Informationsqualität

Kennzeichen	Indikator
M-IN-1	Relevant information is easy to reach within the system.
M-IN-2	I am given the opportunity to search for specific desired pieces of information.
M-IN-3	The information is presented in a format that makes it easy to understand.
M-IN-4	The information format encourages an intuitive conception of its meaning.
M-IN-5	Working with the app, I see the need to interact with other people in order to receive helpful information.
M-IN-6	I would like to perform this interaction (of M-IN-5) with others through the system.
M-IN-7	The app intuitively draws my attention to relevant information.
M-IN-8	It occurs that I am looking for certain information for some time before I can find it.

lich der Entscheidungsgrößen. Die Indikatoren der Informationsqualität sind in Tabelle 4.1 mit den Kennzeichen von *M-IN-1* bis *M-IN-8* dargestellt.

Zur Bestimmung der Qualität der Entscheidungslogik werden Kriterien herangezogen, die die Vorgehensweise und Verständlichkeit der Entscheidungsfunktion eines Entscheidungsmodells bewerten. Kernelemente dieser Qualitätsbewertung sind die verwendeten Rationalitätskalküle der Entscheidungslogik sowie Lerneffekte, die aus dem Entscheidungsprozess entstehen (Dean und Sharfman, 1996; McDaniels und Gregory, 2004). Abgeleitet für den Entscheidungsprozess im IT-Outsourcing überprüfen diese Kriterien die Entscheidungslogik hinsichtlich ihrer Korrektheit, Nachvollziehbarkeit und Anpassungsfähigkeit (Dean und Sharfman, 1996). Weiterhin wird die persönliche Einflussnahme der Entscheider und die Möglichkeit der Erweiterung ihrer Wissensbasis evaluiert (McDaniels und Gregory, 2004). Die Indikatoren der Qualität des Entscheidungsprozesses messen daher die Verständlichkeit der zugrundeliegenden Entscheidungslogik, die Fundierung der Berechnungsweise für die Aggregation, die Anpassungsfähigkeit der Logik, dadurch erzielte Lerneffekte auf den Anwender sowie die Effekte der Eingabe von

Tabelle 4.2.: Indikatoren der Qualität der Entscheidungslogik

Kennzeichen	Indikator
M-VP-1	Working with the app gave me an idea of the outsourcing decision making rationale embodied in the system.
M-VP-2	I believe that working with the system increases my individual skills.
M-VP-3	The evaluation criteria in the system encouraged me to consider factors I would not have considered without the app.
M-VP-4	The app allows for a rapid understanding of the decision rationale embodied.
M-VP-5	Working with the system to support the decision process was exhaustive.
M-VP-6	I feel that the app is based on a theoretically founded rationale.
M-VP-7	The app acknowledges user input and provides feedback on actions taken.
M-VP-8	It is easy to see how different user input effects the system's output.

Anwendern auf das Ergebnis. Sie sind in Tabelle 4.2 mit den Kennzeichen von *M-VP-1* bis *M-VP-8* dargestellt.

Die wahrgenommene Benutzerfreundlichkeit ist definiert als die *Qualität der Implementierung* und ist das Maß für die wahrgenommene Einfachheit der Nutzung des implementierten Entscheidungsmodells. Ein umfassendes Rahmenwerk zur Bestimmung der Implementierungsqualität wird mit dem Standard ISO 9126 bereitgestellt, in dem sich die Qualität von Softwaresystemen aus den Bereichen Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit zusammensetzt. Der Standard wird in der Literatur durch die Bereiche Integration und Wichtigkeit ergänzt (DeLone und McLean, 2003). Zur Erfragung und Vergleichbarkeit der Benutzerfreundlichkeit von Systemen hat Brooke (1996) zehn Indikatoren entwickelt, die über einen Zeitraum von 10 Jahren in über 2.000 Befragungen validiert wurden. Diese Indikatoren werden zur Bewertung der wahrgenommenen Benutzerfreundlichkeit des mobilen EUS herangezogen und sind in Tabelle 4.3 mit den Kennzeichen von *M-SU-1* bis *M-SU-9* dargestellt. Der zehnte Indikator misst die Nutzungsbereitschaft und wird nachfolgend

Tabelle 4.3.: Indikatoren der Qualität der Implementierung

Kennzeichen	Indikator
M-SU-1	I found the system unnecessarily complex.
M-SU-2	I thought the system was easy to use.
M-SU-3	I think that I would need the support of a technical person to be able to use this system.
M-SU-4	I found the various functions in this system were well integrated.
M-SU-5	I thought there was too much inconsistency in this system.
M-SU-6	I would imagine that most people would learn to use this system very quickly.
M-SU-7	I found the system very cumbersome to use.
M-SU-8	I felt very confident using the system.
M-SU-9	I needed to learn a lot of things before I could get going with this system.
M-IM-1	I am satisfied with the look and feel of the app.
M-IM-2	The app frequently becomes inaccessible or breaks down.
M-IM-3	My confusion with the system created situations when I did not know how to continue (break downs).

Tabelle 4.4.: Indikator der Nutzungsbereitschaft

Kennzeichen	Indikator
M-NB-1	I think that I would like to use this system frequently.

erläutert. Zur weiteren Abdeckung der Qualitätsmerkmale von Softwaresystemen (z.B. Zuverlässigkeit und Benutzbarkeit) wird die Liste um die Indikatoren mit den Kennzeichen von *M-IM-1* bis *M-IM-3* erweitert.

Die *Nutzungsbereitschaft* ist das Maß für die Nutzungsabsicht des neu entwickelten Entscheidungsmodells. Dieses Konstrukt wird in der Evaluation durch die direkte Frage nach der Nutzungsbereitschaft gemessen. Der zugehörige Indikator ist mit dem Kennzeichen M-NB-1 in Tabelle 4.4 gelistet und komplettiert damit die Fragen aus der SUS.

Alle Indikatoren werden auf einer fünfstufigen Likert-Skalierung gemessen, um die Zustimmung oder Ablehnung des Befragten zu erfassen (vgl. Tabelle 4.5).

Tabelle 4.5.: Wertzuweisung zu den Stufen der Likert-Skala

Likert-Skala	**Wert**
strongly disagree	1
disagree	2
neither agree nor disagree	3
agree	4
strongly agree	5

Es wird dabei vorausgesetzt, dass die Evaluationspersonen in der Lage sind, ihre Zustimmung oder ihre Ablehnung auf einer Skalierung mit gleichen Abständen auszudrücken. Die Indikatoren wurden in englischer Sprache formuliert, um eine Erhebung in international tätigen Unternehmen durchführen zu können, bei denen nicht alle Outsourcingmanager Deutsch sprechen.

4.2.3. Evaluationsstrategie und Instrumente

Zur Bewertung des in dieser Arbeit entwickelten Artefakts wird das Evaluationsmodell inklusive seiner Messgrößen (vgl. Kapitel 4.2.1 und 4.2.2) verwendet, um die Qualität der Entscheidung, die Qualität der Implementierung und schließlich die Nutzungsbereitschaft der innovativen Entscheidungstechnik durch erfahrene Anwender zu ermitteln. *SmartSourcer* ist eine technische Implementierung bzw. Instanziierung eines neuartigen Entscheidungsmodells zur Unterstützung von Outsourcingentscheidungen. Weil dieses Artefakt einen neuen Lösungsansatz zu einem bisher nicht betrachteten Problem darstellt, ist eine detaillierte und systematische Evaluation erforderlich. Daher wird die Nutzungsbereitschaft anhand der adaptierten Kernmerkmale des TAM (vgl. Abbildung 4.2) mit deskriptiven statistischen Verfahren evaluiert. Weiterhin ermöglicht die Bereitstellung eines Prototyps unter realen Bedingungen die Erhebung qualitativer Daten, die Rückschlüsse auf die Ergebnisse der deskriptiven Analyse zulassen und mögliche Gründe der Bewertungen offenlegen. So lassen sich funktionale Schwachstellen sowie Möglichkeiten zur Verbesserung des Artefakts aufdecken.

Zunächst wird in einer künstlichen *ex post* Evaluation der vollständige Prototyp von *SmartSourcer* im Rahmen eines Experiments Masterstudenten und Doktoranden der Fächer Wirtschaftsinformatik, Betriebswirtschaftslehre und Volkswirtschaftslehre als Proxy-Anwender anhand eines beispielhaften Problems über-

prüft. Dadurch wird das Entscheidungsmodell und dessen Implementierung in einem kontrollierten Setting beurteilt (experimentelle Evaluation). Dies entspricht gemäß Sonnenberg und vom Brocke (2012) jener Aktivität, die zur initialen Demonstration des Artefakts und zur Bewertung seiner Leistungsfähigkeit dient (Sonnenberg und vom Brocke, 2012, S. 77). Zusätzlich prüft dieser Schritt das Design des betreffenden Artefakts und liefert wertvolle Hinweise für die Verbesserung des Prototyps. Eine Konstruktionsphase, die das Artefakt hinsichtlich des wichtigsten Feedbacks verbessert, das im Experiment eingesammelt wurde, vervollständigt den Designzyklus und verleiht der Evaluation einen meliorisierenden Charakter.

Das so präzisierte Artefakt wird anschließend in einer naturalistischen *ex post* Evaluation bewertet. Dabei wird der organisatorische Kontext erfasst und durch branchenspezifisches Wissen sowie Expertise der realen Anwender ergänzt, die das Artefakt in seiner vorgesehenen betrienlichen Umgebung anwenden (Expertenevaluation). Diese Phase entspricht der finalen Evaluationsaktivität aus dem zyklischen Forschungsprozess von Sonnenberg und vom Brocke (2012) und liefert wertvolle Hinweise, ob ein Artefakt für den praktischen Einsatz anwendbar bzw. nützlich ist. Ferner stimuliert die Aufdeckung von Verbesserungsmöglichkeiten das Anstoßen zukünftiger Designzyklen.

In der naturalistischen *ex post* Evaluation spielen reale Anwender, die Branchenexperten sind und den Prototyp in seiner bestimmungsgemäßen Umgebung ausprobieren, die wichtigste Rolle für ein rigoroses Evaluationsergebnis. Da der Einsatz des Artefakts in seiner natürlichen Umgebung, also als Teil der Kollaborationstechnologie eines Softwareunternehmens, kritische Veränderungen in der Systemlandschaft eines Unternehmens erfordert, ist die Durchführung der Evaluation durch Aufgabenträger in der Praxis nur eingeschränkt möglich. Es konnten die Mitarbeiter von zwei mittelständischen Softwarefirmen gewonnen werden, die Teile ihrer Entwicklung auslagern und jahrelange Erfahrung mit dem Thema Outsourcing von Softwarekomponenten aufweisen können. Dadurch konnten für diese naturalistische Evaluation Interviewdaten von Projektleitern, Account-Managern und Architekten der Unternehmen in einer Fallstudie analysiert werden.

Bei der Zuordnung der beiden Strategiequadranten auf das Klassifikationsschema zur Strategiewahl von Venable et al. (2012) zeichnet sich ein Laborexperiment mit der neuen Entscheidungstechnik als geeignete Variante für die experimentelle

Evaluation ab, wohingegen sich eine Fallstudie für die Expertenevaluation eignet. Laborexperimente reduzieren Komplexität, stellen künstliche Einstellungen sicher und werden daher als geeignet für die zielgerichtete Bewertung von interessanten Variablen erachtet, die nicht unternehmensbezogen sind (Easterbrook et al., 2008). Im Gegensatz dazu untersucht eine Fallstudie

> *„ein gegenwärtiges Phänomen innerhalb eines Kontextes aus dem realen Leben, besonders wenn die Grenzen zwischen Phänomen und Kontext nicht klar ersichtlich sind“* (Yin, 2009, S. 18).

Über den soziotechnischen Kontext des Artefakts hinaus ermöglichen Fallstudien auch die Kombination von deskriptiven und explorativen Befragungen für die Evaluation. Die ausgewählten Methoden entsprechen ebenfalls den Empfehlungen für die Evaluationsphasen in der Arbeit von Sonnenberg und vom Brocke (2012) und wurden in Bezug auf den Evaluationszweck dieser Arbeit angepasst.

Für die deskriptive Evaluation des Entscheidungsmodells und des mobilen EUS (vgl. Kapitel 4.2.1) werden die Indikatoren der Konstrukte (vgl. Kapitel 4.2.2) auf einer fünfstufigen Likert-Skala von Ablehnung bis Zustimmung (vgl. Tab. 4.5) ermittelt. Der Fragebogen der künstlichen und naturalistischen Evaluation des Artefakts ist im Anhang G abgedruckt.

In der qualitativen Auswertung im Rahmen dieser Evaluation wird ein enger Fokus auf das implementierte EUS gelegt. Sie dient als Ausgangsbasis für die Durchführung eines zusätzlichen Designzyklus, um das mobile EUS zu verbessern und um Ideen für zukünftige Weiterentwicklungen der Entscheidungstechnik zu liefern. Das Rahmenwerk von Palmius (2007) zum Messen und Vergleichen von Informationssystemen wird hierfür herangezogen, da es umfassende Evaluationskriterien zur Bewertung der Nützlichkeit (*utility*) und Nutzbarkeit (*usability*) enthält (Sonnenberg und vom Brocke, 2012). Es fasst Kriterien zum Messen und Vergleichen von Informationssystemen zusammen. Im Sinne eines betrieblich anwendbaren Ansatzes wurden Qualitätsmerkmale für Informationssysteme von vorherigen Qualitätsmodellen, internationalen Standards, aus der Mensch-Computer-Interaktionsforschung, aus der Forschung zur Informationsqualität sowie anderen Ideenschmieden bei Palmius (2007) zusammengeführt. Die Kriterien sind anhand organisatorischer (*organization*), individueller (*individual*), informationsbezogener (*information*), technologischer (*technology*) und systemischer

(*systemics*) Aspekte unterteilt. Diese Gliederung wird verwendet, um den im Anhang H abgedruckten Fragebogen zu strukturieren. Die darin enthaltenen Fragen repräsentieren die Evaluationskriterien von Palmius (2007) und werden hinter jeder Frage referenziert.

4.2.3.1. Künstliche ex post Evaluation: Studentenexperiment

Die experimentelle Evaluation in dieser Arbeit wird als künstliche *ex post* Evaluation durchgeführt, um in einem ersten Schritt zu reflektieren, wie umfangreich die Definition des Entscheidungsmodells und der zugehörigen Verknüpfungslogik ist und wie gut der Prototyp in Bezug auf Faktoren funktioniert, die nicht unternehmensbezogen sind. Zusätzlich soll sie Feedback für eine zwischengelagerte Konstruktionsphase zur Verfeinerung des EUS geben. Dafür werden die Indikatoren des Bezugsrahmens (vgl. Kapitel 4.2.2) evaluiert und Feedback zur Implementierung in semistrukturierten Interviews eingeholt. Als Teilnehmer für das Laborexperiment werden Masterstudenten und Doktoranden der Wirtschaftsinformatik (WI) sowie Wirtschaftswissenschaftler (WiWi) aus den Bereichen Betriebswirtschaftslehre (BWL) und Volkswirtschaftslehre (VWL) eingeladen. Eine Übersicht der Teilnehmer ist in Tabelle 4.6 dargestellt.

Die Experimente sind nach dem folgenden Schema aufgebaut: Zuerst wird den Teilnehmern eine einführende Präsentation über das Treffen von Outsourcingentscheidungen in der komponentenbasierten Softwareentwicklung sowie dem entwickelten Entscheidungsmodell gegeben, das im *SmartSourcer* verankert ist, um diese mit dem notwendigen Hintergrundwissen über das Thema auszustatten. Sie sollen sich anschließend mit dem Aussehen und der Funktionsweise von *SmartSourcer* vertraut machen, indem sie individuell alle Funktionen und Bedienelemente der Software aufgezeigt bekommen und ausprobieren können. Sie werden gleichzeitig ermutigt, kritisches qualitatives Feedback zum Design, der Positionierung von Bedienelementen, dem Aussehen, dem Bedienkonzept und dem intuitiven Charakter zu geben. Der Inhalt des Feedbacks wird unmittelbar schriftlich festgehalten. Anschließend erhalten die Teilnehmer Aufgaben, die sie unter Verwendung des mobilen EUS sowie einem vorgegebenen Softwareentwicklungsszenario durchführen müssen. Dafür ist in der Kollaborationsplattform *CodeBeamer* im Vorfeld ein Projekt erstellt worden, das die Entwicklung eines Kundenbeziehungs-

Tabelle 4.6.: Teilnehmer der experimentellen Evaluation

Lfd. Nr.	Alter	m/w	Tätigkeit	Studium	Nutzung mobiler Technologie	Outsourcing-bezug
1	23	m	Student	WI	täglich	B
2	23	m	Student	WI	täglich	A
3	29	m	Doktorand	WI	täglich	A
4	31	m	Doktorand	WI	täglich	B
5	23	m	Student	WI	täglich	B
6	23	m	Student	WiWi	täglich	A
7	23	m	Student	WI	manchmal	B
8	23	m	Student	WI	täglich	C
9	22	m	Student	WI	wöchentlich	A
10	22	w	Student	WI	täglich	B
11	32	m	Doktorand	WiWi	täglich	B
12	31	m	Doktorand	WiWi	täglich	B
13	23	w	Student	WiWi	täglich	A
14	23	w	Student	WI	täglich	B
15	32	m	Habilitand	WI	täglich	C

A Kein

B Im Rahmen von Vorlesungen und Übungen

C In der Berufspraxis

managementsystems zum Ziel hat. Sowohl die Anforderungen an die Software als auch das Design der Softwarekomponenten des geplanten Systems sind dabei in der Plattform hinterlegt. Der so gespeicherte Zustand des Entwicklungsprojekts kann dann für alle Teilnehmer des Experiments auf den gleichen Ursprungszustand zurückgesetzt werden. Die Aufgabe jedes einzelnen Teilnehmers besteht daher aus der Verwendung von *SmartSourcer* zum Laden des aktuellen Projektstands, dem vollständigen Bewerten aller Softwarekomponenten des Projekts auf der Grundlage des konfigurierten und gewichteten Entscheidungsmodells sowie aus der finalen Entscheidungsfindung darüber, welche der Komponenten selbst entwickelt werden würden und welche ausgelagert werden sollten. Für die Zeit während der Durchführung der Aufgabe werden die Nutzer gebeten, kritisches und qualitatives Feedback durch regelmäßige Aussprache ihrer Gedanken zu ge-

ben (Holzinger, 2005). Nach Beendigung der Aufgaben erfolgt die Bewertung der Entscheidungstechnik auf der Basis des Fragebogens für die deskriptive Evaluation (siehe Anhang G) sowie des Interviewleitfadens für die qualitative Evaluation (siehe Anhang H).

4.2.3.2. Naturalistische ex post Evaluation: Expertenbefragung

Die Expertenevaluation wurde als naturalistische *ex post* Evaluation durchgeführt, um die Nutzungsbereitschaft der entwickelten Entscheidungstechnik im Unternehmenskontext zu erforschen. Auch hier wurden Daten zur deskriptiven und qualitativen Analyse von *SmartSourcer* und dem zugehörigen Entscheidungsmodell erhoben. Für die Expertenevaluation konnten Mitarbeiter von Softwareunternehmen gewonnen werden, die in ihrer täglichen Arbeit mit Outsourcingentscheidungen in der komponentenbasierten Softwareentwicklung konfrontiert sind. Die Unternehmen der befragten Mitarbeiter sind auf die Entwicklung von Anwendungssoftware spezialisiert und haben bestehende Outsourcingbeziehungen nach Osteuropa. Vor diesem Hintergrund werden den Teilnehmern entsprechendes Wissen und Erfahrungen zugeordnet, um besonders die Qualität des Entscheidungsmodells und der zugehörigen Implementierung kritisch bewerten und im qualitativen Teil hilfreiches Feedback für die Weiterentwicklung der Technologie zur Entscheidungsunterstützung geben zu können. Eine Übersicht über die Teilnehmer der Expertenevaluation mit ihrer zugehörigen Firma, ihrem Alter, ihrem Geschlecht, ihrer Jobbeschreibung sowie ihren Erfahrungen mit Outsourcing (in Jahren) ist in Tabelle 4.7 abgebildet. Die Firmen wurden auf eigenen Wunsch anonymisiert.

Die Vorgehensweise bei der Expertenbefragung besteht aus einer einführenden Präsentation, einem gemeinsamen Durchgang durch die Funktionalitäten der App, einer realen Aufgabe zur Durchführung einer Entscheidungsfindung sowie der Erhebung der deskriptiven und qualitativen Daten in Form von Fragebögen und halbstrukturierten Interviews. Die Teilnehmer der Firma B können aus zeitlichen Gründen nicht an den Einzelinterviews (vgl. Anhang H) für die qualitative Evaluation teilnehmen, haben jedoch den Fragebogen (vgl. Anhang G) ausgefüllt, der bereits bei der künstlichen Evaluation verwendet wurde. Das verwendete Instrument zur Datenerhebung ist daher eng an den Aufbau der La-

Tabelle 4.7.: Teilnehmer der Expertenevaluation

Lfd. Nr.	Firma	m/w	Alter	Berufsbezeichnung	Outsourcing-erfahrung
1	A	m	–	Senior Account Manager	10
2	A	m	–	Project Manager	1
3	A	m	–	Key Account Manager	5
4	B	m	47	Managing Consultant	10
5	B	m	36	Leading Technical Consultant	5
6	B	m	40	Project Manager	7
7	B	m	41	Project Manager	3
8	B	m	38	Leading Technical Consultant	5
9	B	m	33	Senior Technical Consultant	5
10	B	m	40	Project Manager	8
11	B	m	39	Leading Technical Consultant	7
12	B	m	45	Managing Consultant	13
13	B	m	37	Leading Technical Consultant	6
14	B	m	34	Senior Technical Consultant	4
15	B	m	42	Project Manager	9

borexperimente angelehnt. Jeder Experte wird zunächst über das der App zugrundeliegende Entscheidungsmodell aufgeklärt und anschließend in die Funktionsweise von *SmartSourcer* eingeführt. Anschließend müssen die Komponenten eines beispielhaften Softwareprojekts (Entwicklung eines Kundenbeziehungsmanagementsystems) analysiert und nach realen Anforderungen sowie auf der Basis der vorgegebenen Merkmale durch die Entscheidungsunterstützungstechnologie bewertet werden. Die Aufgabe für die Experten wird aus dem Experiment übernommen und gleichermaßen durchgeführt. Eine Bewertung der Ergebnisse trägt zum Verständnis über die im System verankerte Entscheidungsmethode bei. Über das Beobachten ihrer Interaktion mit dem System hinaus werden die Teilnehmer zusätzlich aufgefordert, während der gesamten Nutzungsdauer der App ihre Gedanken zu artikulieren und kritisches qualitatives Feedback zum Design, der

Benutzeroberfläche, dem Aussehen sowie dem intuitiven Empfinden der App zu äußern (Holzinger, 2005). Das Feedback wird unmittelbar schriftlich festgehalten.

4.3. Ergebnisse

Zunächst werden die Ergebnisse der Expertenbefragung und der Laborexperimente ausgewertet und analysiert. Nach Auswertung dieser Daten sowie Beschreibung der Ergebnisse werden durch einen Vergleich der beiden evaluierten Zielgruppen Gemeinsamkeiten und Unterschiede hervorgehoben. Diese werden dann im Licht der verfügbaren qualitativen Daten diskutiert.

4.3.1. Auswertung der deskriptiven Statistiken

Für die deskriptive Analyse der Evaluation werden die Ergebnisse aus den Fragebögen der Teilnehmer der Experimente und der Expertenbefragungen in ihrem theoretischen Bezugsrahmen (vgl. Kapitel 4.2.1) ausgewertet. Die Einzelergebnisse der Indikatoren geben dabei jeweils ein Maß für ihre Konstrukte an, die wiederum Bestandteile des Bewertungsmodells dieser Arbeit sind (vgl. Kapitel 4.2.2). Für die Analyse wird die Software *IBM SPSS*[1] verwendet.

Tabelle 4.8.: Aggregiertes Ergebnis der deskriptiven Analyse durch Studenten und Experten

		utility	usability	M_NB_1
N	Gültig	30	30	30
	Fehlend	0	0	0
Mittelwert		4,0576	4,4861	4,27
Standardabweichung		,36660	,23781	,640
Minimum		3,31	4,00	3
Maximum		4,75	4,92	5

utility Wahrgenommene Nützlichkeit

usability Wahrgenommene Benutzerfreundlichkeit

M_NB_1 Nutzungsbereitschaft

1 http://www.ibm.com/software/de/analytics/spss/products/statistics/ (abgerufen am 10.12.2015)

Im Rahmen der Analyse werden zunächst die Gesamtergebnisse der Erhebung beider Evaluationsgruppen zusammen dargestellt. Aus Gründen der Übersichtlichkeit sind die entsprechenden Tabellen im Anhang I abgedruckt. In der entsprechenden Tabelle I.3 sind die Ergebnisse nach ihren Konstrukten und den zugehörigen Indikatoren gegliedert und geben den Mittelwert, die Standardabweichung, den Stichprobenumfang sowie den Maximal- und Minimalwert der abgegebenen Antworten an. Eine aggregierte Form der Tabellen ist in Tabelle 4.8[2] dargestellt und fasst die Ergebnisse aus dem Anhang zusammen. Dabei fassen *utility* und *usability* die wahrgenommene Nützlichkeit sowie die wahrgenommene Benutzerfreundlichkeit zusammen. Die fünf Stufen der Likert-Skala wurden dabei durchgängig für diese Arbeit mit den Werten aus Tabelle 4.5 belegt. Weiterhin werden die Ergebnisse der beiden Evaluationsgruppen gesondert in Tabelle I.4 für die Teilnehmer des Studentenexperiments und in Tabelle I.5 für die Befragung der Outsourcingexperten aufgezeigt.

Nach Auswertung der Ergebnisse lassen sich die arithmetischen Mittelwerte der Indikatoren und schließlich der Konstrukte des theoretischen Bezugssystems dieser Evaluation (vgl. Kapitel 4.2.1) berechnen. Sie werden im Bewertungsmodell zur Beschreibung der Konstrukte und deren Beziehungen untereinander verwendet. Die arithmetischen Mittelwerte des Evaluationsmodells sind in Abbildung 4.3 dargestellt. Sie fasst die Bewertungen sowohl der Studentenexperimente als auch der Expertenbefragungen zusammen.

Die **wahrgenommene Benutzerfreundlichkeit** ist das am besten bewertete Konstrukt in dieser Evaluation. Sie drückt die Qualität der Implementierung aus und konstatiert mit einem durchschnittlichen Wert von *4,49* eine gute bis sehr gute Umsetzung des mobilen EUS. Der direkte Vergleich der Meinungen zwischen den Experten (*4,43*) und den Teilnehmern des Experiments (*4,54*) zeigt, dass auch hier kaum Differenzen bei der wahrgenommenen Benutzerfreund-

2 Die Ergebniswerte einiger abgefragter Indikatoren mussten für die Analyse invertiert werden, weil die Fragen negativ formuliert waren. So hat die Bewertung der Aussage, dass das System unnötig komplex ist (Kennzeichen *M-SU-1*), einen negativen Einfluss auf die Qualität der Implementierung also die wahrgenommene Benutzerfreundlichkeit. Wird diese Aussage dann mit Ablehnung bewertet, so bedeutet dies jedoch Zustimmung zur Benutzerfreundlichkeit und muss folgerichtig korrigiert werden. Die betreffenden Ergebniswerte wurden daher mit der Formel $(6 - x)$ korrigiert, um für die korrekte Berechnung der Konstrukte verwendet werden zu können. Die betroffenen Indikatoren haben die Kennzeichen: *M-IN-8*, *M-VP-5*, *M-SU-1*, *M-SU-3*, *M-SU-5*, *M-SU-7*, *M-SU-9*, *M-IM-2* und *M-IM-3*.

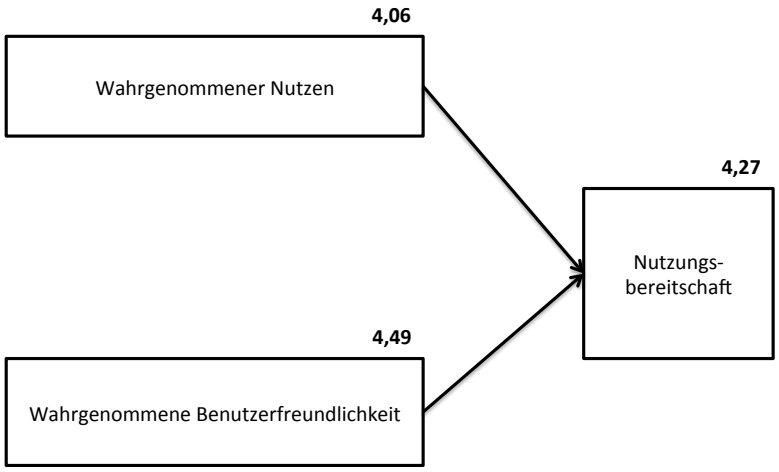

Abbildung 4.3.: Evaluationsmodell mit Ergebnissen

lichkeit auftreten und beide Gruppen eine relativ starke Zustimmung zu diesem Konstrukt zum Ausdruck bringen. Besonders starke Zustimmung der Teilnehmer erfahren die Indikatoren *M-SU-1* (geringe Komplexität), *M-IM-2* und *M-IM-3* (Stabilität), die Aussagen über die Komplexität und Stabilität des Systems erfassen. Offensichtlich arbeitete die Implementierung von *SmartSourcer* in den Tests zuverlässig und hatte keine Probleme im Zusammenspiel mit der verknüpften Kollaborationsplattform, so dass ihr eine hohe Stabilität attestiert wurden. Die starke Zustimmung zu geringer Komplexität wird als einfache Verständlichkeit des Systems zur Entscheidungsunterstützung interpretiert und legt damit offen, dass eine weitgehend intuitive Verwendung für Personen mit Outsourcingbezug möglich ist. Im direkten Vergleich der Bewertungen aus den Experimenten und der Expertenevaluation wird zusätzlich ersichtlich, dass die Stabilität des Systems von den Expertenanwendern mit *4,87* leicht höher eingeschätzt wird als bei den Teilnehmern des Experiments (*4,80*). Im Gegensatz dazu wird von den durchschnittlich jüngeren Teilnehmern des Experiments mit *4,87* eine bessere Verständlichkeit attestiert als von den im Durchschnitt älteren Experten mit *4,53*.

Relativ niedrige Werte zur Berechnung des Mittelwerts lassen sich bei den Indikatoren *M-SU-7*, *M-SU-8* und *M-IM-1* anführen. Diese Indikatoren drücken den vertrauten Umgang mit dem EUS sowie die Wahrnehmung von Optik und Haptik aus. Der vertraute und sichere Umgang mit dem System stellt mit den Werten *4,20* und *4,27* der Indikatoren *M-SU-7* und *M-SU-8* das als am niedrigs-

Tabelle 4.9.: Berechnung der *System Usability Scale* (SUS)

Indikator	**Studenten-experiment**	**Experten-befragung**	**Gesamt**
M-NB-1	3,40	3,13	3,27
M-SU-1	3,87	3,53	3,70
M-SU-2	3,60	3,47	3,53
M-SU-3	3,60	3,53	3,57
M-SU-4	3,27	3,47	3,37
M-SU-5	3,47	3,67	3,57
M-SU-6	3,73	3,27	3,50
M-SU-7	3,27	3,13	3,20
M-SU-8	3,27	3,27	3,27
M-SU-9	3,67	3,13	3,40
Summe	**35,13**	**33,60**	**34,37**
SUS Punktzahl	**87,83**	**84,00**	**85,92**

ten bewertete Kriterium der Implementierungsqualität dar. Die Vorgehensweise der Evaluation (vgl. Kapitel 4.2.3) lässt jedoch nur geringen Spielraum, um einen vertrauten Umgang zu bewerten, weil die Tester mit einem neuen EUS konfrontiert wurden und nur einen begrenzten Zeitraum zur Bewertung zur Verfügung hatten. Der für dieses Konstrukt ebenfalls niedrige Wert *4,30* des Indikators *M-IM-1* zur Bewertung von Optik und Haptik ergibt sich aus den sehr subjektiven Meinungen von Anwendern sowie deren Vorlieben für den optischen Aufbau von mobilen Anwendungen oder verwendeten Farben, Symbolen und Grafiken. Vergleicht man die Experimentteilnehmer mit den Experten, so sind die Experten (*4,07*) offenbar wählerischer beim Aussehen von *SmartSourcer* wie die Gruppe des Experiments (*4,53*). Beim vertrauten Umgang mit der neuen Technologie gibt es hingegen keine erwähnenswerten Unterschiede. Zusammenfassend muss jedoch festgehalten werden, dass selbst die schwachen Indikatoren der Implementationsqualität stets einen Wert über *4,00* besitzen und damit immer noch eine relativ hohe Qualität der Implementierung signalisieren.

Die Bewertung der Qualität der Implementierung wird durch die Einbeziehung der SUS (vgl Kapitel 4.2.1) zusätzlich bestätigt. Dafür müssen die zehn durch Brooke (1996) definierten Messgrößen auf einen standardisierten Wert gebracht

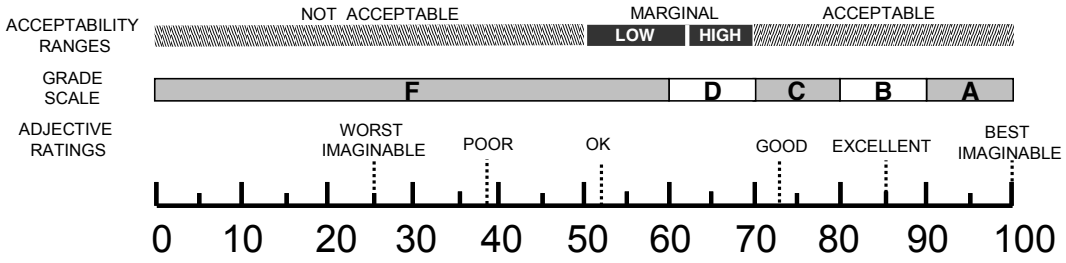

Abbildung 4.4.: SUS-Auswertung (übernommen von Bangor et al., 2009)

werden, um die endgültige Punktzahl der SUS[3] zu berechnen. Dieser errechnet sich für die Indikatoren *M-NB-1* (vgl. Tabelle 4.4) sowie *M-SU-1* bis *M-SU-9* (vgl. Tabelle 4.3). Die inversen Skalen dieser Indikatoren werden bei der Standardisierung berücksichtigt. Deshalb gilt für die Indikatoren *M-NB-1*, *M-SU-2*, *M-SU-4*, *M-SU-6* und *M-SU-8* die Formel $x - 1$, wobei x den errechneten Mittelwert des Indikators angibt. Für die Indikatoren *M-SU-1*, *M-SU-3*, *M-SU-5*, *M-SU-7* und *M-SU-9* gilt folglich die Formel $5 - x$. Der standardisierte Wert liegt daher zwischen 0 und 4. Anschließend werden die Werte der zehn Indikatoren addiert und mit 2,5 multipliziert. Die erreichbare Punktzahl der SUS liegt dann zwischen 0 und 100 (Bangor et al., 2008; Brooke, 1996). Die Berechnung der Punktzahl der SUS für die Teilnehmer des Experiments, für die Experten sowie für alle Evaluationsteilnehmer sind in Tabelle 4.9 aufgelistet. Darin werden bereits die standardisierten Werte verwendet, deren Berechnung in diesem Abschnitt beschrieben wurde.

Die Ergebnisinterpretation erfolgt auf der Basis des Schemas zur Klassifizierung der SUS-Punktzahl, das in Abbildung 4.4 dargestellt ist (Bangor et al., 2008, 2009). Dementsprechend liefert die Auswertung der Ergebnisse der experimentellen Evaluation für die SUS eine Punktzahl von *87,83*. Dies entspricht auf der Akzeptanzskala des Schemas einer akzeptablen Benutzerfreundlichkeit des Systems und lässt sich lt. Bangor et al. (2008) als „*exzellent*“ bezeichnen. Die erzielte Punktzahl von *84,00* bei der Expertenevaluation entspricht in diesem Bewertungsschema ebenfalls einer akzeptablen Lösung, die sich lt. Bangor

3 Das Vorgehen zur Berechnung entspricht der von Bangor et al. (2008) definierten Berechnungsweise und wird detailliert für die Verwendung im Rahmen der Evaluation dieser Arbeit beschrieben.

et al. (2008) als „*gut*“ beschreiben lässt. Die insgesamt erreichte Punktzahl aller Evaluationsteilnehmer von *85,92* kann wiederum unter Berücksichtigung der betreffenden Systematik als „*exzellent*“ eingestuft werden und unterstreicht das hohe Maß der wahrgenommenen Benutzerfreundlichkeit.

Die **wahrgenommene Nützlichkeit** zur Bewertung der Qualität der Entscheidung wird mit einem durchschnittlichen Wert von *4,06* als ebenfalls gut bewertet (vgl. Tabelle 4.8). Auch wenn der Unterschied zwischen den Meinungen der Experten (*4,03*) und der Experimentteilnehmer (*4,01*) gering ist, bewerten die Experten die Qualität des Entscheidungsmodells (Information und Logik) leicht besser (vgl. Tabelle I.4 und Tabelle I.5 im Anhang I). Ihr täglicher Umgang mit solchen Entscheidungen spricht ihrer Aussage allerdings eine hohe Bedeutung zu. Am stärksten wird die wahrgenommene Nützlichkeit durch die Indikatoren *M-IN-3* und *M-VP-6* (theoretische Fundierung der Entscheidungslogik) gestützt. Die Experten attestieren dem Artefakt eine solide und fundierte Entscheidungslogik, um damit den Bedarf aus dem organisatorischen Umfeld zu decken.

Negativ fällt bei der Messung dieses Konstrukts die Bewertung der relativ geringen Interaktionsmöglichkeit bei Nutzung des Systems mit anderen Aufgabenträgern auf (*M-IN-6*). Hierfür wird ein Wert von *2,65* gemessen und liegt damit als einziger Mittelwert im Bereich der Ablehnung auf der verwendeten Skala. Da dieser Indikator ähnlich zum Indikator *M-IN-5* (Bewertung des Interaktionsbedarfs für Gruppenentscheidungen) ist, werden diese gemeinsam analysiert. Generell wird diesem Interaktionsbedarf mit *3,33* eine Zustimmung auf geringem Niveau bescheinigt. Die Kommunikation über *SmartSourcer* wird allerdings abgelehnt. Die stärkere Ablehnung der Experten mit *2,53* im Gegensatz zu *2,82* der Studenten präzisiert diesen Sachverhalt.

Die **wahrgenommene Nutzungsbereitschaft** zur Bewertung der Gesamtqualität der neuartigen Entscheidungstechnik, die aus einem vordefinierten Entscheidungsmodell und einer zugehörigen Implementierung als mobiles EUS besteht, wird durch den Indikator *M-NB-1* gemessen und wird insgesamt mit dem Mittelwert *4,27* bewertet (vgl. Tabelle 4.8). Damit wird dem innovativen Artefakt dieser Arbeit sowohl von den Experten der Evaluation als auch von den Teilnehmern der Laborexperimente ein hohes Maß an Nutzungsbereitschaft zugesprochen. Im Vergleich zu den Ergebnissen aus dem Experiment sind die Experten mit ihrer Bewertung der Nutzungsbereitschaft von *4,13* etwas skeptischer als die

Teilnehmer des Experiments (*4,40*). Jedoch bleibt festzuhalten, dass keiner der Experten eine Ablehnung der Nutzung signalisiert hat. Denn der geringste Wert aller Messungen beträgt über alle Teilnehmer hinweg *3* und bedeutet weder Zustimmung noch Ablehnung. Beim Experiment lassen alle Teilnehmer eine hohe Nutzungsbereitschaft erkennen (vgl. Anhang I).

Die bisherige Analyse der Ergebnisse legt marginale Unterschiede bei den Bewertungen der beiden befragten Gruppen offen. Ob dies statistisch untermauert werden kann, wurde mittels eines t-Tests unter Verwendung der Software SPSS überprüft. Die gesamte Ergebnistabelle ist im Anhang I als Tabelle I.2 abgebildet. Die Auswertung ergibt keinen signifikanten Unterschied zwischen den Mittelwerten beider Stichproben. Es kann also nicht bestätigt werden, dass die beiden Stichproben aus unterschiedlichen Grundgesamtheiten stammen. Aus diesem Grund können keine verlässlichen Aussagen über die Unterschiede bei den Mittelwerten der beiden Gruppen getroffen worden (Gibbons, 1985). Es gilt jedoch zu berücksichtigen, dass bei einer Stichprobengröße von $N = 15$ (pro Gruppe) der t-Test nur bedingte Validität aufweist.

Um den Zusammenhang der Konstrukte des Bewertungsmodells sowie ihre Signifikanz zu überprüfen, werden die Daten in SPSS mit einer Analyse nach Pearson weiter untersucht. Die Ergebnisse davon sind in Tabelle 4.10 abgebildet und ergänzen die bisherigen deskriptiven Ausführungen in diesem Kapitel.

Tabelle 4.10.: Korrelationsmatrix

		Alpha	Mittelw.	Std. abw.	Min.	Max.	1	2	3
1	Nutzungsbereitschaft	-	4,27	0,64	3,00	5,00	1,00		
2	Nützlichkeit	0,78	4,06	0,37	3,31	4,75	0,52**	1,00	
3	Benutzerfreundlichkeit	0,62	4,51	0,31	3,63	5,00	0,40*	0,43*	1,00

N = 30, * p < 0.05, ** p < 0.01

Die Korrelationsmatrix[4] legt offen, dass der Zusammenhang zwischen der wahrgenommenen Nützlichkeit und der Nutzungsbereitschaft signifikant ist. Gleiches

4 Zusätzlich wurden die Variablen *Alter*, *Geschlecht* und *Erfahrung* als Kontrollvariablen in die Analyse mit aufgenommen. Die Ergebnisse zeigen, dass das Alter nicht mit der Benutzerfreundlichkeit oder der Nutzungsbereitschaft korreliert. Damit kann ausgeschlossen werden, dass es Altersgruppen gab, die nicht mit der Nutzung und dem Umgang mobiler Technologien vertraut waren oder den Prototyp weniger benutzerfreundlich fanden als andere und damit das Ergebnis beeinflusst hätten. Andererseits erhöht die signifikante Korrelation zwischen

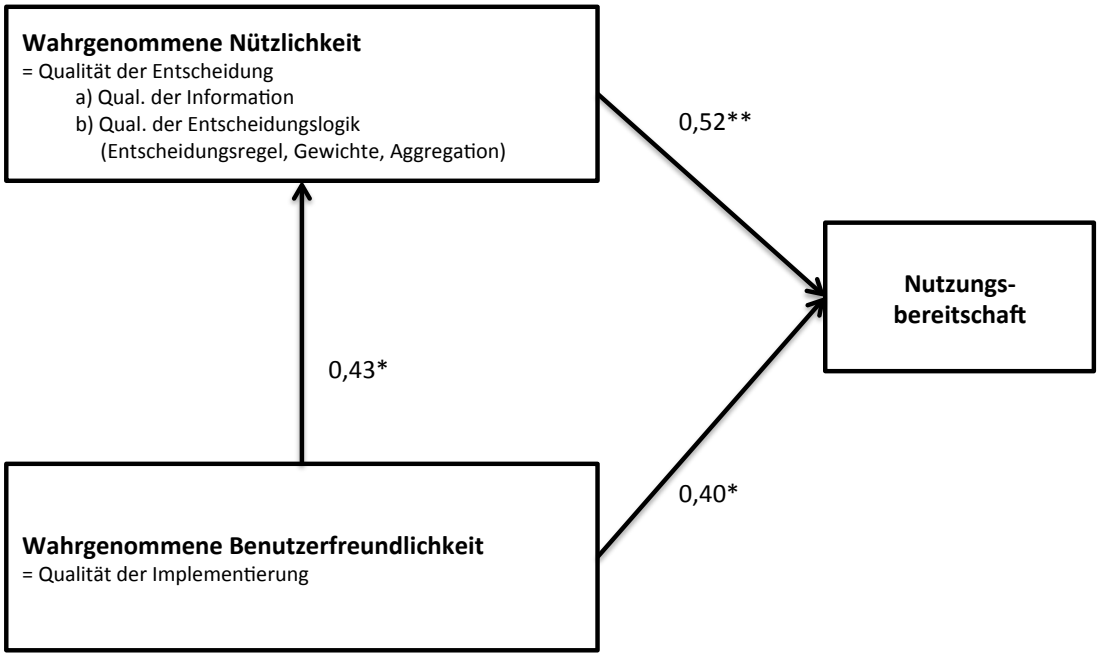

Abbildung 4.5.: Evaluationsmodell mit Effektstärken

gilt für die wahrgenommene Benutzerfreundlichkeit und der Nutzungsbereitschaft des Artefakts. Es kann damit also für das Entscheidungsmodell und seine Implementierung gezeigt werden, dass einerseits eine hohe Nützlichkeit durch die darunterliegende Entscheidungslogik sowie eine gute Nutzbarkeit des Prototyps eine hohe Nutzungsbereitschaft der Technologie mit sich bringen. Außerdem ist der Gesamteffekt (*0,43 * 0,52 + 0,40 = 0,62*), den die Qualität der Implementierung auf die Nutzungsbereitschaft des Artefakts hat, größer als der direkte Effekt (*0,52*) der Qualität der Entscheidung (vgl. Abb. 4.5).

Unabhängig von der experimentellen Nutzung oder der Verwendung der Entscheidungstechnik im organisatorischen Umfeld, lassen sich abschließend mit dem eingeführten Bewertungsmodell und der deskriptiven Analyse Rückschlüsse auf die intendierte Nutzung des in dieser Arbeit entwickelten Entscheidungsmodells und dessen Implementierung *SmartSourcer* ziehen. Es kann dadurch nachgewie-

Alter und Erfahrung die Validität der ausgewerteten Daten. Für die Berechnung der Korrelationsmatrix wurden bei der Berechnung des Konstrukts der wahrgenommenen Benutzerfreundlichkeit die Indikatoren M-SU-3, M-SU-7 und M-SU-9 sowie M-IM-3 (vgl. Tabelle 4.3) nicht mit einbezogen, da sie nicht überschneidungsfrei zu bereits inkludierten Indikatoren sind. Allerdings gilt auch für diese Analyse, dass die Aussagekraft bei $N = 30$ nur bedingt zuverlässig ist.

sen werden, dass die Qualität der Implementierung (bewertet mit der wahrgenommenen Benutzerfreundlichkeit) als hoch bis sehr hoch eingeschätzt wird (*4,49*). Ebenfalls wird die Qualität der Entscheidung, welche die Qualität der Information und der Entscheidungslogik beinhaltet, und damit die wahrgenommene Nützlichkeit als ebenfalls hoch bewertet (*4,06*). Die Gesamtqualität des Artefakts (bewertet mit der wahrgenommenen Nutzungsbereitschaft) wird entsprechend des Bewertungsmodells von der Qualität des Entscheidungsmodells und der Qualität der Implementierung beeinflusst. So ist es nicht verwunderlich, dass die Nutzungsbereitschaft aller Teilnehmer der Evaluation zwischen hoch bis sehr hoch liegt (*4,27*). Dies bestätigt den prognostizierten Zusammenhang der Konstrukte und erlaubt es, aus dem Zusammenhang der wahrgenommenen Nutzungsbereitschaft mit der tatsächlichen Nutzung der in dieser Arbeit entwickelten Entscheidungstechnik, eine Zustimmung der Anwender vorherzusagen. Eine Ablehnung konnte sowohl bei den Experten als auch bei den Experimentteilnehmern für keines der Konstrukte gemessen werden.

4.3.2. Qualitative Auswertung

Bestandteil der Evaluationsstragie ist neben der Erhebung der quantitativen Daten zur Analyse des adaptierten TAM auch ein Interviewleitfaden, um die entwickelte Technologie zur Entscheidungsunterstützung anhand organisatorischer, individueller, informationsbezogener, technologischer und systemischer Aspekte qualitativ zu bewerten (vgl. Kapitel 4.2.3). Dadurch sind alle Konstrukte des theoretischen Bezugssystems abgedeckt und werden in dieser qualitativen Auswertung um Kontextinformationen, sofern erforderlich, ergänzt. Zunächst werden die ergänzenden Bewertungen zur deskriptiven Analyse und Verbesserungsvorschläge der Laborexperimente und anschließend der Expertenbefragung vorgestellt.

4.3.2.1. Qualitative Auswertung der Studentenexperimente

Im Rahmen der experimentellen Evaluation beschränkt sich die qualitative Analyse auf die technologischen Aspekte des Artefakts, da z.B. organisatorische Details und Zusammenhänge aufgrund der künstlichen Umgebung nicht valide beantwortet werden können. Deshalb werden von den Teilnehmern dieses Teils der Eva-

luation in erster Linie technologische Merkmale kritisch diskutiert, die sich mit der Funktionsweise und der Benutzeroberfläche des mobilen EUS *SmartSourcer* befassen. Dadurch sollen Schwachstellen der bisherigen Version und Verbesserungsmöglichkeiten für Weiterentwicklungen identifiziert werden.

Aus technologischer Sicht liefert das qualitative Feedback 23 kleinere Verbesserungshinweise in Bezug auf die Oberflächenvisualisierung, wie Rechtschreibfehler, Formulierungen oder Anzeigefehler. Die meisten der genannten Hinweise beziehen sich hauptsächlich auf die Benutzerfreundlichkeit des Systems. Sie werden direkt für sinnvoll und einfach umzusetzen befunden und werden deshalb direkt im Prototyp umgesetzt. In die Kategorie dieser kleineren Veränderungen fallen die Größenänderung von Schaltflächen, die Farbänderungen von Beschriftungen, das Hinzufügen von Abbruchsfeldern für bestimmte Navigationsanzeigen, die Implementierung von Auswahlfunktionalitäten und die permanente Anzeige von Balken zum Scrollen. Zusätzlich werden weitere Änderungshinweise individueller, technologischer und informationsbezogener Natur auf ihre Machbarkeit evaluiert und ebenfalls umgesetzt.

Auf der *individuellen* Ebene werden ergonomische Erweiterungen vorgenommen, die dem wahrgenommenen Stress durch vermeidbare Menüaufrufe entgegenwirken sollen. So wird mehrmals kritisiert, dass der Startbildschirm, der in einer früheren Version aus einem Auswahlmenü mit großen Icons und Verzweigungsmöglichkeiten in die Funktionalitäten der Bewertung, der Einstellungen, der Statistiken und der Hilfetexte bestand, keinen entscheidenden Mehrwert liefere, sondern lediglich die Anzahl unnötiger Menüaufrufe erhöhe. Dem kritischen Aspekt der Ergonomie des Anwenders wird Genüge geleistet, indem der Startbildschirm nach dem Aufruf von *SmartSourcer* als übersichtliches Menü angezeigt wird, das als zentrale Informationsanzeige für die Projektauswahl, die Komponentenbewertung und die Ergebnisübersicht der Outsourcingempfehlungen dient (vgl. Abbildung 3.16). Von dort aus lassen sich alle wichtigen Funktionen des mobilen EUS direkt anwählen. Dadurch soll unnötigen Navigationsschritten und dem damit verbunden Unmut des Anwenders entgegengewirkt werden.

Weiterhin wird aus den Interviews ersichtlich, dass die Berechnung der Outsourcingentscheidung pro Komponente zwar theoretisch fundiert ist, die Repräsentation des Ergebnisses innerhalb von *SmartSourcer* auf den ersten Blick allerdings zu wenig Details zur Berechnung bietet und somit undurchsichtig wirkt.

Es wird bemängelt, dass eine mühevolle Informationssuche durch Absprung in die Übersicht der konfigurierten Entscheidungstabellen und zurück zur Einstellung der Gewichtung vollzogen werden müsse, bevor erst eine Nachvollziehbarkeit der Entscheidung des Systems möglich sei. Diesem Fall des Informationsmangels wird mit einer Nachbesserung entgegengewirkt, die für jede bewertete Komponente eine detaillierte Übersicht über die Zusammensetzung des Ergebnisses der Entscheidung zur Verfügung stellt (vgl. Abbildung 3.21).

Auf der *informationsbezogenen Ebene* liefern die Interviews der experimentellen Evaluation weitere nützliche Hinweise mit Vorschlägen zur verbesserten Informationsbereitstellung innerhalb des mobilen EUS. Die Integrationsmöglichkeit der neuen Technologie in vorhandene Systemlandschaften von Unternehmen wird als zweckmäßig befunden, allerdings wird kritisiert, dass diese Verbindung zu anderen Systemen während der Benutzung der App nicht ersichtlich sei und keine Möglichkeit bestehe, ein anderes als das eingestellte Kollaborationssystem für das Einlesen von Softwarekomponenten zu verwenden. Daraufhin wird das Menü der Systemeinstellungen um die Möglichkeit erweitert, eine URL für den Zugriff auf ein anderes System einzugeben. Hier ist das Hinzufügen eines weiteren Eingabefelds in der Programmoberfläche der Einstellungen nicht ausreichend (vgl. Abbildung 3.17). Zusätzlich muss hierbei der zwischengeschaltete Proxyserver auf die neue Konfigurationsmöglichkeit angepasst werden (vgl. Abbildung 3.9).

Unter *technologischen Gesichtspunkten* liefern die Interviews aus den Experimenten neben den eingangs erwähnten kleineren Verbesserungen der Oberflächenvisualisierung weitere Hinweise auf Implementierungsmängel, die die Benutzerfreundlichkeit der neuen Entscheidungstechnik einschränken. In diesem Zusammenhang wird erwähnt, dass eine Wiederherstellung der ursprünglichen Systemeinstellungen die Benutzerfreundlichkeit wesentlich erhöhen würde, weil dadurch eine ungewollte Konfiguration des Entscheidungsmodells und der jeweiligen Aggregate durch den Anwender schnell wieder auf den Ursprungszustand zurückgesetzt werden können. Diese Möglichkeit wird den Anwendern durch eine Resetfunktion im Einstellungsmenü eingeräumt (vgl. Abbildung 3.17). Ihnen bleibt dadurch viel Zeit bei der Rücksetzung auf ein bereits validiertes Entscheidungsmodell erspart (vgl. Kapitel 4.3.1).

Zusätzlich wird von einem Teilnehmer des Experiments die unbrauchbare Verwendungsmöglichkeit von *SmartSourcer* im Hochformat des Tablet-PCs ange-

sprochen. Die Anwendung sei deshalb je nach Situation und Kontext nicht vollwertig einsetzbar. Einige Situationen erfordern aufgrund eingeschränkter Haltemöglichkeiten eine Verwendung des Tablet-PCs im Hochformat. In diesem Fall würden die Möglichkeiten eines mobilen EUS nicht vollständig ausgeschöpft werden. Um dem entgegenzuwirken, wird die Darstellungsweise für alle Funktionen in beiden Formaten ermöglicht. Neben einer Anpassung aller Anzeigebildschirme im *Storyboard* (vgl. Abbildung 3.13) für die Verwendung im Hochformat mussten zusätzlich Anpassungen beim Auswahlmenü der Komponentenbewertung vorgenommen werden. *SmartSourcer* ist nun im Quer- und Hochformat des Tablet-PCs verwendbar.

4.3.2.2. Qualitative Auswertung der Expertenbefragungen

Die qualitative Analyse der Expertenevaluation verfolgt ebenfalls die Zielsetzung, das neu entwickelte Entscheidungsmodell und seine Implementierung umfassend zu evaluieren und über die deskriptive Analyse hinaus Schwachstellen und Verbesserungspotenziale aufzudecken. Im Gegensatz zur experimentellen Evaluation wird hierbei der Fokus auf die organisatorischen Aspekte des innovativen Artefakts gelegt, so dass das Expertenwissen und die Branchenkenntnisse der Befragten entsprechend zur Geltung kommen.

Beginnend mit der *organisatorischen Ebene* liefern die Experteninterviews drei nennenswerte Erkenntnisse, die sich durch die Nutzung von *SmartSourcer* ergeben. Zunächst erwähnt ein Proband, dass sich durch Verwendung des Artefakts und der rigorosen Anwendung des darin vorkonfigurierten Entscheidungsmodells Lerneffekte und Produktivitätssteigerungen für Mitarbeiter erzielen lassen, die erstmalig mit Outsourcingentscheidungen konfrontiert und betraut werden. Einerseits werde dadurch eine Reduzierung von Fehlentscheidungen und andererseits ein beschleunigter Entscheidungsprozess erwartet, da das Entscheidungsfeld mit bereits vorkonfigurierten Entscheidungsgrößen direkt für die Bewertung von Softwarekomponenten herangezogen werden könne und nicht erst in einem langwierigen Prozess erstellt und getestet werden müsse.

Laut einem weiteren Probanden der Outsourcingexperten birgt das mobile EUS die Chance, vorkonfigurierte Entscheidungsmodelle unternehmensweit einzusetzen, wenn sie sich nach der Durchführung von Softwareprojekten als besonders

erfolgreich für die Unterstützung der Outsourcingentscheidung herausgestellt haben. Er macht deshalb den Verbesserungsvorschlag, dass sich die Konfiguration des Entscheidungsmodells, das im Falle von *SmartSourcer* die Entscheidungsgrößen und Aggregate umfasst, speichern lasse und auf andere Installationen des mobilen EUS übertragen und laden lasse. Eine derartige Funktionalität könne die Entscheidungsgeschwindigkeit und die Qualität der Outsourcingentscheidungen von Unternehmen insgesamt verbessern.

Weiterhin wird die Implementierung von *SmartSourcer* als Anwendung auf Tablet-PCs mehrmals positiv erwähnt, was einerseits die wahrgenommene Benutzerfreundlichkeit untermauert. Andererseits werde dadurch aber auch die Flexibilität des Unternehmens beim Management und bei der Steuerung ihrer Outsourcingprojekte unterstützt. Der mobile Charakter des EUS mache die Entscheidungsfindung überall und jederzeit möglich und reduziere damit die Gefahr einer Verzögerung bei der Durchführung administrativer Projektaufgaben.

Auf der *individuellen Ebene* wird dem Artefakt von einem der Experten eine Grundlage zur Rechtfertigung getroffener Entscheidungen zugesprochen. Dies ist gerade dann von besonderer Bedeutung, wenn ein Softwareprojekt am Ende fehlschlägt und nach möglichen Gründen gesucht wird. Zur eigenen Zufriedenstellung könne man damit dann auf bewährte Entscheidungsgrößen zurückgreifen und mit diesen argumentieren.

Ein weiterer genannter Aspekt, der die Ergonomie eines Anwenders betrifft, ist die farbliche und grafische Anpassung der App. Dieses sehr subjektive Empfinden wurde von mehreren Experten angesprochen. Bekannte Bezeichnungen und Navigationsbuttons würden dabei helfen, sich noch schneller in der Anwendung zurechtzufinden. Daraus lassen sich zusätzliche Einstellungsoptionen als Verbesserungsmöglichkeit in zukünftigen Versionen von *SmartSourcer* ableiten.

Ebenfalls zu den individuellen Aspekten zählt die Anmerkung eines Experten, dass ein Projektzeitstrahl in Verknüpfung mit einer stets aktuellen Anzeige des verbrauchten Projektbudgets den Bewertungsvorgang im mobilen EUS verbessern könnte. Einerseits würde der Zeitstrahl, der zur Anzeige der Restlaufzeit des Projekts verwendet wird, damit angeben, ob noch mehr Komponenten zur Einhaltung des Fertigstellungstermins ausgelagert werden müssten. Andererseits würde eine aktuelle Übersicht über das Restbudget die Entscheidung entsprechend

beeinflussen, wie viele weitere Komponenten ausgelagert werden könnten. Diese Verbesserungsvorschläge würden zwar zu einer Reduzierung der Informationsunterversorgung bei der Entscheidung auf individueller Ebene führen, beziehen sich allerdings primär auf das strategische Outsourcingverhalten eines Unternehmens und sollten im Vorfeld der Anwendung dieses EUS geklärt sein.

Hinsichtlich *informationsbezogener Aspekte* liefert die qualitative Analyse Erkenntnisse über die Selbstreflexion. So gibt ein Befragter an, dass er einen großen Vorteil dieser Entscheidungsunterstützung darin sieht, dass sich getroffene Entscheidungen im Detail nachvollziehen lassen und dass im Falle einer Abweichung von der eigenen Entscheidung die Unterschiede detailliert nachgeforscht werden können. Dadurch werde die Informations- und Entscheidungsqualität verbessert. Entscheidungen wären so verlässlicher. Eng damit verbunden ist auch die Anmerkung eines anderen Experten, dass das mobile EUS die Zurückverfolgung von getroffenen Entscheidungen jederzeit ermögliche und so auch bei gescheiterten Outsourcingprojekten eine Überprüfung der verwendeten Entscheidungsgrößen und deren Bewertungen möglich sei. Dies habe den Vorteil, dass Konfigurationen des Entscheidungsmodells besser zugänglich und durchsuchbar wären. Dadurch lassen sich Zusammenhänge zwischen getroffenen Ausprägungen und dem Misserfolg des zugehörigen Softwareprojekts ableiten.

Aus *technologischer Sicht* wird dem Artefakt dieser Arbeit eine hohe Benutzerfreundlichkeit ausgesprochen, weil sich die Ergebnisse der Entscheidungsunterstützung als ausführliche Übersicht am Ende der Bewertung mit ihren Begründungszusammenhängen darstellen lassen. Dies untermauert zusätzlich das Konstrukt der wahrgenommenen Benutzerfreundlichkeit in der deskriptiven Analyse. Weil sich diese Übersichten laut einem der Experten optimal für die Verwendung als Entscheidungsvorlage verwenden ließen, wäre eine Exportfunktion der Ergebnisse als PDF-Dokument wünschenswert.

Weiterhin wird der integrative Charakter des mobilen EUS in bestehende Systemlandschaften und Kollaborationsplattformen mehrfach goutiert. Eine Nutzung von *SmartSourcer* wäre aufgrund der guten Softwarekompatibilität problemlos in bestehenden Softwareentwicklungsumgebungen möglich.

Aus *systemischer Sicht* werden in den Interviews weder von den Experten noch von den Teilnehmern der experimentellen Evaluation kritische oder verbessernde

Bemerkungen gemacht. Eine weitere zwischengelagerte und an die Expertenstudie anknüpfende Konstruktionsphase zur Implementierung der Exportfunktionalität als PDF und Korrektur kleinerer Fehler im Design und in der Funktionalität der App wird nach Auswertung der Studienergebnisse eingeschoben, um den iterativen Prozesscharakter der *Design Science* weiter hervorzuheben.

4.4. Zusammenfassung

In diesem Kapitel wurde die Evaluation des Systems zur Entscheidungsunterstützung in der komponentenbasierten Softwareentwicklung im Rahmen des konstruktionswissenschaftlichen Forschungsparadigmas durchgeführt. Dafür wurden zunächst der Hintergrund der Evaluation und die unterschiedlichen Ausprägungen der Ansätze zur Durchführung diskutiert und dabei die künstliche und naturalistische Evaluation *ex post* als geeignet für diese Arbeit identifiziert.

Das in dieser Arbeit herangezogene Bewertungsmodell ist das Technologieakzeptanzmodell. Dadurch wurde ein Bezugsrahmen geschaffen und mit der die Messung der Implementierungsqualität und der Qualität des Entscheidungsmodells verknüpft. Die Datenerhebung wurde im Rahmen eines Studentenexperiments und einer Expertenbefragung durchgeführt. Anschließend wurden die Ergebnisse deskriptiv und qualitativ ausgewertet.

In der deskriptiven Analyse konnte gezeigt werden, dass sowohl die wahrgenommene Benutzerfreundlichkeit als auch die wahrgenommene Nützlichkeit signifikant mit der Nutzungsbereitschaft korrelieren. Es konnte für alle Konstrukte des Bewertungsmodells eine hohe bis sehr hohe Ausprägung erzielt werden und damit die Nützlichkeit und Nutzbarkeit von *SmartSourcer* bestätigt werden. Die Ergebnisse der deskriptiven Analyse wurden durch eine qualitative Analyse untermauert, indem die Vorteilhaftigkeit von *SmartSourcer* unter organisatorischen, individuellen, informationsbezogenen und technologischen Aspekten beleuchtet wurde. Hierbei konnten wertvolle Verbesserungsvorschläge für zukünftige Weiterentwicklungen des mobilen EUS gewonnen werden.

5. Zusammenfassung und Ausblick

In diesem Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst. Zunächst wird daher der Forschungsbeitrag erläutert, den diese Arbeit liefert, bevor ein Ausblick auf eine mögliche zukünftige Weiterentwicklung des Entscheidungsmodells und seiner Instanziierung unter Berücksichtigung der Limitationen das letzte Kapitel abrundet.

5.1. Forschungsbeitrag

Der Forschungsbeitrag dieser konstruktionswissenschaftlichen Arbeit lässt sich einerseits zusammenfassend hinsichtlich der Zielsetzung und andererseits partikular entlang der formulierten Forschungsziele (vgl. Kap. 1.2) beschreiben. In der *Design Science* werden durch die Gestaltung und Anwendung sowie das Testen von neuen Artefakten Erkenntnisse und Wissen über ein abgegrenztes Problemfeld erlangt. Vor diesem Hintergrund werden innovative Artefakte angestrebt, die einen klaren und belegbaren Forschungsbeitrag für das Design von Artefakten, für das Design von Grundlagenwissen oder das Design von Methodologien leisten (Hevner et al., 2004). Diesbezüglich wird im weiteren Verlauf dieses Abschnitts der Forschungsbeitrag dieser Arbeit entlang der beiden einleitend genannten Perspektiven erörtert.

Der bedeutendste Beitrag dieser Arbeit ist der Neuigkeitsgrad des Ansatzes zur Unterstützung von Outsourcingentscheidungen unter Berücksichtigung technischer Eigenschaften eines Softwareprodukts. Diese Arbeit ist daher nach Kenntnissen ihres Verfassers die erste ihrer Art, die einen Beitrag zur Auslagerung konkreter technischer Artefakte in der Form von Softwarekomponenten in Softwareentwicklungsprojekten leistet. Das neu entworfene Entscheidungsmodell trägt in Kombination mit der prototypischen Umsetzung des Modells als Softwarewerkzeug wesentlich zur Entscheidung bei, ob eine Komponente selbst entwickelt oder fremdbezogen wird. Outsourcingentscheidungen müssen daher nicht mehr nur als

„Bauchentscheidungen“ getroffen werden, sondern lassen sich im Vorfeld strukturieren und mit dem vorgestellten Ansatz systematischer und umfassender entscheiden.

Der zweite wesentliche Beitrag im Rahmen dieser Forschung ist die Entwicklung eines normativen Entscheidungsmodells zur Durchführung von Outsourcingentscheidungen in Softwareentwicklungsprojekten. Es wird damit ein Konzept geschaffen, das es erlaubt, anerkannte Entscheidungskriterien für die Bewertung von Softwarekomponenten heranzuziehen und damit deren Eignung für das Outsourcing zu bewerten. Da bislang nur wenige wissenschaftliche Beiträge verfasst wurden, die normative Modelle für das Outsourcing von Informationssystemen entwickelt haben, sondern vielmehr die Zusammenhänge im IT-Outsourcing deskriptiv und explikativ erforscht wurden, ist das Entscheidungsmodell dieser Arbeit eine substantielle Ergänzung des Wissensfundus des IT-Outsourcings, der zudem technisch implementiert und umfassend validiert wurde.

Unter Berücksichtigung der Forschungsziele (vgl. Kap. 1.2) können weitere Beiträge genannt werden, die diese Arbeit leistet. Für die Herleitung und Entwicklung eines Entscheidungsmodells für das Outsourcing von Softwarekomponenten im Softwareentwicklungsprozess, also das erste Forschungsziel dieser Arbeit (**Z1**), wurden verschiedene theoretische Ansätze verwendet, um das Entscheidungsmodell auf einer dreidimensionalen Basis aufzubauen.

Die erste Dimension, die *strukturellen Eigenschaften* von Softwarekomponenten, wurde aus der Systemtheorie heraus entwickelt. Aus ihr konnten die Entscheidungskriterien Kohäsion und Kopplung von Softwaremodulen als relevant für das Modell abgeleitet werden, da sie aus technischer Sicht spezifische Eigenschaften für den modularen Aufbau bei der Entwicklung komplexer Systeme darstellen (vgl. Kap. 3.2.1.1).

Zur Bestimmung der *Prozessmerkmale der Entwicklung* von Softwarekomponenten als zweite Dimension des Entscheidungsmodells wurde die Transaktionskostentheorie herangezogen. Auf deren Basis konnten spezielle Entscheidungsmerkmale zur Bewertung des Erstellungsprozesses von Softwarekomponenten, die von unterschiedlichen Herstellern, von unterschiedlichen Teams oder an verteilten Standorten entwickelt werden, identifiziert werden (vgl. Kap. 3.1). Als Resultat fließen u.a. Merkmale wie die Kommunikationsintensität für die Abstimmung mit

dem Auftraggeber der Software, der erwartete Abstimmungsaufwand zwischen den Entwicklern einzelner Komponenten oder der Integrationsaufwand nach Fertigstellung einzelner Teile des Softwareprodukts mit in den Entscheidungsprozess ein (vgl. Kap. 3.2.1.2).

Als dritte und letzte Dimension des Entscheidungsmodells wurden *Wissensmerkmale von Softwarekomponenten* aus Sicht der ressourcenbasierten Sichtweise von Unternehmen sowie der Transaktionskostentheorie (vgl. Kap. 3.2.1.3) erarbeitet. Diese Merkmale ermöglichen im Entscheidungsmodell die Bewertung, in welchem Ausmaß spezifisches Wissen, besondere Fähigkeiten und Fertigkeiten oder auch spezielle Kenntnisse erforderlich sind, um eine Softwarekomponente entwickeln zu können. Zu diesen Kriterien zählen z.B. die geschäftsprozessbezogene, die funktionale und die technische Spezifität oder auch die Sensitivität von Test- und Nutzungsdaten (vgl. Kap. 3.2.1.3).

Weiterhin ist es in dieser Arbeit gelungen, das Entscheidungsmodell so zu entwickeln und zu gestalten, dass es sich in bestehende Prozessmodelle der Softwareentwicklung integrieren lässt. Der Zeitpunkt der Entscheidung wird dabei der Implementierungsphase vorangestellt. Die Implementierung selbst wird dann in zwei alternative Pfade aufgeteilt, in denen Softwarekomponenten intern entwickelt oder von einem externen Zulieferer umgesetzt werden. Diese Form der einfachen Integration (vgl. Kap. 2.4.3.1) lässt sich gleichermaßen auf weitere Prozessmodelle übertragen.

Hinsichtlich der Instanziierung des Entscheidungsmodells (Forschungsziel **Z2**) liefert diese Arbeit die Umsetzung des neu entwickelten Entscheidungsmodells in ein nützliches und nutzbares Softwarewerkzeug in Form eines mobilen EUS auf dem Stand der Technik (vgl. Kap. 3.3). Die konzeptionellen Anforderungen des Entscheidungsmodells wurden dafür in eine Systemarchitektur überführt, die einerseits den mobilen Charakter des neuen EUS mit dem Namen *SmartSourcer* hervorhebt und sich andererseits nahtlos in bestehende Systemlandschaften von Softwareunternehmen integrieren lässt, ohne dass erhebliche Änderungen an der Kollaborationsinfrastruktur vorgenommen werden müssen (vgl. Kapitel 3.3.1).

Zusätzlich ermöglicht die Einbindung der *SODA*-Methode in das Softwarewerkzeug *SmartSourcer* die vereinfachte Anwendung des Entscheidungsmodells. In diesem Fall ist keine a priori spezifizierte Softwarearchitektur des zu erstellen-

den Softwareprodukts erforderlich, um die strukturellen Eigenschaften von Softwarekomponenten durch das Entscheidungsmodell zu evaluieren. Vielmehr wird hierbei auf eine bereits existierende Menge von Anforderungen und der Verknüpfungen ihrer Elemente zurückgegriffen. Mithilfe graphentheoretischer Analysemethoden ist es möglich, automatisiert die strukturellen Entscheidungskriterien von Softwarekomponenten zu evaluieren, um kohärente Gruppen von Anforderungen zu betrachten. Im Falle einer spezifizierten Softwarearchitektur lassen sich die Resultate dann weiter verfeinern. Dem Anwender wird dadurch eine Möglichkeit gegeben, einen Teil der Entscheidungskriterien durch *SmartSourcer* automatisch evaluieren zu lassen.

Die Ergebnisse zum dritten Forschungsziel (**Z3**) dieser Arbeit liefern im konstruktionswissenschaftlichen Sinne schließlich den konkreten Nachweis der Nützlichkeit und Nutzbarkeit des Entscheidungsmodells und des Prototyps in experimenteller Umgebung. Im Rahmen einer künstlichen Evaluation mit Studenten und einer naturalistischen Evaluation mit Outsourcingexperten wird das Artefakt *SmartSourcer* positiv bewertet. Konkret wird über die Nützlichkeit nachgewiesen, dass sowohl die Entscheidungskriterien als auch die Entscheidungslogik vorteilhaft für das Treffen von Outsourcingentscheidungen über Softwarekomponenten im Softwareentwicklungsprozess sind. Die Ergebnisse der Evaluation legen ebenfalls offen, dass ein signifikanter Zusammenhang sowohl zwischen der Nutzbarkeit des Softwarewerkzeugs und der Nutzungsbereitschaft von *SmartSourcer* als auch zwischen der Nützlichkeit und der Nutzungsbereitschaft von *SmartSourcer* besteht (vgl. Kap. 4.3).

5.2. Limitationen und Ausblick

In diesem Abschnitt werden die Einschränkungen, denen diese Forschungsarbeit unterliegt, aufgezeigt und diskutiert. Dafür werden konzeptionelle, methodische und technische Limitationen in Bezug auf die einzelnen Forschungsziele (vgl. Kap. 1.2) erläutert. Schließlich wird unter Berücksichtigung dieser Einschränkungen jeweils ein Ausblick auf mögliche Weiterentwicklungen des bearbeiteten Themas gegeben.

Für die Entwicklung des Entscheidungsmodells dieser Arbeit und dessen Logik (**Z1**) wurden unterschiedliche Entscheidungsverfahren evaluiert und ihre Taug-

lichkeit für die Anwendung im Outsourcing von Softwarekomponenten überprüft (vgl. Kap. 2.2.1.3). Als passende Verfahren hierfür haben sich die Methoden der multikriteriellen Verfahren erwiesen und wurden entsprechend ausgewählt. Allerdings unterstellen diese Methoden eine Entscheidungsfindung unter Sicherheit, so dass für jede Handlungsalternative sämtliche Konsequenzen bekannt sind. Dies trifft für das Entscheidungsmodell dieser Arbeit nur bedingt zu. Sicherlich sind die Konsequenzen einzelner Entscheidungen für eine einzelne Softwarekomponente bekannt. Sie wird entweder intern entwickelt oder von extern zugeliefert. Allerdings finden diese Entscheidungen stets unter Unsicherheit statt, weil keinesfalls die Konsequenzen des Outsourcingmix bekannt sind, also wie sich das Gesamtprodukt entwickelt, wenn die Implementierung der Komponenten entsprechend des neuen Entscheidungsmodells verteilt wird. Zukünftige Forschungsarbeiten könnten daher weitere Entscheidungsverfahren auswählen und diese mit der Nützlichkeit des in dieser Arbeit entwickelten Modells vergleichen. Alternativ würde eine Weiterentwicklung von *SmartSourcer* zu einem Werkzeug, das Gruppenentscheidungen ermöglicht, dazu beitragen, dass die Standardabweichungen der Einzelbewertungen dafür genutzt werden könnten, um eine Entscheidung unter Risiko zu betrachten.

Eine weitere Limitation stellt die praxeologisch-konzeptionelle Integration der theoretischen Konzepte in das Entscheidungsmodell dar. Anstatt einer stringenten theoretischen Herleitung der Kriterien wurde eine unmittelbar für die praktische Verwendung geeignete Ableitung von Entscheidungskriterien gewählt. Denn dieser Ansatz macht es möglich, dass die Kriterien direkt in die Implementierung des Prototyps übernommen werden können und ohne weitere Bearbeitung (Umformulierung) für Praktiker verwendbar und verständlich sind. Nichtsdestotrotz dient diese Arbeit als Ausgangsbasis für die Einbindung zusätzlicher Theorien und die stringentere Herleitung von weiteren Entscheidungskriterien in zukünftigen Arbeiten.

Bei der prototypischen Umsetzung des Entscheidungsmodells (**Z2**) wurde die Instanziierung ausschließlich als native Anwendung für das iOS-Betriebssystem entwickelt. Die Anwendung ist daher auf sämtlichen *iPads* neuerer Generationen lauffähig, kann in seiner aktuellen Version aber nicht mit den Tablet-PCs anderer Hersteller verwendet werden. Da es sich bei diesem System um einen Prototyp zur Konzeptdemonstration handelt, wurde dieses Vorgehen als ausreichend erachtet.

Bei einer Weiterentwicklung des mobilen EUS für den produktiven Einsatz ist es daher empfehlenswert, die Implementierung auf die Betriebssysteme mobiler Tablet-PCs auszuweiten, die am meisten Verwendung finden.

Gleichermaßen wurde die Entwicklung des Proxyservers zum Zweck der Konzeptdemonstration ausschließlich für den Zugriff auf die Kollaborationsplattform *CodeBeamer*[1] konzipiert und umgesetzt. Eine situationsbedingte Erweiterung auf zusätzliche Plattformen (wie in Kap. 3.3.1.2 vorgestellt) wird bei einer Weiterentwicklung von *SmartSourcer* ebenfalls empfohlen.

Hinsichtlich der Evaluation des Modells und seiner Instanziierung (**Z3**) müssen weitere Einschränkungen beachtet werden. Die Bewertung des Artefakts konnte in dieser Arbeit nur teilweise und in einem begrenzten Umfang in seiner natürlichen Umgebung mit realen Anwendern durchgeführt werden. Die gewählte Evaluationsstrategie ermöglichte es zwar, reale Anwender zu befragen, allerdings konnte die Verwendung des Systems nur in einer künstlichen Umgebung und nicht in einem Feldtest getestet werden. Dabei wurden ausschließlich hypothetische Aufgaben verwendet, die für eine realitätsgetreue Anwendung des Entscheidungsmodells und seiner Instanziierung trotz experimenteller Evaluationsumgebung sorgen sollten (vgl. Kap. 4.2.3).

Abschließend ist die Aussagekraft der Methoden der deskriptiven und induktiven Statistik in der Evaluation dieser Arbeit zu nennen. Obwohl mit diesen statistischen Verfahren signifikante Zusammenhänge zwischen den Konstrukten innerhalb des Bewertungsmodells nachgewiesen werden konnten, ist die Aussagekraft dieser Ergebnisse aufgrund der überschaubaren Stichprobengröße (Anzahl der Evaluationsteilnehmer) relativ gering (vgl. Kap. 4.3.1). Dies gilt ebenfalls für die Aussagekraft des t-Tests in Tab. I.2, also den Vergleich der beiden Evaluationsgruppen. Aus diesem Grund wurde die quantitative Analyse durch eine qualitative Bewertung ergänzt. Ergänzende Forschungsarbeiten könnten durch Einbindung weiterer Studienteilnehmer und der Durchführung einer Feldstudie, bei der das mobile EUS in einem realen Softwareentwicklungsprojekt eingesetzt und bewertet wird, den naturalistischen Charakter dieser Evaluation untermauern und damit dem Entscheidungsmodell und dessen Instanziierung weitere Plausibilität zusprechen.

1 http://intland.com/ (abgerufen am 01.12.2014)

Trotz der genannten Limitationen wird in dieser Arbeit ein innovatives Artefakt für die Entscheidungsunterstützung beim Outsourcing von Softwarekomponenten entwickelt und bereitgestellt. Das Artefakt besteht einerseits aus einem normativen Entscheidungsmodell und der dazugehörigen Entscheidungslogik zur Bewertung sämtlicher Komponenten eines Softwareentwicklungsprojekts. Die aus der Theorie hergeleiteten und im Modell berücksichtigten Entscheidungskriterien stellen zusammen mit der Entscheidungslogik ein neuartiges Konzept dar, das erstmalig technische Eigenschaften eines Softwareprodukts in die Outsourcingentscheidung mit einbezieht. Andererseits umfasst das Artefakt sowohl die Implementierung als auch Validierung des Entscheidungsmodells.

Insgesamt wird mit *SmartSourcer* ein praktikabler Ansatz sowie ein innovatives Softwarewerkzeug für Outsourcingmanager geschaffen, um ihnen eine strukturierte Herangehensweise und eine systematische Entscheidungsfindung bei der Klassifikation von Softwarekomponenten für das IT-Outsourcing zu ermöglichen. Eine positive Evaluation bestätigt die Qualität sowohl des Entscheidungsmodells als auch der Implementierung und unterstreicht damit die Nützlichkeit und Nutzbarkeit von *SmartSourcer* zum Treffen von Outsourcingentscheidungen in der Softwareentwicklung.

A. Entscheidungstabelle für die strukturellen Eigenschaften einer Komponente in Bezug auf das Softwareprodukt

Die verwendeten Abkürzungen sind wie folgt definiert:

E_i Entscheidungsgröße i (vgl. Tabelle 3.3)

L_1 Lösungsalternative *Hoher strukturspezifischer Einfluss*

L_2 Lösungsalternative *Mittlerer strukturspezifischer Einfluss*

L_3 Lösungsalternative *Niedriger strukturspezifischer Einfluss*

R_j Regel j als Ausprägungsvektor der Entscheidungsgrößen

H / M / L hoch / mittel / niedrig

		R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9
E_1	Kohäsion	H	H	H	M	M	M	L	L	L
E_2	Kopplung	L	M	H	L	M	H	L	M	H
L_1	Hohe Spezifität						✓		✓	✓
L_2	Mittlere Spezifität			✓		✓		✓		
L_3	Niedrige Spezifität	✓	✓		✓					

B. Entscheidungstabelle für die Prozesssmerkmale der Entwicklung von Softwarekomponenten

Die nachfolgende Tabelle wurde aus Darstellungsgründen im Vergleich zu Tabelle 3.6 transponiert. Weiterhin sind die verwendeten Abkürzungen wie folgt definiert:

E_i Entscheidungsgröße i (vgl. Tabelle 3.3)

L_1 Lösungsalternative *Hoher prozessspezifischer Einfluss*

L_2 Lösungsalternative *Mittlerer prozessspezifischer Einfluss*

L_3 Lösungsalternative *Niedriger prozessspezifischer Einfluss*

R_j Regel j als Ausprägungsvektor der Entscheidungsgrößen

H / M / L hoch / mittel / niedrig

	E_3	E_4	E_5	E_6	E_7	L_1	L_2	L_3
R_1	H	H	H	H	H	✓		
R_2	H	H	H	H	M	✓		
R_3	H	H	H	H	L	✓		
R_4	H	H	H	M	H	✓		
R_5	H	H	H	M	M	✓		
R_6	H	H	H	M	L	✓		
R_7	H	H	H	L	H	✓		
R_8	H	H	H	L	M	✓		
R_9	H	H	H	L	L		✓	
R_{10}	H	H	M	H	H	✓		
R_{11}	H	H	M	H	M	✓		
R_{12}	H	H	M	H	L	✓		
R_{13}	H	H	M	M	H	✓		
R_{14}	H	H	M	M	M	✓		
R_{15}	H	H	M	M	L		✓	
R_{16}	H	H	M	L	H	✓		
R_{17}	H	H	M	L	M		✓	
R_{18}	H	H	M	L	L		✓	
R_{19}	H	H	L	H	H	✓		
R_{20}	H	H	L	H	M	✓		
R_{21}	H	H	L	H	L		✓	
R_{22}	H	H	L	M	H	✓		
R_{23}	H	H	L	M	M		✓	
R_{24}	H	H	L	M	L		✓	
R_{25}	H	H	L	L	H		✓	

	E_3	E_4	E_5	E_6	E_7	L_1	L_2	L_3
R_{26}	H	H	L	L	M		✓	
R_{27}	H	H	L	L	L		✓	
R_{28}	H	M	H	H	H	✓		
R_{29}	H	M	H	H	M	✓		
R_{30}	H	M	H	H	L	✓		
R_{31}	H	M	H	M	H	✓		
R_{32}	H	M	H	M	M	✓		
R_{33}	H	M	H	M	L		✓	
R_{34}	H	M	H	L	H	✓		
R_{35}	H	M	H	L	M		✓	
R_{36}	H	M	H	L	L		✓	
R_{37}	H	M	M	H	H	✓		
R_{38}	H	M	M	H	M	✓		
R_{39}	H	M	M	H	L		✓	
R_{40}	H	M	M	M	H	✓		
R_{41}	H	M	M	M	M		✓	
R_{42}	H	M	M	M	L		✓	
R_{43}	H	M	M	L	H		✓	
R_{44}	H	M	M	L	M		✓	
R_{45}	H	M	M	L	L		✓	
R_{46}	H	M	L	H	H	✓		
R_{47}	H	M	L	H	M		✓	
R_{48}	H	M	L	H	L		✓	
R_{49}	H	M	L	M	H		✓	
R_{50}	H	M	L	M	M		✓	
R_{51}	H	M	L	M	L		✓	
R_{52}	H	M	L	L	H		✓	
R_{53}	H	M	L	L	M		✓	
R_{54}	H	M	L	L	L			✓
R_{55}	H	L	H	H	H	✓		
R_{56}	H	L	H	H	M	✓		
R_{57}	H	L	H	H	L		✓	
R_{58}	H	L	H	M	H	✓		
R_{59}	H	L	H	M	M		✓	
R_{60}	H	L	H	M	L		✓	
R_{61}	H	L	H	L	H		✓	
R_{62}	H	L	H	L	M		✓	
R_{63}	H	L	H	L	L		✓	
R_{64}	H	L	M	H	H	✓		
R_{65}	H	L	M	H	M		✓	
R_{66}	H	L	M	H	L		✓	
R_{67}	H	L	M	M	H		✓	
R_{68}	H	L	M	M	M		✓	
R_{69}	H	L	M	M	L		✓	
R_{70}	H	L	M	L	H		✓	
R_{71}	H	L	M	L	M		✓	
R_{72}	H	L	M	L	L			✓
R_{73}	H	L	L	H	H		✓	
R_{74}	H	L	L	H	M		✓	
R_{75}	H	L	L	H	L		✓	
R_{76}	H	L	L	M	H		✓	
R_{77}	H	L	L	M	M		✓	
R_{78}	H	L	L	M	L			✓
R_{79}	H	L	L	L	H		✓	
R_{80}	H	L	L	L	M			✓
R_{81}	H	L	L	L	L			✓
R_{82}	M	H	H	H	H	✓		
R_{83}	M	H	H	H	M	✓		
R_{84}	M	H	H	H	L	✓		
R_{85}	M	H	H	M	H	✓		
R_{86}	M	H	H	M	M	✓		
R_{87}	M	H	H	M	L		✓	
R_{88}	M	H	H	L	H	✓		
R_{89}	M	H	H	L	M		✓	
R_{90}	M	H	H	L	L		✓	
R_{91}	M	H	M	H	H	✓		

	E_3	E_4	E_5	E_6	E_7	L_1	L_2	L_3
R_{92}	M	H	M	H	M	✓		
R_{93}	M	H	M	H	L		✓	
R_{94}	M	H	M	M	H	✓		
R_{95}	M	H	M	M	M		✓	
R_{96}	M	H	M	M	L		✓	
R_{97}	M	H	M	L	H		✓	
R_{98}	M	H	M	L	M		✓	
R_{99}	M	H	M	L	L		✓	
R_{100}	M	H	L	H	H	✓		
R_{101}	M	H	L	H	M		✓	
R_{102}	M	H	L	H	L		✓	
R_{103}	M	H	L	M	H		✓	
R_{104}	M	H	L	M	M		✓	
R_{105}	M	H	L	M	L		✓	
R_{106}	M	H	L	L	H		✓	
R_{107}	M	H	L	L	M		✓	
R_{108}	M	H	L	L	L			✓
R_{109}	M	M	H	H	H	✓		
R_{110}	M	M	H	H	M	✓		
R_{111}	M	M	H	H	L		✓	
R_{112}	M	M	H	M	H	✓		
R_{113}	M	M	H	M	M		✓	
R_{114}	M	M	H	M	L		✓	
R_{115}	M	M	H	L	H		✓	
R_{116}	M	M	H	L	M		✓	
R_{117}	M	M	H	L	L		✓	
R_{118}	M	M	M	H	H	✓		
R_{119}	M	M	M	H	M		✓	
R_{120}	M	M	M	H	L		✓	
R_{121}	M	M	M	M	H		✓	
R_{122}	M	M	M	M	M		✓	
R_{123}	M	M	M	M	L		✓	
R_{124}	M	M	M	L	H		✓	
R_{125}	M	M	M	L	M		✓	
R_{126}	M	M	M	L	L			✓
R_{127}	M	M	L	H	H		✓	
R_{128}	M	M	L	H	M		✓	
R_{129}	M	M	L	H	L		✓	
R_{130}	M	M	L	M	H		✓	
R_{131}	M	M	L	M	M		✓	
R_{132}	M	M	L	M	L			✓
R_{133}	M	M	L	L	H		✓	
R_{134}	M	M	L	L	M			✓
R_{135}	M	M	L	L	L			✓
R_{136}	M	L	H	H	H	✓		
R_{137}	M	L	H	H	M		✓	
R_{138}	M	L	H	H	L		✓	
R_{139}	M	L	H	M	H		✓	
R_{140}	M	L	H	M	M		✓	
R_{141}	M	L	H	M	L		✓	
R_{142}	M	L	H	L	H		✓	
R_{143}	M	L	H	L	M		✓	
R_{144}	M	L	H	L	L			✓
R_{145}	M	L	M	H	H		✓	
R_{146}	M	L	M	H	M		✓	
R_{147}	M	L	M	H	L		✓	
R_{148}	M	L	M	M	H		✓	
R_{149}	M	L	M	M	M		✓	
R_{150}	M	L	M	M	L			✓
R_{151}	M	L	M	L	H		✓	
R_{152}	M	L	M	L	M			✓
R_{153}	M	L	M	L	L			✓
R_{154}	M	L	L	H	H		✓	
R_{155}	M	L	L	H	M		✓	
R_{156}	M	L	L	H	L			✓
R_{157}	M	L	L	M	H		✓	

	E_3	E_4	E_5	E_6	E_7	L_1	L_2	L_3
R_{158}	M	L	L	M	M			✓
R_{159}	M	L	L	M	L			✓
R_{160}	M	L	L	L	H			✓
R_{161}	M	L	L	L	M			✓
R_{162}	M	L	L	L	L			✓
R_{163}	L	H	H	H	H	✓		
R_{164}	L	H	H	H	M	✓		
R_{165}	L	H	H	H	L		✓	
R_{166}	L	H	H	M	H	✓		
R_{167}	L	H	H	M	M		✓	
R_{168}	L	H	H	M	L		✓	
R_{169}	L	H	H	L	H		✓	
R_{170}	L	H	H	L	M		✓	
R_{171}	L	H	H	L	L		✓	
R_{172}	L	H	M	H	H	✓		
R_{173}	L	H	M	H	M		✓	
R_{174}	L	H	M	H	L		✓	
R_{175}	L	H	M	M	H		✓	
R_{176}	L	H	M	M	M		✓	
R_{177}	L	H	M	M	L		✓	
R_{178}	L	H	M	L	H		✓	
R_{179}	L	H	M	L	M		✓	
R_{180}	L	H	M	L	L			✓
R_{181}	L	H	L	H	H		✓	
R_{182}	L	H	L	H	M		✓	
R_{183}	L	H	L	H	L		✓	
R_{184}	L	H	L	M	H		✓	
R_{185}	L	H	L	M	M		✓	
R_{186}	L	H	L	M	L			✓
R_{187}	L	H	L	L	H		✓	
R_{188}	L	H	L	L	M			✓
R_{189}	L	H	L	L	L			✓
R_{190}	L	M	H	H	H	✓		
R_{191}	L	M	H	H	M		✓	
R_{192}	L	M	H	H	L		✓	
R_{193}	L	M	H	M	H		✓	
R_{194}	L	M	H	M	M		✓	
R_{195}	L	M	H	M	L		✓	
R_{196}	L	M	H	L	H		✓	
R_{197}	L	M	H	L	M		✓	
R_{198}	L	M	H	L	L			✓
R_{199}	L	M	M	H	H		✓	
R_{200}	L	M	M	H	M		✓	
R_{201}	L	M	M	H	L		✓	
R_{202}	L	M	M	M	H		✓	
R_{203}	L	M	M	M	M		✓	
R_{204}	L	M	M	M	L			✓
R_{205}	L	M	M	L	H		✓	
R_{206}	L	M	M	L	M			✓
R_{207}	L	M	M	L	L			✓
R_{208}	L	M	L	H	H		✓	
R_{209}	L	M	L	H	M		✓	
R_{210}	L	M	L	H	L			✓
R_{211}	L	M	L	M	H		✓	
R_{212}	L	M	L	M	M			✓
R_{213}	L	M	L	M	L			✓
R_{214}	L	M	L	L	H			✓
R_{215}	L	M	L	L	M			✓
R_{216}	L	M	L	L	L			✓
R_{217}	L	L	H	H	H		✓	
R_{218}	L	L	H	H	M		✓	
R_{219}	L	L	H	H	L		✓	
R_{220}	L	L	H	M	H		✓	
R_{221}	L	L	H	M	M		✓	
R_{222}	L	L	H	M	L			✓
R_{223}	L	L	H	L	H		✓	

	E_3	E_4	E_5	E_6	E_7	L_1	L_2	L_3
R_{224}	L	L	H	L	M			✓
R_{225}	L	L	H	L	L			✓
R_{226}	L	L	M	H	H		✓	
R_{227}	L	L	M	H	M		✓	
R_{228}	L	L	M	H	L			✓
R_{229}	L	L	M	M	H		✓	
R_{230}	L	L	M	M	M			✓
R_{231}	L	L	M	M	L			✓
R_{232}	L	L	M	L	H			✓
R_{233}	L	L	M	L	M			✓
R_{234}	L	L	M	L	L			✓
R_{235}	L	L	L	H	H		✓	
R_{236}	L	L	L	H	M			✓
R_{237}	L	L	L	H	L			✓
R_{238}	L	L	L	M	H			✓
R_{239}	L	L	L	M	M			✓
R_{240}	L	L	L	M	L			✓
R_{241}	L	L	L	L	H			✓
R_{242}	L	L	L	L	M			✓
R_{243}	L	L	L	L	L			✓

C. Entscheidungstabelle für die Wissensmerkmale von Softwarekomponenten

Die nachfolgende Tabelle wurde aus Darstellungsgründen im Vergleich zu Tabelle 3.6 transponiert. Weiterhin sind die verwendeten Abkürzungen wie folgt definiert:

E_i Entscheidungsgröße i (vgl. Tabelle 3.4)

L_1 Lösungsalternative *Hoher wissensspezifischer Einfluss*

L_2 Lösungsalternative *Mittlerer wissensspezifischer Einfluss*

L_3 Lösungsalternative *Niedriger wissensspezifischer Einfluss*

R_j Regel j als Ausprägungsvektor der Entscheidungsgrößen

H / M / L hoch / mittel / niedrig

	E_8	E_9	E_{10}	E_{11}	E_{12}	L_1	L_2	L_3
R_1	H	H	H	H	H	✓		
R_2	H	H	H	H	M	✓		
R_3	H	H	H	H	L	✓		
R_4	H	H	H	M	H	✓		
R_5	H	H	H	M	M	✓		
R_6	H	H	H	M	L	✓		
R_7	H	H	H	L	H	✓		
R_8	H	H	H	L	M	✓		
R_9	H	H	H	L	L		✓	
R_{10}	H	H	M	H	H	✓		
R_{11}	H	H	M	H	M	✓		
R_{12}	H	H	M	H	L	✓		
R_{13}	H	H	M	M	H	✓		
R_{14}	H	H	M	M	M	✓		
R_{15}	H	H	M	M	L		✓	
R_{16}	H	H	M	L	H	✓		
R_{17}	H	H	M	L	M		✓	
R_{18}	H	H	M	L	L		✓	
R_{19}	H	H	L	H	H	✓		
R_{20}	H	H	L	H	M	✓		
R_{21}	H	H	L	H	L		✓	
R_{22}	H	H	L	M	H	✓		
R_{23}	H	H	L	M	M		✓	
R_{24}	H	H	L	M	L		✓	
R_{25}	H	H	L	L	H		✓	
R_{26}	H	H	L	L	M		✓	
R_{27}	H	H	L	L	L		✓	
R_{28}	H	M	H	H	H	✓		

	E_3	E_4	E_5	E_6	E_7	L_1	L_2	L_3
R_{29}	H	M	H	H	M	✓		
R_{30}	H	M	H	H	L	✓		
R_{31}	H	M	H	M	H	✓		
R_{32}	H	M	H	M	M	✓		
R_{33}	H	M	H	M	L		✓	
R_{34}	H	M	H	L	H	✓		
R_{35}	H	M	H	L	M		✓	
R_{36}	H	M	H	L	L		✓	
R_{37}	H	M	M	H	H	✓		
R_{38}	H	M	M	H	M	✓		
R_{39}	H	M	M	H	L		✓	
R_{40}	H	M	M	M	H	✓		
R_{41}	H	M	M	M	M		✓	
R_{42}	H	M	M	M	L		✓	
R_{43}	H	M	M	L	H		✓	
R_{44}	H	M	M	L	M		✓	
R_{45}	H	M	M	L	L		✓	
R_{46}	H	M	L	H	H	✓		
R_{47}	H	M	L	H	M		✓	
R_{48}	H	M	L	H	L		✓	
R_{49}	H	M	L	M	H		✓	
R_{50}	H	M	L	M	M		✓	
R_{51}	H	M	L	M	L		✓	
R_{52}	H	M	L	L	H		✓	
R_{53}	H	M	L	L	M		✓	
R_{54}	H	M	L	L	L			✓
R_{55}	H	L	H	H	H	✓		
R_{56}	H	L	H	H	M	✓		
R_{57}	H	L	H	H	L		✓	
R_{58}	H	L	H	M	H	✓		
R_{59}	H	L	H	M	M		✓	
R_{60}	H	L	H	M	L		✓	
R_{61}	H	L	H	L	H		✓	
R_{62}	H	L	H	L	M		✓	
R_{63}	H	L	H	L	L		✓	
R_{64}	H	L	M	H	H	✓		
R_{65}	H	L	M	H	M		✓	
R_{66}	H	L	M	H	L		✓	
R_{67}	H	L	M	M	H		✓	
R_{68}	H	L	M	M	M		✓	
R_{69}	H	L	M	M	L		✓	
R_{70}	H	L	M	L	H		✓	
R_{71}	H	L	M	L	M		✓	
R_{72}	H	L	M	L	L			✓
R_{73}	H	L	L	H	H		✓	
R_{74}	H	L	L	H	M		✓	
R_{75}	H	L	L	H	L		✓	
R_{76}	H	L	L	M	H		✓	
R_{77}	H	L	L	M	M		✓	
R_{78}	H	L	L	M	L			✓
R_{79}	H	L	L	L	H		✓	
R_{80}	H	L	L	L	M			✓
R_{81}	H	L	L	L	L			✓
R_{82}	M	H	H	H	H	✓		
R_{83}	M	H	H	H	M	✓		
R_{84}	M	H	H	H	L	✓		
R_{85}	M	H	H	M	H	✓		
R_{86}	M	H	H	M	M	✓		
R_{87}	M	H	H	M	L		✓	
R_{88}	M	H	H	L	H	✓		
R_{89}	M	H	H	L	M		✓	
R_{90}	M	H	H	L	L		✓	
R_{91}	M	H	M	H	H	✓		
R_{92}	M	H	M	H	M	✓		
R_{93}	M	H	M	H	L		✓	
R_{94}	M	H	M	M	H	✓		

	E_3	E_4	E_5	E_6	E_7	L_1	L_2	L_3
R_{95}	M	H	M	M	M		✓	
R_{96}	M	H	M	M	L		✓	
R_{97}	M	H	M	L	H		✓	
R_{98}	M	H	M	L	M		✓	
R_{99}	M	H	M	L	L		✓	
R_{100}	M	H	L	H	H	✓		
R_{101}	M	H	L	H	M		✓	
R_{102}	M	H	L	H	L		✓	
R_{103}	M	H	L	M	H		✓	
R_{104}	M	H	L	M	M		✓	
R_{105}	M	H	L	M	L		✓	
R_{106}	M	H	L	L	H		✓	
R_{107}	M	H	L	L	M		✓	
R_{108}	M	H	L	L	L			✓
R_{109}	M	M	H	H	H	✓		
R_{110}	M	M	H	H	M	✓		
R_{111}	M	M	H	H	L		✓	
R_{112}	M	M	H	M	H	✓		
R_{113}	M	M	H	M	M		✓	
R_{114}	M	M	H	M	L		✓	
R_{115}	M	M	H	L	H		✓	
R_{116}	M	M	H	L	M		✓	
R_{117}	M	M	H	L	L		✓	
R_{118}	M	M	M	H	H	✓		
R_{119}	M	M	M	H	M		✓	
R_{120}	M	M	M	H	L		✓	
R_{121}	M	M	M	M	H		✓	
R_{122}	M	M	M	M	M		✓	
R_{123}	M	M	M	M	L		✓	
R_{124}	M	M	M	L	H		✓	
R_{125}	M	M	M	L	M		✓	
R_{126}	M	M	M	L	L			✓
R_{127}	M	M	L	H	H		✓	
R_{128}	M	M	L	H	M		✓	
R_{129}	M	M	L	H	L		✓	
R_{130}	M	M	L	M	H		✓	
R_{131}	M	M	L	M	M		✓	
R_{132}	M	M	L	M	L			✓
R_{133}	M	M	L	L	H		✓	
R_{134}	M	M	L	L	M			✓
R_{135}	M	M	L	L	L			✓
R_{136}	M	L	H	H	H	✓		
R_{137}	M	L	H	H	M		✓	
R_{138}	M	L	H	H	L		✓	
R_{139}	M	L	H	M	H		✓	
R_{140}	M	L	H	M	M		✓	
R_{141}	M	L	H	M	L		✓	
R_{142}	M	L	H	L	H		✓	
R_{143}	M	L	H	L	M		✓	
R_{144}	M	L	H	L	L			✓
R_{145}	M	L	M	H	H		✓	
R_{146}	M	L	M	H	M		✓	
R_{147}	M	L	M	H	L		✓	
R_{148}	M	L	M	M	H		✓	
R_{149}	M	L	M	M	M		✓	
R_{150}	M	L	M	M	L			✓
R_{151}	M	L	M	L	H		✓	
R_{152}	M	L	M	L	M			✓
R_{153}	M	L	M	L	L			✓
R_{154}	M	L	L	H	H		✓	
R_{155}	M	L	L	H	M		✓	
R_{156}	M	L	L	H	L			✓
R_{157}	M	L	L	M	H		✓	
R_{158}	M	L	L	M	M			✓
R_{159}	M	L	L	M	L			✓
R_{160}	M	L	L	L	H			✓

	E_3	E_4	E_5	E_6	E_7	L_1	L_2	L_3
R_{161}	M	L	L	L	M			✓
R_{162}	M	L	L	L	L			✓
R_{163}	L	H	H	H	H	✓		
R_{164}	L	H	H	H	M	✓		
R_{165}	L	H	H	H	L		✓	
R_{166}	L	H	H	M	H	✓		
R_{167}	L	H	H	M	M		✓	
R_{168}	L	H	H	M	L		✓	
R_{169}	L	H	H	L	H		✓	
R_{170}	L	H	H	L	M		✓	
R_{171}	L	H	H	L	L		✓	
R_{172}	L	H	M	H	H	✓		
R_{173}	L	H	M	H	M		✓	
R_{174}	L	H	M	H	L		✓	
R_{175}	L	H	M	M	H		✓	
R_{176}	L	H	M	M	M		✓	
R_{177}	L	H	M	M	L		✓	
R_{178}	L	H	M	L	H		✓	
R_{179}	L	H	M	L	M		✓	
R_{180}	L	H	M	L	L			✓
R_{181}	L	H	L	H	H		✓	
R_{182}	L	H	L	H	M		✓	
R_{183}	L	H	L	H	L		✓	
R_{184}	L	H	L	M	H		✓	
R_{185}	L	H	L	M	M		✓	
R_{186}	L	H	L	M	L			✓
R_{187}	L	H	L	L	H		✓	
R_{188}	L	H	L	L	M			✓
R_{189}	L	H	L	L	L			✓
R_{190}	L	M	H	H	H	✓		
R_{191}	L	M	H	H	M		✓	
R_{192}	L	M	H	H	L		✓	
R_{193}	L	M	H	M	H		✓	
R_{194}	L	M	H	M	M		✓	
R_{195}	L	M	H	M	L		✓	
R_{196}	L	M	H	L	H		✓	
R_{197}	L	M	H	L	M		✓	
R_{198}	L	M	H	L	L			✓
R_{199}	L	M	M	H	H		✓	
R_{200}	L	M	M	H	M		✓	
R_{201}	L	M	M	H	L		✓	
R_{202}	L	M	M	M	H		✓	
R_{203}	L	M	M	M	M		✓	
R_{204}	L	M	M	M	L			✓
R_{205}	L	M	M	L	H		✓	
R_{206}	L	M	M	L	M			✓
R_{207}	L	M	M	L	L			✓
R_{208}	L	M	L	H	H		✓	
R_{209}	L	M	L	H	M		✓	
R_{210}	L	M	L	H	L			✓
R_{211}	L	M	L	M	H		✓	
R_{212}	L	M	L	M	M			✓
R_{213}	L	M	L	M	L			✓
R_{214}	L	M	L	L	H			✓
R_{215}	L	M	L	L	M			✓
R_{216}	L	M	L	L	L			✓
R_{217}	L	L	H	H	H		✓	
R_{218}	L	L	H	H	M		✓	
R_{219}	L	L	H	H	L		✓	
R_{220}	L	L	H	M	H		✓	
R_{221}	L	L	H	M	M		✓	
R_{222}	L	L	H	M	L			✓
R_{223}	L	L	H	L	H		✓	
R_{224}	L	L	H	L	M			✓
R_{225}	L	L	H	L	L			✓
R_{226}	L	L	M	H	H		✓	

	E_3	E_4	E_5	E_6	E_7	L_1	L_2	L_3
R_{227}	L	L	M	H	M		✓	
R_{228}	L	L	M	H	L			✓
R_{229}	L	L	M	M	H		✓	
R_{230}	L	L	M	M	M			✓
R_{231}	L	L	M	M	L			✓
R_{232}	L	L	M	L	H			✓
R_{233}	L	L	M	L	M			✓
R_{234}	L	L	M	L	L			✓
R_{235}	L	L	L	H	H		✓	
R_{236}	L	L	L	H	M			✓
R_{237}	L	L	L	H	L			✓
R_{238}	L	L	L	M	H			✓
R_{239}	L	L	L	M	M			✓
R_{240}	L	L	L	M	L			✓
R_{241}	L	L	L	L	H			✓
R_{242}	L	L	L	L	M			✓
R_{243}	L	L	L	L	L			✓

D. Entscheidungstabelle für das Outsourcingpotenzial

Die nachfolgende Tabelle wurde aus Darstellungsgründen im Vergleich zu Tabelle 3.6 transponiert. Weiterhin sind die verwendeten Abkürzungen wie folgt definiert:

A_1 Aggregat der strukturellen Eigenschaften einer Komponente in Bezug auf das Softwareprodukt (vgl. Tabelle 3.6)

A_2 Aggregat der Prozessmerkmale der Entwicklung einer Komponente (vgl. Anhang B)

A_3 Aggregat der Wissensmerkmale einer Komponente (vgl. Anhang C)

L_1 Lösungsalternative *Hohes Outsourcingpotenzial*

L_2 Lösungsalternative *Mittleres Outsourcingpotenzial*

L_3 Lösungsalternative *Niedriges Outsourcingpotenzial*

R_i Regel i als Ausprägungsvektor der Entscheidungsgrößen

H / M / L hoch / mittel / niedrig

	A_1	A_2	A_3	L_1	L_2	L_3
R_1	H	H	H			✓
R_2	H	H	M			✓
R_3	H	H	L			✓
R_4	H	M	H			✓
R_5	H	M	M			✓
R_6	H	M	L		✓	
R_7	H	L	H			✓
R_8	H	L	M		✓	
R_9	H	L	L	✓		
R_{10}	M	H	H			✓
R_{11}	M	H	M			✓
R_{12}	M	H	L		✓	
R_{13}	M	M	H			✓
R_{14}	M	M	M		✓	
R_{15}	M	M	L	✓		
R_{16}	M	L	H		✓	
R_{17}	M	L	M	✓		
R_{18}	M	L	L	✓		
R_{19}	L	H	H			✓

	$\mathbf{A_1}$	$\mathbf{A_2}$	$\mathbf{A_3}$	$\mathbf{L_1}$	$\mathbf{L_2}$	$\mathbf{L_3}$
R_{20}	L	H	M		✓	
R_{21}	L	H	L	✓		
R_{22}	L	M	H		✓	
R_{23}	L	M	M	✓		
R_{24}	L	M	L	✓		
R_{25}	L	L	H	✓		
R_{26}	L	L	M	✓		
R_{27}	L	L	L	✓		

E. Beispielcode der mobilen App

Methode zur Instanziierung des Models

```
/*  Method that initializes the project model
 *  It downloads all information about a project
 *  including its components from the webservice
 */

- (void)initializeProjectModel
{
  //build project Model
  NSArray *project = [self.platformModel getSelectedProject];
  self.projectModel = [[ProjectModel alloc] initWithProjectID:[
    project objectAtIndex:0] useSoda:YES];
  //self.projectModel = [[ProjectModel alloc] initWithProjectID:[
    project objectAtIndex:0]];
  self.projectInfo = [self.projectModel getProjectInfoArray];
}
```

Generierung des PDF-Reports

```
/*  Method that generates the pdf report
 *  including all results
 */

- (void)generatePdfWithFilePath:(NSString *)thefilePath
{
  UIGraphicsBeginPDFContextToFile(thefilePath, CGRectZero, nil);

  self.currentPage = 0;

  if (self.currentPage == 0) {
      //draw overview over outsourcing recommendations
      [self drawOverview];
  }
```

```
15
16   for (NSArray *category in self.resultArray) {
17       for (Component *currComponent in category) {
18           [self drawComponentExplanationForComponent:currComponent];
19       }}
20   //draw overview table
21   [self drawOverViewTable];
22   // Close the PDF context and write the contents out.
23   UIGraphicsEndPDFContext();
24 }
```

Algorithmus zur Berechnung von Kopplung und Kohäsion

```
1 //
2 //  SODAFunctions.m
3 //  SmartSourcer
4
5 #import "SODAFunctions.h"
6 #import "GraphEdge.h"
7 #import "GraphNode.h"
8 #import "Requirement.h"
9
10 @implementation SODAFunctions
11
12 /*  calculates the cohesion value of a subgraph of an entire
      requirementsGraph
13  *  the subgraph is defined and built up by a set of requirements (
      nodes) and the edges between
14  *  them in the entire requirements graph
15  */
16 + (CGFloat)getCohesionOfClusterWithRequirementsSubset:(NSSet *)
      requirementsSubset inRequirementsGraph:(Graph *)requirementsGraph
17 {
18
19     //get total weight of edges of subgraph - only internal edges
20     Graph *subGraph = [requirementsGraph getSubgraphForNodeSet:
      requirementsSubset];
21     CGFloat weightOfIntraEdges = 0.0;
22     for (GraphEdge *intraEdge in [subGraph getAllEdges]) {
23         weightOfIntraEdges += [intraEdge getWeight];
24     }
```

```
25      //get total weight of edges that belong to internal vertices -
    includes edges to external vertices
26      NSMutableSet *interEdges = [[requirementsGraph
    getAllEdgesLinkedWithNodes:requirementsSubset] mutableCopy];
27      CGFloat weightOfInterEdges = 0.0;
28      for (GraphEdge *interEdge in interEdges) {
29          weightOfInterEdges += [interEdge getWeight];
30      }
31      //avoid error
32      if (weightOfInterEdges == 0) {
33          return 0.0;
34      }
35      //calculate intraEdges/allEdges --> cohesion
36      CGFloat cohesion = (weightOfIntraEdges/weightOfInterEdges);
37      return  cohesion;
38  }
39
40  + (CGFloat)getCouplingOfClusterWithRequirementsSubset:(NSSet *)
    requirementsSubset inRequirementsGraph:(Graph *)requirementsGraph
41  {
42      //weight of edges that link requirements of the subgraph to
    external requirements
43      CGFloat weightOfEdgesToExternalNode = 0.0;
44      //subgraph built by requirements subset
45      Graph *subGraph = [requirementsGraph getSubgraphForNodeSet:
    requirementsSubset];
46      NSSet *internalNodes = [subGraph getAllNodes];
47      //iterate internal nodes
48      for (GraphNode *internalNode in internalNodes) {
49          //look for same node in complete graph
50          GraphNode *nodeInEntireGraph = [requirementsGraph
    nodeWithValue:[internalNode getValue]];
51          //iterate edges
52          for (GraphEdge *edgeFromNode in [nodeInEntireGraph getEdges
    ]) {
53              //if edge leads to node outside the subgraph, add its
    weight
54              if (![subGraph containsNode:[edgeFromNode
    getNodeOtherThan:nodeInEntireGraph]] ) {
55                  weightOfEdgesToExternalNode += [edgeFromNode
    getWeight];
```

```
56              }
57          }
58      }
59      return weightOfEdgesToExternalNode;
60  }
61
62
63  /*  takes a dictionary of cluster requirements and a graph of their
        interdependencies
64   *  parameter clusters: dictionary with
65   *          keys: nsstring of cluster identifiers
66   *          values: nsset of nodes from the requirements graph that
        belong to the cluster
67   *
68   *  the method calculates the quotient of the total weight of the
        edges that lead out of one cluster and the total weight of all
        inter-cluster edges (edges that connect the clusters of the
        entire graph)
69   *  this provides a relative value for the coupling of these
        clusters
70   *
71   *  return value: dictionary of cluster identifiers and relative
        coupling values
72   */
73  + (NSDictionary *)getCouplingValuesForClusteringDictionary:(
        NSDictionary *)clusters inRequirementsGraph:(Graph *)
        requirementsGraph
74  {
75      //all interedges of the graph
76      NSMutableSet *overAllInterEdges = [NSMutableSet set];
77      NSMutableDictionary *sumWeightOfInterEdges = [
        NSMutableDictionary dictionaryWithCapacity:[clusters count]];
78      //iterate clusters
79      NSArray *clusterIDArray = [clusters allKeys];
80      for (NSString *clusterID in clusterIDArray) {
81          CGFloat weightInterEdgesThisCluster = 0.0;
82          //build subgraph for this cluster
83          Graph *subGraph = [requirementsGraph getSubgraphForNodeSet:[
        clusters objectForKey:clusterID]];
84          NSSet *internalNodes = [subGraph getAllNodes];
85          for (GraphNode *internalNode in internalNodes) {
```

```
86              //look for same node in complete graph
87              GraphNode *nodeInEntireGraph = [requirementsGraph
    nodeWithValue:[internalNode getValue]];
88              //iterate edges
89              for (GraphEdge *edgeFromNode in [nodeInEntireGraph
    getEdges]) {
90                  //if edge leads to node outside the subgraph, add
    its weight
91                  if (![subGraph containsNode:[edgeFromNode
    getNodeOtherThan:nodeInEntireGraph]] ){
92                      //add weight to sum of this cluster
93                      weightInterEdgesThisCluster += [edgeFromNode
    getWeight];
94                      //add edge to set of overall interedges
95                      if (![overAllInterEdges containsObject:
    edgeFromNode]) {
96                          [overAllInterEdges addObject:edgeFromNode];
97                      }
98                  }
99              }
100             //write sum into dictionary
101             [sumWeightOfInterEdges setObject:[NSNumber
    numberWithFloat:weightInterEdgesThisCluster] forKey:clusterID];
102         }
103     }
104     //get total weight of interedges
105     CGFloat totalWeightOfInterEdges = 0.0;
106     for (GraphEdge *oneInterEdge in overAllInterEdges) {
107         totalWeightOfInterEdges += [oneInterEdge getWeight];
108     }
109     //prevent 0 as a divisor
110     if (totalWeightOfInterEdges > 0) {
111         //calculate relative value for each cluster (aka subset of
    requirements)
112         for (NSString *clusterID in clusterIDArray) {
113             NSNumber *weightInterEdgesThisCluster = [
    sumWeightOfInterEdges objectForKey:clusterID];
114             CGFloat relativeValue = [weightInterEdgesThisCluster
    floatValue]/totalWeightOfInterEdges;
115             [sumWeightOfInterEdges setObject:[NSNumber
    numberWithFloat:relativeValue] forKey:clusterID];
```

```
116         }
117         NSDictionary *output = [sumWeightOfInterEdges copy];
118         return output;
119     }
120     //return array of 0s in case totalWeightOfInterEdges = 0
121     for (NSString *clusterID in clusterIDArray) {
122         [sumWeightOfInterEdges setObject:[NSNumber numberWithFloat
        :0.0] forKey:clusterID];
123     }
124     NSDictionary *output = [sumWeightOfInterEdges copy];
125     return output;
126 }
127
128 /*  takes a relative value between 0 and 1 and returns
129  *  a categorization into low(1), medium(2), or high(3)
130  */
131 + (NSNumber *)get123ValueForLinearValue:(CGFloat)linearFloatValue
132 {
133     if (linearFloatValue <= 0.33) {
134         return [NSNumber numberWithInt:1];
135     } else if (linearFloatValue <= 0.66) {
136         return [NSNumber numberWithInt:2];
137     } else if (linearFloatValue <= 1.0) {
138         return [NSNumber numberWithInt:3];
139     } else {
140         return [NSNumber numberWithInt:0];
141     }
142 }
143 @end
```

Für die Entwicklung wurde der Code im Verwaltungssystem von *Github*[1] abgespeichert. Der gesamte Code ist dort für die Öffentlichkeit zugänglich und kann unter https://github.com/ISDepartment-UMA/DecisionApp abgerufen werden.

1 https://github.com (abgerufen am 01.12.2015)

F. Beispielcode des Proxyservers

Methode zum Abruf aller Projekte

```
1  /**
2     * Method returns all Projects from a platform
3     * @param url : url of the codebeamer instance - needs to be
     encoded by UTF-8
4     * @param login : username of codebeamer user
5     * @param password : password of codebeamer user
6     * @return : Stringarray of all projects from the platform
7     *
8     */
9    @GET
10   @Path("/getAllProjects")
11   @Produces(MediaType.APPLICATION_JSON)
12   public String[][] getAllProjects(@QueryParam("url") @Encoded
     String url, @QueryParam("login") String login, @QueryParam("
     password") String password){
13
14
15     // connecting
16     RemoteApi api;
17     try {
18       String codeBeamerUrl = "";
19       try {
20         codeBeamerUrl = URLDecoder.decode(url, "UTF-8");
21       } catch (Exception e) {
22       }
23
24       api = RemoteApiFactory.getInstance().connect(codeBeamerUrl);
25       if (api == null) {
26         String[][] out = new String[1][1];
27         out[0][0] = "wrong url";
28         return out;
29       }
```

```
30
31        // signing in
32        String token;
33        try{
34          token = api.login(login, password);
35        } catch (AccessControlException e){
36          String[][] out = new String[1][1];
37          out[0][0] = "wrong login";
38          return out;
39        }
40
41        if (token == null) {
42          String[][] out = new String[1][1];
43          out[0][0] = "token null";
44          return out;
45        }
46
47        // retrieving project names
48        ProjectDto projects[] = api.findAllProjects(token);
49        String[][] aus = new String[projects.length][3];
50        for (int i = 0; i < projects.length; i++) {
51          ProjectDto currentProject = projects[i];
52          aus[i][0] = currentProject.getId().toString();
53          aus[i][1] = currentProject.getName();
54          aus[i][2] = currentProject.getDescription();
55        }
56        return aus;
57
58      } catch (MalformedURLException e) {
59        String[][] out = new String[1][1];
60        out[0][0] = "malformed url";
61        return out;
62      }
63
64    }
```

Für die Entwicklung wurde ebenfalls das Verwaltungssystem von *Github* verwendet. Der gesamte Code ist dort für die Öffentlichkeit zugänglich und kann unter https://github.com/ISDepartment-UMA/DecisionAppProxy abgerufen werden.

G. Fragebogen für die künstliche und naturalistische Evaluation des Artefakts

Information Quality		strongly disagree				strongly agree
M-IN-1	Relevant information is easy to reach within the system.	☐	☐	☐	☐	☐
M-IN-2	I am given the opportunity to search for specific desired pieces of information.	☐	☐	☐	☐	☐
M-IN-3	The information is presented in a format that makes it easy to understand.	☐	☐	☐	☐	☐
M-IN-4	The information format encourages an intuitive conception of its meaning.	☐	☐	☐	☐	☐
M-IN-5	Working with the app, I see the need to interact with other people.	☐	☐	☐	☐	☐
M-IN-6	I would like to perform this interaction with others through the system.	☐	☐	☐	☐	☐
M-IN-7	The app intuitively draws my attention to relevant information.	☐	☐	☐	☐	☐
M-IN-8	It occurs that I am looking for certain information for some time before I can find it.	☐	☐	☐	☐	☐

Quality of Decision Logic		strongly disagree				strongly agree
M-VP-1	Working with the app gave me an idea of the outsourcing decision making rationale embodied in the system.	☐	☐	☐	☐	☐
M-VP-2	I believe that working with the system increases my individual skills.	☐	☐	☐	☐	☐
M-VP-3	The evaluation criteria in the system encouraged me to consider factors I would not have considered without the app.	☐	☐	☐	☐	☐
M-VP-4	The app allows for a rapid understanding of the decision rationale embodied.	☐	☐	☐	☐	☐
M-VP-5	Working with the system to support the decision process was exhaustive.	☐	☐	☐	☐	☐
M-VP-6	I feel that the app is based on a theoretically founded rationale.	☐	☐	☐	☐	☐
M-VP-7	The app acknowledges user input and provides feedback on actions taken.	☐	☐	☐	☐	☐
M-VP-8	It is easy to see how different user input effects the system's output.	☐	☐	☐	☐	☐

Usability		strongly disagree				strongly agree
M-SU-1	I found the system unnecessarily complex.	☐	☐	☐	☐	☐
M-SU-2	I thought the system was easy to use.	☐	☐	☐	☐	☐
M-SU-3	I think that I would need the support of a technical person to be able to use this system.	☐	☐	☐	☐	☐
M-SU-4	I found the various functions in this system were well integrated.	☐	☐	☐	☐	☐
M-SU-5	I thought there was too much inconsistency in this system.	☐	☐	☐	☐	☐
M-SU-6	I would imagine that most people would learn to use this system very quickly.	☐	☐	☐	☐	☐
M-SU-7	I found the system very cumbersome to use.	☐	☐	☐	☐	☐
M-SU-8	I felt very confident using the system.	☐	☐	☐	☐	☐
M-SU-9	I needed to learn a lot of things before I could get going with this system.	☐	☐	☐	☐	☐
M-IM-1	I am satisfied with the look and feel of the app.	☐	☐	☐	☐	☐
M-IM-2	The app frequently becomes inaccessible or breaks down.	☐	☐	☐	☐	☐

M-IM-3	My confusion with the system created situations when I did not know how to continue (break downs).	☐	☐	☐	☐	☐

Use		strongly disagree				strongly agree
M-NB-1	I think that I would like to use this system frequently.	☐	☐	☐	☐	☐

H. Interviewleitfaden für die qualitative Evaluation

Personal Data

1. What is your position in the company?
2. How often do you have to deal with the decision of software component outsourcing?
3. How long have you been involved in outsourcing decision making?

Individual

4. Do you see room for improvement of the look and feel of the app? *(Satisfaction)*
5. Does the positioning of buttons and other user interface elements allow for an intuitive navigation through the app? *(Learnability, Satisfaction)*
   a) Which elements could be placed somewhere else?
   b) Which elements should be stressed? *(Amplifiers)*
   c) Which elements should provide more feedback? *(Feedback)*

Organizational Fit

6. Do you consider a rationale to base your outsourcing decision upon helpful for your company? *(Relevance)*
7. To what extend does the method embodied in the system fulfill this need? *(Efficacy, Impact on environment)* / What are the weaknesses of the system?

8. Could the app be used effectively on the operational level by members within your organization? *(Operationality)*
9. Do you believe that using the system can make a difference for your organization? *(Impact on environment)* And what could be improved?
    a) Decision Speed
    b) Decision Quality
    c) Return on Investment *(ROI)*
    d) Competitiveness
    e) Individual Skills *(Learning)*
10. Does the method contain any parts that reflect current working practices within your organization? *(Timliness and Actuality)*
11. Which could be possible upcoming business needs which require a modification to the system? *(Flexibility)*
12. How strong is your trust in the correctness and accuracy of the app's resulting recommendations? *(Decision Acuracy)*
13. Do you believe that using the system could decrease decision maker's stress while making the outsourcing decision? *(Stress)*
14. Do you see the need for functionality to modify the displayed information in order to adapt it to the user's needs? *(Control)*

Information Quality

15. Is there any missing, misplaced or irrelevant information at any point during the workflow? *(Information relevance, Overload, Underfeed)*
    a) If information is missing, does your project collaboration platform provide this information?
16. Is there any point during the workflow that should provide further search functionality to find desired pieces of information? *(Searchability)*
17. Do you see the need to archive the output generated by the app? *(Information Archivability)*

a) If yes, what format should the archive have? *(Compatibility)*

18. Does the app provide sufficient possibilities to trace the output generation back to its origins? *(Information traceability)*

Technology

19. Which project collaboration platforms does the system need to support in order to be compatible within your company? *(Compatibility)*
20. Which other project collaboration platforms do you consider relevant for the system in order to be compatible across the industry? *(Compatibility)*

Systemics

21. Are decision tables implemented in the system accurately? *(Efficacy)*
22. How easy do you perceive wrong input to be corrected or to try out the effect of different inputs on the end result? *(Playfulness)*

I. Ergebnistabellen der deskriptiven Evaluation

Tabelle I.1.: Gruppenstatistik beim t-Test zum Vergleich der beiden Gruppen (Experten / Studenten)

	EXP	H	Mittelwert	Standardabweichung	Standardfehler Mittelwert
utility	Experte	15	4,0867	,35501	,09166
	kein Experte	15	4,0286	,38802	,10019
usability	Experte	15	4,4500	,24458	,06315
	kein Experte	15	4,5667	,36249	,09359
M_NB_1	Experte	15	4,13	,743	,192
	kein Experte	15	4,40	,507	,131

Tabelle I.2.: Ergebnisse des t-Tests

		Levene-Test der Varianzgleichheit		T-Test für die Mittelwertgleichheit		
		F	Sig.	t	df	Sig. (2-seitig)
utility	Varianzgleichheit angenommen	,065	,800	,428	28	,672
	Varianzgleichheit nicht angenommen			,428	27,782	,672
usability	Varianzgleichheit angenommen	1,968	,172	-1,033	28	,310
	Varianzgleichheit nicht angenommen			-1,033	24,559	,312
M_NB_1	Varianzgleichheit angenommen	,700	,410	-1,148	28	,261
	Varianzgleichheit nicht angenommen			-1,148	24,713	,262

Tabelle I.3.: Gesamtergebnis der deskriptiven Analyse

Konstrukt		Kennz.	Mittel-wert	Std.-abw.	n	Max. Wert	Min. Wert
Wahrgenommene Nützlichkeit (Qualität der Entscheidung)	**Qualität der Information**	M-IN-1	4,33	0,55	30	5	3
		M-IN-2	4,00	0,87	30	5	2
		M-IN-3	4,57	0,57	30	5	3
		M-IN-4	4,27	0,52	30	5	3
		M-IN-5	3,33	1,30	30	5	1
		M-IN-6	2,65	1,50	26	5	1
		M-IN-7	4,27	0,74	30	5	2
		M-IN-8	4,23	0,82	30	5	2
	Qualität der Entscheidungslogik	M-VP-1	4,23	0,57	30	5	3
		M-VP-2	3,77	0,77	30	5	2
		M-VP-3	4,10	0,76	30	5	3
		M-VP-4	4,17	0,59	30	5	3
		M-VP-5	4,00	0,91	30	5	2
		M-VP-6	4,37	0,89	30	5	2
		M-VP-7	4,07	0,78	30	5	2
		M-VP-8	3,93	0,98	30	5	1
			4,06				
Wahrgenommene Benutzerfreundlichkeit	**Qualität der Implementierung**	M-SU-1	4,70	0,47	30	5	4
		M-SU-2	4,53	0,57	30	5	3
		M-SU-3	4,57	0,68	30	5	3
		M-SU-4	4,37	0,67	30	5	2
		M-SU-5	4,57	0,68	30	5	2
		M-SU-6	4,50	0,57	30	5	3
		M-SU-7	4,20	0,89	30	5	2
		M-SU-8	4,27	0,58	30	5	3
		M-SU-9	4,40	0,62	30	5	3
		M-IM-1	4,30	0,70	30	5	3
		M-IM-2	4,83	0,46	30	5	3
		M-IM-3	4,60	0,62	30	5	3
			4,49				
Nutzungsber.		M-NB-1	**4,27**	0,64	30	5	3

Tabelle I.4.: Ergebnis der deskriptiven Analyse des Studentenexperiments

Konstrukt		Kennz.	Mittel-wert	Std.-abw.	n	Max. Wert	Min. Wert
Wahrgenommene Nützlichkeit (Qualität der Entscheidung)	**Qualität der Information**	M-IN-1	4,40	0,51	15	5	4
		M-IN-2	3,80	1,01	15	5	2
		M-IN-3	4,80	0,41	15	5	4
		M-IN-4	4,27	0,46	15	5	4
		M-IN-5	2,80	1,21	15	5	1
		M-IN-6	2,82	1,47	11	5	1
		M-IN-7	4,40	0,83	15	5	2
		M-IN-8	4,27	0,80	15	5	2
	Qualität der Entscheidungslogik	M-VP-1	4,27	0,59	15	5	3
		M-VP-2	3,73	0,80	15	5	2
		M-VP-3	3,93	0,80	15	5	3
		M-VP-4	4,33	0,62	15	5	3
		M-VP-5	4,20	0,86	15	5	2
		M-VP-6	4,27	0,96	15	5	2
		M-VP-7	4,00	0,93	15	5	2
		M-VP-8	3,80	1,15	15	5	1
			4,01				
Wahrgenommene Benutzerfreundlichkeit	**Qualität der Implementierung**	M-SU-1	4,87	0,35	15	5	4
		M-SU-2	4,60	0,63	15	5	3
		M-SU-3	4,60	0,63	15	5	3
		M-SU-4	4,27	0,80	15	5	2
		M-SU-5	4,47	0,83	15	5	2
		M-SU-6	4,73	0,46	15	5	4
		M-SU-7	4,27	0,96	15	5	2
		M-SU-8	4,27	0,59	15	5	3
		M-SU-9	4,67	0,49	15	5	4
		M-IM-1	4,53	0,52	15	5	4
		M-IM-2	4,80	0,56	15	5	3
		M-IM-3	4,40	0,74	15	5	3
			4,54				
Nutzungsber.		M-NB-1	**4,40**	0,51	15	5	4

Tabelle I.5.: Ergebnis der deskriptiven Analyse der Expertenbefragung

Konstrukt		Kennz.	Mittel-wert	Std.-abw.	n	Max. Wert	Min. Wert
Wahrgenommene Nützlichkeit (Qualität der Entscheidung)	**Qualität der Information**	M-IN-1	4,27	0,59	15	5	3
		M-IN-2	4,20	0,68	15	5	3
		M-IN-3	4,33	0,62	15	5	3
		M-IN-4	4,27	0,59	15	5	3
		M-IN-5	3,87	1,19	15	5	1
		M-IN-6	2,53	1,55	15	5	1
		M-IN-7	4,13	0,64	15	5	3
		M-IN-8	4,20	0,86	15	5	2
	Qualität der Entscheidungslogik	M-VP-1	4,20	0,56	15	5	3
		M-VP-2	3,80	0,77	15	5	3
		M-VP-3	4,27	0,70	15	5	3
		M-VP-4	4,00	0,53	15	5	3
		M-VP-5	3,80	0,94	15	5	2
		M-VP-6	4,47	0,83	15	5	2
		M-VP-7	4,13	0,64	15	5	3
		M-VP-8	4,07	0,80	15	5	2
			4,03				
Wahrgenommene Benutzerfreundlichkeit	**Qualität der Implementierung**	M-SU-1	4,53	0,52	15	5	4
		M-SU-2	4,47	0,52	15	5	4
		M-SU-3	4,53	0,74	15	5	3
		M-SU-4	4,47	0,52	15	5	4
		M-SU-5	4,67	0,49	15	5	4
		M-SU-6	4,27	0,59	15	5	3
		M-SU-7	4,13	0,83	15	5	2
		M-SU-8	4,27	0,59	15	5	3
		M-SU-9	4,13	0,64	15	5	3
		M-IM-1	4,07	0,80	15	5	3
		M-IM-2	4,87	0,35	15	5	4
		M-IM-3	4,80	0,41	15	5	4
			4,43				
Nutzungsber.		M-NB-1	**4,13**	0,74	15	5	3

Literaturverzeichnis

[Abrahamsson et al. 2004]

Abrahamsson, P. ; Hanhineva, A. ; Hulkko, H. ; Ihme, T. ; Jäälinoja, J. ; Korkala, M. ; Koskela, J. ; Kyllönen, P. ; Salo, O.: Mobile-D: An Agile Approach for Mobile Application Development. In: *Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, 2004, S. 174–175.

[Abran et al. 1999]

Abran, A. ; Bourque, P. ; Dupuis, R. ; Moore, J. W. ; Tripp, L. L.: Guide to the Software Engineering Body of Knowledge. In: *IEEE Software* 16 (1999), Nr. 6, S. 35–44.

[Adipat und Zhang 2005]

Adipat, B. ; Zhang, D.: Interface Design for Mobile Applications. In: *AMCIS 2005 Proceedings*, 2005, S. 2281–2290.

[Al-Otaiby und AlSharif 2007]

Al-Otaiby, T. N. ; AlSharif, M.: Software requirements modularization using partitioning clustering technique. In: *Proceedings of the 45th annual Southeast Regional Conference*, 2007, S. 65–69.

[Alahuhta et al. 2005]

Alahuhta, P. ; Ahola, J. ; Hakala, H.: Mobilizing Business Applications. In: *Technology Review, Tekes, Helsinki* 167 (2005).

[Aloul et al. 2009]

Aloul, F. ; Zahidi, S. ; El-Hajj, W.: Two Factor Authentication Using Mobile Phones. In: *2009 IEEE/ACS International Conference on Computer Systems and Applications*, IEEE Computer Society, 2009, S. 641–644.

[Ang und Slaughter 1998]

Ang, S. ; Slaughter, S. A.: Organizational Psychology and Performance

in IS Employment Outsourcing and Insourcing. In: *Proceedings of the 31st Hawaii International Conference on System Sciences* Bd. 6, 1998, S. 635–643.

[Applegate und Montealegre 1991]
APPLEGATE, L. ; MONTEALEGRE, R.: Eastman Kodak Organization: Managing Information Systems Through Strategic Alliances / Harvard Business School Case 9-192-030. Boston, USA, 1991. – Forschungsbericht.

[Applegate et al. 2003]
APPLEGATE, L. M. ; MCFARLAN, F. W. ; AUSTIN, R. D.: *Corporate Information Strategy and Management: Text and Cases.* New York : McGraw-Hill, 2003.

[Apte 1990]
APTE, U.: Global Outsourcing of Information Systems and Processing Services. In: *The Information Society* 7 (1990), S. 287–303.

[Ariav und Ginzberg 1985]
ARIAV, G. ; GINZBERG, M. J.: DSS Design: a Systemic View of Decision Support. In: *Communications of the ACM* 28 (1985), Nr. 10, S. 1045–1052.

[Arnett und Jones 1994]
ARNETT, K. P. ; JONES, M. C.: Firms That Choose Outsourcing: A Profile. In: *Information and Management* 26 (1994), Nr. 4, S. 179–188.

[Arnott und Pervan 2008]
ARNOTT, D. ; PERVAN, G.: Eight Key Issues for the Decision Support Systems Discipline. In: *Decision Support Systems* 44 (2008), Nr. 3, S. 657–672.

[Aspray et al. 2006]
ASPRAY, W. ; MAYADAS, F. ; VARDI, M. Y.: *Globalization and Offshoring of Software – A Report of the ACM Job Migration Task Force.* New York : Association for Computing Machinery, 2006.

[Atkinson et al. 2002]
ATKINSON, C. ; BAYER, J. ; BUNSE, C. ; KAMSTIES, E. ; LAITENBERGER, O. ; LAQUA, R. ; MUTHIG, D. ; PAECH, B. ; WÜST, J. ; ZETTEL, J.: *Component-based Product Line Engineering with UML.* London : Addison-Wesley, 2002.

[Atkinson et al. 1983]
Atkinson, M. ; Bailey, P. ; Chisholm, K. ; Cockshott, W. ; Morrison, R.: An Approach to Persistent Programming. In: *Computer Journal* 26 (1983), Nr. 4, S. 360–365.

[Auster und Choo 1993]
Auster, E. ; Choo, C. W.: Environmental Scanning by CEOs in two Canadian Industries. In: *Journal of the American Society for Information Science* 44 (1993), S. 194–203.

[Bair 1989]
Bair, J. H.: Supporting Cooperative Work with Computers: Addressing Meetingmania. In: *Proceedings of the 34th IEEE Computer Society International Conference (COMPCON Spring '89)*, 1989, S. 208–217.

[Balagtas-Fernandez et al. 2009]
Balagtas-Fernandez, F. ; Forrai, J. ; Hussmann, H.: Evaluation of User Interface Design and Input Methods for Applications on Mobile Touch Screen Devices. In: *Human-Computer Interaction – INTERACT 2009, Lecture Notes in Computer Science* Bd. 5726, 2009, S. 243–246.

[Baldwin und Clark 2000]
Baldwin, C. Y. ; Clark, K. B.: *Design Rules: The Power of Modularity.* Cambridge, USA : MIT Press, 2000.

[Baldwin und Clark 2006]
Baldwin, C. Y. ; Clark, K. B.: The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model? In: *Management Science* 52 (2006), S. 1116–1127.

[Balzert 2009]
Balzert, H.: *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering.* Spektrum, 2009.

[Bamberg und Coenenberg 1996]
Bamberg, G. ; Coenenberg, A. G.: *Betriebswirtschaftliche Entscheidungstheorie.* München : Vahlen, 1996.

[Bangor et al. 2008]
Bangor, A. ; Kortum, P. ; Miller, J.: An Empirical Evaluation of the System Usability Scale. In: *International Journal of Human-Computer Interaction* 24 (2008), Nr. 6, S. 574–594.

[Bangor et al. 2009]
Bangor, A. ; Kortum, P. ; Miller, J.: Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. In: *Journal of Usability Studies* 4 (2009), Nr. 3, S. 114–123.

[Bartlett und Ghoshal 2000]
Bartlett, C. A. ; Ghoshal, S.: *Transnational Management: Text, Cases, and Readings in Cross-Border Management*. New York, USA : McGraw-Hill, 2000.

[Baskerville et al. 2009]
Baskerville, R. ; Pries-Heje, J. ; Venable, J.: Soft Design Science Methodology. In: *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, DESRIST'09*, 2009.

[Bass et al. 2003]
Bass, L. ; Clements, P. ; Kazman, R.: *Software Architecture in Practice*. Munich : Addison-Wesley, 2003.

[Beck 1999]
Beck, K.: *Extreme Programming Explained: Embrace Change*. Reading, USA : Addison-Wesley, 1999.

[Benbasat und Barki 2007]
Benbasat, I. ; Barki, H.: Quo vadis TAM? In: *Journal of the Association for Information Systems* 8 (2007), Nr. 4.

[Blume et al. 1991]
Blume, L. ; Brandenburger, A. ; Dekel, E.: Lexicographic Probabilities and Choice Under Uncertainty. In: *Econometrica* 59 (1991), Nr. 1, S. 61–79.

[Bode und Mertens 2006]
Bode, A. ; Mertens, P.: Globalization and Offshoring of Software. In: *Informatik Spektrum* 29 (2006), Nr. 3, S. 171–173.

[Boehm 1999]
Boehm, B.: Escaping the Software Tar Pit: Model Clashes and How to Avoid Them. In: *ACM SIGSOFT Software Engineering Notes* 24 (1999), January, Nr. 1, S. 36–48.

[Bonczek et al. 1981]
Bonczek, R. H. ; Holsapple, C. W. ; Whinston, A. B.: *Foundations of Decision Support Systems.* New York : Academic Press, 1981.

[Booch et al. 2007]
Booch, G. ; Maksimchuk, R. A. ; Engel, M. W. ; Young, B. J. ; Conallen, J. ; Houston, K. A.: *Object-Oriented Analysis and Design with Applications.* Upper Saddle River, USA : Addison-Wesley, 2007.

[Bosch 2000]
Bosch, J.: *Design and Use of Software Architectures.* Amsterdam, Netherlands : Addison-Wesley, 2000.

[Brans und Vincke 1985]
Brans, J. P. ; Vincke, P.: A Preference Ranking Organisation Method: The PROMETHEE Method for Multiple Criteria Decision-Making. In: *Management Science* 31 (1985), Nr. 6, S. 647–656.

[Brans et al. 1986]
Brans, J. P. ; Vincke, P. ; Mareschal, B.: How to Select and How to Rank Projects: The PROMETHEE Method. In: *European Journal Of Operational Research* 24 (1986), Nr. 2, S. 228–238.

[Breidert 2006]
Breidert, C.: *Estimation of Willingness-to-Pay: Theory, Measurement, Application.* Wiesbaden : DUV Gabler, 2006.

[Briand et al. 1996]
Briand, L. ; Morasca, S. ; Basili, V.: Property-based software engineering

Measurement. In: *IEEE Transactions on Software Engineering* 22 (1996), Nr. 1, S. 68–86.

[Brooke 1996]
Kapitel 21. In: Brooke, J.: *SUS: a 'quick and dirty' Usability Scale.* Tayler and Francis, 1996, S. 189–194.

[Brooks 2003]
Brooks, F. P.: *The Mythical Man-month: Essays on Software Engineering.* Addison-Wesley, 2003.

[Burstein und Holsapple 2008]
Burstein, F. ; Holsapple, C.: *Handbook on Decision Support Systems 1.* Berlin : Springer, 2008.

[Butter 2009]
Butter, T.: *Privacy-preserving Framework for Context-aware Mobile Applications.* Online-Ressource. https://ub-madoc.bib.uni-mannheim.de/3036. Version: 2009.

[Buxmann et al. 2011]
Buxmann, P. ; Diefenbach, H. ; Hess, T.: *Die Softwareindustrie.* 2. Auflage. Heidelberg : Springer, 2011.

[Buxmann und Hess 2008]
Buxmann, P. ; Hess, T.: Software as a Service. In: *Wirtschaftsinformatik* 50 (2008), Nr. 6, S. 500–503.

[Böhm et al. 2009]
Böhm, M. ; Leimeister, S. ; Riedl, C. ; Krcmar, H.: Cloud Computing: Outsourcing 2.0 oder ein neues Geschäftsmodell zur Bereitstellung von IT-Ressourcen? In: *Information Management and Consulting* 24 (2009), Nr. 2, S. 6–14.

[Cai et al. 2000]
Cai, X. ; Lyu, M. R. ; Wong, K.: Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes. In: *Asia-Pacific Software Engineering Conference*, 2000, S. 372–379.

[Carmel und Agarwal 2001]
CARMEL, E. ; AGARWAL, R.: Tactical Approaches for Alleviating Distance in Global Software Development. In: *IEEE Software* 18 (2001), Nr. 2, S. 22–29.

[Carmel und Nicholson 2005]
CARMEL, E. ; NICHOLSON, B.: Small Firms and Offshore Software Outsourcing: High Transaction Costs and Their Mitigation. In: *Journal of Global Information Management* 13 (2005), Nr. 3, S. 33–54.

[Carmel und Tjia 2006]
CARMEL, E. ; TJIA, P.: *Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce*. Cambridge, USA : Cambridge Univ. Press, 2006.

[Chalos und Sung 1998]
CHALOS, P. ; SUNG, J.: Outsourcing Decisions and Managerial Incentives. In: *Decision Sciences* 29 (1998), Nr. 4, S. 901–919.

[Checkland und Scholes 1990]
CHECKLAND, P. ; SCHOLES, J.: *Soft systems methodology in practice*. Chichester, UK : Wiley, 1990.

[Cheesman und Daniels 2001]
CHEESMAN, J. ; DANIELS, J.: *UML Components – A Simple Process for Specifying Component-Based Software*. Addison-Wesley, 2001.

[Chen 1976]
CHEN, P. P.: The Entity-Relationship Model: Toward a Unified View of Data. In: *ACM Transactions on Database Systems (TODS)* 1 (1976), March, Nr. 1, S. 9–36.

[Chen et al. 1992]
CHEN, S. J. ; HWANG, C. L. ; HWANG, F. P.: *Fuzzy Multiple Attribute Decision Making, Methods and Applications*. Berlin : Springer, 1992 (Lecture Notes in Economics and Mathematical Systems).

[Choudhary 2007]
CHOUDHARY, V.: Software as a Service: implications for Investment in Soft-

ware Development. In: Sprague, R. H. (Hrsg.): *Proceedings of the 40th Annual Hawaii International Conference on System Sciences, HICSS, Hawaii*, 2007.

[Clermont 1991]
Clermont, P.: Outsourcing Without Guilt. In: *Computerworld* 9 (1991), S. 67–68.

[Conway 1968]
Conway, M. E.: How Do Committees Invent? In: *Datamation* 14 (1968), Nr. 5, S. 28–31.

[Cormen et al. 2009]
Cormen, T. H. ; Leiserson, C. E. ; Rivest, R. L.: *Introduction to algorithms*. 3. Ed. Cambridge : MIT Press, 2009.

[Courtney und Paradice 1993]
Courtney, J. F. ; Paradice, D. B.: Studies in Managerial Problem Formulation Systems. In: *Decision Support Systems* 9 (1993), S. 413–423.

[Cramer und Kamps 2014]
Cramer, E. ; Kamps, U.: *Grundlagen der Wahrscheinlichkeitsrechnung und Statistik*. 3. Ed. Berlin : Springer, 2014.

[Cullen et al. 2005]
Cullen, S. ; Seddon, P. ; Willcocks, L. P.: Managing Outsourcing: The Life Cycle Imperative. In: *MIS Quarterly Executive* 4 (2005), Nr. 1, S. 229–246.

[Culnan 1985]
Culnan, M. J.: The Dimensions of Perceived Accessibility to Information: Implications for the Delivery of Information Systems and Services. In: *Journal of the American Society for Information Science* 36 (1985), S. 302–308.

[Cyert und March 1963]
Cyert, R. M. ; March, J. G.: *A Behavioral Theory of the Firm*. Prentice-Hall, 1963.

[Dahlstedt und Persson 2005]
Dahlstedt, S. G. ; Persson, A.: Requirements Interdependencies: State of the Art and Future challenges. In: Aurum, A. (Hrsg.) ; Wohlin, C. (Hrsg.): *Engineering and Managing Software Requirements*. 2005, S. 95–116.

[Davis et al. 1989]
Davis, F. D. ; Bagozzi, R. P. ; Warshaw, P. R.: User Acceptance of Computer Technology: A Comparison of Two Theoretical Models. In: *Management Science* 35 (1989), Nr. 8, S. 982–1003.

[Davis und Olson 1985]
Davis, G. B. ; Olson, M. H.: *Management Information Systems Conceptual Foundations, Structure, and Development*. 2nd ed. New York : McGraw-Hill, 1985.

[Dean und Sharfman 1996]
Dean, J. W. J. ; Sharfman, M. P.: Does Decision Process Matter? A Study of Strategic Decision-Making Effectiveness. In: *The Academy of Management Journal* 39 (1996), Nr. 2, S. 368–396.

[DeLone und McLean 2003]
DeLone, W. H. ; McLean, E. R.: The DeLone and McLean Model of Information Systems Success: A Ten-Year Update. In: *Journal of Management Information Systems* 19 (2003), Nr. 4, S. 9–30.

[DeSanctis und Gallup 1987]
DeSanctis, G. ; Gallup, B.: A Foundation for the Study of Group Decision Support Systems. In: *Management Science* 33 (1987), Nr. 12, S. 1589–1609.

[Dibbern et al. 2005]
Dibbern, J. ; Chin, W. W. ; A., H.: The Impact Of Human Asset Specificity on the Sourcing of Application Services. In: *13th European Conference of Information Systems, Regensburg*, 2005.

[Dibbern et al. 2012a]
Dibbern, J. ; Chin, W. W. ; Heinzl, A.: Systemic Determinants of the Information Systems Outsourcing Decision: A Comparative Study of German

and United States Firms. In: *Journal of the Association for Information Systems* 13 (2012), Nr. 6, S. 446–497.

[Dibbern et al. 2004]
DIBBERN, J. ; GOLES, T. ; HIRSCHHEIM, R. ; JAYATILAKA, B.: Information Systems Outsourcing: A Survey and Analysis of the Literature. In: *Communications of the ACM* 35 (2004), Nr. 4, S. 6–102.

[Dibbern et al. 2012b]
DIBBERN, J. ; HEINZL, A. ; CHIN, W. W.: Systemic Determinants of the Information Systems Outsourcing Decision: A Comparative Study of German and United States Firms. In: *Journal of the Association for Information Systems* 13 (2012), Nr. 6.

[Dibbern et al. 2008]
DIBBERN, J. ; WINKLER, J. ; HEINZL, A.: Explaining Variations in Client Extra Costs Between Software Projects Offshored to India. In: *MIS Quarterly* 32 (2008, June), Nr. 2, S. 1–30.

[Diestel 2010]
DIESTEL, R.: *Graphentheorie*. 4. Berlin : Springer, 2010.

[Dietterich 2000]
DIETTERICH, T. G.: Ensemble Methods in Machine Learning. In: KITTLER, J. (Hrsg.) ; ROLI, F. (Hrsg.): *First International Workshop on Multiple Classifier Systems*. New York : Springer, 2000 (Lecture Notes in Computer Science), S. 1–15.

[Dressler 2007]
DRESSLER, S.: *Shared Services, Business Process Outsourcing und Offshoring*. Wiesbaden : Gabler, 2007.

[Dumke 2003]
DUMKE, R.: *Software Engineering*. Wiesbaden : Vieweg, 2003.

[Dutoit et al. 2006]
DUTOIT, A. H. ; MCCALL, R. ; MISTRIK, I. ; PAECH, B.: *Rationale Management in Software Engineering*. New York, USA : Springer, 2006.

[Duvall et al. 2007]
DUVALL, P. M. ; MATYAS, S. ; GLOVER, A.: *Continuous Integration*. Upper Saddle River, NJ, USA : Addison-Wesley, 2007.

[Earle und Keen 2000]
EARLE, N. ; KEEN, P.: *From .com to .profit: Inventing Business Models that Deliver Value and Profit*. San Francisco, CA, USA : Jossey-Bass, 2000.

[Easterbrook et al. 2008]
EASTERBROOK, S. ; SINGER, J. ; STOREY, M. ; DAMIAN, D.: Selecting Empirical Methods for Software Engineering Research. In: SHULL, F. (Hrsg.) ; SINGER, J. (Hrsg.) ; SJOBERG, D. I. K. (Hrsg.): *Guide to Advanced Empirical Software Engineering*. London : Springer, 2008, S. 285–311.

[Edwards 1971]
EDWARDS, W.: Social utilities. In: *The Engineering Economist* 6 (1971), S. 119–129.

[Edwards 1977]
EDWARDS, W.: How to Use Multiattribute Utility Measurement for Social Decisionmaking. In: *IEEE Transactions On Systems Man And Cybernetics* 7 (1977), Nr. 5, S. 326–340.

[Edwards und Barron 1994]
EDWARDS, W. ; BARRON, F. H.: SMARTS and SMARTER: Improved Simple Methods for Multiattribute Utility Measurement. In: *Organizational Behavior and Human Decision Processes* 60 (1994), Nr. 3, S. 306–325.

[Eisenführ et al. 2010]
EISENFÜHR, F. ; WEBER, M. ; LANGER, T.: *Rational Decision Making*. Dordrecht, London, New York : Springer, 2010.

[Endres und Rombach 2003]
ENDRES, A. ; ROMBACH, D.: *A Handbook of Software and Systems Engineering*. Harlow, England : Pearson / Addison-Wesley, 2003.

[Figueira et al. 2005a]
FIGUEIRA, J. ; GRECO, S. ; EHRGOTT, M.: *Multiple Criteria Decision Analysis - State of the Art Surveys*. New York : Springer, 2005.

[Figueira et al. 2005b]
FIGUEIRA, J. ; MOUSSEAU, V. ; ROY, B.: ELECTRE Methods. In: FIGUEIRA, J. (Hrsg.) ; GRECO, S. (Hrsg.) ; EHRGOTT, M. (Hrsg.): *Multiple Criteria Decision Analysis: State of the Art Surveys*. Boston : Springer, 2005, S. 133–153.

[Fishburn 1974]
FISHBURN, P. C.: Lexicographic Orders, Utilities and Decision Rules: A Survey. In: *Management Science* 20 (1974), Nr. 11, S. 1442–1471.

[Fishburn 1980]
FISHBURN, P. C.: Lexicographic Additive Differences. In: *Journal of Mathematical Psychology* 21 (1980), Nr. 3, S. 191–218.

[Fitsilis et al. 2010]
FITSILIS, P. ; GEROGIANNIS, V. ; ANTHOPOULOS, L. ; SAVVAS, I.: Supporting the Requirements Prioritization Process Using Social Network Analysis Techniques. In: *19th IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE)*, 2010, S. 110–115.

[Fjällström 1998]
FJÄLLSTRÖM, P.-O.: Algorithms for Graph Partitioning: A Survey. In: *Linköping Electronic Articles in Computer and Information Science* 3 (1998), Nr. 10, S. 1–37.

[Fjermestad und Saitta 2005]
FJERMESTAD, J. ; SAITTA, J.: A Strategic Management Framework for IT Outsourcing: A review of the literature and the development of a success factors model. In: *Journal of Information Technology Case and Application Research* 7 (2005), Nr. 3, S. 42–60.

[Frank 2000]
FRANK, U.: Evaluation von Artefakten in der Wirtschaftsinformatik. In: HEINRICH, L. J. (Hrsg.) ; HÄNTSCHEL, I. (Hrsg.): *Evaluation und Evaluationsforschung in der Wirtschaftsinformatik: Handbuch für Praxis, Lehre und Forschung*. Munich : Oldenbourg, 2000, S. 35–48.

[Friedrich et al. 2008]
FRIEDRICH, J. ; HAMMERSCHALL, U. ; KUHRMANN, M. ; SIHLING, M.: *Das V-Modell XT. Für Projektleiter und QS-Verantwortliche - kompakt und übersichtlich*. Berlin : Springer, 2008.

[Gagsch 1971]
Kapitel 34. In: GAGSCH, S.: *Probleme der Partition und Subsystembildung in betrieblichen Informationssystemen*. Gabler, 1971, S. 623–652.

[Gagsch 1980]
GAGSCH, S.: Subsystembildung. In: GROCHLA, E. (Hrsg.): *Handwörterbuch der Organisation*. 2. ed. Stuttgart : Poeschel, 1980, S. 2156 – 2171.

[Geisser 2008]
GEISSER, M.: *Requirements Engineering von Informationssystemen*. Saarbrücken : VDM, 2008.

[Gewald 2010]
GEWALD, H.: The Perceived Benefits of Business Process Outsourcing: An Empirical Study of the German Banking Industry. In: *Strategic Outsourcing: An International Journal* 3 (2010), Nr. 2, S. 89–105.

[Gibbons 1985]
GIBBONS, J. D.: *Nonparametric Statistical Inference*. 2. New York : M. Dekker, 1985.

[Goetz et al. 2012]
GOETZ, T. ; FELDMANN, N. ; SCHMIDT, S.: Smarter Apps – Motor für Geschäftsmodellinnovationen. In: VERCLAS, S. (Hrsg.) ; LINNHOFF-POPIEN, C. (Hrsg.): *Smart Mobile Apps*. Berlin : Springer, 2012, S. 507–518.

[Gopal et al. 2003]
GOPAL, A. ; SIVARAMAKRISHNAN, K. ; KRISHNAN, M. ; MUKHOPADHYAY, T.: Contracts in Offshore Software Development: An empirical Analysis. In: *Management Science* 49 (2003), Nr. 12, S. 1671–1683.

[Gorlenko und Merrick 2003]
GORLENKO, L. ; MERRICK, R.: No wires attached: Usability challenges in

the connected mobile world. In: *IBM Systems Journal* 42 (2003), Nr. 4, S. 639–651.

[Gorry und Scott Morton 1971]
Gorry, G. A. ; Scott Morton, M. S.: A framework for Management Information Systems. In: *Sloan Management Review* 13 (1971), Nr. 1, S. 50–70.

[Gotel und Finkelstein 1994]
Gotel, O. ; Finkelstein, A.: An Analysis of the Requirements Traceability Problem. In: *Proceedings of the 1st International Conference on Requirements Engineering (ICRE'94)*, IEEE Computer Society, 1994, S. 94–101.

[Grant 1991]
Grant, R. M.: The Resource-Based Theory of Competitive Advantage: Implications for Strategy Formulation. In: *California Management Review* 33 (1991), Nr. 3, S. 114–135.

[Green und Rao 1971]
Green, P. E. ; Rao, V.: Conjoint Measurement for Quantifying Judgemental Data. In: *Journal of Marketing Research* 8 (1971), S. 355–363.

[Gregor und Jones 2007]
Gregor, S. ; Jones, D.: The Anatomy of a Design Theory. In: *Journal of the Association for Information Systems* 8 (2007), Nr. 5, S. 312–335.

[Gregory et al. 2013]
Gregory, R. ; Beck, R. ; Keil, M.: Control Balancing in Information Systems Development Offshoring Projects. In: *MIS Quarterly* 37 (2013), Nr. 4, S. 1211–1232.

[Grinter et al. 1999]
Grinter, R. E. ; Herbsleb, J. D. ; Perry, D. E.: The Geography of Coordination: Dealing with Distance in R&D Work. In: *Proceedings of International ACM SIGGROUP Conference Supporting Group Work*. New York, USA : ACM Press, 1999, S. 306–315.

[Gross und Yellen 2004]
Gross, J. L. ; Yellen, J.: Fundamentals of Graph Theory. In: Gross, J. L.

(Hrsg.) ; YELLEN, J. (Hrsg.): *Handbook of Graph Theory*. Boca Raton, FL : CRC Press, 2004, S. 2–19.

[Grover et al. 1996]
GROVER, V. ; CHEON, M. ; TENG, J.: The Effect of Service Quality and Partnership on the Outsourcing of Information Systems Functions. In: *Journal of Management Information Systems* 12 (1996), Nr. 4, S. 89–116.

[Grover et al. 1994]
GROVER, V. ; CHEON, M. J. ; TENG, J. T. C.: An Evaluation of the Impact of Corporate Strategy and the Role of Information Technology on IS Functional Outsourcing. In: *European Journal of Information Systems* 3 (1994), Nr. 3, S. 179–191.

[Grünig und Kühn 2005]
GRÜNIG, R. ; KÜHN, R.: *Successful Decision-Making*. Springer, 2005.

[Hall 1990]
HALL, A.: The Seven Myths of Formal Methods. In: *IEEE Software* (1990), Nr. 9, S. 11–19.

[Hall und Liedtka 2005]
HALL, J. ; LIEDTKA, S.: Financial Performance, CEO Compensation, and Large-scale Information Technology Outsourcing Decisions. In: *Journal of Management Information Systems* 22 (2005), Nr. 1, S. 193–222.

[Heeks et al. 2001]
HEEKS, R. ; KRISHNA, S. ; NICHOLSON, B. ; SAHAY, S.: Synching or Sinking: Global Software Outsourcing Relationships. In: *IEEE Software* 18 (2001), Nr. 2, S. 56–60.

[Heijden 2006]
HEIJDEN, v. d. H.: Mobile Decision Support for In-store Purchase Decisions. In: *Decision Support Systems* 42 (2006), Nr. 2, S. 656–663.

[Heim et al. 2008]
HEIM, P. ; LOHMANN, S. ; LAUENROTH, K. ; ZIEGLER, J.: Graph-based Visualization of Requirements Relationships. In: *Requirements Engineering Visualization* (2008), S. 51–55.

[Heindl und Biffl 2006]
HEINDL, M. ; BIFFL, .: Risk Management with Enhanced Tracing of Requirements Rationale in Highly Distributed Projects. In: *Proceedings of the 2006 International Workshop on Global Software Development for the Practitioner (GSD '06)*. New York, USA : ACM Press, 2006, S. 20–26.

[Heinrich 1996]
HEINRICH, L. J.: *Systemplanung: Planung und Realisierung von Informatik-Projekten*. Munich : Oldenbourg, 1996.

[Heinrich et al. 2011]
HEINRICH, L. J. ; HEINZL, A. ; RIEDL, R.: *Wirtschaftsinformatik: Einführung und Grundlegung*. 4. Auflage. Heidelberg : Springer, 2011.

[Heinrich et al. 2004]
HEINRICH, L. J. ; ROITHMAYR, F. ; HEINZL, A.: *Wirtschaftsinformatik-Lexikon*. 7. ed. München : Oldenbourg, 2004.

[Heinzl 1993]
HEINZL, A.: *Die Ausgliederung der betrieblichen Datenverarbeitung: Eine empirische Analyse der Motive, Formen und Wirkungen*. Poeschel, 1993.

[Heinzl und Sinß 1993]
HEINZL, A. ; SINSS, M.: Kooperationen zur zwischenbetrieblichen Entwicklung von Anwendungssystemen. In: *Information Management* (1993), S. 60–67.

[Herbsleb 2007]
HERBSLEB, J. D.: Global Software Engineering: The Future of Socio-technical Coordination. In: *Proceedings of Future of Software Engineering (FOSE'07)*. 2007, S. 188–198.

[Herbsleb und Moitra 2001]
HERBSLEB, J. D. ; MOITRA, D.: Global Software Development. In: *IEEE Software* 18 (2001), March - April, S. 16–20.

[Hevner et al. 2004]
HEVNER, A. R. ; MARCH, S. T. ; PARK, J. ; SUDHA, R.: Design Science in Information Systems Research. In: *MIS Quarterly* 28 (2004), Nr. 1, S. 75–105.

[Hilbert und Ackermann 1928]
HILBERT, D. ; ACKERMANN, W.: *Grundzüge der theoretischen Logik*. Springer, 1928.

[Hildenbrand 2008]
HILDENBRAND, T.: *Improving Traceability in Distributed Collaborative Software Development*. Frankfurt : Lang, 2008.

[Hildenbrand et al. 2007a]
HILDENBRAND, T. ; KORCHMINSKAYA, A. ; OSWALD, S. ; BIEBER, E. ; BERCHEZ, J. ; MACHÉ, N.: Konzeption einer Kollaborationsplattform für die zwischenbetriebliche Softwareerstellung. In: *WIRTSCHAFTSINFORMATIK* 49 (2007), Nr. 4, S. 247–256.

[Hildenbrand et al. 2007b]
HILDENBRAND, T. ; ROTHLAUF, F. ; HEINZL, A.: Ansätze zur kollaborativen Softwareerstellung. In: *Wirtschaftsinformatik* 49 (2007), S. 72–80.

[Hodel et al. 2006]
HODEL, M. ; BERGER, A. ; RISI, P.: *Outsourcing realisieren - Vorgehen für IT und Geschäftsprozesse zur nachhaltigen Steigerung des Unternehmenserfolgs*. 2. ed. Wiesbaden : Vieweg, 2006.

[Hofer et al. 2003]
HOFER, T. ; SCHWINGER, W. ; PICHLER, M. ; LEONHARTSBERGER, G. ; ALTMANN, J. ; RETSCHITZEGGER, W.: Context-Awareness on Mobile Devices - The Hydrogen Approach. In: *Proceedings of the 36th Annual Hawaii International Conference on System Sciences, HICSS, Hawaii*, 2003.

[Hofmeister et al. 2000]
HOFMEISTER, C. ; NORD, R. ; SONI, D.: *Applied Software Architecture*. Amsterdam, Netherlands : Addison-Wesley Object Technology, 2000.

[Holzinger 2005]
HOLZINGER, A.: Usability Engineering Methods for Software Developers. In: *Communications of the ACM* 48 (2005), Nr. 1, S. 71–74.

[Homburg 2000]
HOMBURG, C.: *Quantitative Betriebswirtschaftslehre: Entscheidungsunterstützung durch Modelle*. 3. Gabler, 2000.

[Huber et al. 2014]
HUBER, T. ; FISCHER, T. ; KIRSCH, L. ; DIBBERN, J.: Explaining Emergence and Consequences of Specific Formal Controls in IS Outsourcing - A Process-View. In: *47th Hawaii International Conference on System Science*, 2014, S. 4276–4285.

[Huysmans et al. 2011]
HUYSMANS, J. ; DEJAEGER, K. ; MUES, C. ; VANTHIENEN, J. ; BAESENS, B.: An Empirical Evaluation of the Comprehensibility of Decision Table, Tree and Rule Based Predictive Models. In: *Decision Support Systems* 51 (2011), Nr. 1, S. 141–154.

[Hättenschwiler 2001]
HÄTTENSCHWILER, P.: Neues anwenderfreundliches Konzept der Entscheidungsunterstützung. In: MEY, H. (Hrsg.) ; POLLHEIMER, D. L. (Hrsg.): *Gutes Entscheiden in Wirtschaft, Politik und Gesellschaft: Absturz im freien Fall – Anlauf zu neuen Höhenflügen*. Zürich : VDF, 2001, S. 1–4.

[Inmon 2005]
INMON, W. H.: *Building the Data Warehouse*. Indianapolis, IN, USA : Wiley, 2005.

[Jacobson et al. 2003]
JACOBSON, I. ; BOOCH, G. ; RUMBAUGH, J.: *The Unified Software Development Process*. Boston, USA : Addison-Wesley, 2003.

[Jahn 2004]
JAHN, J.: *Vector Optimization: Theory, applications, and extensions*. Heidelberg : Springer, 2004.

[Jeong und Lambert 2001]
JEONG, M. ; LAMBERT, C. U.: Adaptation of an Information Quality Framework to Measure Customers' Behavioral Intentions to Use Lodging Web Sites. In: *Hospitality Management* 20 (2001), S. 129–146.

[Kahraman 2008]
KAHRAMAN, C.: *Fuzzy Multi-Criteria Decision Making: Theory and Applications with Recent Developments*. New York : Springer, 2008.

[Karlsson und Ryan 1997]
KARLSSON, J. ; RYAN, K.: A Cost-Value Approach for Prioritizing Requirements. In: *IEEE Software* 14 (1997), Nr. 5, S. 67–74.

[Karlsson et al. 1998]
KARLSSON, J. ; WOHLIN, C. ; REGNELL, B.: An Evaluation of Methods for Prioritizing Software Requirements. In: *Information and Software Technology* 39 (1998), Nr. 14 - 15, S. 939–947.

[Keen und Scott Morton 1978]
KEEN, P. G. W. ; SCOTT MORTON, M. S.: *Decision Support Systems: An Organizational Perspective*. Reading u.a., 1978.

[Kern und Willcocks 2002]
KERN, T. ; WILLCOCKS, L.: Exploring Relationships in Information Technology Outsourcing: The Interaction Approach. In: *European Journal of Information Systems* 11 (2002), S. 3–19.

[Kim 2009]
KIM, J.: Online Reverse Auctions for Outsourcing Small Software Projects: Determinants of Vendor Selection. In: *E-Service Journal* 6 (2009), Nr. 3, S. 40–55.

[King 2005]
KING, W. R.: Outsourcing Becomes More Complex. In: *Information Systems Management* 22 (2005), Nr. 2, S. 89–90.

[King und Torkzadeh 2008]
KING, W. R. ; TORKZADEH, G.: Information Systems Offshoring: Research Status and Issues. In: *MIS Quarterly* 32 (2008), Nr. 2, S. 205–225.

[Klimpke 2013]
KLIMPKE, L.: *Konzeption und Realisierung eines integrierten Mikroblog-basierten Kommunikationsansatzes für die verteilte Softwareentwicklung*. Frankfurt : Peter Lang, 2013.

[Klimpke und Hildenbrand 2009]

KLIMPKE, L. ; HILDENBRAND, T.: Towards End-to-End Traceability: Insights and Implications from Five Case Studies. In: *Proceedings of the 4th International Conference on Software Engineering Advances*, 2009, S. 465–470.

[Klimpke et al. 2011]

KLIMPKE, L. ; KRAMER, T. ; BETZ, S. ; NORDHEIMER, K.: Globally Distributed Software Development in Small and Medium-Sized Enterprises in Germany: Reasons, Locations, and Obstacles. In: *Proceedings of the 19th European Conference on Information Systems (ECIS2011), Helsinki, Finland*, 2011.

[Koh et al. 2004]

KOH, C. ; ANG, S. ; STRAUB, D.: IT Outsourcing Success: A Psychological Contract Perspective. In: *Information Systems Research* 15 (2004), Nr. 4, S. 356–373.

[Kohli und Jedidi 2007]

KOHLI, R. ; JEDIDI, K.: Representation and Inference of Lexicographic Preference Models and Their Variants. In: *Marketing Science* 26 (2007), Nr. 3, S. 380–299.

[Kotonya und Sommerville 2004]

KOTONYA, G. ; SOMMERVILLE, I.: *Requirements Engineering*. Chichester, USA : Wiley, 2004.

[Kramer und Eschweiler 2013]

KRAMER, T. ; ESCHWEILER, M.: Outsourcing Location Selection with SODA: A Requirements based Decision Support Methodology and Tool. In: *Proceedings of the 25th International Conference on Advanced Information Systems Engineering, CAiSE, Valencia, Spain*, 2013.

[Kramer et al. 2014]

KRAMER, T. ; HEINZL, A. ; ESCHWEILER, M.: Software Outsourcing Decision Aid (SODA): A Requirements Based Decision Support Method and Tool. In: HIRSCHHEIM, R. (Hrsg.) ; HEINZL, A. (Hrsg.) ; DIBBERN, J. (Hrsg.): *Information Systems Outsourcing: Towards Sustainable Business Value; Progress*

in IS. 4th edition. Heidelberg, New York, Dodrecht, London : Springer, 2014, S. 115–140.

[Kramer et al. 2011]
KRAMER, T. ; HEINZL, A. ; SPOHRER, K.: Should this Software Component be Developed Inside or Outside our Firm? - A Design Science Perspective on the Sourcing of Application Systems. In: KOTLARSKY, J. (Hrsg.) ; WILLCOCKS, L. P. (Hrsg.) ; ILAN, O. (Hrsg.): *New Studies in Global IT and Business Service Outsourcing: 5th Global Scourcing Workshop 2011, Courchevel, France, March 14-17, 2011, Revised Selected Papers*. Heidelberg, Dordrecht, London, New York : Springer, 2011, S. 115–132.

[Kramer et al. 2009]
KRAMER, T. ; HILDENBRAND, T. ; ACKER, T.: Enabling Social Network Analysis in Distributed Collaborative Software Development. In: *Software Engineering 2009, Workshopband, International Workshop on Software Engineering within Social Software Environments (SENSE09)*, 2009, S. 255–266.

[Kramer et al. 2013]
KRAMER, T. ; KLIMPKE, L. ; HEINZL, A.: Outsourcing Decisions of Small and Medium-Sized Enterprises: A Multiple-Case Study Approach in the German Software Industry. In: *Proceedings of the 46th Annual Hawaii International Conference on System Sciences, HICSS, Hawaii*, 2013, S. 4236–4245.

[Kraut und Streeter 1995]
KRAUT, R. E. ; STREETER, L. A.: Coordination in Software Development. In: *Communications of the ACM* 38 (1995), Nr. 3, S. 69–81.

[Kruchten 2000]
KRUCHTEN, P.: *The Rational Unified Process*. Boston, USA : Addison-Wesley, 2000.

[Kwon et al. 2005]
KWON, O. ; YOO, K. ; SUH, E.: UbiDSS: A Proactive Intelligent Decision Support System as an Expert System Deploying Ubiquitous Computing Technologies. In: *Expert Systems with Applications* 28 (2005), Nr. 1, S. 149–161.

[Königstorfer 2008]
KÖNIGSTORFER, J.: *Akzeptanz von technologischen Innovationen*. Wiesbaden : Gabler, 2008.

[Lacity et al. 2010]
LACITY, M. C. ; KHAN, S. A. ; YAN, A. ; WILLCOCKS, L. P.: A Review of the IT Outsourcing Empirical Literature and Future Research Directions. In: *Journal of Information Technology* 25 (2010), S. 395–433.

[Lacity et al. 2011]
LACITY, M. C. ; SOLOMON, S. ; YAN, A. ; WILLCOCKS, L. P.: Business Process Outsourcing Studies: A Critical Review and Research Directions. In: *Journal of Information Technology* 26 (2011), S. 221–258.

[Lacity und Willcocks 1995]
LACITY, M. C. ; WILLCOCKS, L. P.: Interpreting Information Technology Sourcing Decisions from a Transaction Cost Perspective: Findings and Critique. In: *Accounting, Management and Information Technologies* 5 (1995), Nr. 3/4, S. 203–244.

[Lacity und Willcocks 1998]
LACITY, M. C. ; WILLCOCKS, L. P.: An Empirical Investigation of Information Technology Sourcing Practices: Lessons from Experience. In: *MIS Quarterly* 22 (1998), Nr. 3, S. 363–408.

[Lacity und Willcocks 2000]
LACITY, M. C. ; WILLCOCKS, L. P.: *Global Information Technology Outsourcing: In Search of Business Advantage*. New York : Wiley, 2000.

[Lacity et al. 1996]
LACITY, M. C. ; WILLCOCKS, L. P. ; FEENY, D. F.: The Value of Selective IT Sourcing. In: *Sloan Management Review* 37 (1996), Nr. 3, S. 13–25.

[Lacity et al. 2009]
LACITY, M. C. ; KHAN, S. A. ; WILLCOCKS, L. P.: A review of the IT outsourcing literature: Insights for practice. In: *The Journal of Strategic Information Systems* 18 (2009), Sept., Nr. 3, S. 130–146.

[Lahres und Rayman 2009]
LAHRES, B. ; RAYMAN, G.: *Objektorientierte Programmierung*. Bonn : Galileo Press, 2009.

[Langlois 1995]
LANGLOIS, R. N.: Capabilities and Coherence in Firms and Markets. In: MONTGOMERY, C. A. (Hrsg.): *Resource-Based and Evolutionary Theories of the Firm: Towards a Synthesis*. Boston, USA : Kluwer Academic, 1995, S. 71–100.

[Larman 2002]
LARMAN, C.: *Applying UML and Patterns*. Upper Saddle River, NJ, USA : Prentice Hall, 2002.

[Larman und Vodde 2008]
LARMAN, C. ; VODDE, B.: *Scaling lean & agile development: thinking and organizational tools for large-scale Scrum*. Boston, MA, USA : Pearson, 2008.

[Laux et al. 2004]
LAUX, H. ; GILLENKIRCH, R. M. ; SCHENK-MATHES, H. Y.: *Entscheidungstheorie*. 9. Springer, 2004.

[Layman et al. 2006]
LAYMAN, L. ; WILLIAMS, L. ; DAMIAN, D. ; BURES, H.: Essential Communication Practices for Extreme Programming in a Global Software Development Team. In: *Information and Software Technology* 48 (2006), Nr. 9, S. 781–794.

[Lee und Xia 2010]
LEE, G. ; XIA, W.: Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility. In: *MIS Quarterly* 34 (2010), Nr. 1, S. 87–114.

[Lee et al. 2003]
LEE, J. ; HUYNH, M. Q. ; R., K. ; PI, S.: IT Outsourcing Evolution – Past, Present, and Future. In: *Communications of the ACM* 46 (2003), Nr. 5, S. 84–89.

[Lee 2010]

Lee, S.: Using Data Envelopment Analysis and Decision Trees for Efficiency Analysis and Recommendation of B2C Controls. In: *Decision Support Systems* 49 (2010), Nr. 4, S. 486–497.

[Leff und Rayfield 2001]

Leff, A. ; Rayfield, J. T.: Web-application Development Using the Model/View/Controller Design Pattern. In: *Proceedings of the 5th Enterprise Distributed Object Computing Conference, EDOC*, 2001, S. 118–127.

[Lehman et al. 2010]

Lehman, S. ; Draisbach, T. ; Koll, C. ; Buxmann, P. ; Diefenbach, H.: Preisgestaltung für Software-as-a-Service - Ergebnisse einer empirischen Analyse mit Fokus auf nutzungs-abhängige Preismodelle. In: *Tagungsband Multikonferenz Wirtschaftsinformatik*, 2010, S. 105–106.

[Levina und Vaast 2008]

Levina, N. ; Vaast, E.: Innovating or Doing as Told? Status Differences and Overlapping Boundaries in Offshore Collaboration. In: *MIS Quarterly* 32 (2008), Nr. 2, S. 307–332.

[Li et al. 2008]

Li, Y. ; Li, J. ; Yang, Y. ; Li, M.: Requirement-Centric Traceability for Change Impact Analysis: A Case Study. In: Wang, Q. (Hrsg.) ; Pfahl, D. (Hrsg.) ; Raffo, D. (Hrsg.): *Making Globally Distributed Software Development a Success Story (5007)*. 2008, S. 100–111.

[Li et al. 2009]

Li, Z. ; Rahman, Q. ; Ferrari, R. ; Madhavji, N.: Does Requirements Clustering Lead to Modular Design? In: Glinz, M. (Hrsg.) ; Heymans, P. (Hrsg.): *Requirements Engineering: Foundation for Software Quality (5512)*. 2009, S. 233–239.

[Lindvall und Sandahl 1996]

Lindvall, M. ; Sandahl, K.: Practical Implications of Traceability. In: *Software - Practice & Expererience* 26 (1996), Nr. 10, S. 1161–1180.

[Lizhe et al. 2008]
Lizhe, W. ; Jie, T. ; Kunze, M. ; Castellanos, A. C. ; Kramer, D. ; Karl, W.: Scientific Cloud Computing: Early Definition and Experience. In: *10th IEEE International Conference on High Performance Computing and Communications, HPCC*, 2008, S. 825–830.

[Loelinger und McCullough 2012]
Loelinger, J. ; McCullough, M.: *Version Control with Git*. 2nd ed. Sebastopol : O'Reilly, 2012.

[Loh und Venkatraman 1992]
Loh, L. ; Venkatraman, N.: Diffusion of Information Technology Outsourcing Influence Sources and the Kodak Effect. In: *Information Systems Research* (1992), December, S. 334–358.

[Luxburg 2007]
Luxburg, U.: A tutorial on spectral clustering. In: *Statistics and Computing* 17 (2007), Nr. 4, S. 395–416.

[MacCormack et al. 2006]
MacCormack, A. ; Rusnak, J. ; Baldwin, C. Y.: Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. In: *Management Science* 52 (2006), S. 1015–1030.

[Malone und Crowston 1994]
Malone, T. W. ; Crowston, K.: The Interdisciplinary Study of Coordination. In: *ACM Computing Surveys* 26 (1994), Nr. 1, S. 87–119.

[Manz et al. 1993]
Manz, K. ; Dahmen, A. ; Hoffmann, L.: *Entscheidungstheorie*. München : Vahlen, 1993.

[March und Smith 1995]
March, S. T. ; Smith, G. F.: Design and Natural Science Research on Information Technology. In: *Decision Support Systems* 26 (1995), Nr. 4, S. 251–266.

[Marciniak 2002]
MARCINIAK, J. J.: *Encyclopedia of Software Engineering.* 2. ed. New York, USA : Wiley, 2002.

[Mata et al. 1995]
MATA, F. J. ; FUERST, W. L. ; BARNEY, J. B.: Information Technology and Sustained Competitive Advantage: A Resource-Based Analysis. In: *MIS Quarterly* 19 (1995), Nr. 4, S. 487–505.

[McDaniels und Gregory 2004]
MCDANIELS, T. L. ; GREGORY, R.: Learning as an Objective within a Structured Risk Management Decision Process. In: *Environmental Science & Technology* 38 (2004), Nr. 7, S. 1921–1926.

[McFarlan 1981]
MCFARLAN, F. W.: Portfolio approach to information-systems. In: *Harvard Business Review* (1981).

[Michlmayr und Hill 2003]
MICHLMAYR, M. ; HILL, B. M.: Quality and the Reliance on Individuals in Free Software Projects. In: *3rd Workshop on Open Source Software Engineering*, ICSE, 2003, S. 105–109.

[Mikkola 2003]
MIKKOLA, J. H.: Modularity, component outsourcing, and inter-firm learning. In: *R & D Management* 33 (2003), Nr. 4, S. 439–454.

[Mintzberg 1979]
MINTZBERG, H.: An Emerging Strategy of „Direct“ Research. In: *Administrative Science Quarterly* 24 (1979), Nr. 4, S. 582–589.

[Mintzberg et al. 1976]
MINTZBERG, H. ; RAISINGHANI, D. ; THÉORÊT, A.: The Structure of „Unstructured“ Decision Processes. In: *Administrative Science Quarterly* 21 (1976), Nr. 2, S. 246–275.

[Mojsilovic et al. 2007]
MOJSILOVIC, A. ; RAY, B. ; LAWRENCE, R. ; TAKRITI, S.: A Logistic Regres-

sion Framework for Information Technology Outsourcing Lifecycle Management. In: *Computers & Operations Research* 34 (2007), Nr. 12, S. 3609–3627.

[Moreira und Araujo 2011]
MOREIRA, A. ; ARAUJO, J.: The Need for Early Aspects. In: FERNANDES, J. (Hrsg.) ; LÄMMEL, R. (Hrsg.) ; VISSER, J. (Hrsg.) ; SARAIVA, J. (Hrsg.): *Generative and Transformational Techniques in Software Engineering III (6491)*. 2011, S. 386–407.

[Morieux 2011]
MORIEUX, Y.: Smart Rules: Six Ways to Get People to Solve Problems without You. In: *Harvard Business Review* (2011), S. 78–85.

[Mullur et al. 2003]
MULLUR, A. A. ; MATTSON, C. A. ; MESSAC, A.: New Decision Matrix Based Approach for Concept Selection Using Linear Physical Programming. In: *Proceedings of the 44th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Materials Conference*, 2003.

[Murray und Crandall 2006]
MURRAY, M. J. ; CRANDALL, R. E.: IT Offshore Outsourcing Requires a Project Management Approach. In: *S.A.M. Advanced Management Journal* 71 (2006), Nr. 1, S. 4–12.

[Myers 1978]
MYERS, G. J.: *Composite/Structured Design*. Van Nostrand Reinhold International, 1978.

[Mühlbauer 2007]
MÜHLBAUER, S.: *iX-Studie: Collaboration Software*. Hannover, Germany : Heise Zeitschriften Verlag, 2007.

[Mühlbauer und Versteegen 2007]
MÜHLBAUER, S. ; VERSTEEGEN, G.: Collaboration in der Softwareentwicklung: Automatisierbare Zusammenarbeit. In: *iX Magazin fuer professionelle Informationstechnik* 2 (2007), S. 68–73.

[Nah et al. 2005]
Nah, F. ; Siau, K. ; Sheng, H.: The Value of Mobile Applications. In: *Communications of the ACM* 48 (2005), Nr. 2, S. 85–90.

[Nelson und Winter 1982]
Nelson, R. ; Winter, S.: *An Evolutionary Theory of Economic Change.* Cambridge, MA : Harvard University Press, 1982.

[Newman 2006]
Newman, M. E. J.: Modularity and Community Structure in Networks. In: *Proceedings of the National Academy of Sciences of the United States of America* Bd. 103, 2006, S. 8577–8582.

[Noll et al. 2010]
Noll, J. ; Beecham, S. ; Richardson, I.: Global software development and collaboration: barriers and solutions. In: *ACM SIGCSE Bulletin – Special Section on Global Intercultural Collaboration* 1 (2010), Nr. 3, S. 66–78.

[Padberg und Tichy 2007]
Padberg, F. ; Tichy, W.: Schlanke Produktionsweisen in der modernen Softwareentwicklung. In: *WIRTSCHAFTSINFORMATIK* 49 (2007), Nr. 3, S. 162–170.

[Palmius 2007]
Palmius, J.: Criteria for Measuring and Comparing Information Systems. In: *Proceedings of the 30th Information Systems Research Seminar in Scandinavia, IRIS*, 2007.

[Pan und Flynn 2003]
Pan, G. S. C. ; Flynn, D. J.: Towards a Stakeholder Analysis of Information Systems Development Project Abandonment. In: *Proceedings of the 11th European Conference on Information Systems (ECIS'03)*, AIS, 2003.

[Patane und Jurison 1994]
Patane, J. R. ; Jurison, J.: Is global outsourcing diminishing the prospects for American programmers? In: *Journal of Systems Management* 45 (1994), Nr. 6, S. 6–10.

[Peffers et al. 2007]
PEFFERS, K. ; TUUNANEN, T. ; ROTHENBERGER, M. A. ; CHATTERJEE, S.: A Design Science Research Methodology for Information Systems Research. In: *Journal of Management Information Systems* 24 (2007), Nr. 3, S. 45–78.

[Pfleeger 2001]
PFLEEGER, S. L.: *Software Engineering.* 2. ed. Upper Saddle River, USA : Prentice Hall, 2001.

[Pfohl und Braun 1981]
PFOHL, H.-C. ; BRAUN, G. E.: *Entscheidungstheorie: Normative und deskriptive Grundlagen des Entscheidens.* Verl. Moderne Industrie, 1981.

[Picot und Baumann 2007]
PICOT, A. ; BAUMANN, O.: Modularität in der verteilten Entwicklung komplexer Systeme: Chancen, Grenzen, Implikationen. In: *Journal für Betriebswirtschaft* 57 (2007), Nr. 3–4, S. 221–246.

[Pomberger und Pree 2004]
POMBERGER, G. ; PREE, W.: *Software Engineering.* 3. Aufl. München : Hanser, 2004.

[Pooch 1974]
POOCH, U. W.: Translation of Decision Tables. In: *ACM Computing Surveys* 6 (1974), S. 125–151.

[Poppo und Zenger 2002]
POPPO, L. ; ZENGER, T.: Do Formal Contracts and Relational Governance Function as Substitutes or Complements? In: *Strategic Management Journal* 23 (2002), S. 707–725.

[Power 1997]
POWER, D. J.: What is a DSS? In: *The On-Line Executive Journal for Data-Intensive Decision Support* 1 (1997), Nr. 3.

[Power 2002]
POWER, D. J.: *Decision Support Systems: Concepts and Resources for Managers.* Greenwood Publishing Group, 2002.

[Power 2008]
Power, D. J.: Decision Support Systems: A Historical Overview. In: *Handbook on Decision Support Systems 1.* Springer, 2008, S. 121–140.

[Pries-Heje et al. 2007]
Pries-Heje, J. ; Baskerville, R. ; Venable, J.: Soft Design Science Research: Extending the Boundaries of Evaluation in Design Science Research. In: *Proceedings from the 2nd International Conference on Design Science Research in IT (DESRIST)*, 2007, S. 18–38.

[Pries-Heje et al. 2008]
Pries-Heje, J. ; Baskerville, R. ; Venable, J.: Strategies for Design Science Research Evaluation. In: *Proceedings of the European Conference on Information Systems*, 2008.

[Prikladnicki et al. 2007]
Prikladnicki, R. ; Audy, J. N. L. ; Damian, D. ; Oliveira, T. C.: Distributed Software Development: Practices and challenges in different business strategies of offshoring and onshoring. In: *International Conference on Global Software Engineering (ICGSE 2007)*, 2007.

[Pérez et al. 2010]
Pérez, I. J. ; Cabrerizo, F. J. ; Herrera-Viedma, E.: A Mobile Decision Support System for Dynamic Group Decision-Making Problems. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 40 (2010), Nr. 6, S. 1244–1256.

[Qu und Brocklehurst 2003]
Qu, Z. ; Brocklehurst, M.: What will it take for China to Become a Competitive Force in Offshore Outsourcing? An Analysis of the Role of Transaction Costs in Supplier Selection. In: *Journal of Information Technology* 18 (2003), Nr. 1, S. 53–67.

[Ranyard et al. 1997]
Ranyard, R. ; Crozier, W. ; Svenson, O.: *Decision Making - Cognitive models and explanations.* London, New York : Routledge, 1997.

[Ranyard 1976]

Ranyard, R. H.: Elimination by Aspects as a Decision Rule for Risky Choice. In: *Acta Psychologica* 40 (1976), Nr. 4, S. 299–310.

[Redmiles et al. 2007]

Redmiles, D. ; Hoek, A. van d. ; Al-Ani, B. ; Hildenbrand, T. ; Quirk, S. ; Sarma, A. ; Silveira, R. ; Filho, S. ; Souza, C. de ; Trainer, E.: Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects. In: *WIRTSCHAFTSINFORMATIK* 49 (2007), S. 28–38.

[Reinecke und Tomczak 2010]

Reinecke, S. ; Tomczak, T.: *Handbuch Marketingcontrolling: Effektivität und Effizienz einer marktorientierten Unternehmensführung*. Springer, 2010.

[Reinertson 2009]

Reinertson, D. G.: *The Principles of Product Development Flow*. Redondo Beach, California : Celeritas, 2009.

[Rifkin 1991]

Rifkin, G.: Heads that Roll if Computers Fail. In: *The New York Times* 14 (1991), May, S. 1.

[Robertson und Robertson 2006]

Robertson, S. ; Robertson, J.: *Mastering the Requirements Process*. 2. ed. Upper Saddle River, USA : Addison-Wesley, 2006.

[Royce 1970]

Royce, W. W.: Managing the Development of Large Software Systems. In: *Proceedings IEEE WESCON*, IEEE Computer Society, 1970, S. 1–9.

[Ruhe et al. 2002]

Ruhe, G. ; Eberlein, A. ; Pfahl, D.: Quantitative Win-Win - A New Method for Decision Support in Requirements Negotiation. In: *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*, ACM Press, 2002, S. 159–166.

[Rupp 2009]
RUPP, C.: *Requirements-Engineering und -Management*. 4th. Munich : Carl Hanser Verlag, 2009.

[Saaty 1986]
SAATY, T. L.: Axiomatic Foundation of the Analytic Hierarchy Process. In: *Management Science* 32 (1986), Nr. 7, S. 841–855.

[Saaty 2003]
SAATY, T. L.: Decision-making with the AHP: Why is the principal eigenvector necessary. In: *European Journal Of Operational Research* 145 (2003), Nr. 1, S. 85–91.

[Saaty 2005]
SAATY, T. L.: The Analytic Hiearchy and Analytic Network Processes for the Measurement of Intangible Criteria and for Decision-Making. In: FIGUEIRA, J. (Hrsg.) ; GRECO, S. (Hrsg.) ; EHRGOTT, M. (Hrsg.): *Multiple Criteria Decision Analysis: State of the Art Surveys*. Boston : Springer, 2005, S. 345–405.

[Saaty und Vargas 2001]
SAATY, T. L. ; VARGAS, L. G.: *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*. Boston, USA : Kluwer, 2001.

[Sahay et al. 2003]
SAHAY, S. ; NICHOLSON, B. ; KRISHNA, S.: *Global IT Outsourcing: Software Development across Borders*. Cambridge, USA : Cambridge University Press, 2003.

[Sahay et al. 2007]
SAHAY, S. ; NICHOLSON, B. ; KRISHNA, S.: *Global IT Outsourcing: Software Development Across Borders*. Cambridge : Cambridge University Press, 2007.

[Sanders 1999]
SANDERS, R.: *The Executive Decision-making Process*. Westport, 1999.

[Satyanarayanan 1996]
SATYANARAYANAN, M.: Fundamental Challenges in Mobile Computing. In:

Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, PODC, 1996, S. 1–7.

[Saxenian 2002]
SAXENIAN, A.: The Silicon Valley Connection: Transnational Networks and Regional Development in Taiwan, China and India. In: *Science Technology & Society* 7 (2002), S. 117–149.

[Schierenbeck 2003]
SCHIERENBECK, H.: *Grundzüge der Betriebswirtschaftslehre*. Munich : Oldenbourg, 2003.

[Schneeweiß 1991]
SCHNEEWEISS, C.: *Planung 1: Systemanalytische und entscheidungstheoretische Grundlagen*. Berlin : Springer, 1991.

[Schwaber et al. 2006]
SCHWABER, C. ; RYMER, J. R. ; STONE, J.: The Changing Face of Application Life-Cycle Management - Tomorrow's ALM Platforms will Deliver on the Promise of Today's ALM Suites / Forrester Research. 2006 (37653). – Trend Study.

[Schwaber und Beedle 2002]
SCHWABER, K. ; BEEDLE, M.: *Agile Software Development with Scrum*. Upper Saddle River, USA : Prentice Hall, 2002.

[Schwarz et al. 2010]
SCHWARZ, H. ; EBERT, J. ; WINTER, A.: Graph-based traceability: a comprehensive approach. In: *Software and Systems Modeling* 9 (2010), Nr. 4, S. 473–492.

[Scott 2013]
SCOTT, J.: *Social Network Analysis*. 3. Ed. London : Sage, 2013.

[Sein et al. 2011]
SEIN, M. ; HENFRIDSSON, O. ; PURAO, S. ; ROSSI, M. ; LINDGREN, R.: Action Design Research. In: *MIS Quarterly* 35 (2011), Nr. 1, S. 37–56.

[Shaw et al. 2001]
Shaw, M. J. ; Subramaniam, C. ; Tan, G. W. ; Welge, M. E.: Knowledge management and data mining for marketing. In: *Decision Support Systems* 31 (2001), Nr. 1, S. 127–137.

[Shim et al. 2002]
Shim, J. P. ; Warkentin, M. ; Courtney, J. F. ; Power, D. J. ; Sharda, R. ; Carlsson, C.: Past, Present, and Future of Decision Support Technology. In: *Decision Support Systems* 33 (2002), Nr. 2, S. 111–126.

[Sieben und Schildbach 1994]
Sieben, G. ; Schildbach, T.: *Betriebswirtschaftliche Entscheidungstheorie.* Düsseldorf : Werner, 1994.

[Simon 1960]
Simon, H. A.: *The New Science of Management Decision.* Englewood Cliffs, USA : Prentice Hall, 1960.

[Simon 1962]
Simon, H. A.: The Architecture of Complexity. In: *Proceedings of the American Philosophical Society* Bd. 106, 1962, S. 467–482.

[Simon 1976]
Simon, H. A.: *Administrative Behavior: A Study of Decision-making Processes in Administrative Organization.* The Free Press, 1976.

[Simon und Newell 1958]
Simon, H. A. ; Newell, A.: Heuristic Problem Solving: The Next Advance in Operations Research. In: *Operations Research* 6 (1958), Nr. 1, S. 1–10.

[Smith und McKeen 2004]
Smith, H. ; McKeen, J.: Developments in Practice XIV: IT Outsourcing – How far can you go? In: *Communications of the AIS* 14 (2004), S. 508–520.

[Smith et al. 1996]
Smith, M. A. ; Mitra, S. ; Narasimhan, S.: Offshore outsourcing of software development and maintenance: A framework for issues. In: *Information and Management* 31 (1996), S. 165–175.

[Sommerville 2007]
SOMMERVILLE, I.: *Software Engineering.* 8. ed. Munich : Addison-Wesley, 2007.

[Sonnenberg und vom Brocke 2012]
SONNENBERG, C. ; BROCKE, J. vom: Evaluation Patterns for Design Science Research Artefacts. In: HELFERT, M. (Hrsg.) ; DONNELLAN, B. (Hrsg.): *Practical Aspects of Design Science.* Berlin, Heidelberg : Springer, 2012, S. 71–83.

[Sprague 1980]
SPRAGUE, R. H.: A Framework for the Development of Decisoin Support Systems. In: *MIS Quarterly* 4 (1980), Nr. 4, S. 1–26.

[Stratman 2008]
STRATMAN, J. K.: Facilitating Offshoring with Enterprise Technologies: Reducing Operational Friction in the Governance and Production of Services. In: *Journal of Operations Management* 26 (2008), Nr. 2, S. 275–287.

[Straub et al. 2008]
STRAUB, D. ; WEILL, P. ; SCHWAIG, K.: Strategic Dependence on the IT Resource and Outsourcing: A Test of the Strategic Control Model. In: *Information Systems Frontiers* 10 (2008), Nr. 2, S. 195–211.

[Stuckenberg 2014]
STUCKENBERG, S.: *Exploring the Organizational Impact of Software-as-a-Service on Software Vendors.* Peter Lang, 2014.

[Stuckenberg et al. 2014]
STUCKENBERG, S. ; KUDE, T. ; HEINZL, A.: Understanding the role of organizational integration in developing and operating Software-as-a-Service. In: *Journal of Business Economics* 84 (2014), Nr. 8, S. 1019–1050.

[Subramani und Venkatraman 2003]
SUBRAMANI, M. R. ; VENKATRAMAN, N.: Safeguarding Investments in Asymmetric Interorganizational Relationships: Theory and Evidence. In: *Academy of Management Journal* 46 (2003), Nr. 1, S. 46–62.

[Subramanyam et al. 2012]
SUBRAMANYAM, R. ; RAMASUBBU, N. ; KRISHNAN, M. S.: In Search of Efficient Flexibility: Effects of Software Component Granularity on Development Effort, Defects, and Customization Effort. In: *Information Systems Research* 23 (2012), Nr. 3–1, S. 787–803.

[Sun und Kantor 2006]
SUN, Y. ; KANTOR, P. B.: Cross-Evaluation: A new Model for Information System Evaluation. In: *Journal of the American Society for Information Science and Technology* 57 (2006), Nr. 5, S. 614–628.

[Svenson 1979]
SVENSON, O.: Process Descriptions of Decision Making. In: *Organizational Behavior and Human Performance* 23 (1979), Nr. 1, S. 86–112.

[Székely et al. 2013]
SZÉKELY, A. ; TALANOW, R. ; BÁGYI, P.: Smartphones, Tablets and Mobile Applications for Radiology. In: *European Journal of Radiology* (2013).

[Takeuchi und Nonaka 1986]
TAKEUCHI, H. ; NONAKA, I.: The New New Product Development Game. In: *Harvard Business Review* (1986), January - February, S. 137–146.

[Tarasewich et al. 2002]
TARASEWICH, P. ; NICKERSON, R. C. ; WARKENTIN, M.: Issues in Mobile E-Commerce. In: *Communications of the Association for Information Systems* 8 (2002), Nr. 3, S. 41–65.

[Teng et al. 1995]
TENG, J. T. C. ; CHEON, M. J. ; GROVER, V.: Decisions to Outsource Information Systems Functions: Testing a Strategy-Theoretic Discrepancy Model. In: *Decision Sciences* 26 (1995), Nr. 1, S. 75–103.

[Thompson 1967]
THOMPSON, J. D.: *Organizations in Action: Social Science Bases of Administrative Theory.* New York, USA : MacGraw-Hill, 1967.

[Totok 2001]
TOTOK, A.: *Modellierung von OLAP- und Data-Warehouse-Systemen*. Düsseldorf : Gabler, 2001.

[Troy und Zweben 1981]
TROY, D. A. ; ZWEBEN, S. H.: Measuring the Quality of Structured Designs. In: *Journal of Systems and Software* 2 (1981), S. 113–120.

[Turban 1990]
TURBAN, E.: *Decision Support and Expert Systems: Management Support Systems*. Upper Saddle River, NJ : Prentice Hall, 1990.

[Tversky 1969]
TVERSKY, A.: Intransitivity of Preferences. In: *Psychological Review* 76 (1969), Nr. 1, S. 31–48.

[Tversky 1972]
TVERSKY, A.: Elimination by Aspects: A Theory of Choice. In: *Psychological Review* 79 (1972), Nr. 4, S. 281–299.

[Varian 2003]
VARIAN, H. R.: *Intermediate Economics – A Modern Approach*. 6. New York, London : W. W. Norton and Company, 2003.

[Venable 2006]
VENABLE, J.: A Framework for Design Science Research Activities. In: *Proceedings of the 2006 Information Resource Management Association Conference, Washington, DC, USA*, 2006.

[Venable et al. 2012]
VENABLE, J. ; PRIES-HEJE, J. ; BASKERVILLE, R.: A Comprehensive Framework for Evaluation in Design Science Research. In: PEFFERS, K. (Hrsg.) ; ROTHENBERGER, M. (Hrsg.) ; KUECHLER, B. (Hrsg.): *Design Science Research in Information Systems. Advances in Theory and Practice* Bd. 7286. Berlin, Heidelberg : Springer, 2012, S. 423–438.

[Venkatesh und Bala 2008]
VENKATESH, V. ; BALA, H.: Technology Acceptance Model 3 and a Research Agenda on Interventions. In: *Decision Sciences* 39 (2008), Nr. 2, S. 273–315.

[Venkatesh et al. 2003]
VENKATESH, V. ; MORRIS, M. ; DAVIS, F. ; DAVIS, G.: User Acceptance of Information Technology: Toward a Unified View. In: *MIS Quarterly* 27 (2003), Nr. 3, S. 425–478.

[Verclas und Linnhoff-Popien 2012]
VERCLAS, S. ; LINNHOFF-POPIEN, C.: Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse. In: VERCLAS, S. (Hrsg.) ; LINNHOFF-POPIEN, C. (Hrsg.): *Smart Mobile Apps*. Berlin : Springer, 2012, S. 3–15.

[Versteegen 2004]
VERSTEEGEN, G.: *Anforderungsmanagement*. Berlin ; Heidelberg : Springer, 2004.

[Vlaar et al. 2008]
VLAAR, P. W. L. ; FENEMA, P. C. ; TIWARI, V.: Cocreating Understanding and Value in Distributed Work: How Members of Onsite and Offshore Vendor Teams Give, Make, Demand, and Break Sense. In: *MIS Quarterly* 32 (2008), Nr. 2, S. 227–255.

[Walls et al. 1992]
WALLS, J. G. ; WIDMEYER, G. R. ; EL SAWY, O. A.: Building an Information System Design Theory for Vigilant EIS. In: *Information Systems Research* 3 (1992), Nr. 1, S. 36–59.

[Walther und Burgoon 1992]
WALTHER, J. B. ; BURGOON, J. K.: Relational Communication in Computer-mediated Interaction. In: *Human Communication Research* 19 (1992), Nr. 1, S. 50–88.

[Wang und Strong 1996]
WANG, R. ; STRONG, D.: Beyond Accuracy: What Data Quality Means to Data Consumers. In: *Journal of Management Information Systems* 12 (1996), Nr. 4, S. 5–34.

[Warkentin et al. 1997]
WARKENTIN, M. E. ; SAYEED, L. ; HIGHTOWER, R.: Virtual Teams ver-

sus Face-to-face Teams: An Exploratory Study of a Web-based Conference System. In: *Decision Sciences* 28 (1997), Nr. 4, S. 975–996.

[Warshaw und Davis 1985]
WARSHAW, P. R. ; DAVIS, F. D.: Disentangling Behavioral Intention and Behavioral Expectations. In: *Journal of Experimental Social Psychology* 21 (1985), S. 213–228.

[Wasserman und Faust 1994]
WASSERMAN, S. ; FAUST, K.: *Social Network Analysis: Methods and Applications*. Cambridge : Cambridge University Press, 1994.

[Weber 1983]
WEBER, M.: *Entscheidungen bei Mehrfachzielen*. Wiesbaden : Gabler, 1983.

[Wich und Kramer 2015]
WICH, M. ; KRAMER, T.: Enhanced Human-Computer Interaction for Business Applications on Mobile Devices: A Design-Oriented Development of a Usability Evaluation Questionnaire. In: *Proceedings of the 48th Annual Hawaii International Conference on System Sciences, HICSS, Hawaii*, 2015, S. 472–481.

[Wiener 2006]
WIENER, M.: *Critical Success Factors of Offshore Software Development Projects*. Wiesbaden : Gabler, 2006.

[Williamson 1985]
WILLIAMSON, O. E.: *The Economic Institutions of Capitalism: Firms, Markets, Relational Contracting*. New York, USA : Free Press, 1985.

[Williamson 1990]
WILLIAMSON, O. E.: Transaction Cost Economics. In: SCHMALENSEE, R. (Hrsg.) ; WILLIG, R. D. (Hrsg.): *Handbook of Industrial Organization*. Amsterdam, Netherlands : North-Holland, 1990, S. 135–182.

[Wind und Saaty 1980]
WIND, Y. ; SAATY, T. L.: Marketing Applications of the Analytic Hierarchy Process. In: *Management Science* 26 (1980), Nr. 7, S. 641–658.

[Winkler et al. 2008]
WINKLER, J. ; DIBBERN, J. ; HEINZL, A.: The Impact of Cultural Differences in Offshore Outsourcing - Case Study Results from German-Indian Application Development Projects. In: *Information Systems Frontiers* 10 (2008), Nr. 2, S. 243–258.

[Winkler et al. 2009]
WINKLER, J. ; DIBBERN, J. ; HEINZL, A.: The Impact of Software Product and Service Characteristics on International Distribution Arrangements for Software Solutions. In: *Proceedings of the 30th International Conference on Information Systems, Phoenix, Arizona*, 2009.

[Xiaojun et al. 2004]
XIAOJUN, D. ; JUNICHI, I. ; SHO, H. O.: Unique Features of Mobile Commerce. In: *Journal of Electronic Science and Technology of China* 2 (2004), Nr. 3, S. 205–210.

[Xin und Levina 2008]
XIN, M. ; LEVINA, N.: Software-as-a Service Model: Elaborating Client-Side Adoption Factors. In: BOLAND, R. (Hrsg.) ; LIMAYEM, M. (Hrsg.) ; PENTLAND, B. (Hrsg.): *29th International Conference on Information Systems*, 2008.

[Yaung 1992]
YAUNG, A. T.: Design and implementation of a requirements clustering analyzer for software system decomposition. In: *Proceedings of the 1992 ACM/-SIGAPP Symposium on Applied Computing: Technological Challenges of the 1990s*, 1992, S. 1048–1054.

[Yin 2009]
YIN, R. K.: *Case Study Research: Design and Methods.* Fourth Edition. Thousand Oaks, CA, USA : Sage Publications, 2009.

[Young 2000]
YOUNG, A.: The Future of Outsourcing: Thriving or Surviving? / Dataquest Service Trends, Gartner Group Report. 2000. – Forschungsbericht.

[Zack 1993]
ZACK, M. H.: Interactivity and Communication Mode Choice in Ongoing Management Groups. In: *Information Systems Research* 4 (1993), Nr. 3, S. 207–239.

[Zangemeister 1976]
ZANGEMEISTER, C.: *Nutzwertanalyse in der Systemtechnik*. 4. Auflage. München : Wittemann, 1976.

[Zangemeister 2003]
ZANGEMEISTER, C.: Nutzwertanalyse von Projektalternativen. In: *Logistik Management* 5 (2003), Nr. 2, S. 50–59.

[Zmund 1978]
ZMUND, R. W.: Empirical Investigation of the Dimensionality of the Concept of Information. In: *Decision Sciences* 9 (1978), S. 187–195.

Lebenslauf

Persönliches

Name	Tommi Dirk Kramer
Familienstand	ledig
Staatsangehörigkeit	deutsch

Ausbildung

07/1993 – 06/2002	Privatgymnasium St. Paulusheim, Bruchsal (Abschluss: Abitur)
10/2003 – 01/2010	Studium, Universität Mannheim (Abschluss: Diplom-Wirtschaftsinformatiker)
02/2010 – 04/2016	Wissenschaftlicher Mitarbeiter, Lehrstuhl für ABWL und Wirtschaftsinformatik (Prof. Dr. Armin Heinzl), Universität Mannheim