

Unterstützung der Koexistenz von agilen und traditionellen Anforderungsartefakten

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des akademischen Grades
Doktor-Ingenieurin
(abgekürzt: Dr.-Ing.)
genehmigte Dissertation

von

Dipl.-Math. Olga Boruszewski

(geb.Liskin)

geboren am 27. Juni 1986

in Kiew

2016

1. Referent: Prof. Dr. Kurt Schneider
2. Referent: Prof. Dr. Martin Glinz
Tag der Promotion: 01.12.2016

Olga Boruszewski, *Unterstützung der Koexistenz von agilen und traditionellen Anforderungsartefakten*. Dissertation. Gottfried Wilhelm Leibniz Universität Hannover, 2016.

Abschnitt 4 dieser Arbeit basiert auf der Veröffentlichung
Olga Liskin, „How Artifacts Support and Impede Requirements Communication“ in S.A. Fricker and K. Schneider (Eds.): REFSQ 2015, LNCS 9013, pp. 132 -147, 2015.

Die finale Veröffentlichung ist erhältlich bei Springer via:

http://dx.doi.org/10.1007/978-3-319-16101-3_9

Die Veröffentlichung trägt das folgende Copyright:

© Springer International Publishing Switzerland 2015

© 2016 Olga Boruszewski
Alle Rechte vorbehalten
olga.boruszewski@gmail.com

Druck: epubli, ein Service der Neopubli GmbH, Berlin, <http://www.epubli.de>

ISBN 978-3-7418-8005-6

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Zusammenfassung

Anforderungsartefakte, wie Use Cases, Geschäftsprozessmodelle oder GUI-Mockups, werden für viele Aktivitäten in der Softwareentwicklung eingesetzt. Sie stellen einen wichtigen Bezugspunkt bei der Kommunikation und Klärung von Anforderungen dar. Wie Anforderungen repräsentiert sind, beeinflusst dabei, welche Aktivitäten damit besonders gut durchführbar sind. Beispielsweise können Entwickler User Stories gut für die Schätzung und Priorisierung von Anforderungen einsetzen. Dafür bieten Geschäftsprozessmodelle einen besseren Gesamtüberblick. Einen besonderen Fall stellen hybride Entwicklungsansätze dar, die agile und traditionelle Methoden kombinieren. Agile und traditionelle Methoden verwenden grundsätzlich unterschiedliche Arten von Anforderungsartefakten, sodass sich die Frage stellt, wie eine Anforderungslösung aussieht, die Elemente beider Methoden unterstützt.

Erstes Ziel dieser Arbeit ist, dazu näher herauszufinden, wie Anforderungsartefakte in der Praxis verwendet werden und welche Vor- und Nachteile verschiedene Rollen in verschiedenen Artefakten sehen. Dazu wird im Rahmen der Arbeit eine Interview-Studie durchgeführt. Das Resultat sind vor allem zwei Erkenntnisse: Erstens werden häufig mehrere Arten von Anforderungsartefakten gemeinsam verwendet. Zweitens ergibt sich in der Praxis das Problem, mit den so entstandenen Abhängigkeiten zwischen Anforderungen umzugehen. Informationen sind oft über mehrere Artefakte verstreut, was es erschwert, sie zu finden oder über alle Artefakte hinweg konsistent anzupassen. Diese Probleme führen zu Missverständnissen, Fehlern und nachträglichen Änderungen aufgrund von mangelnder Anforderungskommunikation.

Aufbauend auf diesen Erkenntnissen zum Umgang mit Artefakten stellt diese Arbeit ein Konzept vor, das dem festgestellten Problem auf zwei Ebenen begegnet. Zum einen werden auf methodischer Ebene Ansätze zur Modellierung und inhaltlichen Verbesserung des Aufbaus von Anforderungslandschaften entwickelt. Hier geht es darum, zu entscheiden, ob beispielsweise neben User Stories zusätzliche Artefakte, wie Spezifikationen, eingesetzt und verknüpft werden sollen. Auf der zweiten Ebene, der Projektebene, helfen werkzeuggestützte Ansätze, welche Verknüpfungen automatisiert nutzen, Projektteilnehmern beim Umgang mit konkreten Anforderungen und Abhängigkeiten. Hier geht es beispielsweise darum, Projektteilnehmer bei der Änderung einer Anforderung zu erinnern und ihnen zu helfen, abhängige Anforderungen ebenfalls anzupassen. Die Evaluation der werkzeuggestützten Mechanismen zeigt, dass sie tatsächlich dabei helfen, mit Abhängigkeiten umzugehen. Durch den Einsatz von Verknüpfungen und automatisierten Operationen auf diesen Verknüpfungen können Projektteilnehmer abhängige Artefakte vollständiger und schneller identifizieren.

ren. Jedoch reicht die bloße Hervorhebung verknüpfter Anforderungen nicht aus, um Inkonsistenzen ausreichend zu vermeiden. In einigen Situationen müssen Projektteilnehmer stärker daran erinnert werden, abhängige Anforderungen zu betrachten.

Die vorgestellten Erkenntnisse und Ansätze helfen Requirements Ingenieuren und –Methodikern dabei, verwendete Anforderungsartefakte projektspezifisch zu planen und dabei besonders auf die Koexistenz von agilen und traditionellen Artefakten einzugehen. Die vorgestellten werkzeuggestützten Mechanismen unterstützen Softwareentwickler im effizienteren Umgang mit Abhängigkeiten. Forscher und Werkzeughersteller können die Erkenntnisse dieser Arbeit nutzen, um bisherige Traceability-Methoden besser in die Anwendungsfälle des Requirements Engineerings zu integrieren und um neue Methoden zu finden, welche die Herausforderungen der Koexistenz von Anforderungsartefakten aufgreifen.

Schlagwörter: Software Engineering, Requirements Engineering, Anforderungsartefakte, Artefaktmodelle, Abhängigkeiten, Tracing, Anforderungslandschaft, Trace-Operation, hybride Entwicklungsansätze, agile Softwareentwicklung, User Stories, Use Cases, risikobasierte Software Engineering-Methoden, Anforderungskommunikation

Abstract

Requirements artifacts, such as use cases, business process models or GUI mockups are used for many activities in software development. They are an important reference during requirements communication and clarification. The representation of a requirement has an influence on which activities can be efficiently performed with it. For example, user stories are easy to estimate and prioritize. However, business process models provide a better overview. One special case of applications are hybrid software development approaches, which combine agile and traditional methods. Agile and traditional methods utilize fundamentally different types of requirements artifacts. This raises the question for a suitable requirements artifact strategy, which supports elements of both methodologies.

A first goal of this dissertation is to find out more specifically how requirements artifacts are used in practice and which benefits and drawbacks different roles see in different artifacts. For this, an interview study has been conducted as a part of the dissertation. The results show two insights in particular: First, projects often use multiple types of requirements artifacts in parallel. Second, this causes dependencies between requirements artifacts, which in turn raise problems in handling the requirements in practice. Often, information is scattered among multiple artifacts which makes it difficult to locate or update a requirement in a way that keeps requirements artifacts consistent. These problems lead to misunderstandings, mistakes and subsequent changes due to poor requirements communication.

Based on the insights about the usage of artifacts in practice, this dissertation presents a concept which addresses the observed problems on two levels. First, on a methodological level, it presents approaches for modeling and optimizing the landscape of used requirements artifacts. This part deals with decisions, such as whether to add a specification in addition to employing user stories and whether to use traceability links to connect both artifact types. On the other level, the project level, tool-based approaches automate the usage of traceability links and help project participants with handling requirements and their dependencies. For example, they remind project participants during a requirement update to also check dependent artifacts. The evaluation of the tool-based approach indicates that it actually can help with the handling of requirements artifacts and their dependencies. The usage of traceability links and automated operations based on the links allows project participants to identify dependent artifacts more quickly and more completely. However, just highlighting dependent requirements is not enough in order to avoid inconsistencies sufficiently. In some situations, stronger mechanisms for reminding project participants of dependencies are needed.

The presented insights and concepts help requirements engineers and methodologists with planning the usage of requirements artifacts in a software development project. They allow to find project-specific solutions and to particularly address the coexistence of agile and traditional artifacts. The presented tool-based mechanisms help software developers to handle dependencies more efficiently. Researchers and manufacturers of requirements engineering tools can use the insights of this dissertation to better integrate current traceability methods into requirements engineering use cases and also to create new methods which address the challenges of the coexistence of requirements artifacts.

Keywords: software engineering, requirements engineering, requirements artifacts, artifact models, dependencies, tracing, requirements landscape, trace-operation, hybrid software development, agile software development, user stories, use cases, risk-based software engineering methods, requirements communication

Danksagung

Diese Arbeit wäre ohne die Unterstützung, das Wissen und die Geduld vieler fantastischer Menschen nicht entstanden. Ich kann mich gar nicht genug dafür bedanken.

Mein herzlicher Dank gilt Prof. Dr. Kurt Schneider. Seine Begeisterung und die Fähigkeit, stets spannend über Softwareentwicklung zu berichten und zu schreiben, lassen das Fach nie langweilig werden. Ich habe von ihm eine Menge über die Softwareentwicklung und die daran beteiligten Menschen, aber auch über wissenschaftliches Arbeiten gelernt. Viele Gespräche sowie das Feedback zu unzähligen Themen, Texten und Präsentationen haben dazu geführt, dass ich mich Schritt für Schritt weiterentwickeln konnte. Danke für die Zeit und die Zuversicht.

Auch bei Prof. Dr. Martin Glinz möchte ich mich herzlich für die Anregungen zum Thema und das konstruktive und anspruchsvolle Feedback zur Arbeit bedanken.

Ich bedanke mich bei allen Mitarbeiterinnen und Mitarbeitern des Fachgebietes Software Engineering für die schöne, konstruktive und anregende Atmosphäre. Es gibt nichts Bereichernderes, als täglich von tollen Menschen umgeben zu sein. Besonders meinen Bürokollegen Leif, Raphael, Tobi und Daniel G. danke ich für die gegenseitige Motivation und den Beistand. Unsere Gespräche waren alles: mitreißend, fachlich interessant, lustig, energisch, aufmunternd, philosophisch, verrückt. Danke, Daniel L., dass du mich von der Studienarbeit an mit deiner Begeisterung motiviert hast und dass du mich immer wieder überzeugt hast, dass alles gut wird.

Danke allen direkt und indirekt Beteiligten der Interview-Studie, die mir mit ihren Aussagen und Reflexionen Einblicke in ihre Welt der Softwareentwicklung gegeben und mich nachhaltig motiviert haben, die Anforderungskommunikation in Projekten fortwährend verbessern zu wollen.

Ein ganz besonderer Dank geht an meine Eltern Elena und Viktor Liskin, an meinen Mann Björn und seine Familie sowie an meine Freunde. Danke, dass ihr einfach so jederzeit an mich geglaubt habt. Das hat einen unschätzbaren Wert für mich.

Hannover, Dezember 2016
Olga Boruszewski

Inhalt

Zusammenfassung	i
Abstract.....	iii
Danksagung	v
Inhalt	vii
1 Einführung.....	1
1.1 Beiträge der Dissertation	3
1.2 Forschungsmethodik	5
1.3 Aufbau der Arbeit.....	6
2 Grundlagen	9
2.1 Hybride Entwicklungsansätze	9
2.2 Artefaktororientierung in Entwicklungsansätzen	10
2.3 Anforderungsartefakte.....	12
2.4 Tracing von Anforderungen	14
3 Verwandte Arbeiten.....	19
3.1 Anforderungsartefakte.....	19
3.2 Verwendung mehrerer Artefakttypen.....	20
3.3 Artefaktmodelle.....	23
3.4 Automatisierte Operationen auf Traceability Links.....	25
3.5 Wertbasierte Softwareentwicklungsmethoden	29
4 Stand der Praxis: Interview-Studie zum Umgang mit Anforderungsartefakten.....	31
4.1 Einleitung	31
4.2 Aufbau der Studie.....	32
4.3 Durchführung der Studie.....	33
4.4 Ergebnisse	35
4.5 Diskussion	47
4.6 Validität der Ergebnisse	49
4.7 Zusammenfassung.....	51
5 Ein Konzept zur Unterstützung der Koexistenz von agilen und traditionellen Anforderungsartefakten	53

5.1	Zielsetzung – Erstellung einer Methodik zur Unterstützung der Koexistenz.....	53
5.2	Zusammenhang zwischen Methodenebene und Projektebene.....	55
6	Hybride Anforderungslandschaften.....	59
6.1	Anforderungsartefakte.....	59
6.2	Beziehungen zwischen Anforderungsartefakten.....	67
6.3	Anforderungslandschaften als Modell der Typen von Anforderungen und Verknüpfungen.....	75
6.4	Ein Metamodell für Anforderungslandschaften.....	83
6.5	Eine Notation für Anforderungslandschaften.....	85
6.6	Verwendung von Anforderungslandschaften.....	90
6.7	Typische hybride Landschaften und deren Entstehung.....	93
6.8	Herausforderungen aus der Koexistenz von Anforderungsartefakten.....	100
7	Risikobasierte Optimierung von Anforderungslandschaften.....	117
7.1	Risk-based Balancing als Optimierungsmethode.....	117
7.2	Optimierungsdimensionen.....	123
7.3	Umgang mit mehreren Optimierungsdimensionen.....	126
7.4	Systematischer Aufbau einer Anforderungslandschaft.....	127
7.5	Vorgehen zur Bewertung der Risikofaktoren.....	129
7.6	Ergebnis: Ermittelte Risikofaktoren.....	140
8	Werkzeugbasierte Unterstützung anforderungsbezogener Aktivitäten: der IRE-Ansatz.....	161
8.1	Leitprinzipien für die Lösungserstellung.....	162
8.2	Überblick über den IRE-Ansatz.....	165
8.3	Integrierte Sichten.....	172
8.4	Trace-Operationen.....	176
8.5	Ausführung der Trace-Operationen in einem Werkzeug.....	197
9	Validierung des IRE-Ansatzes.....	211
9.1	Evaluationsziele.....	213
9.2	Evaluiierungsmethode.....	215
9.3	Experimentdurchführung und Datensammlung.....	218
9.4	Ergebnisse.....	220
9.5	Diskussion.....	223
9.6	Validität der Ergebnisse.....	223
9.7	Zusammenfassung.....	225
10	Zusammenfassung und Ausblick.....	227
10.1	Zusammenfassung.....	227

10.2	Kritischer Rückblick	227
10.3	Ausblick	230
Anhang A – Interview-Aussagen mit Ableitung von Risiken		235
Anhang B – Unterlagen zur Experiment-Studie		250
Anhang C – Mathematische Erläuterung zur Addition von Risk Exposure-Werten		255
Literaturverzeichnis		260
Lebenslauf.....		275

1 Einführung

Der Erfolg von Softwareprojekten hängt in großem Umfang davon ab, wie gut die Anforderungen an die Software verstanden, ausgearbeitet und umgesetzt werden. Fehler in einer dieser Aktivitäten können dazu führen, dass fertigestellte Software die Probleme der Kunden nicht löst oder nicht deren Wünschen entspricht [144]. Auch qualitativ hochwertige Software kann so dennoch zu Unzufriedenheit bei den Kunden führen.

Requirements Engineering-Methoden legen Prozesse, Aktivitäten, Rollen und Artefakte fest, die dabei helfen, Anforderungen effektiv zu erarbeiten und für die weitere Entwicklung bereitzustellen. Sie stellen bewährte Vorgehensweisen dar, die helfen Ingenieursprinzipien, wie Kostendenken und Wiederverwendung auch in komplexen Projekten einzuhalten. Für den Umgang mit Anforderungen spielt der Aspekt *Artefakte* eine wichtige Rolle. Neben Anforderungsartefakten finden in der Softwareentwicklung auch weitere Artefakte, wie Entwurfsdokumente, Tests und der Code selbst Verwendung. Besonders im Tracing werden diese unterschiedlichen Arten von Artefakten häufig verknüpft. In dieser Arbeit liegt jedoch das Augenmerk auf dem Zusammenspiel zwischen Anforderungsartefakten unter sich, da diese die Kommunikation und den Umgang mit Anforderungen maßgeblich beeinflussen.

Anforderungsrepräsentationen werden in vielen Aktivitäten unterschiedlicher Rollen innerhalb des Softwareprojektes verwendet. Die Repräsentation von Anforderungen stellt einen wichtigen Bezugspunkt für die Kommunikation von Anforderungen dar. Dieser hilft, durch Externalisierung von Wissen ein gemeinsames Verständnis herzustellen [58], Missverständnisse aufzudecken und Informationen weiter zu konkretisieren [125]. Zudem werden mittels Artefakten Informationen festgehalten, sodass auch im weiteren Verlauf eines Projektes ein Bezug zu diesen Informationen möglich ist [141]. Die Artefakte müssen zu ihren Nutzern passen, um effektiv verwendet zu werden.

Je nach Art der Artefakte werden unterschiedliche Aspekte von Anforderungen besonders in den Vordergrund gerückt oder weggelassen. Story Cards stellen Anforderungen als einzelne nutzerorientierte Interaktionen dar und fördern so die Priorisierung, Schätzung und Detail-Diskussion einzelner Anforderungen. Abhängigkeiten zwischen Anforderungen sind so jedoch häufig nicht gut sichtbar. Geschäftsprozessmodelle hingegen bringen Aktivitäten der Softwarenutzer in eine Reihenfolge und weisen so auf Abhängigkeiten hin. Die einzelnen Modellelemente lassen sich jedoch wiederum schlecht für sich priorisieren oder beispielsweise zu Komponenten zuordnen. Nichtfunktionale Anforderungen oder allgemeine Visionen lassen sich abermals mit keiner der beiden erwähnten

1 Einführung

Artefaktarten gut darstellen. Hierfür eignen sich eher Spezifikationsdokumente, die auch textuelle Anforderungen ohne besondere Struktur zulassen.

Oft werden in einem Projekt mehrere Arten von Artefakten gemeinsam verwendet, um für die unterschiedlichen Aktivitäten jeweils eine passende Repräsentation parat zu haben. So kann beispielsweise die Iterationsplanung mithilfe von User Stories durchgeführt werden, während zur Erstellung von Systemtests die Geschäftsprozessmodelle herangezogen werden. Die Informationen in den unterschiedlichen Artefakten überlappen sich dabei teilweise, was zu Abhängigkeiten zwischen Artefaktarten führt. Wird am Prozessmodell eine Aktivität verändert, so muss diese Änderung möglicherweise auch in den User Stories durchgeführt werden, um Inkonsistenzen zu vermeiden. Die Abhängigkeiten führen zu neuen Herausforderungen, die in einem guten Requirements Engineering-Ansatz ebenfalls berücksichtigt werden müssen.

Auf dem Zusammenspiel von Anforderungsartefakten unterschiedlicher Art liegt ein wichtiges Augenmerk dieser Arbeit. Zum einen wird durch eine *Koexistenz* von Anforderungsartefakten ermöglicht, mehrere Blickwinkel auf die gewünschte Software bereitzustellen und so mehrere Arbeitsweisen zu unterstützen. Auf der anderen Seite wird der Umgang mit den Anforderungen so auch erschwert, weil Information verteilt wird und mehr Abhängigkeiten und auch Inkonsistenzen auftreten. Ein gutes Zusammenspiel von Anforderungsartefakten ist hier wichtig. Dieses ist auch in hybriden Entwicklungsansätzen, die immer häufiger anzutreffen sind [157], relevant.

Die meisten Vorgehensmodelle, wie SCRUM [145], V-Modell [134] oder RUP [87], folgen einem agilen oder einem traditionellen, das bedeutet plangetriebenen, Entwicklungsansatz. Diese beiden Arten von Entwicklungsansätzen unterscheiden sich unter anderem darin, welche Artefaktarten sie verwenden. In agilen Ansätzen wird die Dokumentation so leichtgewichtig wie möglich gestaltet. Es werden einige wenige Artefaktarten, wie Epics, User Stories und Abnahmetests verwendet und die einzelnen Artefakte so simpel wie möglich gehalten [8]. In traditionellen Methoden wird Wert darauf gelegt, dass die Dokumentation möglichst vollständig und möglichst detailliert ist [16]. Anforderungen werden aus mehreren Blickwinkeln dargestellt, sodass Sichten auf funktionale und nichtfunktionale Anforderungen, Datenmodelle, Geschäftsprozessmodelle und Oberflächen abgedeckt werden.

Neben rein agilen oder traditionellen Ansätzen werden auch immer häufiger hybride Entwicklungsansätze eingesetzt, die agile und plangetriebene Methoden vereinen [16], [34], [66]. Da die typischen Artefakte der einzelnen Ansätze grundsätzlich verschieden sind, fällt es besonders schwer, die Dokumentationsweisen zu verweben [66]. Oft werden mehrere verschiedene Arten von Artefakten zusammen verwendet. Neben User Stories können beispielsweise Spezifikationen [27], Use Cases [94] oder eine Reihe von UML-Modellen zusammen mit

verschieden granularen textuellen Anforderungen [5] hinzugenommen werden. Auch die iterative Weiterentwicklung von Use Cases und weiteren Artefakten, wie im RUP [87] angedeutet, lässt sich in einen agilen Prozess integrieren.

So können für die einzelnen Projektaktivitäten die jeweils passenden Artefakte gewählt werden. Einige der Artefakte werden bevorzugt für agile Elemente des Entwicklungsansatzes eingesetzt und andere Artefakte für die plangetriebenen Elemente. Da dann aber mit den jeweiligen Artefakten unterschiedlich umgegangen wird, können hier besondere Herausforderungen durch deren Abhängigkeiten entstehen. Während die Artefakte, die agil verwendet werden, sich häufig und leicht ändern, sollen Artefakte im plangetriebenen Vorgehen eher einen festen Planungs-Rahmen darstellen und gleich bleiben. Existieren viele Abhängigkeiten zwischen den Artefakten, so führt dies schnell zu Konflikten.

1.1 Beiträge der Dissertation

Zunächst fehlt in der Wissenschaft bisher ein Überblick darüber, wie in der Praxis Anforderungsartefakte verwendet werden. Dazu gehören Fragestellungen, wie, welche Artefakte Projektteilnehmer aus welchen Gründen einsetzen, welche Herausforderungen sie im Umgang mit Anforderungsartefakten sehen und wie sie das Zusammenspiel zwischen verschiedenen Artefakten bewerten. Mit einer Studie über den Stand der Praxis leistet diese Dissertation einen ersten Beitrag, indem sie diesen Überblick herstellt.

Wie die Studie zeigt, werden in der Praxis oft mehrere verschiedene Artefaktarten gemeinsam verwendet. Dabei treten häufig Probleme auf, die auf das Zusammenspiel und die Abhängigkeiten zwischen Anforderungsartefakten zurückzuführen sind. Oftmals müssen Informationen an mehreren Stellen festgehalten und genauso auch an mehreren Stellen angepasst werden, wenn sich Änderungen ergeben. Nicht immer ist klar, in welchen Artefakten Informationen zu einem bestimmten Thema abgelegt sind, was die Suche nach Informationen erschwert. Für einen effektiveren Umgang mit Anforderungsartefakten müssen diese Abhängigkeiten vermindert oder zumindest gut beherrscht werden. Ein gutes Zusammenspiel zwischen Anforderungsartefakten ist also ein relevanter Faktor, ohne den auch die Anforderungskommunikation schnell ins Stocken kommt. Wie ein gutes Zusammenspiel zwischen verschiedenen Anforderungsartefakten – vor allem auch in hybriden Ansätzen – erreicht werden kann, bildet daher die Forschungsfrage für diese Arbeit.

Bei der Behandlung dieser Frage tun sich bisher einige Lücken auf. Bisherige Ansätze in der Wissenschaft beschäftigen sich damit, wie die in einem Projekt verwendeten Artefakttypen und deren Verknüpfungen modelliert werden können. Dazu werden sogenannte Traceability-Modelle aufgestellt [2], [85], [96], [110]. Ein Defizit dieser Modelle in Bezug auf die oben genannten Probleme ist, dass sie nur explizit definierte Verknüpfungen betrachten, jedoch keine weiteren

1 Einführung

Aspekte modellieren, die das Zusammenspiel von Artefakten beeinflussen. Weitere Aspekte wären zum Beispiel Abhängigkeiten zwischen Artefaktarten oder die Anpassbarkeit eines Artefakts.

Außerdem fehlt in der Forschung zu Traceability-Modellen eine Hilfestellung zur Erstellung inhaltlich guter Modelle. Beispielsweise wird gezeigt, welche Attribute eine Verknüpfung besitzen sollte [2], [96], allerdings bleibt offen, wann nun zwischen zwei bestimmten Artefakttypen ein bestimmter Verknüpfungstyp eingesetzt werden sollte und wann nicht. Dadurch bleibt offen, wie ein gutes Zusammenspiel zwischen unterschiedlichen Anforderungsartefakttypen unterstützt werden kann. Genau das ist aber wichtig sowohl für die Erstellung eigener hybrider Entwicklungsansätze als auch für die Individualisierung bestehender Vorgehensmodelle, wie RUP, SCRUM oder V-Modell XT.

Die vorliegende Dissertation schließt diese beiden Lücken, indem sie eine Erweiterung des Traceability-Modell-Konzeptes erstellt. Mithilfe von *Anforderungslandschaften* werden neben den Artefakttypen und ihren Verknüpfungen weitere Eigenschaften abgebildet, die für die *Koexistenz* von Artefakten von Bedeutung sind. Dabei spielen Faktoren, wie die Anpassbarkeit der Artefakte oder spezielle Typen von Abhängigkeiten eine Rolle. Aufbauend darauf stellt die Arbeit ein Verfahren zur inhaltlichen Planung und Optimierung von Anforderungslandschaften in Hinblick auf Koexistenz vor. Ein risikobasierter Ansatz gibt Hilfestellung dabei, ob beispielsweise bestimmte Artefakttypen eingeführt und wann diese durch Traceability-Links verknüpft werden sollen. Der Ansatz basiert auf der oben genannten Praxisstudie und greift damit genau die Probleme aus der Praxis auf. Damit kann bereits auf Methodenebene für jedes Projekt eine individuell passende Zusammenstellung gefunden werden, die berücksichtigt, welche Blickwinkel besonders gut abgedeckt sein sollten und die hilft, Abhängigkeiten zu beherrschen und unnötige Abhängigkeiten zu vermeiden.

Dennoch lassen sich nicht alle Abhängigkeiten vermeiden. Um mit den Abhängigkeiten gut umgehen zu können, sind Tracing-Methoden notwendig. Diese dokumentieren Abhängigkeiten durch explizite Verknüpfungen und erlauben es, durch Folgen von Verknüpfungen zu passenden Informationen zu gelangen. Jedoch werden Tracing-Methoden oft nur in komplexen oder umfangreichen Projekten verwendet. In kleineren Projekten wird der Aufwand als zu hoch im Vergleich zum Nutzen gesehen. Dort sind die Abhängigkeiten zwischen Anforderungsartefakten oft ein Randthema, das nicht explizit angegangen wird. Es stehen die eigentlichen anforderungsbezogenen Aktivitäten, wie die Ausarbeitung, Schätzung und Priorisierung von Anforderungen im Vordergrund.

Lösungsansätze aus der Wissenschaft begegnen diesem Problem durch Automatisierung von Tracing-Aktivitäten. Die automatisierte Verwendung von Verknüpfungen zur Identifikation von relevanten Anforderungsartefakten wird durch das Konzept der Trace Queries ermöglicht. Damit werden Suchanfragen

auf Projektartefakten formuliert, die dann automatisiert durchgeführt werden. Allerdings müssen Trace Queries bisher manuell und auch selbstmotiviert angestoßen werden.

Um diese Lücke zu schließen, stellt diese Dissertation das Konzept der *Trace-Operationen* vor. Trace-Operationen bauen auf Trace Queries auf, erweitern diese aber um weitere Elemente, die es erlauben, Trace Queries automatisiert anzustoßen und die Ergebnisse automatisiert auszuwerten oder darzustellen. Hierzu werden in Trace-Operationen zusätzlich Auslöser und Überprüfungen für die Trace Queries festgelegt. Durch die Erweiterung wird es möglich, Tracing-Aktivitäten in die eigentlichen anforderungsbezogenen Aktivitäten, wie die Ausarbeitung oder Priorisierung von Anforderungen, zu integrieren. Die Integration wiederum steigert den Nutzen von Tracing-Methoden und hilft so, Traceability lohnenswerter zu machen.

1.2 Forschungsmethodik

Um die Koexistenz von Anforderungsartefakten besser zu unterstützen, sind sowohl neue Erkenntnisse als auch neue konzeptuelle Lösungen notwendig. Diese Arbeit wendet verschiedene Methoden an, um diese zu erreichen.

Zunächst wird die **Problemstellung** näher untersucht und definiert. Um zu neuen Erkenntnissen bezüglich der Verwendung und des Zusammenspiels von Anforderungsartefakten in der Praxis zu gelangen, wird eine qualitative Studie durchgeführt. In einer Interview-Studie – durchgeführt und ausgewertet nach dem Grounded Theory-Ansatz [56], [69] – werden praktische Erfahrungen und Herausforderungen im Umgang mit Anforderungen erhoben. Für diese Studie wurde der Grounded Theory-Ansatz gewählt, um die Fragestellung möglichst offen zu behandeln. Leitfragen für die Interviews werden bereits im Vorfeld festgelegt, um eine grundsätzliche Richtung vorzugeben. Basierend auf ersten Interview-Erkenntnissen wurden diese Fragen jedoch mit der Zeit erweitert.

Basierend auf den festgestellten Problemen werden dann **Lösungen** konstruiert, die auf den festgestellten Problemen aufbauen. Dabei gibt es drei Ansatzpunkte für Lösungen. Zum einen wird eine Möglichkeit benötigt, den Aufbau einer Anforderungslandschaft zu beschreiben. Ein Metamodell zusammen mit einer grafischen Notation formalisieren hier die Darstellung. Zweitens werden Ansätze gebraucht, um den Aufbau einer Anforderungslandschaft auch inhaltlich so zu gestalten, dass das Zusammenspiel der Anforderungsartefakte begünstigt wird. Hierzu wird ein systematisches risikobasiertes Verfahren erstellt, das es erlaubt, Entscheidungen anhand von Risikoeinschätzungen zu treffen. Die dazu aufgestellten Risikofaktoren werden wiederum durch Abstraktion aus den berichteten Erfahrungen in der Interview-Studie erstellt. Der dritte Lösungsansatz bezieht sich auf die Unterstützung des täglichen Umgangs mit Anforderungen

1 Einführung

und deren Abhängigkeiten. Auch hier orientiert sich die Lösung an den Problemen aus der Interview-Studie.

Eine **Evaluation** prüft dann Teile der Lösungsansätze darauf, ob sie hilfreich sind. Hierbei wird das Augenmerk auf den Nutzen des Ansatzes beim täglichen Umgang mit Anforderungen gelegt. In einem kontrollierten Experiment wird dazu überprüft, ob die Einführung von Verknüpfungen und Trace-Operationen einen Einfluss auf den Umgang mit Anforderungen und deren Abhängigkeiten hat.

1.3 Aufbau der Arbeit

Die zwei zentralen Konzepte dieser Arbeit sind Anforderungslandschaften und Trace-Operationen. *Anforderungslandschaften* stellen den Aufbau und die Zusammensetzung von Anforderungsartefakten in einem Softwareprojekt dar. Sie dienen dazu, auf Methodenebene festzulegen, welche Artefakte im Projekt verwendet werden und wie diese verknüpft werden sollen. Diese explizite Planung der zu verwendenden Anforderungsartefakte dient dazu, benötigte Blickwinkel auf die Anforderungen bereitzustellen und die anforderungsbezogenen Aktivitäten unterschiedlicher Projektteilnehmer zu unterstützen. *Trace-Operationen* werden dann verwendet, um in der täglichen Arbeit mit Anforderungen den Umgang mit Abhängigkeiten zwischen den Artefakten zu unterstützen. Sie werden mithilfe der Verknüpfungen aus der Anforderungslandschaft definiert. Ausgelöst durch bestimmte Zustände oder Aktionen auf den Anforderungen, werden automatisierte Tracing-Aktivitäten durchgeführt und die Ergebnisse den Projektteilnehmern während der Arbeit mit Anforderungen präsentiert. Die verschiedenen zur Konzepterstellung bearbeiteten Aspekte und deren Aufteilung auf die Kapitel sind in Abbildung 1 dargestellt.

Zunächst werden in Kapitel 2 die Grundlagen und in Kapitel 3 die verwandten Arbeiten beschrieben. Hier wird besonders auf Artefaktmodelle sowie Tracing-Ansätze eingegangen.

In Kapitel 4 wird eine Studie vorgestellt, die untersucht, wie Anforderungslandschaften in der Praxis beschaffen sind. Es werden Gründe zur Wahl bestimmter Anforderungsartefakte untersucht und die Herausforderungen, die in der Praxis im Umgang mit Anforderungslandschaften entstehen, beleuchtet.

Kapitel 5 zeigt einen Lösungsansatz der die festgestellten Herausforderungen betrachtet. Der Ansatz wird dann in den weiteren Kapiteln näher beschrieben. In Kapitel 6 werden Anforderungslandschaften mit den dafür relevanten Elementen, wie Artefakten und Verknüpfungen, definiert. Außerdem wird eine Notation vorgestellt, welche die relevanten Aspekte darstellt und zur Planung und Dis-

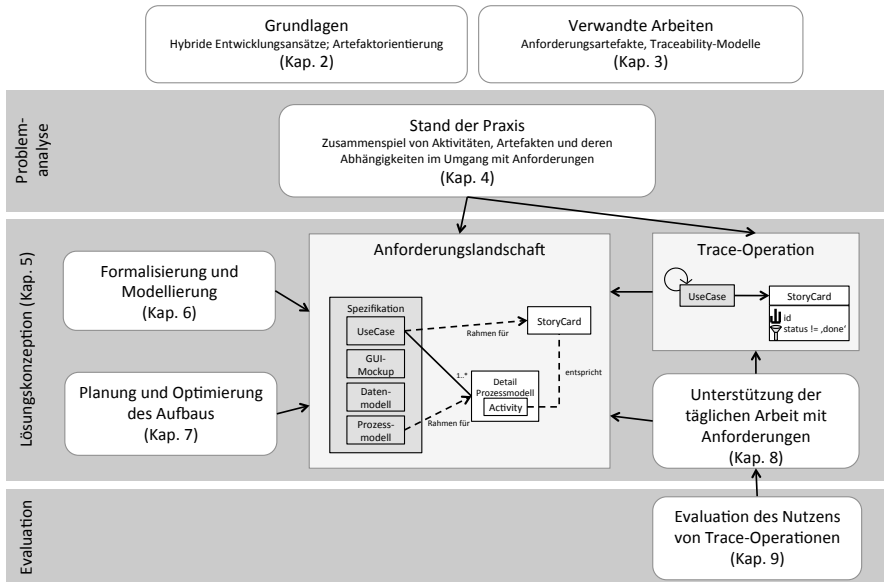


Abbildung 1: Aufbau der Arbeit

kussion von Anforderungslandschaften verwendet wird. Kapitel 7 gibt eine Vorgehensweise zur Planung und Optimierung von Landschaften an. Darin werden Vor- und Nachteile von Artefakten und Verknüpfungen systematisch ausgewertet, um Entscheidungen zu einzelnen Optimierungsschritten zu unterstützen. In Kapitel 8 wird dann der Umgang mit Anforderungslandschaften in der täglichen Arbeit beleuchtet. Zur Unterstützung des Umgangs mit Abhängigkeiten wird das Konzept der Trace-Operationen vorgestellt.

Die Nützlichkeit eines Systems, das Trace-Operationen verwendet, wird anschließend in einer Evaluation in Kapitel 9 untersucht.

In der Zusammenfassung und dem Ausblick in Kapitel 10 wird schließlich diskutiert, an welchen Stellen die beiden Konzepte weiter verbessert werden können.

Die Anhänge A und B enthalten Basisdaten, die für die risikobasierte Optimierungsmethode und die Auswertung der Evaluierung herangezogen wurden. Die Rohdaten zur Interview-Studie liegen dem Fachgebiet Software Engineering der Leibniz Universität Hannover als anonymisierte Interview-Transkripte in – aufgrund von Vertraulichkeitsgründen – nicht öffentlich zugänglicher Form vor.

2 Grundlagen

2.1 Hybride Entwicklungsansätze

Immer mehr Softwareprojekte wählen hybride Entwicklungsansätze, die sowohl aus agilen als auch traditionellen Elementen¹ bestehen. Das Ziel dieser Herangehensweise ist, Vorteile aus beiden Ansätzen zu vereinen. Beispielsweise werden agile Elemente eingesetzt, um gute Kundenkommunikation und effektiven Umgang mit Änderungen zu erreichen [8], [17]. Traditionelle Elemente kommen zum Einsatz, um die Planbarkeit, Zuverlässigkeit und Nachvollziehbarkeit im Projekt zu stärken [9], [141].

Ein wichtiger Aspekt bei der Kombination der Entwicklungsansätze ist die Dokumentation von Anforderungen. Die verwendeten Anforderungsartefakte in einem Projekt beeinflussen die Kommunikation sowie weitere Aktivitäten in der Softwareentwicklung. Traditionelle und agile Entwicklungsansätze fußen auf grundsätzlich unterschiedlichen Artefakten. Häufig werden in traditionellen Methoden Spezifikationen eingesetzt, die aus Use Case-Tabellen, Prozess- und Datenmodellen, textuellen Beschreibungen funktionaler und nichtfunktionaler Anforderungen bestehen [137]. Diese werden aus agiler Sicht als unflexibel angesehen. Agile Methoden hingegen sind um User Stories herum aufgebaut [8], [30]. Diese sind aus traditioneller Sicht keine „angemessene Dokumentation von vertraglich vereinbarten Requirements“ ([9], S. 102). Die bloße Einschränkung auf entweder typisch agile oder typisch traditionelle Anforderungsartefakte führt bei hybriden Methoden selten zum gewünschten Ziel. Jedoch ist es ebenfalls schwierig, Mischformen für einzelne Anforderungsartefakte zu finden [67]. Häufig wird eher über Tailoring bzw. individuelle Erweiterung bestehender Methoden versucht, mehrere Artefakte der verschiedenen Ansätze gemeinsam zu verwenden [5], [16]. Die Idee ist hier, die Vorteile der verschiedenen Artefakte zu vereinen und die Nachteile zu meiden.

In dieser Arbeit werden genau solche hybriden Entwicklungsansätze betrachtet, in denen agile und traditionelle Artefakte gemeinsam verwendet werden. Ziel ist es hier, eine effektive Koexistenz zwischen agilen und traditionellen Anforderungsartefakten zu schaffen. Um die Anforderungsartefakte in einem hybriden Entwicklungsansatz zu beschreiben, wird in dieser Arbeit der Begriff *hybride Anforderungslandschaft* verwendet. Dieser wird in Abschnitt 6.3.2 definiert. Die Arbeit wird für solche hybriden Anforderungslandschaften Planungs- bzw. Optimierungsmechanismen erörtern sowie Unterstützungsoperationen entwickeln.

¹ Was genau unter einem Element verstanden werden kann, wird in Abschnitt 5.2 (Methodik) erklärt, wo es um die Bestandteile von Methoden geht. Prinzipiell können zu einer Methode beispielsweise Prozesse, Artefakte, Rollen sowie Aktivitäten bzw. Praktiken festgelegt werden.

Allgemein ist mit einem Projekt mit hybrider Anforderungslandschaft ein Projekt gemeint, in dem agile und traditionelle Anforderungsartefakte gemeinsam verwendet werden. Für sämtliche Herausforderungen kommt es jedoch nicht unbedingt darauf an, ob ein Artefakt agil bzw. traditionell ist, sondern eher allgemeiner darauf, wie sich das Artefakt bei Änderungen verhält – also ob es leicht änderbar ist oder ob es bei Änderungen Widerstände gibt. Diese Arbeit differenziert dazu zwischen sogenannten *zähen* und *flexiblen Anforderungsartefakten*. Charakteristisch für hybride Anforderungslandschaften ist, dass dort zähe und flexible Anforderungsartefakte zusammenkommen, wodurch Herausforderungen in ihrer gemeinsamen Handhabung verursacht werden.

2.2 Artefaktorientierung in Entwicklungsansätzen

Softwareentwicklungs-Methoden beschreiben, wie bei der Softwareentwicklung vorgegangen werden soll und helfen den Projektteilnehmern so, ihre Aktivitäten und Ergebnisse aufeinander abzustimmen. Softwareentwicklungs-Methoden (in dieser Arbeit auch als Entwicklungsansätze bezeichnet) sind komplex. Es gibt viele Faktoren, die festzulegen oder zu berücksichtigen sind. Die wichtigsten Faktoren sind ([28], [72], [76], [87], [89], [148]):

- die Rollen, die am Projekt beteiligt sind
- die Artefakte bzw. Work Products, die im Zuge des Projektes erstellt oder manipuliert werden
- der Prozess als Folge von Tätigkeiten und Definition von Phasen und Meilensteinen
- die genaueren Aktivitäten, die zur Erstellung der Work Products ausgeführt werden
- Standards und Werkzeuge, welche die Entwicklung unterstützen

Weitere Faktoren, die in Bezug auf eine Methodik berücksichtigt werden müssen, sind nach Cockburn [28]: Prozesse, Meilensteine, Qualität, Techniken, Teams, Fähigkeiten, Menschen, Persönlichkeiten sowie Werte im Team.

Manchmal wird diese gesamte Sammlung aus Aspekten, also die Methodik, auch als *Prozess* bezeichnet. Dies lässt jedoch keine klare Abgrenzung zwischen der Sammlung aller Methodik-Bestandteile und des konkreten Unterelementes Prozess, als Folge von Tätigkeiten, zu. In dieser Arbeit wird daher in Anlehnung an die Definition aus dem IEEE-Standard 24765:2010 (Systems and Software Engineering – Vocabulary) [72] ein **Prozess** als Folge von Tätigkeiten bezeichnet und eine **Methodik** als Sammlung von Prozessen, Artefakten, Rollen und weiteren Faktoren, die zur Durchführung eines Softwareprojektes festgelegt werden.

In der *Requirements Engineering*-Methodik kommt den Artefakten eine wichtige Rolle zu. Sie dienen der Konservierung von Wissen, als Hilfsmittel zur Er-

zeugung von Gesamtüberblick und Detailtiefe sowie zur Aufdeckung von Missverständnissen [141]. Auch innerhalb agiler Vorgehensweisen, zu deren Prinzipien es gehört, unnötige Dokumentation durch Kommunikation zu ersetzen [51], stellen Artefakte in Form von User Stories doch ein wichtiges Hilfsmittel dar. Laut Jeffries gehören zur User Story die Konzepte *Card* (physische Karte, auf welche die Story grob notiert wird), *Communication* (Gespräche, die der Klärung der Details dienen), und *Confirmation* (formale Tests zur Validierung) [30], [79]. Auch wenn hier das Artefakt nicht alle Anforderungsdetails enthält, dient es als Repräsentant für diese Details [125] und als Verankerung der *Communication*- und *Confirmation*-Aspekte. Mithilfe der Karte wird Aufwand geschätzt, Funktionalität priorisiert und die Kommunikation fokussiert.

Die Fokussierung auf Artefakte im Gegensatz zu Prozessen im Requirements Engineering bietet mehrere Vorteile, die im Folgenden erläutert werden.

Artefakte *unterstützen die Kommunikation*, indem sie die Anforderungen und deren Auswirkungen aus verschiedenen Blickwinkeln darstellen. Durch Externalisierung von Ideen und Wissen sowie Konkretisierung helfen sie zudem, ein gemeinsames Verständnis zwischen mehreren Personen herzustellen [58], [102], [125].

Doch nicht nur die Kommunikation, sondern eine *ganze Fülle an anderen Aktivitäten* von unterschiedlichsten Rollen wird durch Anforderungsartefakte beeinflusst. Auf viele Aktivitäten, wie die Klärung von Anforderungsdetails, die Iterationsplanung, oder das Projektmonitoring wirken Anforderungsartefakte ein. Verschiedene Artefakte, wie Spezifikationen, Geschäftsprozessmodelle, GUI-Mockups oder User Stories stellen unterschiedliche Aspekte von Anforderungen in den Vordergrund und unterstützen damit verschiedene Aktivitäten besser oder schlechter. Die Wahl passender Artefakte für ein Projekt und die Bewertung ihrer Qualität hängt damit auch vom Kontext im Sinne der im Projekt vertretenen Rollen und ihrer durchgeführten Aktivitäten ab [47].

Méndez unterscheidet zwischen prozessorientierten und artefaktorientierten Methoden [113]. Eine prozessorientierte Methode betont, welche Phasen und Aktivitäten in der Softwareentwicklung durchgeführt werden. Sie sieht Artefakte als Ergebnis dieser Aktivitäten, legt aber nicht immer fest, wie die Artefakte aussehen und zusammenhängen. Méndez zitiert auch eine Studie von Braun aus 2005 [20], in der festgestellt wurde, dass nur ca. 50% der dort untersuchten Vorgehensmodelle überhaupt Beschreibungen der Artefakte enthielten.

Während es in einem prozessorientierten Ansatz also eher darum geht, *wie* Artefakte hergestellt werden, fokussiert eine artefaktorientierte Methode hingegen, *welche* Artefakte erstellt werden, wie diese aussehen und insbesondere wie sie zusammenhängen. Das Vorgehen zur Erstellung im einzelnen, wird dabei für die Projektteilnehmer offen gelassen. Ein artefaktorientierter Ansatz hilft dabei,

konsistente Artefakte zu erzeugen und einen *flexiblen Prozess* zu unterstützen [112], [113].

Zudem haben Méndez et al. festgestellt, dass Artefaktorientierung auch für Aktivitäten im *Software Process Improvement* hilfreich sein kann: Artefaktmodelle abstrahieren Prozess- und Methodenaspekte und können so einen Teil der Komplexität von Prozessen verstecken [115]. Dies unterstützt die Kommunikation und ermöglicht unter anderem einen besseren Wissenstransfer bei Prozessverbesserungsanstrengungen [114]. Kuhrmann et al. stellen ebenfalls fest, dass die Sammlung und Analyse relevanter Informationen innerhalb Prozessverbesserungsvorhaben durch Fokussierung auf Artefakte vereinfacht werden kann [90], [91].

2.3 Anforderungsartefakte

Im Lauf eines Softwareprojektes werden unterschiedliche Arten von Anforderungsartefakten erzeugt. In diesem Abschnitt wird kurz beleuchtet, welche Artefakte typisch für einige bekannte Entwicklungsansätze sind und welche Elemente generell als Anforderungsartefakte gelten. Eine formale Definition für Anforderungsartefakte befindet sich in Abschnitt 6.1.

In agilen Ansätzen werden Anforderungen auf möglichst simple Art dokumentiert. Es wird Wert darauf gelegt, Details erst kurz vor der Entwicklung der entsprechenden Funktionalität festzulegen und sie dann auch eher mündlich zu besprechen bzw. als Akzeptanztests festzuhalten anstatt sie separat als Anforderungen zu dokumentieren.

Ein zentraler Typ von Anforderungsartefakten in der agilen Entwicklung sind **User Stories**. Diese beschreiben auf grobe, benutzerorientierte Art, was der Nutzer mit dem System tun können möchte. Die zu einer Story zugehörigen **Akzeptanzkriterien** werden oft ebenfalls als Anforderungsartefakte angesehen, da sie Anforderungsdetails in formalisierter Form enthalten [8]. Neben User Stories können auch größere Elemente, wie **Features**, **Epics** oder **Visionsstatements** erstellt werden [94], [125]. Diese enthalten allgemeinere Ziele und werden im Verlauf eines Softwareprojektes zu User Stories konkretisiert.

Die Granularität der Artefakte beeinflusst, welche Aspekte gut kommuniziert werden können. Feingranulare Anforderungsartefakte auf Ebene einzelner Nutzerinteraktionen sind wichtig, um Details zu klären, während jedoch auch grobgranulare Artefakte auf Ebene der Nutzerziele benötigt werden, um Stakeholder in die Lage zu versetzen, kritisch abzuwägen, ob eine Anforderung tatsächlich hilft, die Nutzerziele zu erreichen [101], [103].

Häufig liegt der Fokus der agilen Anforderungsartefakte auf funktionalen Anforderungen, da die Systeminteraktionen in den User Stories genau beschreiben, *was* der Nutzer mit dem System tun können möchte. Nichtfunktionale Anforderungen können aber ebenfalls festgehalten werden [94]. Hierbei ist zu berücksichtigen, dass nichtfunktionale Anforderungen in unterschiedlichen Beziehungen zum System bzw. zu den funktionalen Anforderungen stehen können, so dass auch unterschiedliche Formen der Repräsentation sinnvoll sind [57]. Nichtfunktionale Anforderungen können das gesamte System betreffen und dann separat dokumentiert oder in die Definition of Done der User Stories aufgenommen werden [94]. Beziehen sie sich eher auf einzelne User Stories bzw. Features, so können sie an diese angehängt werden. Wenn sie nur einige konkrete Funktionen des Systems betreffen und in Verbindung mit diesen implementiert werden können, können sie auf speziellen User Stories und Akzeptanztests festgehalten werden [94].

In traditionellen, also plangetriebenen, Entwicklungsansätzen ist von vornherein eine größere Fülle an Blickwinkeln und dokumentierten Arten von Anforderungen vorgesehen. Hier gibt es Entwicklungsansätze, wie das V-Modell [134] und den Rational Unified Process [87], die viele einzelne Dokumente für die einzelnen Informationsarten vorsehen. Dabei wird insbesondere auch darauf eingegangen, wie Informationen aus einem Artefakt in einem weiteren Artefakt weiter konkretisiert werden. Alternativ existieren Templates für Anforderungsspezifikationen, wie das Volere-Template [137] oder das *Software Requirements Specification*-Template im IEEE Standard 29148:2011 ([73], Kapitel 9.5), welche die unterschiedlichen Blickwinkel in verschiedenen Kapiteln nur eines Dokumentes zeigen.

Als Anforderungsartefakte werden hier einzelne Elemente innerhalb dieser Dokumente oder Dokumentkapitel verstanden. Da im späteren Verlauf der Arbeit Abhängigkeiten zwischen einzelnen Anforderungen betrachtet und dokumentiert werden sollen, ist es sinnvoll, von Artefakten auf möglichst feiner Ebene zu sprechen. Wenn beispielsweise ein einzelner Use Case, anstatt beispielsweise der ganzen Spezifikation, als Anforderungsartefakt gilt, können auch einzelne Use Cases explizit verknüpft werden.

Zu den wichtigsten Blickwinkeln, die alle diese Ansätze abdecken, gehören textuelle **funktionale Anforderungen**, die nutzerorientiert in Form von *Use Cases* [29], [87] oder als **einzelne systemorientierte Anforderungen**, in der Form „Das System muss/soll <funktionale Anforderung>“ – bzw. in ähnlicher Form mithilfe von Textschablonen [141] – beschrieben werden. Weiterhin vertreten sind **nichtfunktionale Anforderungen**, die ebenfalls textuell ausgedrückt werden. Auch hier müssen, wie schon bei den agilen Artefakten beschrieben, bei der Repräsentation der nichtfunktionalen Anforderungen ihre Beziehungen zum System bzw. zu den anderen Anforderungen berücksichtigt werden. Während es Sinn macht, Anforderungen, die sich auf das gesamte System beziehen,

in einem separaten Spezifikationskapitel festzuhalten, sollten Anforderungen, die konkrete funktionale Anforderungen – wie einzelne Use Cases – betreffen, innerhalb dieser repräsentiert oder mit diesen verlinkt werden [57]. Funktionale und nichtfunktionale Anforderungen können in verschiedene Ebenen unterteilt werden, welche dann einzelne, sich konkretisierende, Anforderungsartefakte besitzen (z.B. Kontextspezifikation, Anforderungsspezifikation und Systemspezifikation aus [113] oder Businessanforderungen, Nutzeranforderungen und Systemanforderungen aus [151]). Weitere wichtige Blickwinkel werden durch **Datenmodelle**, **Oberflächenmodelle** sowie **Geschäftsprozessmodelle** und deren **Prozessaktivitäten** dargestellt. Mit Prozessmodellen im Sinne von Anforderungsartefakten sind in dieser Arbeit stets Modelle von fachlichen Prozessen, wie Geschäftsprozessen oder beispielsweise Berechnungsprozessen gemeint. Das sind die Prozesse, die durch die Software unterstützt werden sollen. Eine Prozessaktivität stellt dann eine Aktivität innerhalb eines solchen Geschäftsprozesses dar. Sie kann nur durch ein entsprechendes grafisches Element – wie eine BPMN-Aktivität – repräsentiert sein, aber auch über eine zusätzliche Beschreibung verfügen.

Neben funktionalen und nichtfunktionalen Anforderungen sind Projekt- und Prozessanforderungen von Bedeutung, die oftmals ebenfalls durch textuelle Anforderungen oder Softwareentwicklungs-Prozessmodelle ausgedrückt werden. Weitere textuelle Anforderungsartefakte sind Beschreibungen der **Stakeholder** und **Stakeholderziele** sowie des **Projektkontexts** [141].

2.4 Tracing von Anforderungen

Als **Requirements Traceability** wird die Fähigkeit bezeichnet, Wege von Anforderungsartefakten zu zusammenhängenden Artefakten zu verfolgen. Eine gängige Definition ist die von Gotel und Finkelstein:

"Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins through its development and specification to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)." [61]

Um Requirements Traceability zu erlangen, werden zusammenhängende Artefakte über sogenannte **Traceability Links** [132] miteinander verbunden, sodass später über die Links eine Navigation von einem Artefakt zu allen zusammenhängenden Artefakten möglich wird. Traceability Links werden in dieser Arbeit als **Verknüpfungen** bezeichnet. Verknüpfungen spielen in dieser Arbeit eine wichtige Rolle, da sie für den parallelen Umgang mit mehreren Artefakten benötigt werden.

Traceability Links können auf unterschiedliche Weise repräsentiert werden ([160] [127]): Sie können in Tracing-Matrizen festgehalten werden, in denen

verknüpfbare Artefakte in den Reihen und Spalten aufgeführt sind, sodass ein Eintrag in der Matrix einen Link darstellt. Verknüpfungen können in ER-Diagrammen dargestellt werden, in denen die Artefakte als Entities und die Verknüpfungen als Relations-Objekte dargestellt werden. Außerdem können Links als Querverweise in Texten bzw. in Artefakten dargestellt werden, bei denen ein Ziel-Element durch Angabe seiner ID oder eines expliziten Links referenziert wird. Ebenfalls gängig ist die Darstellung als Graphen, wie beispielsweise als UML-Objektmodelle, bei denen Artefakte als Knoten und Verknüpfungen als Kanten visualisiert werden.

In dieser Arbeit werden Verknüpfungen als Querverweise in Artefakten dargestellt. Zur Veranschaulichung von Verknüpfungen wird teilweise auch die graphenbasierte Notation verwendet, wie zum Beispiel im unteren Teil von Abbildung 2 angegeben.

2.4.1 Traceability-Modelle

Traceability-Modelle legen auf Typebene fest, welche Arten von Artefakten miteinander verknüpft werden dürfen bzw. sollen. Da die Erstellung von Verknüpfungen aufwändig ist, sollten nur solche Verknüpfungen erzeugt werden, die später auch verwendet werden. Im Traceability-Modell wird dazu ein struktureller Überblick erzeugt, der die Planung der gewünschten Verknüpfungen erleichtert. Abbildung 2 zeigt oben ein Beispiel für ein Traceability-Modell. Darunter sind konkrete Artefakte und Verknüpfungen angegeben, die diesem Modell folgen.

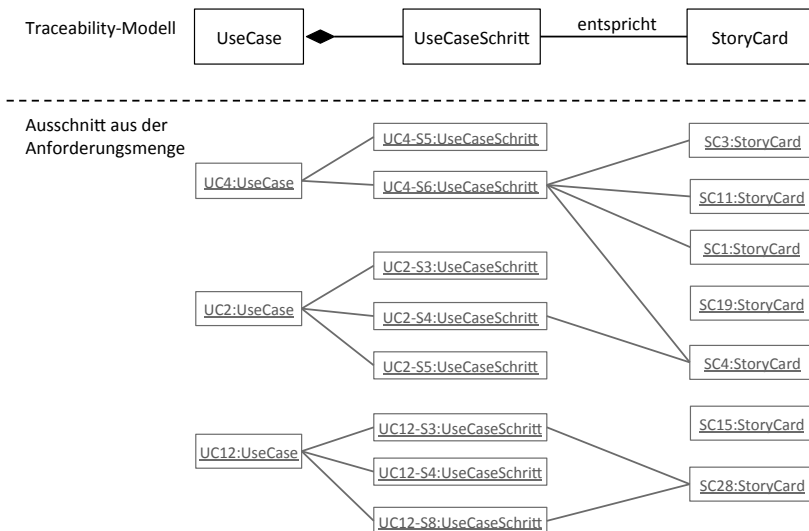


Abbildung 2: Traceability-Modell und zugehörige konkrete Anforderungen und Verknüpfungen

Traceability-Modelle dienen der Planung und Steuerung der Traceability-Struktur in einem Projekt. Sie stellen eine Übersicht dar, die zeigt, welche Abhängigkeiten die Projektteilnehmer festlegen müssen bzw. welche verknüpften Elemente sie zu einem Artefakt erwarten können [110]. So helfen sie dabei, einheitliche und vollständige Verknüpfungen zu erzeugen, was für einen effektiven Einsatz von Traceability wichtig ist [110]. Außerdem sind Traceability-Modelle eine Voraussetzung für die automatisierte Handhabung und Analyse von Verknüpfungen [110], [127]. Die in dieser Arbeit verwendeten Traceability-Modelle werden *Anforderungslandschaften* genannt.

2.4.2 Typen von Traceability Links

Ramesh und Jarke beschreiben in [132] (Seite 3), dass die Effizienz und Effektivität der Unterstützung durch Traceability auch davon abhängt, welche Artefakt- und Verknüpfungstypen angeboten werden, um Inhalte festzuhalten. Die angebotenen Typen sollten den Erfordernissen der Anforderungsdokumentation im konkreten Projekt entsprechen. Da es nicht praktikabel ist, alle möglichen Beziehungen festzuhalten, sollte hier im Vorfeld definiert werden, welche Tracing-Beziehungen im späteren Verlauf nützlich sind [108].

Verknüpfungen können verschiedene *Relationen*² zwischen Anforderungsartefakten dokumentieren. Es gibt viele Relationstypen, wie ‚*verfeinert*‘ und ‚*ist abhängig von*‘, die Abhängigkeitsbeziehungen beschreiben. In der Literatur sind mehrere Ansätze zu finden, die diese Relationstypen beispielsweise in Fallstudien, Interviews oder durch Zusammenfassen weiterer Literatur ermitteln und kategorisieren [35], [46], [111], [128], [132], [153].

Die Relationstypen lassen sich dabei nach verschiedenen Dimensionen kategorisieren: Dahlstedt und Persson [35] unterscheiden zwischen *strukturellen, einschränkenden und wert-/kostenbezogenen Relationstypen*. Während strukturelle Relationen zeigen, ob Artefakte beispielweise hierarchisch aufgebaut sind oder zueinander ähnlich sind, geben einschränkende Relationen an, ob Anforderungen inhaltlich voneinander abhängen oder sich widersprechen. Abbildung 3 zeigt die Klassifikation, wie in [35] angegeben.

Méndez [111] baut auf dieser Klassifikation auf, unterscheidet jedoch zwischen *syntaktischen und semantischen Relationstypen*. Syntaktische Relationstypen werden verwendet, um anzugeben, wie sich Artefakte prinzipiell aufeinander

² Auf den Unterschied zwischen den Begriffen Verknüpfung und Relation wird in Abschnitt 6.2.1 genauer eingegangen. Grundsätzlich lässt sich eine Verknüpfung als ein explizites Objekt auffassen, das eine Relation zwischen zwei Elementen dokumentiert. Eine Relation zwischen zwei Elementen, zum Beispiel dass ein Element das andere verfeinert, besteht dabei unabhängig davon, ob diese durch eine Verknüpfung explizit verlinkt sind.

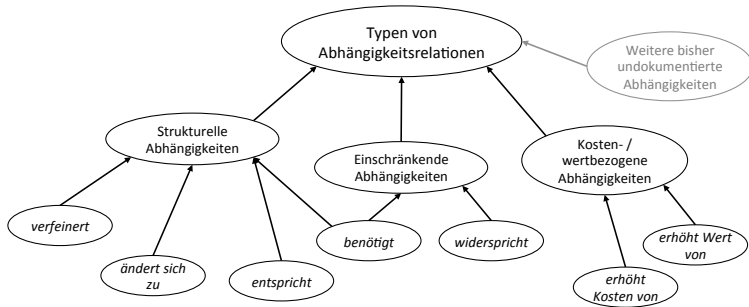


Abbildung 3: Klassifikation von Relationen zwischen Anforderungen (nach Dahlstedt und Persson [35])

beziehen sollen. Semantische Relationstypen können dann innerhalb syntaktischer Relationen genauer angeben, ob die Inhalte tatsächlich zusammenpassen oder sich widersprechen. Zusätzlich führt Méndez konkrete Abstraktionsebenen ein und unterscheidet zwischen Relationen, die zwischen mehreren (Inter-Abstraction) oder innerhalb einer Abstraktionsebene (Intra-Abstraction) verlaufen.

Ramesh und Jarke [132] trennen zwischen *produkt- und prozessbezogenen Relationstypen*. Produktbezogene Relationen zeigen, welche inhaltlichen (also produktbezogenen) Abhängigkeiten zwischen Projektartefakten (die nicht nur Anforderungsartefakte sind) herrschen. Sie zeigen auch, wie mehrere konkretere Artefakte ein übergeordnetes Artefakt erfüllen können. Prozessbezogene Relationen hingegen treten auf, wenn sich Anforderungen im Prozess ändern. Sie entstehen, wenn sich ein Anforderungsartefakt in ein weiteres weiterentwickelt und werden ebenfalls eingesetzt, um die damit verbundene Begründung zu dokumentieren.

Dahlstedt und Persson [35] stellen heraus, dass es schwierig ist, ein für alle gültiges und geeignetes Referenzmodell von Relationstypen und deren Beziehungen zu erstellen, weil die Relationstypen sich teilweise überlappen und nicht klar voneinander abgrenzen lassen. In Abschnitt 6.2.2 wird herausgearbeitet, welche Relationstypen in dieser Arbeit relevant sind. Es wird die oben angegebene Klassifikation von Dahlstedt und Persson als Basis verwendet, da diese die benötigten Typen am besten unterstützt.

3 Verwandte Arbeiten

Es gibt zahlreiche Ansätze in der Wissenschaft, die den besseren Umgang mit Anforderungsartefakten unterstützen und das Zusammenspiel unterschiedlicher Anforderungsartefakte verbessern. Dieser Abschnitt stellt einen Überblick über solche Ansätze her und stellt den Bezug zur vorliegenden Dissertation heraus.

3.1 Anforderungsartefakte

Anforderungsartefakte spielen eine wichtige Rolle im Requirements Engineering. Sie dienen als Bezugsquelle in der Kommunikation über Anforderungen und helfen so, ein gemeinsames Verständnis herzustellen, Widersprüche aufzudecken und Details zu veranschaulichen. Es gibt viele Untersuchungen dazu, wie Anforderungsartefakte bei der Softwareentwicklung helfen und wie sie beschaffen sein sollten, um in verschiedenen Aktivitäten hilfreich zu sein.

Zunächst lassen sich hier Standards und Templates nennen, wie der ISO/IEEE-Standard 29148 (Life Cycle Processes – Requirements Engineering) [73] und das Volere-Template [137], die Elemente nennen, welche in vielen Softwareprojekten hilfreich waren. Méndez [111] beschreibt den Ansatz des artefaktbasierten Requirements Engineering als Gegensatz zum prozess- bzw. aktivitätsbasierten Requirements Engineering. Hierbei werden statt der Prozesse und Aktivitäten die geforderten Ergebnisse des RE-Prozesses festgelegt und eher offen gelassen, wie genau die Projektteilnehmer zu diesen Ergebnissen kommen sollen.

Er stellt konkrete Referenzmodelle von Anforderungsartefakten für verschiedene Domänen, wie Business Information Systems [112] und Cyber-Physical Systems [126] auf und kombiniert diese auch zu einem domänenunabhängigen Ansatz (AMDiRE) [113]. Dazu zeigt er, wie Referenzmodelle allgemein anhand eines Metamodells erstellt werden können und wie Referenzmodelle mittels Tailoring für einzelne Projekte angepasst werden [111]. Er stellt fest, dass der Ansatz ausreichend Flexibilität schafft, um auf einzelne Projekte anwendbar zu sein und dass die Fokussierung des Prozesses auf die geforderten Artefakte dazu führt, dass syntaktisch bessere und konsistentere Artefakte erzeugt werden [112].

Gross und Dörr [62] haben untersucht, welche Elemente von Anforderungsspezifikationen wie wichtig für verschiedene Leser sind. Ihre Grundidee ist es, verschiedene Sichten auf eine Spezifikation anzubieten, die für verschiedene Rollen, wie Architekten oder UI-Designer, jeweils unterschiedliche Inhalte hervorhebt. Sie haben herausgefunden, dass sowohl die Inhalte für verschiedene Rollen unterschiedlich wichtig sind als auch, dass unterschiedliche Repräsentationen bevorzugt werden. Außerdem haben sie festgestellt, dass die UI-Designer relevante Zusatz-Informationen wie Datenmodelle lieber direkt in Interaktions-

beschreibungen lesen wollten, als in separat aufgeführten Informationen. Dies zeigt, dass der passende Aufbau von Anforderungsartefakten in einem konkreten Projekt auch davon abhängt, welche Rollen und welche Personen daran beteiligt sind.

Femmer et al. [47] schlagen ein kontextspezifisches Qualitätsmodell für Anforderungen vor. Die Autoren sind der Ansicht, dass die Anforderungsartefakte durch ihre Beschaffenheit eine Reihe von Aktivitäten positiv oder negativ beeinflussen. Sie sagen, dass sich die einzelnen Aktivitäten, die mithilfe von Anforderungen durchgeführt werden, so sehr von Projekt zu Projekt unterscheiden, dass sich nicht pauschal festlegen lässt, wie ein Artefakt beschaffen sein sollte. Daher schlagen die Autoren ein Qualitätsmodell für Anforderungsartefakte vor, das die vorgesehenen Projektaktivitäten einbezieht. Eine erste Evaluation ergibt, dass dieses Modell zu wertvollen Analysen der Richtlinien an Anforderungsartefakte führt.

Bezug zu dieser Arbeit: Auch in dieser Arbeit wird eine Studie durchgeführt, die untersucht, wie Anforderungsartefakte in der Praxis verwendet werden. Damit erweitert diese Arbeit den Wissenstand auf diesem Gebiet. Insbesondere stellt sich in der Studie heraus, dass besonders die Abhängigkeiten zwischen verschiedenen Arten von Anforderungsartefakten zu Herausforderungen führen, die Projektteilnehmer in ihren Aktivitäten behindern. Auf dieser Basis entwickelt diese Arbeit neue Sichtweisen auf Anforderungsartefakte, deren Zusammenspiel und auf anforderungsbezogene Aktivitäten, die zu neuen konzeptionellen Lösungen führen.

3.2 Verwendung mehrerer Artefakttypen

Zunächst lässt sich feststellen, dass in vielen Projekten mehrere Arten von Anforderungsartefakten verwendet werden. Praktisch alle traditionellen Vorgehensmodelle und Templates, wie das V-Modell [134], der RUP [87], das Vole-re-Spezifikationstemplate [137] und der ISO/IEEE-Standard 29148:2011 zu Softwarespezifikationen [73] beinhalten eine Vielzahl an Anforderungsartefakten, die bei der Dokumentation abgedeckt werden sollten. Auch in der agilen Entwicklung, die scheinbar hauptsächlich nur auf User Stories beruht, werden weitere Anforderungsartefakte als sinnvoll erachtet. Zusätzliche Artefakte können (i) Features und Epics sein, die eine gröbere Granularität erlauben [8], [94], [125], (ii) zusätzliche grafische Modelle, wie Daten- oder Geschäftsprozessmodelle, die einen guten Überblick bieten [5], [8], bis hin zu (iii) zusätzlichen Spezifikationsdokumenten, die für die Abstimmung in besonders großen Teams hilfreich sind [27].

Viele Autoren beschreiben, dass es wichtig ist, Anforderungen aus unterschiedlichen Blickwinkeln darzustellen. Rupp schreibt, dass ein Softwareprojekt sowohl *textuelle* als auch *grafische* Anforderungsrepräsentationen verwenden

sollte, um ein umfassendes Verständnis der zu entwickelnden Software herzustellen [141]. Finkelstein, Sommerville, Nuseibeh und weitere haben sich mit dem Konzept von *Viewpoints* beschäftigt, welches die Sichten bzw. Sichtweisen der unterschiedlichen Projektteilnehmer auf die zu erstellende Software darstellt [120], [49]. Viewpoints-basierte Methoden rücken die unterschiedlichen Sichten in den Vordergrund, erlauben es, diese zu sammeln und zu organisieren [48], [86] sowie die Beziehungen zwischen den Sichten zu berücksichtigen [120]. Die zu erstellende Software wird dann basierend auf den identifizierten Viewpoints beschrieben.

Ein dritter Aspekt, der für die Verwendung mehrerer Anforderungsrepräsentationen spricht, ist die Verwendung mehrerer *Granularitäts- bzw. Abstraktionsebenen*. Rupp [141] beschreibt, wie wichtig Artefakte auf verschiedenen Abstraktionsebenen sind. Sie stellt fünf Spezifikationsebenen auf, die von Systemüberblick über grobe Anwendungsfälle, Nutzeranforderungen und technische Anforderungen, bis hin zu Komponentenanforderungen oder Interface-Design-Beschreibungen reichen, und zeigt, welche agilen oder traditionellen Artefakte die jeweiligen Ebenen ausfüllen können. Pohl und Sikora [130] unterteilen in ihrem Ansatz COSMOD-RE Anforderungen in die Abstraktionsebenen System, Funktionen, Hardware-/Softwarekomponenten und Software-Deployment. Auf jeder Abstraktionsebene beschreiben sie Anforderungen, Ziele, Szenarien, und Architekturaspekte um alle wichtigen Blickwinkel abzudecken. Bühne et al. [21] stellen ähnlich ein Projekt aus dem Automotive-Bereich vor, in dem die Anforderungen auf Fahrzeug-, System-, Funktions- und dann auf Softwareebene beschrieben werden. Van Lamsweerde [92] stellt mit *Goal-Oriented Requirements Engineering* einen Ansatz vor, bei dem Beschreibungen von Zielen die Basis für Anforderungen darstellen und stellt dabei auch heraus, dass auch die Unterteilung von Zielen auf verschiedene Abstraktionsebenen wichtig ist.

In der Literatur werden viele konkrete Kombinationen vorgestellt, bei denen verschiedene Artefakte gemeinsam verwendet und einander zugeordnet werden.

Antonino et al. [6] stellen eine Methode zur leichtgewichtigen Verknüpfung von Anforderungen und Entwicklungsartefakten vor. Diese enthält auch die Erzeugung von User Stories und zusätzlichen Spezifikationen, die beispielsweise Anforderungen aus Normen oder Vorschriften enthalten. Sie stellen eine Vorgehensweise und ein Werkzeug vor, die helfen, beide Arten von Anforderungen in Enterprise Architect-Elemente zu verwandeln, sodass diese in derselben Umgebung erreichbar sind und verknüpft werden können.

Imaz und Benyon [75] betrachten die Unterschiede zwischen User Stories und Use Cases. Für sie sind User Stories informeller als Use Cases und adressieren ein konkretes Beispiel, während Use Cases eine allgemeine Interaktion be-

schreiben. Die Autoren sagen, dass beide Artefaktarten wichtig sind. Sie schlagen vor, beides im Projekt zu verwenden und User Stories als Basis für die Anforderungen in den Use Cases aufzufassen und sie entsprechend der Pre-Requirements-Traceability mit Use Cases zu verknüpfen.

Gallardo-Valencia et al. [53] haben in einem kontrollierten Experiment untersucht, ob Softwareentwickler durch das Hinzufügen von Use Cases zu agilen Artefakten, konkret User Stories, Vorteile hatten. Use Cases wurden hier vor allem als detaillierter im Vergleich zu den User Stories charakterisiert. Sie haben herausgefunden, dass Softwareentwickler, die Use Cases verwendet haben, weniger Zeit benötigt haben, um gegebene Requirements zu verstehen und dann auch bessere – d.h. weniger, dafür relevantere – Fragen an den On-Site Customer stellen konnten.

Weitere Veröffentlichungen beschäftigen sich mit der Zuordnung von Anforderungen zu allgemeineren bzw. abstrakteren Elementen, die eine Basis für die Anforderungen darstellen. Abelein und Paech [1] beschreiben die Zuordnung von Design-Entscheidungen zu Anforderungen und zeigen, wie dies besonders in großen Projekten hilft, das gemeinsame Verständnis zwischen Nutzern und Entwicklern zu stärken. Rashid et al. [133] verwenden zusätzlich zu Anforderungen sogenannte *Aspekte*, die übergeordnete querschnittliche Elemente darstellen. Die Verwendung von Aspekten macht es beispielsweise möglich, Konflikte von querschnittlichen Aspekten zu entdecken und bereits früh im Projekt passende Abwägungen zu treffen. Auch im Goal-Oriented Requirements Engineering, wie von van Lamsweerde [92] vorgestellt, werden Ziele und Anforderungen einander zugeordnet, um so eine Basis für die Anforderungen darzustellen. Creighton et al. [33] verwenden Sequenzen von Videoclips, um Anforderungen nutzerverständlich zu visualisieren und diese zu formalen Elementen wie Use Cases oder Sequenzdiagramme zuzuordnen. Sie beschreiben, wie diese Technik dabei hilft, die Kommunikation von Nutzern und Entwicklern zu verbessern und eine Reihe von Herausforderungen im Requirements Engineering zu lösen.

Bezug zu dieser Arbeit: Auch in dieser Arbeit wird die parallele Verwendung mehrerer Artefakte betrachtet. Im Gegensatz zu den hier erwähnten Arbeiten, werden dabei aber grundsätzlich keine konkreten Artefakttypen vorgegeben. Nur in Kapitel 6.7 (Typische hybride Landschaften) wird auf konkrete Konstellationen eingegangen, um sinnvolle hybride Anforderungslandschaften zu beleuchten. Die Konzepte sind aber sonst auf allgemeine Anforderungslandschaften ausgelegt, die aus beliebigen Artefakttypen bestehen. Ein Fokus liegt auf hybriden Landschaften, bei denen agile und traditionelle Anforderungsartefakte gemeinsam verwendet werden. Dabei wird zusätzlich beleuchtet, welche Aspekte diese parallele Verwendung besonders schwierig machen können. Entspre-

chend werden neue Konzepte, wie Zähheit von Artefakten und verschiedene Abhängigkeiten bzw. Relationen definiert.

Einige der Autoren schlagen vor, Artefakte unterschiedlicher Typen durch Traceability Links zu verknüpfen. Auch dies wird in dieser Arbeit im Konzept der Anforderungslandschaften aufgegriffen. Zusätzlich ermöglicht aber das Konzept der Trace-Operationen, das Zusammenwirken unterschiedlicher Artefakte noch stärker zu unterstützen. Dazu werden die Verknüpfungen durch Trace-Operationen bereits teil-automatisiert verwendet und können den Nutzern so bereits früh relevante Informationen über abhängige Elemente präsentieren.

3.3 Artefaktmodelle

Artefaktmodelle – auch als Traceability-Modelle bezeichnet – werden zur Planung und Diskussion der Artefaktstruktur in einem Projekt eingesetzt. Sie können aber auch ein Basis-Modell für automatisierte Operationen wie der Durchführung von Anfragen an die Instanzmenge oder der Konsistenzsicherung dienen. Artefakt- und Traceability-Modelle beziehen sich dabei in der Regel auf alle Artefakte, die in einem Projekt verwendet werden. Dazu gehören auch Design-Artefakte oder Code. Anforderungslandschaftsmodelle, die in dieser Arbeit verwendet werden, sind dabei auf solche Artefakte eingeschränkt, die Informationen zu Anforderungen enthalten. Die grundsätzlichen Einsatzzwecke und Herausforderungen sind dabei aber dieselben wie bei allgemeinen Artefaktmodellen.

Konrad und Degen [85] haben in vier Fallstudien untersucht, wie Artefaktmodelle in der Industrie verwendet werden und dazu Erkenntnisse zum effektiven Einsatz der Modelle aufgestellt. Sie stellen fest, dass es wichtig ist, dass die Repräsentation der Modelle zur Zielgruppe passt, da die Modelle sonst nicht beachtet werden. Sie berichten außerdem, dass es ein hoher Aufwand ist, detaillierte Artefaktmodelle aufzustellen und dass es sich lohnt, dazu existierende Frameworks zu benutzen und bestehende Modelle als Basis wiederzuverwenden. Ein gut eingesetztes Artefaktmodell kann dann dabei helfen, ein konsistentes Verständnis über die verwendeten Artefakttypen und vor allem deren Zwecke bei allen Projektteilnehmern herzustellen.

Mäder et al. [109], [110] stellen *Traceability Information Models* vor, um Informationen über die verwendeten Artefakte und vor allem über die gewünschten Verknüpfungen festzuhalten. Diese Informationen sollen genutzt werden, um einheitliche und vollständige Tracing Links zu bewirken. Traceability Information Modelle werden auch zur automatisierten Unterstützung der Traceability verwendet [107]–[109]. Hierauf wird im folgenden Abschnitt 3.4 näher eingegangen. Traceability Information Models sind sehr simpel gehalten, um auch in der Praxis eine pragmatische Art zur Definition von Tracing-Modellen

dazustellen. Die Modelle sind an UML-Modellen orientiert, welche die Artefakttypen als Klassen und deren erlaubte Traces als Assoziationen enthalten. Eine Besonderheit der Modelle ist die Unterscheidung zwischen Projektartefakten und den eigentlichen Artefakten, die dafür im Tool bereitstehen. Beispielsweise kann ein Tool-Artefakt *Use Case* in einem konkreten Projekt für die beiden Projektartefakte *Business Use Case* und *technischer Use Case* verwendet werden. Das Mapping von Projektartefakten auf Tool-Artefakte ermöglicht es in der Praxis, besser auf die Tool-Eigenschaften einzugehen und eine konkrete Tool-Unterstützung für gewünschte Traces bereitzustellen. Die Anforderungslandschaftsmodelle in dieser Arbeit enthalten nur Projektartefakte, da sie auch Tool-unabhängig eingesetzt werden sollen. Sie sind jedoch prinzipiell auf explizite Projekt- und Tool-Artefakte erweiterbar.

Méndez [111], [113] beleuchtet artefaktbasiertes Requirements Engineering und zeigt, dass es hilfreich ist, in Projekten genau zu definieren, welche Artefakte verwendet werden sollen und wie sie miteinander zusammenhängen. Artefaktmodelle – dort als Referenzmodelle bezeichnet – sind dabei ein wichtiges Hilfsmittel, um die beabsichtigte Artefaktstruktur zu visualisieren. Méndez stellt solche Referenzmodelle auf und stellt auch ein Metamodell und eine Vorgehensweise für die Anpassung der Referenzmodelle in Projekten zur Verfügung.

Es gibt eine Reihe von Formalisierungen für Traceability-Modelle, die beispielsweise mithilfe von Metamodellen genau definieren, welche Attribute die Elemente eines Traceability-Modells besitzen sollten, festlegen, welche Modellierungssprache verwendet wird und zeigen, wie einzelne Modelle erweitert werden können.

Adersberger [2] stellt mit *TraceML* eine Modellierungssprache vor, die an die UML angelehnt ist und vor allem auf einer unifizierten, semantisch fundierten und erweiterbaren Beschreibung von Traceability Links beruht. Adersberger hat bestehende Ansätze untersucht und fasst diese in seinem Ansatz zusammen. Zu UML-basierten Modellierungsansätzen zählen die Arbeiten von Espinoza [96], Letelier [95], Walderhaug et al. [158] und Schwarz et al. [146]. Auch Drivalos et al. [42] stellen ein Metamodell auf, verwenden dann zur Modellierung aber eine textuelle domänenspezifische Sprache.

Bezug zu dieser Arbeit: Auch in dieser Arbeit werden Artefaktmodelle formalisiert. Die Notation lehnt sich an UML-Klassendiagramme an, erweitert die Darstellung aber in einigen Punkten. Diese Arbeit grenzt sich von den vorgestellten Ansätzen dadurch ab, dass nicht versucht wird, in den Modellen alle möglichen auftretenden Traceability-Informationen darzustellen, sondern ein möglichst simples Modell zu erhalten, das speziell für die Planung der Koexistenz von Anforderungsartefakten eingesetzt werden kann. Bei der Erstellung des Anforderungslandschaftskonzeptes sind daher vor allem die Erfahrungen von

Projektteilnehmern bei der parallelen Verwendung mehrerer Anforderungsarten in der Praxis maßgebend. Entsprechend werden spezielle Aspekte, wie zähe und zusammengesetzte Artefakte, sowie Abhängigkeitsrelationen, die nicht durch Traces abgedeckt sind, in der Notation hervorgehoben.

Zusätzlich zur bloßen Notation stellt diese Arbeit außerdem Ansätze zur Anwendung von Anforderungslandschaften bereit. Damit wird Requirements Engineers eine Hilfestellung zur Erstellung von *inhaltlich geeigneten* Modellen gegeben. Dazu werden in Kapitel 4 Herausforderungen beim Umgang mit mehreren Artefakten herausgestellt, in den Kapiteln 6.7 und 6.8 konkrete sinnvolle Konstellationen und beachtenswerte Eigenschaften von hybriden Anforderungslandschaften erarbeitet und in Kapitel 7 eine Methode zur Planung und Optimierung von Anforderungslandschaften angegeben.

3.4 Automatisierte Operationen auf Traceability Links

3.4.1 Automatisierte Verwendung von Traceability Links

In Anforderungslandschaften, in denen Artefakte durch Tracing Links verknüpft sind, können nun die Links dazu verwendet werden, automatisierte Unterstützung für den Umgang mit den Artefakten bereitzustellen. Innerhalb der Tracing-Literatur gibt es dazu viele unterschiedliche Ansätze.

Mäder und Cleland-Huang [108], [109] haben eine visuelle Notation für Trace Queries entwickelt (VTML), mit denen Nutzer Anfragen über Artefakte und deren Verknüpfungen stellen können. Die Trace Queries beruhen auf Traceability Information Models, die im vorigen Abschnitt 3.3 beleuchtet wurden. VTML-Queries sind UML-Klassendiagramme, die aus Teilen des Traceability Information Models bestehen und spezielle Stereotypen für Anfrageinformationen enthalten. Mithilfe von Trace Queries können Requirements Engineers flexibel und einfach beliebige Anfragen an die Anforderungsmenge stellen, die eine ähnliche Mächtigkeit wie SQL-Anfragen an eine entsprechende Datenbank besitzen. Die Autoren zeigen in einem Experiment, dass Trace Queries mit VTML schneller gelesen und erstellt werden konnten, als textuelle Trace Queries mit SQL [109].

Weitere visuelle Notationen für Trace Queries werden von Jaakkola und Thalheim [77] sowie von Störrle [156] vorgestellt. Maletic und Collard [80] und Schwarz et al. [146] stellen textbasierte Methoden zur Abfrage von Artefakten vor. Pruski et al. [131] entwickeln mit TiQi ein Werkzeug, das natürlichsprachliche Trace Queries auswerten kann.

Eine weitere automatisierte Form der Verwendung von Traceability Links ist die automatisierte *Konsistenzsicherung* bei Änderungen von Anforderungsartefakten oder Verknüpfungen. Zettel [165] stellt ein Konzept zur anpassbaren Konsistenzsicherung von Artefakten innerhalb von CASE-Tools vor. Durch

Konstrukte aus OCL-Ausdrücken und Funktionsaufrufen, die auf einem Artefaktmodell basieren, werden Konsistenzregeln und Berichtigungs-Mechanismen festgelegt. Das Werkzeug SPEARMINT, das im Zuge von Zettels Arbeit mit entwickelt wurde, implementiert 53 solcher Konsistenzregeln und Berichtigungen für Softwareentwicklungs-Prozessmodelle. Zettel zeigt weiterhin, dass die Anpassbarkeit der Methodenassistenz durch Konsistenzregeln die Akzeptanz von CASE-Tools beeinflusst.

Neben der Rekonstruktion von *Artefakten* nach Änderungen, ist es auch wichtig, die *Verknüpfungen* und ihre Gültigkeit nach Änderungen zu berücksichtigen. Mäder [107] hat sich mit der automatisierten Wartung von Verknüpfungen innerhalb von Anforderungs- Analyse- und Entwurfs-Modellen in UML beschäftigt. Er hat Regeln für unterschiedliche Muster und Änderungsszenarien aufgestellt und zu diesen festgelegt, wie Verknüpfungen angepasst werden müssen, um weiterhin gültig zu sein.

Eine weitere Anwendung der automatisierten Auswertung von Tracing-Strukturen ist die Analyse der Auswirkungen von Änderungen an Artefakten (Impact Analyse). Von Knehten und Grund [84] stellen einen Ansatz vor, bei dem zu einem Anforderungsartefakt, das geändert werden soll, weitere primär, sekundär und tertiär betroffene Artefakte automatisch ermittelt und angezeigt werden. Darauf basierend können die Projektplaner dann die Kosten schätzen. Dazu legen die Autoren auch fest, wie die Tracing-Struktur vorher aufgebaut werden muss, um sinnvolle Vorschläge für die Auswirkungs-Analyse zu erhalten.

Bezug zu dieser Arbeit: In dieser Arbeit werden Trace Queries basierend auf den VTML-Queries von Mäder und Cleland-Huang herangezogen. Aufgrund ihrer Einfachheit und ihres klaren Bezugs zu einem Basis-Artefaktmodell sind sie gut geeignet, um in das Konzept der Anforderungslandschaftsmodelle integriert zu werden. Zusätzlich werden auch textuelle Trace Queries, die mithilfe von OCL-Ausdrücken über dem Artefaktmodell formuliert werden, beleuchtet. Während beide Darstellungen die gleiche ausreichende Mächtigkeit zur Darstellung der notwendigen Trace Queries besitzen, unterscheiden Sie sich in ihrer Lesbarkeit.

Das in dieser Arbeit verwendete Konzept der Trace Operationen baut auf Trace Queries auf, erweitert diese aber um weitere Elemente, die es erlauben, Trace Queries automatisiert durchzuführen und die Ergebnisse auch automatisiert auszuwerten oder darzustellen. Hierzu werden in Trace Operationen zusätzlich Auslöser und Überprüfungen für die Trace Queries festgelegt. Durch die Erweiterung wird es möglich, die Trace Queries bzw. allgemein Trace-Aktivitäten in die eigentlichen anforderungsbezogenen Aktivitäten zu integrieren, bei denen es

beispielsweise in erster Linie darum geht, Anforderungen zu formulieren, anzupassen oder zu priorisieren.

Einige der Trace Operationen in dieser Arbeit enthalten automatisierte Überprüfungen der abgefragten Anforderungsmenge. Damit ähneln sie den Konsistenzregeln von Zettel, in denen ebenfalls automatisierte Überprüfungen durchgeführt werden. Jedoch weisen die Überprüfungen in dieser Arbeit auf potenzielle Inkonsistenzen hin, können aber nicht feststellen, ob es tatsächlich eine Inkonsistenz gibt und damit auch keine automatisierte Korrektur veranlassen. Die Anforderungsartefakte müssen inhaltlich überprüft werden, um festzustellen, ob sie nach einer Änderung ebenfalls angepasst werden müssen.

Damit ähnelt der Ansatz dem von von Knethen und Grund. Es werden potentiell relevante Artefakte präsentiert, aber die Auswertung muss manuell vorgenommen werden. Von Knethen und Grund stellen dazu einen Ansatz vor, der basierend auf Verknüpfungsarten immer nach demselben Prinzip primär, sekundär und tertiär beeinflusste Artefakte ermittelt. In dieser Arbeit wird ein Ansatz gewählt, bei dem für jede Trace Operation eine eigene Regel mit eigenen Verknüpfungen festgelegt wird. Auf diese Art kann für jedes Projekt individuell festgelegt werden, welche der Verknüpfungen für welche Art von Artefakten relevant sind.

3.4.2 Automatisierte Erstellung von Traceability Links

Neben der automatisierten Verwendung von Traceability Links ist auch die automatisierte Unterstützung bei der *Erstellung* von Traceability Links von Bedeutung, da sie die Kosten zum Aufbau einer nutzbaren Link-Struktur reduziert und damit die Effizienz eines Tracing-Ansatzes steigert.

Delater [36] präsentiert einen Prozess und einen Algorithmus zur semi-automatisierten Erstellung von Verknüpfungen zwischen Code und Anforderungen. Hierzu werden Arbeitsaufträge, die innerhalb der Versionsverwaltung mit Code assoziiert sind, mit Anforderungsartefakten verknüpft. Das System leitet daraufhin Verknüpfungen zwischen Anforderungen und Code-Elementen ab.

Auch Ben Charrada [24] hat die beiden konkreten Artefakte Anforderungsspezifikation und Code betrachtet und eine Methode entwickelt, mit der Code-Änderungen automatisiert zu entsprechenden Stellen in der Spezifikation zugeordnet werden können. Die Zuordnung zeigt Anforderungen, die mit besonders hoher Wahrscheinlichkeit von einer Code-Änderung betroffen sind. Basierend darauf erhält ein Entwickler eine Liste mit Stellen der Spezifikation, die überprüft und ggf. nachgepflegt werden sollten.

Weitere Ansätze befassen sich damit, allgemein Informationen in den Artefakten auszuwerten, um automatisiert Vorschläge für Verknüpfungen zwischen

diesen zu erstellen. Hier ist beispielsweise der Ansatz von Hayes et al. [65] zu nennen, bei dem Techniken des Information Retrieval eingesetzt werden, um ähnliche Artefakte zu bestimmen und zu verknüpfen. Ähnlich gehen Natt och Dag et al. [116] vor, welche die Kerninformationen von Artefakten mittels Natural Language Processing bestimmen und dann ebenfalls Information Retrieval Techniken einsetzen, um die Ähnlichkeit zu bestimmen.

Niklas et al. [119] wenden ähnliche Prinzipien an, um Anforderungs- und Entwurfs-Artefakte zu betrachten. Sie stellen Inkonsistenzen zwischen diesen fest, indem sie mittels Methoden des Natural Language Processing aus den textuellen Anforderungen ein Modell über die Konzepte und deren Beziehungen extrahieren und dieses mit dem Entwurfsmodell vergleichen. Potenziell abhängige Artefakte und deren Inkonsistenzen können so automatisiert aufgespürt und mit Korrekturvorschlägen versehen werden.

Bezug zu dieser Arbeit: Diese Arbeit ist auf die automatisierte Verwendung von Tracing Links fokussiert, die zuvor manuell erstellt worden sind. Die automatisierte Erstellung von Verknüpfungen wird nicht explizit aufgegriffen. Dennoch lässt sich der Ansatz um die genannten oder auch andere Methoden erweitern. Die Anforderungslandschaft gibt dazu an, welche Artefakttypen miteinander verknüpft werden sollen. Zusätzlich zu vorhandenen Verknüpfungen kann das System dann Vorschläge zu weiteren Verknüpfungen machen. In [162] wurde beispielsweise das im Zuge dieser Arbeit entwickelte RE-Werkzeug IRE prototypisch um zusätzliche Trace Operationen ergänzt, die mittels Natural Language Processing Vorschläge für Verknüpfungen zwischen den Artefakten Use Case-Schritt und User Story erzeugen.

Potenziell lassen sich auch die hier vorgestellten Konzepte *Trace Operationen* und *integrierte Sichten* zur Unterstützung der Erstellung von Traceability Links einsetzen. Durch integrierte Sichten können abhängige Elemente potenziell schneller entdeckt und auch leichter verknüpft werden. Trace Operationen können verwendet werden, um auf fehlende Verknüpfungen hinzuweisen und so die Nutzer dazu anzuregen, mehr Verknüpfungen zu erstellen. Das Konzept der Trace Operationen ließe sich außerdem auch um automatisierte Aktionen erweitern, die zur Erstellung oder Anpassung von Verknüpfungen verwendet werden.

Techniken zur Unterstützung der Erstellung von Verknüpfungen sind eine wichtige Ergänzung zum Konzept der Trace Operationen, da diese durch Reduktion des Aufwandes dafür, eine verknüpfte Landschaft aufzusetzen, die Effizienz des Konzeptes insgesamt erhöhen.

3.5 Wertbasierte Softwareentwicklungsmethoden

Wertbasierte Softwareentwicklungsmethoden beschäftigen sich damit, Entscheidungen oder Praktiken daran auszurichten, welchen Wert sie bringen. Hierzu stellen Grünbacher et al. [63] beispielsweise mit EasyWinWin einen Ansatz vor, der es erlaubt, mit Stakeholdern den Wert, den bestimmte Lösungen oder Vorgehensweisen für diese haben, zu erarbeiten. Wohlin und Aurum [163] zeigen Kriterien, die dabei helfen, Anforderungen wertbasiert für Releases zu priorisieren. Ein wichtiges Mittel für wertbasierte Ansätze ist die Bewertung von Vorgehensweisen über deren Risiken. Hierzu stellen Boehm und Turner risikobasierte Ansätze zur normalen Verwendung [12], aber auch zur Abwägung mehrerer Vorgehensweisen [17] vor.

Auch im Tracing werden wertbasierte Verfahren eingesetzt, beispielsweise von Egyed et al. [44], Heindl und Biffel [68] und Cleland-Huang et al. [26]. Dabei geht es darum, Traceability-Verknüpfungen nur für solche Anforderungsartefakte zu erzeugen, die am meisten davon profitieren. Beispielsweise können dies besonders hoch priorisierte User Stories sein.

Bezug zu dieser Arbeit: Um zu beurteilen, ob die Anforderungslandschaft, als Teil der Vorgehensweise, gut geeignet für ein Projekt ist, verwendet diese Arbeit ebenfalls wertbasierte bzw. genauer risikobasierte Verfahren. Der vorgestellte Ansatz lehnt sich an die Methode zur risikobasierten Bewertung von agilen und traditionellen Entwicklungsmethoden von Boehm und Turner [16] an. Die Arbeit erweitert diese Methode, indem sie neue Risiken erarbeitet, welche sich auf den Umgang mit Anforderungsartefakten beziehen. Die neuen Risiken sind feingranularer, weil sie sich auf einzelne Aspekte, wie die Anforderungsdokumentation oder Verknüpfungen beziehen. Damit erlauben sie aber auch eine feingranulare Abwägung, wie beispielsweise die Entscheidung, ob ein bestimmtes Anforderungsartefakt in die Anforderungslandschaft aufgenommen werden sollte.

4 Stand der Praxis: Interview-Studie zum Umgang mit Anforderungsartefakten

Um einen Einblick in den Stand der Praxis zu erlangen und die Herausforderungen im Umgang mit Anforderungslandschaften zu erfahren, wurde im Zuge dieser Arbeit eine Interview-Studie mit 21 Industrievertretern aus 6 Unternehmen durchgeführt. Das Ziel ist es, zu verstehen, wie Praktiker in verschiedenen Rollen Anforderungsartefakte nutzen, wie sie mit mehreren Artefakten umgehen und ob sie aktuelle Praktiken zur Verknüpfung von verwandten Artefakten nutzen. Die Ergebnisse dieser Studie wurden in Teilen in [97] veröffentlicht.

Die Studie weist darauf hin, dass oftmals eine Fülle an verschiedenen Artefakten benötigt wird, um ein Projekt erfolgreich durchzuführen. Gleichzeitig verursacht der Einsatz mehrerer Artefakttypen Herausforderungen, wie manuellen Übersetzungsaufwand und Inkonsistenzen. Es werden Zuordnungsmechanismen benötigt, die abhängige Artefakte explizit verknüpfen. Jedoch werden existierende Ansätze, beispielsweise aus dem Tracing, in der Praxis oftmals nicht verwendet. Daher wird in der Studie auch untersucht, warum existierende Methoden nicht verwendet werden.

4.1 Einleitung

Anforderungsartefakte, wie Spezifikationen, Diagramme oder User Stories unterstützen verschiedenste anforderungsbezogene Aktivitäten, wie die Klärung von Details, die Priorisierung und Validierung von Anforderungen sowie das Einarbeiten von Anforderungsänderungen. Wie gut dabei ein Artefakt eine Aktivität unterstützen kann, hängt von der Art des Artefaktes ab. Verschiedene Artefakte stellen bestimmte Aspekte der Anforderungsmenge in den Vordergrund und verstecken andere. Damit beeinflussen sie beispielsweise, welche Information konkretisiert wird oder wie gut Abhängigkeiten sichtbar werden. Nicht alle Artefakttypen sind somit gleichermaßen gut geeignet, um eine bestimmte Aktivität zu unterstützen. Beispielsweise eignet sich ein Spezifikationsdokument gut dazu, Informationen zum Projektkontext bereitzustellen, ist jedoch umständlich, um häufige Änderungen an Anforderungen zu pflegen.

Anforderungsartefakte werden von vielen Projektteilnehmern mit verschiedenen Rollen verwendet, die alle unterschiedliche Aufgaben und damit unterschiedliche Erfordernisse an die Artefakte haben. Oft existiert nicht der eine perfekte Artefakttyp, der die Erfordernisse aller Projektteilnehmer erfüllen kann, sodass es notwendig ist, eine Vielzahl verschiedener Artefakttypen einzusetzen. Dies wiederum birgt das Risiko, aufgrund von Abhängigkeiten Inkonsistenzen und Zusatzaufwand zu erzeugen. Erfolgreiche Integration der Anforderungsartefakte ist ein wichtiger Aspekt im Requirements Engineering. Die vorliegende Studie leistet einen Betrag dazu, das Verständnis hierfür zu verbessern.

4.2 Aufbau der Studie

In der Studie wird der Umgang mit verschiedenen Anforderungsartefakten in der Praxis untersucht. Dies erfolgt in *zwei Phasen*: Zunächst werden die Artefakte selbst und ihre Unterstützung von Entwicklungsaktivitäten untersucht. Dann wird auf die parallele Verwendung mehrerer Artefakte eingegangen.

In der ersten Phase der Studie ist es das Ziel, die Werte und Hindernisse verschiedener Artefakttypen zu verstehen sowie zu untersuchen, welche Konsequenzen die Verwendung mehrerer Artefakttypen hat. Diese Phase wird durch vier Forschungsfragen angeleitet:

Forschungsfrage 1: Welche Anforderungsartefakte werden in der Praxis verwendet? Zunächst wird berichtet, welche konkreten Anforderungsartefakte in den untersuchten Projekten verwendet werden. Dabei wird vor allem darauf eingegangen, welche Eigenschaften aus Sicht der Projektteilnehmer diese Artefakte ausmachen.

Forschungsfrage 2: Ändern sich die verwendeten Artefakte mit der Zeit? In verschiedenen Phasen eines Projektes wird in unterschiedlicher Weise auf Anforderungen eingegangen. Während die Projektteilnehmer zu Beginn des Projektes einen gemeinsamen Informationsstand erlangen und die grobe Vorgehensweise durch Priorisierung und Architekturentscheidungen festlegen wollen, werden später eher Details ausgehandelt, Änderungen eingepflegt und die Software getestet und validiert. Es stellt sich die Frage, ob eine Anforderungslandschaft, die auf solche Änderungen eingeht, von vornherein auf alle Aktivitäten ausgelegt wird oder ob sich die Landschaft mit der Zeit den Erfordernissen anpasst. Dazu wird innerhalb dieser Forschungsfrage untersucht, ob sich die verwendeten Artefakttypen mit der Zeit ändern, obsolet werden, ob sie durch andere Typen abgelöst werden oder ob mit der Zeit neue Artefakte in die Landschaft aufgenommen werden, um neue Blickwinkel abzudecken.

Forschungsfrage 3: Welche Werte und Störfaktoren sehen Praktiker in den einzelnen Anforderungsartefakten? Innerhalb eines Projektes kommen unterschiedliche Rollen in Berührung mit Anforderungen und führen auf deren Basis unterschiedliche Aktivitäten durch. Die Repräsentationen von Anforderungen können besser oder schlechter dazu geeignet sein, diese Aktivitäten durchzuführen. In Zusammenhang mit dieser Forschungsfrage wird ein Überblick über relevante Aktivitäten geschaffen und gezeigt, welche Artefakte diese Aktivitäten gut oder schlecht unterstützen können.

Forschungsfrage 4: Welche Vorteile und Probleme begegnen Praktikern, wenn sie mehrere verschiedene Anforderungsartefakte in einem Projekt verwenden? Die Studie zeigt, dass in einem Projekt oft mehrere verschiedene Artefaktarten zusammen verwendet werden, um die verschiedenen Aktivitäten zu unterstützen. Oft haben dabei verschiedene Artefakte auch überlappende

Inhalte, sodass deren Bearbeitung zu Inkonsistenzen führen kann. Im Zuge dieser Forschungsfrage wird herausgearbeitet, auf welche Probleme Praktiker tatsächlich gestoßen sind und welche sie relevant finden.

Im Umgang mit mehreren Artefakten könnten viele Probleme verringert werden, wenn abhängige Artefakteile explizit zueinander zugeordnet würden. Die zweite Phase der Studie konzentriert sich daher darauf, herauszuarbeiten, ob Methoden zur Verknüpfung in der Praxis verwendet werden und welche Gründe die Praktiker davon abhalten, solche Methoden zu verwenden. Während in dieser Phase nach wie vor Einsichten und Validierung für die ersten vier Forschungsfragen erarbeitet wurden, wurden die folgenden zwei Forschungsfragen hinzugefügt.

Forschungsfrage 5: Welche Methoden zur Verknüpfung mehrerer verschiedener Artefakte werden in der Industrie angewendet? Die Verknüpfung eines Artefaktes mit einem anderen – beispielsweise indem die ID des einen Artefaktes im anderen Artefakt referenziert wird – kann später dabei helfen, verwandte Inhalte zu identifizieren. Dies kann verwendet werden, um bei der Dokumentation von Änderungen Inkonsistenzen zu vermeiden. Eine fortgeschrittenere Methode ist der Einsatz klickbarer Links, die den Nutzer direkt zu verwandten Inhalten führen oder diese direkt anzeigen. Weiterhin könnten zwei Artefakte auf denselben Inhalten operieren und einfach zwei Sichten auf diese Inhalte darstellen. Es ist nicht gut dokumentiert, welche dieser Methoden in der Industrie bekannt oder verbreitet sind. Mit dieser Forschungsfrage wird diese Lücke geschlossen.

Forschungsfrage 6: Welche Herausforderungen treten bei der Verknüpfung mehrerer Anforderungsartefakte auf? Oftmals werden nur die simpleren Methoden zur Verknüpfung von Artefakten verwendet. Gleichzeitig finden Entwickler es schwierig, mit mehreren Artefakten parallel zu arbeiten. Hier wird darauf eingegangen, was die Entwickler davon abhält, fortgeschrittene Zuordnungsmethoden zu verwenden, welche die Schwierigkeiten beheben könnten. Insbesondere wird darauf eingegangen, ob eher die Erzeugung der Links Probleme verursacht oder ob die Entwickler zu wenig Wert in der späteren Verwendung von Links sehen.

4.3 Durchführung der Studie

Es wurden 21 Praktiker aus 6 Unternehmen in Interviews befragt. Tabelle 1 zeigt einen Überblick über die Interview-Teilnehmer, deren Rollen und Unternehmen. Um eine Abdeckung von vielen verschiedenen anforderungsbezogenen Aktivitäten zu erreichen, wurden Personen aus verschiedenen Rollen interviewt. Zudem wird die Art der Unternehmen differenziert, weil diese beeinflusst, wie das Verhältnis der Entwicklungsseite zur Kundenseite ist und damit auch, wie die Anforderungskommunikation abläuft. *IT Service Provider* erstellen spezielle

4 Stand der Praxis: Interview-Studie zum Umgang mit Anforderungsartefakten

Software für einen externen Kunden und stehen daher in einem typischen Auftraggeber-Auftragnehmer-Verhältnis. *In-House-IT-Abteilungen* sind Abteilungen, die zu einem Unternehmen gehören, welches nicht primär Software herstellt. Die *In-House-IT-Abteilung* erstellt Software (oder stellt IT-Dienstleistungen bereit), welche die Fachabteilungen des Unternehmens und ihre Prozesse unterstützen. Hier gehören die Nutzer und auch die Geldgeber der Software zum selben Unternehmen, wie die Entwickler, sodass oft ein lockeres Verhältnis mit kürzeren Dienstwegen möglich ist. Oftmals werden vertragliche Vereinbarungen getroffen, die jedoch leichter änderbar sind, als in einem Auftraggeber-Auftragnehmer-Verhältnis. *Standardsoftware-Hersteller* wiederum sind Unternehmen, die eine Software erstellen, welche sie dann an mehrere Kunden verkaufen. Auch hier ist es möglich, individuell angepasste Versionen der Software anzubieten, doch wird vorrangig eine Basissoftware entwickelt. Hier ist das Unternehmen selbst dafür verantwortlich, die umzusetzenden Anforderungen festzulegen und zu priorisieren. Meist tun dies Marketingabteilungen oder Produktmanager des Unternehmens.

Tabelle 1: Übersicht über Interview-Teilnehmer

Unternehmensart	Unternehmens-ID	Größe (Mitarbeiter in der IT)	ID	Rolle
IT Service Provider	C1	500 - 1000	I1	Projektleiter
	C2	1000 - 1500	I2	Product Owner & Projektleiter
In-House-IT	C3	100 - 500	I3	Entwickler
			I4	Projektleiter
			I5	Fachbereichs- (bzw. Kunden-) vertreter
			I6	Product Owner
			I7	Projektleiter
			I8	Fachbereichsvertreter
			I9	Architekt
			I10	Fachbereichsvertreter
	C4	1000 - 1500	I11	Fachbereichsvertreter
			I12	Projektleiter
			I13	Entwickler
Standardsoftware-Hersteller	C5	<100	I14	Prozess-Engineer
	C6	<100	I15	Teamleiter
			I16	Entwickler
			I17	Entwickler
			I18	Entwickler
			I19	Teamleiter
			I20	Teamleiter
			I21	Teamleiter

Es wurden semi-strukturierte Interviews mit einer durchschnittlichen Dauer von ca. 75 Minuten durchgeführt. Die Interviews wurden aufgezeichnet, transkribiert und die Aussagen im Anschluss kodiert und kategorisiert im Sinne der Aktivitäten aus der Grounded Theory (*Open Coding*, *Selective Coding* und *Categorization*) [56], [69].

4.4 Ergebnisse

4.4.1 Verwendete Anforderungsartefakte und ihre Klassifikation (Forschungsfrage 1)

Die Vielfalt an Anforderungsartefakten, die in Softwareprojekten verwendet werden, ist sehr hoch. Die Interview-Teilnehmer haben vor allem drei Arten von Artefakten erwähnt: Container, Einzelemente und Lösungsmodelle. Im Lauf der Interviews stellten sich weitere Charakteristiken von Artefakten heraus, die ihre Handhabung beeinflussen. Die drei Kategorien werden daher weiter unterteilt, um auf diese Charakteristiken eingehen zu können. Abbildung 4 zeigt die ermittelten Artefaktkategorien, sowie die konkreten Artefakte, die dazu gehören.

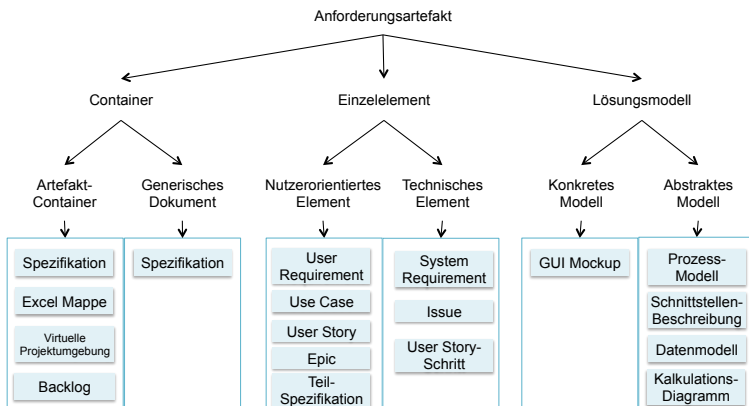


Abbildung 4: Klassifikation verwendeter Artefakte

Container sind gekennzeichnet durch ihren Wert, mehrere Dinge, wie Anforderungen und zugehörige Informationen, alle an einer Stelle zusammenzuhalten. Ein Unterschied ist, ob ein Container nur aus anderen Artefakten (Einzelementen oder Lösungsmodellen) besteht, oder ob das Container-Artefakt selbst die Eingabe generischer Inhalte erlaubt. Generische Elemente, wie ein Textdokument, erleichtern es, beliebige wichtige Information schnell festzuhalten, da diese zunächst einfach in freier Form dokumentiert werden kann. Gleichzeitig besteht jedoch dann das Risiko, dass die Information in späteren Aufgaben nicht gut verarbeitet werden kann, da sie nicht die richtige Form besitzt oder den Anforderungen relevante Attribute fehlen.

Virtuelle Projektumgebungen, wie beispielsweise die anforderungsbezogenen Umgebungen in Werkzeugen, wie JIRA [7] oder DOORS [71], oder Backlogs sind dabei Artefakt-Container, während Spezifikationsdokumente und Excel-Arbeitsmappen auf beide Arten – als Artefakt-Container oder generisches Dokument – genutzt werden können. Generell sind Textdokumente generisch und erlauben, Informationen in verschiedensten Formen festzuhalten. Textblöcke können ein oder mehrere Anforderungen auf einmal enthalten und diese mit Zielen, Hintergrundinformationen oder Richtlinien vermischen. Am anderen Ende des Spektrums können Dokumente eine starke formale Struktur besitzen, die nur Elemente zulässt, welche einen bestimmten Typ, eine bestimmte Form und eine ID enthalten. Auch Excel-Arbeitsmappen können generische Information enthalten. Innerhalb der untersuchten Projekte wurden diese jedoch nur als Artefakt-Container verwendet, in dem jedes Element ein konkretes Artefakt mit einem definierten Typ, wie User Story, GUI-Mockup oder Prozessmodell war.

Für *Einzelelemente* ist der wichtigste zu unterscheidende Aspekt, ob diese nutzerorientiert sind oder nicht. Dieser bestimmt, wie gut die Nutzer selbst zur Erstellung oder Beurteilung solcher Elemente beisteuern können. Ein Element gilt als nutzerorientiert, wenn für einen Nutzer klar ist, wie sich das System nach der Umsetzung des Elementes ändern würde.

Elemente, die als *User Stories* bezeichnet wurden, kamen in beiden Formen vor – als nutzerorientierte, aber auch als technischer orientierte Elemente. Während User Stories im ursprünglichen Sinne [8], [145] beschreiben, was die Nutzer tun, um ihre fachlichen Aufgaben zu erledigen, beschreiben technische User Stories meist kleinere einzelne Schritte, die benötigt werden, um eine User Story erfüllen zu können. Nutzer können sie im System erkennen, testen und auch Feedback zu ihnen abgeben. Dennoch können Nutzer nicht unbedingt verstehen, wie sie an ihre täglichen Aufgaben anknüpfen und entsprechend auch nicht kritisch beurteilen, ob die Umsetzung wirklich bei der Erreichung der Ziele helfen wird. Um solche technischen User Stories unterscheiden zu können, werden diese hier als *User Story-Schritte* bezeichnet.

Beispielsweise ist, wie in Abbildung 5 gezeigt, eine User Story denkbar, in der ein Bankmitarbeiter alle Akten von Kontobesitzern überprüfen will, deren Kredit in sechs Monaten zurückgezahlt sein muss. Dazu werden beispielsweise die User Story-Schritte *Hauptmenü mit verschiedenen Kontotypen anzeigen*, *Datensätze filtern*, *Detailinformationen zu Datensatz anzeigen* und *Notiz an Datensatz anhängen* erstellt. Die Aufteilung erlaubt es, die Funktionen nach und nach umzusetzen und auf verschiedene Iterationen aufzuteilen. Jedoch ist es – laut in den Interviews berichteter analoger Situationen – möglich, dass für den Nutzer beispielsweise die Filterfunktion für sich stehend ausreichend aussieht und er erst bemerkt, dass die Datumseingabe nicht dem üblichen Format folgt, wenn er sich tatsächlich in seine Arbeitsaufgaben versetzt und versucht, konkrete Datensätze, die in sechs Monaten ablaufen, aufzurufen.

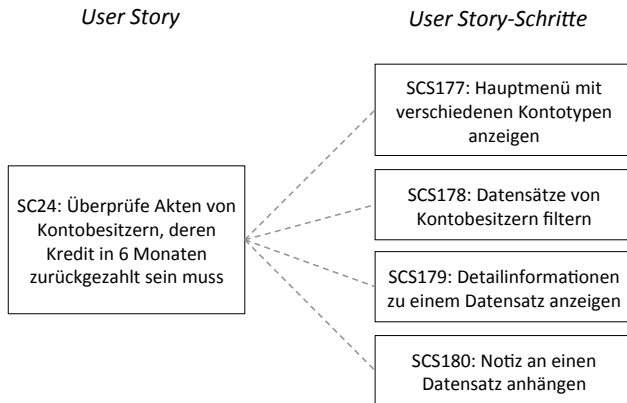


Abbildung 5: Beispiel für Unterscheidung zwischen User Story und User Story-Schritt

Als *Teil-Spezifikationen* werden kleinere Spezifikationen bezeichnet, die nur einen Teil des Systems anstelle des gesamten Systems beschreiben. Sie werden verwendet, um die Anforderungen und deren Details nach und nach auszuarbeiten. Dies kommt beispielsweise in iterativen bzw. inkrementellen Entwicklungsansätzen oder in Wartungs- bzw. Erweiterungsprojekten zu bestehender Software vor. Die Interview-Teilnehmer beschrieben, dass Teil-Spezifikationen relativ leicht handhabbar sind. Herausforderungen treten aber vor allem auf, weil es zu einem System viele solche Teil-Spezifikationen geben kann, die zudem in Wechselbeziehung stehen. Daher werden Teil-Spezifikationen als Einzelelemente und nicht als Container klassifiziert.

Lösungsmodelle illustrieren Aspekte der zukünftigen Lösung mittels formaler oder grafischer Modelle. Dabei stellen konkrete Modelle konkrete Situationen dar, die ein Nutzer tatsächlich erfährt. Im Gegensatz dazu zeigen abstrakte Modelle Verallgemeinerungen von mehreren konkreten Situationen. Beispielsweise zeigen Geschäftsprozessmodelle oft allgemeine abstrakte Workflows, die mehrere Rollen und Phasen umfassen. Die schrittweise Beschreibung eines spezifischen Teilpfades innerhalb dieses Workflows ist dann eine konkrete Instanziierung für eine einzelne Person. Die Interview-Teilnehmer haben von Situationen berichtet, in denen einige Kundenvertreter Schwierigkeiten damit hatten, in abstrakten Aspekten zu denken und darin Fehler oder Missverständnisse ausfindig zu machen. Damit kann die Abstraktheit eines Modells dessen Eignung, bei der Anforderungskommunikation zu helfen, beeinflussen.

4.4.2 Veränderung der verwendeten Anforderungsartefakte mit der Zeit (Forschungsfrage 2)

Es wurden zwei Arten von Änderungen beschrieben. Einerseits hat sich in einigen der besprochenen Projekte mit der Zeit der Aufbau der Anforderungslandschaft geändert. Eine weitere Feststellung ist die Änderung die Nutzung einer Anforderungslandschaft in einigen Projekten. Tabelle 2 gibt einen Überblick über die konkreten Situationen, in denen sich die verwendeten Anforderungsartefakte ändern können.

Tabelle 2: Situationen, in denen sich Anforderungslandschaften ändern

		# Int.
Änderung des Aufbaus einer Anforderungslandschaft		
Ä1	Zusätzliche Artefakte als Anpassung an das Verständnis der Stakeholder	6
Ä2	Zusätzliche Artefakte für Höhere Separation of Concerns	2
Änderung der Nutzung einer Anforderungslandschaft		
Ä3	Eigentliche Software kommt mit der Zeit hinzu und macht andere Artefakte weniger wichtig	5
Ä4	Kunden und Entwickler lernen mit der Zeit die typischen Notationsformen der anderen Seite und verwenden diese häufiger auch selbst	3

In einigen Fällen traten nachträgliche Änderungen des *Aufbaus einer Anforderungslandschaft* auf. Als wichtigster Grund wurde eine Anpassung der Anforderungslandschaft an das Verständnis der Stakeholder genannt. Oft sollen in einem Projekt die Stakeholder verstärkt in die Entwicklung einbezogen werden, beispielsweise durch Priorisierung einzelner Anforderungen oder Validierung und Feedback zu einzelnen Implementierungen. Hierzu ist wichtig, dass auch die Stakeholder so gut mit den Anforderungsartefakten umgehen können, dass sie diese selbst erstellen oder zumindest gut beurteilen können. Jedoch ist nicht immer zu Beginn eines Projektes klar, mit welchen Repräsentationen die Stakeholder gut umgehen können. Oft haben die Interview-Teilnehmer mit typischen Artefakten begonnen und mit der Zeit weitere hinzugefügt, wenn sie merkten, dass die Stakeholder mit einer bestimmten Repräsentation nicht ausreichend gut zurechtkamen oder wenn sie Kommunikationsprobleme zu bestimmten Aspekten hatten.

Weiterhin wurden in einigen Fällen nachträglich neue Artefakttypen hinzugefügt, um ähnliche Konzepte, die vorher in einem Artefakttyp zusammengefasst waren, klarer zu trennen. Beispielsweise wurde ein Projekt anhand von Entwicklungstasks, wie *Issues*, verwaltet. Nachträglich wurden Use Cases als explizite Artefakte hinzugefügt und mit den Tasks verknüpft, um klarer zu trennen, wann nur ein Task und wann ein ganzer Use Case fertiggestellt worden ist. Die Fertigstellung eines Use Cases zog zusätzliche Aktivitäten nach sich und

konnte nun transparenter behandelt werden. Ähnlich sind in einem Wartungsprojekt im Zeitverlauf Anforderungen mit anderen IT-Aufträgen in derselben Artefaktform vermischt worden. Jedoch hat sich gezeigt, dass beide unterschiedlich zu behandeln sind. Durch eine Aufteilung auf die verschiedenen Artefakttypen wurde diese unterschiedliche Behandlung möglich.

In diesen Fällen wurde das Prinzip der Separation of Concerns [39] auf Anforderungsartefakte angewendet. Dies wird auch in der Literatur beispielsweise von [49], [133] aufgegriffen. Durch die Aufteilung auf mehrere Artefaktarten wird die Handhabung verbessert, da die Artefakttypen besser zu ihren Aktivitäten passen, wenn sie nicht mehrere Zwecke vermischen.

Beachtenswert ist, dass die Änderungen am Aufbau oft zusammen mit der Identifikation von Problemen auftraten. Projektteilnehmer konnten Informationen nicht vermitteln oder Aktivitäten nicht durchführen und haben daraufhin den Aufbau geändert.

Für die Änderung der *Nutzung einer Anforderungslandschaft* wurden zwei Gründe genannt. Nachdem ein nutzbares Grundgerüst der Software aufgestellt ist, kommt die Software selbst als ein Artefakt hinzu, das zur Anforderungskommunikation genutzt werden kann. Da es konkret ist, können Stakeholder gut damit umgehen und daran gut veranschaulichen, welche Wünsche oder Probleme sie haben. Die Software als Kommunikationsartefakt kann andere Artefakte, die zudem das Risiko tragen zu veralten, weiter in den Hintergrund rücken oder ganz ablösen. Diese Entwicklung wurde als positiv angesehen.

Eine zweite Art von Veränderung tritt dadurch auf, dass Projektteilnehmer einige Repräsentationen zu Beginn des Projektes nicht gut selbst erstellen können, dies aber mit der Zeit lernen. Entwickler mussten beispielsweise erst lernen, welche Arten von Berechnungen im Versicherungswesen relevant sind und wie diese dargestellt werden können. Vor allem wurde aber die Situation genannt, dass Stakeholder zu Beginn nicht gut selbst User Stories oder auch Geschäftsprozessmodelle verfassen konnten und diese Aufgabe an die Entwickler abgaben. Beispielsweise schickten sie ihnen E-Mails mit den gewünschten Anforderungen und die Entwickler pflegten diese als User Stories in den Backlog ein. Mit der Zeit lernten sie diese Dinge – auch auf Basis der vorhandenen Artefakte – und konnten dann auch selbst User Stories verfassen. Dies war von den Entwicklern als positiv angesehen, weil sie so die Anforderungen direkter erhielten und keine unnötigen Verzögerungen oder Übersetzungsschritte entstanden.


4.4.3 Werte und Störfaktoren Einzelner Anforderungsartefakte (Forschungsfrage 3)

Der größte Wert von Anforderungsartefakten liegt in ihrer Eignung, eine Softwareentwicklungsaktivität zu unterstützen. Um auf diesen Aspekt näher einzugehen, drehte sich ein Teil der Interviews darum, welche Softwareentwicklungsaktivitäten die Interview-Teilnehmer durchführen, ob sie dazu Artefakte verwenden und ob sie die Artefakte eher als hilfreich oder störend für die entsprechenden Aktivitäten empfinden.

Tabelle 3 zeigt die Aktivitäten, über die in den Interviews berichtet wurde. In jeder Zelle wird angezeigt, wie viele Interview-Teilnehmer erwähnt haben, dass sich ein bestimmter Artefakttyp (Spalte) in einer bestimmten Aktivität (Zeile) gut eignet, beziehungsweise potenziell Probleme verursacht. Um zudem besser zu verstehen, was genau zur Einschätzung der Interview-Teilnehmer führte, wurden zusätzlich die Eigenschaften von Artefakten abgefragt, die es hilfreich oder störend machten. Diese Eigenschaften sind in Tabelle 4 aufgeführt.

Tabelle 3: Anforderungsbezogene Aktivitäten und positive sowie negative Erfahrungen mit bestimmten Artefaktarten darin

ID	Aktivität	Container		Einzelelement		Lösungsmodell		Verbale Kommunikation, E-Mail
		Artefakt-container	Generisches Dokument	nutzer-orientiert	technisch	abstrakt	konkret	
A1	Übergreifende Aspekte verstehen und dokumentieren	1	7 1	2 4	2 5		2	2
A2	Allgemeine Projektplanung		5 1				1	
A3	Anforderungen sammeln	2	6 1	7	1 1	1 2	4	1
A4	Widersprüche/ Inkonsistenzen aufdecken	1	1 1	3 1		2	2	
A5	Anforderungen priorisieren		1 1	1	3 2			4
A6	Iterationen planen und steuern		1 3	4 2	9 1	1		1 1
A7	Anforderungen klären	1	1 2	7 2	2 2	4 6	7	2
A8	Details außerhalb der Analysephase klären	1	1		1			5
A9	Anforderungen verwalten		3 4	4 4	4 2	1 1	1	4 6
A10	Release-Erfüllung prüfen und berichten		2 1		1	1	2	6 1
A11	Vertragsangelegenheiten		4	1	1			1

 # Interview-Teilnehmer, die erwähnt haben, dass Artefakttyp gut geeignet für Aktivität war


 # Interview-Teilnehmer, die erwähnt haben, dass Artefakttyp potenziell Probleme in der Aktivität verursacht

Tabelle 4: Positive und Negative Eigenschaften verschiedener Artefaktarten

Artefakt	Positive Eigenschaften	# Int.	Negative Eigenschaften	# Int.
Container	sammelt alles an einem zentralen Ort	6		
	ermöglicht Abnahmen	4		
Artefakt-Container			neues Tool, das erlernt werden muss	5
Generisches Dokument	universell (jeder kann mit Office-Dokumenten umgehen)	1	schwierig zu durchsuchen	6
	erlaubt es, Informationen auf viele Arten aufzuschreiben	2	Inhalte bleiben zu vage oder generisch	4
			Inhalte werden zu detailliert spezifiziert	4
Einzelelement	teilt große Blöcke in kleine handhabbare Teile	9	Abhängigkeiten zu übergreifenden Elementen unklar	5
	besitzt gute Granularität, um auf Detail-Informationen einzugehen	4	Abhängigkeiten zu anderen Elementen desselben Typs unklar	3
	einfach, Attribute anzuhängen (Autor, Release)	4	schwierig für Personen, die nicht regelmäßig mit den Artefakten umgehen, die Struktur zu verstehen	3
Nutzerorientiertes Einzelelement	verständlich für Stakeholder	3	Zu grob für die Planung der Entwicklung	2
Technisches Einzelelement	gute Granularität, um zu prüfen ob gesamte notwendige Information vorhanden ist	3	nicht verständlich für Stakeholder	6
			genauer Scope ist nicht klar	2
Lösungsmodell	lässt wenig Raum für (Fehl-) Interpretation	4	Begrenzte Ausdruckskraft	1
	gut, um Informationen schnell zu finden	4		
Konkretes Modell	für Stakeholder am besten verständlich	8		
Abstraktes Modell			Nicht für alle Stakeholder verständlich	4

Eine der am meisten diskutierten Aktivitäten ist die *Klärung von Anforderungsdetails* innerhalb der Analysephase. Viele Interview-Teilnehmer hatten Schwierigkeiten damit, die richtige Artefaktart zur guten Kommunikation mit den Kunden bzw. Nutzern zu finden. Auch wenn es viele Spezifikationstemplates und Diagrammarten gibt, die für diese Aktivität empfohlen werden [73], [137], haben die Interview-Teilnehmer berichtet, dass ihre Kunden damit oft nicht gut umgehen konnten. Es wurde als besonders hilfreich angesehen, sehr konkrete Modelle für die Kommunikation zu verwenden. Stakeholder konnten Anforde-

rungsrepräsentationen am besten evaluieren und zu ihnen beisteuern, wenn diese in Form von GUI-Mockups oder konkreten Anwendungsszenarien auftraten. Abstrakte Modelle, wie Flow Charts [3] eines Prozesses, waren hilfreich für einige Kunden, aber gefährlich für andere. Kunden, welche die Notation oder die abstrakten Inhalte nicht komplett verstanden – und aufgrund von Zeitdruck keine Zeit hatten, sich damit näher zu beschäftigen – betrachteten die Modelle manchmal zu oberflächlich und stimmten vorgeschlagenen Modellen zu schnell zu. Dies führte jedoch zu falschen Annahmen und auch zu einem falschen Gefühl der Sicherheit bei den Entwicklern. Solche Missverständnisse wurden erst spät aufgedeckt.

In den meisten Situationen, in denen User Stories verwendet wurden, waren diese interessanterweise nicht gut verständlich für die Stakeholder. Oftmals waren sie so aufgeilt, dass sie nicht mehr die fachlichen Arbeitsaufgaben der Nutzer betrafen, sondern eher einzelne Funktionen auf Systemebene repräsentierten. Das Team war von User Stories zu User Story-Schritten abgedriftet und hatte damit Potenzial verloren, die Nutzer mit in die Entwicklung und vor allem die Priorisierung einzubeziehen. Teilweise konnte dies durch mehr Kommunikation zwischen Kunden und Entwicklern kompensiert werden, indem die Entwickler den Kunden die Bedeutung und Notwendigkeit der User Story-Schritte erklärten. Der zusätzliche Aufwand für die kontinuierliche Erklärung bzw. Übersetzung der User Story-Schritte war jedoch merkbar.

Oftmals müssen Details auch nach der Analysephase, also beispielsweise während der Implementierung oder des Testens, geklärt werden. In diesen Situationen wurden sehr schnelle Kommunikations- und Dokumentationsformen, wie E-Mails oder Telefonate, als gut geeignet angesehen. Da dies jedoch keine sehr beständige Dokumentation erzeugt, wurden spätere Anforderungsmanagement-Aktivitäten, wie die Rekonstruktion der Historie einer Anforderung schwierig bis hin zu unmöglich. Genauso haben einige Interview-Teilnehmer berichtet, dass das Anforderungsmanagement erschwert wurde, wenn ein Spezifikationsdokument verwendet wurde. Zum einen ist es hier schwierig, eine bestimmte Anforderung beispielsweise bei einer Änderung im Dokument zu identifizieren. Hinzu kommt das Problem, dass oft Zusatzinformationen, wie Ersteller oder Bearbeiter einer Anforderung, ihre Begründung (Rationale) oder das Release, in dem sie umgesetzt wurde, nicht für jede Anforderung einzeln festgehalten werden. Dadurch kann es später zu Unklarheiten und zusätzlichen Kosten zur Klärung einzelner Anforderungen kommen.

Zur *Planung und Steuerung der Entwicklung* wurden Einzelelemente als hilfreich genannt. Die Aufspaltung der Spezifikation in Einzelelemente macht es möglich, zusätzliche Informationen zu einzelnen Artefakten hinzuzufügen. Außerdem werden die einzelnen Elemente durch das *Teile und Herrsche*-Prinzip

besser greifbar und handhabbar. Dies entspricht größtenteils dem, was aus der Literatur zu agilen Methoden bekannt ist.

Jedoch wurden in der Studie auch Aktivitäten festgestellt, die problematisch sind, wenn im Projekt nur Einzelelemente, wie User Stories verwendet werden. Die erwähnten Probleme entstehen meist in Situationen, in denen ein Gesamtüberblick über Anforderungen benötigt wird. Zum einen haben einige Interview-Teilnehmer erwähnt, dass es wichtig war, die *Vision eines Projektes zu verstehen und zu dokumentieren*. Sie berichteten über Situationen, in denen eine Anforderung für sich in Ordnung war – beispielsweise weil sie klar und in sich geschlossen war und Akzeptanzkriterien besaß – jedoch auf allgemeinerer Ebene keinen Sinn ergab, weil sie nicht die Probleme der Nutzer löste. Um solche Situationen zu identifizieren, müssen Entwickler den Kontext einer User Story verstehen, der sich zum Beispiel über verwandte Stories oder Ziele ergibt. Jedoch war dies nicht immer mit den vorhandenen Anforderungsartefakten in Form von Einzelelementen möglich.

Eine weitere wichtige übergreifende Aktivität ist das Aufstellen von Tests, die über den Scope eines einzelnen Elementes, wie einer Story hinausgehen. Beispielsweise müssen Entwickler auch automatisierte Tests schreiben, welche die Zusammenarbeit von Funktionen mehrerer User Stories abdecken. Genauso, wurde zur Vorbereitung von Akzeptanztests zu einem Release zusätzliches Wissen über allgemeine Nutzerziele benötigt. Wenn Entwickler nur mit den Informationen arbeiteten, die auf den einzelnen User Stories standen, fehlten ihnen einige solche verknüpfenden Testfälle.

Es ist wichtig, solche übergreifende Informationen zu dokumentieren, sodass sie nicht in späteren Projektphasen übersehen werden. Jedoch wurde es als schwierig empfunden, sie nur mittels Einzelelementen, wie User Stories oder Use Cases, festzuhalten. Hier wurden Spezifikationsdokumente als hilfreich empfunden, weil sie dem Autor viele Freiheiten lassen, solche Information festzuhalten. Manchmal wurden auch konkrete oder abstrakte Modelle – beispielsweise in Foliensätzen – als Referenz für die Projektvision und -ziele verwendet.

4.4.4 Vorteile und Probleme der Verwendung mehrerer verschiedener Anforderungsartefakte in einem Projekt (Forschungsfrage 4)

Wie die Ergebnisse zu den Forschungsfragen 2 und 3 andeuten, ist ein Grund, mit mehreren verschiedenen Anforderungsartefakten zu arbeiten, dass so die verschiedenen Aktivitäten besser unterstützt werden können. Dies stellt einen Vorteil dar. Ein weiterer erwähnter Vorteil war, dass die Darstellung derselben Ideen auf mehrere verschiedene Weisen es erlaubt, besser zu überprüfen, ob man die Anforderungen richtig interpretiert hat.

Jedoch beinhaltet die Verwendung mehrerer Artefakttypen zusätzliche Kosten zur Erstellung, Wartung und gegenseitigen Übersetzung von Anforderungsarte-

fakten. Tabelle 5 zeigt die Probleme, die von den Interview-Teilnehmern erwähnt wurden sowie die zugehörigen Ursachen und Effekte. #Int. stellt die Anzahl an Interview-Teilnehmern dar, die ein Thema erwähnt haben. Die Anzahlen #Int. in den Zeilen *Ursachen*, *Probleme* und *Effekte* zeigt die Gesamtzahl der Personen an, die über diese Aspekte gesprochen haben.

Tabelle 5: Probleme bei der Verwendung mehrerer verschiedener Anforderungsartefakte

		# Int.
Ursachen/Verstärker		5
U1	Einige Inhalte überlappen sich, andere sind disjunkt	2
U2	Nichttriviale Relationen zwischen Artefakten oder Artefakteilen	5
Probleme		15
Pr1	(R1 ->) Mehrfacher Aufwand zur Erstellung der Artefakte	3
Pr2	(R2 ->) Unsicherheit zur Vollständigkeit der Übersetzung	6
Pr3	(R1 ->) Änderungen müssen an mehreren Orten dokumentiert werden	4
Pr4	(R1, R2 ->) Inkonsistenzen	3
Pr5	(R1, R2 ->) Schwieriger, relevante Informationen an mehreren Orten ausfindig zu machen	4
Effekte		5
E1	Höhere Kosten zur Durchführung von Aktivitäten oder zur Verhinderung von Problemen	2
E2	Höhere Kosten bei Auftreten der Probleme (Fehler, Missverständnisse)	1
E3	Gemindertes Vertrauen in Aktualität der Artefakte	2

Mehr als 70% der Interview-Teilnehmer sind beim Umgang mit mehreren Artefakten auf Probleme gestoßen. Neben dem Zusatzaufwand zum Dokumentieren und Finden von Informationen an mehreren Orten (Pr1, Pr3, Pr5), hatten sie Schwierigkeiten durch Inkonsistenzen erlebt (Pr4). Außerdem wurde erwähnt, dass es schwierig war, zu prüfen, ob alle relevanten Elemente aus einem Artefakttyp auch korrekt in einem anderen abhängigen Artefakt auftraten (Pr2).

Die potenziellen Probleme führten zu zusätzlichem Aufwand zur Vermeidung des Eintretens der Probleme (E1). Jedoch konnten nicht immer Probleme vermieden werden. Wenn eine Inkonsistenz oder weiterführende Informationen übersehen wurden, so traten Missverständnisse oder falsche Annahmen auf, welche wiederum potenziell zu höheren Kosten aufgrund von Fehlern führten (E2). Ein weiterer berichteter Effekt war, dass Projektteilnehmer sehr schnell das Vertrauen in ein Dokument verloren (E3) – und aufhörten es zu verwenden – wenn sie mehrmals auf Inhalte stießen, die nicht aktuell oder inkonsistent waren.

Es sind insbesondere zwei Umstände, welche die Arbeit mit mehreren Artefakttypen in der Praxis schwierig machen. Zum einen beschreiben die Artefakte oft nicht nur disjunkte Informationen, sondern eher verschiedene Aspekte derselben Anforderungen (U1). Daher sind einige – jedoch wiederum nicht alle – Informationen in mehreren Artefakten enthalten. Da nicht alle Informationen auf mehrere Artefakte verteilt sind, sondern sich einige Informationen auch nur an einem

Ort befinden, kann nicht immer davon ausgegangen werden, dass es zu Anforderungen immer ein passendes Pendant in einer anderen Repräsentation gibt. Dies erschwert auch die automatisierte Prüfung.

Außerdem stehen Anforderungen nicht immer in einer simplen hierarchischen 1:n-Beziehung (U2), wie dies bei der Dekomposition von abstrakten Anforderungen zu konkreteren Unterelementen oder bei der Unterteilung einer Story in Tasks üblich wäre. Beispielsweise können ein Geschäftsprozessmodell und eine User Story-Menge komplexe Relationen besitzen. Das Geschäftsprozessmodell kann die Interaktion zwischen mehreren Stories darstellen. Zur selben Zeit kann eine Teilmenge der Aktivitäten im Prozessmodell eine User Story veranschaulichen.

4.4.5 In der Praxis verwendete Methoden zur Verlinkung verschiedener Anforderungsartefakte (Forschungsfrage 5)

Es wurden verschiedene Methoden erwähnt, die in der Praxis zur Verknüpfung von Anforderungsartefakten verwendet werden. Diese wurden in vier Verknüpfungsarten klassifiziert, wie Tabelle 6 zeigt.

Tabelle 6: In der Praxis angewendete Methoden zur Verknüpfung von Anforderungen

ID	Verknüpfungsmethode	# Int.
M1	Manuelle textuelle Referenz	6
M2	Anhang	10
M3	Link	3
M4	Generiertes Artefakt	1

Die simpelste Variante ist, zu einem Artefakt ein verwandtes Artefakt zuzuordnen, indem im Text des Artefaktes die ID des verwandten Artefaktes notiert wird. Diese Technik wird vor allem verwendet, um auf relevante Teile eines Spezifikationsdokumentes hinzuweisen. Beispielsweise enthält ein Change Request das Spezifikationskapitel, in dem die ursprünglichen Anforderungen enthalten waren. Ähnlich hatte ein Entwickler eine Spezifikation in User Stories übersetzt und zu jeder User Story das Kapitel mit den ursprünglichen Anforderungen hinzugefügt. Da sich die Struktur der Spezifikation jedoch mit der Zeit änderte, wurden einige der Referenzen obsolet. In einem anderen Projekt wurden in User Stories die Titel von GUI-Mockups oder Teile eines Überblicks-Prozessmodells textuell referenziert.

Eine weitere häufig verwendete Technik ist die Verknüpfung zweier Artefakte, indem ein Artefakt als Anhang zum anderen hinzugefügt wird. Wird mit dem übergeordneten Element gearbeitet, so kann so der Anhang direkt abgerufen werden, was es erleichtert, beispielsweise Detailinformationen nachzuschlagen. Jedoch existiert der Anhang nur innerhalb des übergeordneten Elementes und kann nicht von anderen Stellen aus abgerufen werden. Daher eignet sich diese Technik für hierarchische Strukturen. Oft erzeugen Werkzeuge diese Art der

Verknüpfung direkt während ein Artefakt angehängt wird, sodass kein besonderer Verknüpfungsschritt notwendig ist.

Die Verlinkung von zwei Elementen, die unabhängig von anderen existieren, wurde ebenfalls in den diskutierten Projekten erwähnt, jedoch nur in wenigen Fällen. Die explizite Verlinkung von zwei Elementen erlaubt es, von einem Element direkt zum anderen zu navigieren und dieses zu betrachten oder zu ändern. In einem beschriebenen Projekt wurde ein iterativer Prozess angewendet, der auf technischen Artefakten, nämlich Tasks, basierte. Sie fügten einen speziellen Artefakttyp hinzu, der Nutzerziele repräsentierte und verlinkten die Ziele mit allen Tasks, die dazu notwendig waren, um das Ziel erreichen zu können. Dies erlaubte es den Kunden, den Fortschritt des Projektes nun basierend auf ihren Zielen zu sehen, während die Entwickler ihre Arbeit nach wie vor basierend auf den Tasks und deren Abhängigkeiten strukturieren konnten.

Eine weitere erwähnte Technik ist die Erstellung oder Generierung eines Artefaktes aus mehreren anderen Artefakten. Diese Technik wird verwendet, um Spezifikationsdokumente aus einzelnen Anforderungen und Modellen zu erzeugen. Sie verhindert die Doppelung von Inhalten, indem die Inhalte an einer Stelle stehen und an anderen Stellen nur mittels anderer Sichten anders dargestellt werden. Um diese Technik zu verwenden, muss die Reihenfolge und Struktur der Einzelelemente beachtet werden, damit die generierten Artefakte lesbar sind.

4.4.6 Herausforderungen bei der Verknüpfung mehrerer Anforderungsartefakte (Forschungsfrage 6)

Es zeigen sich Hinweise auf beide Arten von Herausforderungen: sowohl ein als hoch empfundener Aufwand zur Erstellung von Verknüpfungen als auch die Erwartung, dass der Wert zu niedrig sein wird, wurden als Gründe dafür erwähnt, auch an problematischen Stellen auf Verknüpfungen zu verzichten. Tabelle 7 zeigt die konkreten Herausforderungen, welche die Interview-Teilnehmer störten oder an der Erstellung von Verknüpfungen hinderten.

Der am häufigsten angegebene Grund dafür, dass Interview-Teilnehmer in konkreten Situationen eine Verknüpfung nicht erstellt haben, war dass sie keine Zeit dazu hatten. Bei präziserem Nachfragen, was genau die Interview-Teilnehmer davon abhielt, Zeit in die Erstellung von Verknüpfungen zu investieren, zeigte sich ein weiterer Aspekt: meist sind die Projektteilnehmer mitten in einer anderen Aktivität, wenn sie gerade Informationen in mehreren Artefakten zusammensuchen. Beispielsweise implementieren sie gerade Code, erstellen andere Artefakte oder schätzen Iterationsaufwand. In solchen Situationen wollen sie die aktuelle Aufgabe nicht unterbrechen, um zusätzlich Artefakt-Verknüpfungen zu erstellen.

Tabelle 7: Herausforderungen bei der Verknüpfung mehrerer Anforderungsartefakte

ID	Herausforderung	# Int.
C1	Zeitdruck	5
C2	Unterbrechung anderer Aktivitäten	4
C3	Benötigt klare Guidelines/Richtlinien	1
C4	Schwierig, wenn die Anforderungen nicht voneinander isoliert aufgeführt sind	2
C5	Manuelle Referenzen können obsolet werden	2

Ein Interview-Teilnehmer, der in der Vergangenheit mit Links und Tracing gearbeitet hatte, erwähnte, dass klare Richtlinien notwendig sind, um eine gute und brauchbare Verknüpfungsstruktur aufzubauen. Beispielsweise muss festgelegt sein, dass jede User Story mit einem Nutzerziel zu verknüpfen ist. Wenn dann eine User Story mit keinem Nutzerziel verknüpft ist, muss klar sein, ob es kein zugehöriges Ziel gibt oder ob einfach die Verknüpfung fehlt. Der zusätzliche Aufwand zur Erarbeitung solcher Richtlinien ist eine weitere Hürde, die Praktiker davon abhält, Verknüpfungen zu verwenden.

Die Verknüpfung von Elementen ist besonders schwierig, wenn die Teile, die man verknüpfen möchte, nicht isoliert vorliegen. Wenn beispielsweise nur ein Textblock mit mehreren Anforderungen vorliegt oder Modellelemente nicht einzeln außerhalb vom gesamten Modell referenziert werden können, muss eine Verknüpfung zum gesamten Artefakt – also Textblock oder Modell – erstellt werden. Dadurch ist es schwieriger und weniger präzise, abhängige Elemente zu referenzieren. Ob einzelne Teile eines Modells mit anderen Artefakten verknüpft werden können, hängt auch von den Werkzeugen für die Erstellung der Modelle ab. Dies kann ein zusätzliches Problem in Erweiterungs- bzw. Wartungsprojekten darstellen, in denen die Entwickler auf bereits bestehender Dokumentation aufsetzen müssen.

Ein weiterer genannter Hinderungsaspekt ist die Gefahr, dass Verknüpfungen bei Änderungen obsolet werden. Ein Team hat beispielsweise versucht, mit manuellen Referenzen von User Stories auf Detailinformationen in einer Spezifikation zu verweisen. Da die Kapitel der Spezifikation sich jedoch von Zeit zu Zeit änderten, wurden die Links immer wieder nutzlos, sodass das Team diese Praktik letztlich aufgab.

4.5 Diskussion

Aus der Interpretation der Ergebnisse ergeben sich Anregungen, die auch für den weiteren Verlauf der Arbeit interessant sind. Sie zeigen, dass die Unterstützung im Umgang mit Anforderungsartefakten, wie sie in den Kapiteln 7 und 8 erarbeitet wird, hilfreich sein können, um Anforderungskommunikation und Softwareprojekte im allgemeinen zu verbessern.

Die Handhabung mehrerer unterschiedlicher Anforderungsartefakte ist schwierig. Die Ergebnisse deuten an, dass oftmals eine einzige Art von Anforderungsartefakten nicht ausreicht, um alle Projektteilnehmer in all ihren Aktivitäten zu unterstützen. In den meisten Projekten werden daher mehrere Arten von Artefakten gemeinsam verwendet um die Kommunikation zu unterstützen. Jedoch zeigen die Studienergebnisse auch, dass unterschiedliche Anforderungsartefakte oft nicht gut gegenseitig integriert sind. Relationen und Abhängigkeiten zwischen den Artefakten sind nicht sichtbar. Wenn Entwickler und Kunden diese Abhängigkeiten nicht im Blick behalten, so fehlen ihnen später wichtige Informationen.

Anforderungskommunikation mit Stakeholdern wird nicht gut unterstützt. Es zeigt sich ein Bedarf nach mehr Forschung, um Stakeholder oder Businessanalysten dabei zu unterstützen, ihre Anforderungen zu kommunizieren. Stakeholder sind oft gezwungen Anforderungsartefakte zu erstellen oder abzunehmen, deren Formate nur für die Entwickler gut geeignet sind. Oft können sie die „Sprachen“ dieser Artefakte nicht verstehen und können sie aufgrund von Zeitdruck auch nicht lernen. Vielmehr sollten Stakeholder in die Lage gebracht werden, Anforderungen in einer Form zu kommunizieren, die für sie greifbar und verständlich ist. Entwickler sollten die Möglichkeit haben, solche Formen von Anforderungen in andere Artefakte, die sie dann im Verlauf der Entwicklung verwenden, zu integrieren.

User Stories erfüllen bereits den Zweck, ein greifbares Kommunikationsmittel für Stakeholder darzustellen und ihnen sogar zu erlauben, die Entwicklung zu leiten. Oftmals werden sie jedoch nicht zu diesem Zweck verwendet. Die Studie zeigt, dass User Stories am häufigsten dazu eingesetzt werden, um Arbeitspakete für Entwickler in kleinere, besser handhabbare Arbeitsschritte aufzuspalten. Solche (technischen) User Stories müssen dann jedoch permanent übersetzt werden, um in der Kommunikation mit Stakeholdern besprochen zu werden. Um die Erfordernisse beider Gruppen – der Entwickler sowie der Stakeholder – zu unterstützen, **werden Stories auf verschiedenen Granularitätsebenen benötigt.** Wie auch von Interview-Teilnehmern vorgeschlagen, macht es Sinn, mit einer Kombination aus User Stories auf fachlicher Ebene („Business User Stories“) und technischen User Stories zu arbeiten (siehe auch [101], [103]).

Die Verknüpfung von Anforderungsartefakten hat hohes Potenzial. Es wurden viele Probleme genannt, die abgeschwächt werden können, wenn zusammenhängende Artefakte explizit verknüpft werden. Es kann viel Aufwand gespart werden, der zur manuellen Prüfung von Inkonsistenzen oder zur Überprüfung der Vollständigkeit von Übersetzungen zwischen Artefakttypen notwendig ist. Auch die Softwareentwicklungsaktivitäten selbst können so verbessert werden. Beispielsweise könnten Entwickler über Abhängigkeiten gewarnt werden, bevor sie eine Story Card implementieren oder wenn sich Änderungen von Kundenseite ergeben. Wenn abstrakte Modelle mit konkreten Modellen

oder anderen konkreten Anforderungsartefakten verlinkt werden können, könnten Requirements Engineers auch abstrakte Modelle zur Kommunikation mit Kunden verwenden: Teile des abstrakten Modells könnten direkt in konkrete Modellformen übersetzt werden und so eine verständliche Sicht auf die Details erlauben. In den Interviews kam der Eindruck auf, dass viele der potenziellen Vorteile von Verknüpfungen für Praktiker nicht deutlich sind.

Leichtgewichtige Anforderungskommunikation funktioniert gut. Viele Interview-Teilnehmer haben berichtet, dass sie viele Probleme durch direkte Kommunikation mit den Kunden lösen. Wie Tabelle 3 aufzeigt, wurden viele der diskutierten Aktivitäten durch direkte Kommunikation unterstützt. Viele Interview-Teilnehmer haben berichtet, dass sie die verbale Kommunikation zwischen verschiedenen Rollen – hauptsächlich durch wöchentliche oder zweiwöchentliche Treffen – erst in den letzten Jahren verstärkt haben. Seit der Einführung solcher Treffen haben sie viele Verbesserungen bemerkt. Dies ist ein positiver Fortschritt.

Jedoch haben sich in den Interviews auch neue Probleme gezeigt, wenn die Abhängigkeit von verbaler Kommunikation zu hoch war. Interview-Teilnehmer haben berichtet dass manchmal die einzige Möglichkeit, eine Abhängigkeit oder ein Missverständnis aufzudecken, war, wenn eine bestimmte Person – die meist die einzige Person mit einem bestimmten Wissen war – dieses Thema in einem der Meetings zur Diskussion brachte. Diese Strategie hat meist funktioniert, ist jedoch durchaus riskant, da sie vom Zufall abhängt. Die beschriebenen Situationen bringen neue Fragen auf, wie, ob mehr Unterstützung bei der Verbreitung von Wissen in Teams benötigt wird und wie solche kommunikationsabhängigen Ansätze skaliert werden können.

4.6 Validität der Ergebnisse

Bei der durchgeführten Studie handelt es sich um eine qualitative Studie, die einen Einblick in die Praxis vermittelt. Die Ergebnisse zeigen konkrete Eindrücke, Probleme und Erfahrungen, die von Praktikern als relevant erachtet wurden. Sie lassen jedoch keine quantitativen Rückschlüsse darauf zu, in wie vielen Projekten sich ein ähnliches oder ein anderes Bild zeigt. Zu den Ergebnissen wurde dennoch stets die Anzahl an Personen, die einen Aspekt erwähnt haben, angegeben. Zwar sind diese aufgrund des zu niedrigen Umfangs der Studie nicht im statistischen Sinn übertragbar, doch vermitteln sie dennoch einen Eindruck über das Gewicht, das ein Aspekt in der Gesamtheit der Gespräche hatte.

Um viele Perspektiven abzudecken und keine wichtigen Inhalte auszulassen, wurden Personen aus verschiedenen Projekten und Rollen befragt und der Umfang der Studie so gewählt, dass theoretische Sättigung der Inhalte eintrat [56]. Das bedeutet, dass die Studie so lange fortgesetzt wurde, bis mit der Zeit keine neuen Inhalte mehr in den Gesprächen aufkamen.

Weitere Einflüsse auf die Validität der Ergebnisse werden im Folgenden nach dem Schema von Runeson und Höst für Gefährdungen der Validität [140] diskutiert.

Validität des Aufbaus (Construct Validity)

Die tatsächlich „gemessenen“ Größen in dieser Studie sind zunächst die konkreten Aussagen der Interview-Teilnehmer. Die Konzepte, die dann als Ergebnisse zur Beantwortung der Forschungsfragen präsentiert werden, sind jedoch Abstraktionen der in den Aussagen erwähnten Aspekte, die durch Interpretation entstehen.

Eine Gefährdung der Konstruktvalidität besteht darin, dass die *gemessenen Größen* nicht geeignet sind, um das *präsentierte Konzept* zu beschreiben. Dies kann sich hier dadurch ereignen, dass die Interpretation der konkreten Aussagen falsch oder die Abstraktion ungeeignet ist. Ein Problem stellt die Tatsache dar, dass die Interpretation nur durch die Autorin der Studie stattfand. Die Interpretation ist dadurch prinzipiell subjektiv. Um jedoch sicherzugehen, dass die Abstraktionen zutreffen und dass keine Missverständnisse oder Missinterpretationen auftreten, wurde die Studie als Interview-Studie durchgeführt und bei Unsicherheiten die Korrektheit der Abstraktion durch Nachfragen bestätigt.

Interne Validität (Internal Validity)

Die interne Validität der Ergebnisse wird durch verschiedene Aspekte beeinflusst. Zum einen wird die Anforderungskommunikation – und damit auch die Effektivität von anforderungsbezogenen Aktivitäten sowie der verwendeten Anforderungsartefakte – stark von der Art eines Projektes und auch von seinen Kunden beeinflusst. Damit variieren die Eindrücke abhängig von den befragten Projekten. Um diese Gefährdung zu minimieren, wurden Praktiker aus verschiedenen Projekten und verschiedenen Unternehmensformen interviewt.

Die Interview-Teilnehmer wussten bereits vor dem Interview, dass es um Anforderungsartefakte gehen würde und haben auf dieser Basis entschieden, ob sie am Interview teilnehmen wollen. Daher ist nicht auszuschließen, dass die Teilnehmer ein erhöhtes Interesse in Requirements Engineering-Praktiken und der Verwendung von Anforderungsartefakten besitzen, als der Durchschnitt aller Softwareentwickler. Die Requirements Engineering-Methoden, die sie verwenden, die Artefakte, die sie verwenden und ihre Eindrücke des Nutzens dieser Artefakte kann durch ihr generelles Interesse für Requirements Engineering beeinflusst sein.

Externe Validität (External Validity)

Die Anzahl der Interview-Teilnehmer beeinflusst die externe Validität. Es wurden nur 21 Praktiker befragt, sodass es möglich ist, dass nicht alle Situationen zur Anforderungskommunikation abgedeckt werden konnten. Zudem waren die Teilnehmer alle aus deutschen Unternehmen. Da jedoch Personen aus verschie-

denen Unternehmensumfeldern, verschiedenen Projekten und verschiedenen Rollen interviewt wurden, ist die Vielfalt an abgedeckten Perspektiven dennoch sehr hoch. Zudem wurde ein Zustand erreicht, in dem neue Aussagen bereits denen aus vorausgegangenen Interviews ähnelten, sodass die Menge an neuen Erkenntnissen in Bezug auf die Forschungsfragen gegen Null ging (analog zur *Theoretical Saturation* aus der Grounded Theory [56]).

Validität der Schlussfolgerung (Reliability)

Da es sich um eine qualitative Untersuchung handelt, ist es nicht sinnvoll, die statistische Signifikanz der Ergebnisse zu untersuchen. Daher wird hier allgemein darauf eingegangen, welche Aspekte einen Einfluss darauf haben, dass die Ergebnisse bei einer anderen Durchführung der Studie zu anderen Ergebnissen führen könnten. Insbesondere ist hier zu erwähnen, dass die Interviews und ihre Analyse von nur einer Person durchgeführt wurden. Eine andere Person könnte sowohl die Fragen anders vermitteln, als auch die Antworten anders interpretieren, was zu anderen Ergebnissen und anderen Kategorisierungen führen kann.

4.7 Zusammenfassung

Es wurde eine Studie durchgeführt, bei der 21 Praktiker zu ihrem Umgang mit Anforderungsartefakten befragt wurden. Der Bericht zeigt die Erfahrungen aus der Software Engineering-Praxis, identifizierte Herausforderungen im Umgang mit Artefakten und den Fortschritt im Einsatz von Verknüpfungstechniken. Es hat sich gezeigt, dass mehrere Artefakttypen in einem Projekt benötigt werden. Entwickler brauchen detaillierte, feingranulare Elemente für die Implementierung, aber müssen auch allgemeinere, übergreifende Aspekte im Blick behalten. Kunden brauchen sehr konkrete Artefakte, um ihre Wünsche auszudrücken. Projektleiter wiederum brauchen Möglichkeiten, die einzelnen Anforderungen im Kontext zur Gesamtmenge der noch bevorstehenden Arbeit sehen zu können. Die Verwendung mehrerer Anforderungsartefakttypen führt zu neuen Herausforderungen, wie der Verteilung von Informationen, unvollständige Übersetzungen oder Inkonsistenzen zwischen mehreren Artefakten. Um mit diesen Herausforderungen umzugehen, sollten Methoden zur Verknüpfung von Anforderungsartefakten – wie aus dem Tracing bekannt – zu den Standardwerkzeugen im Requirements Engineering gehören. Jedoch wurde die Verwendung solcher Methoden in der Praxis selten in Erwägung gezogen. Viele Interview-Teilnehmer sind der Meinung, dass die explizite Verknüpfung von Anforderungsartefakten in ihrem Projektkontext zu aufwändig ist. Tatsächlich ist die Verknüpfung von Artefakten auch nicht in allen Situationen notwendig. Wenn die Verknüpfung von Artefakten jedoch durch bessere Methoden und Tools besser anwendbar wäre, könnte in vielen Softwareprojekten der Umgang mit Anforderungen und Anforderungskommunikation verbessert werden.

Nutzen der Ergebnisse.

Die Ergebnisse der Studie helfen Praktikern, ein besseres Verständnis des Nutzens eines Artefaktes für bestimmte Aktivitäten zu erlangen. Sie stellen einen Überblick über Verknüpfungsmethoden dar und helfen einzuschätzen, welche Gründe Projektteilnehmer davon abhalten, diese zu verwenden. Forscher erhalten einen Überblick über anforderungsbezogene Aktivitäten und die Handhabung von Anforderungsartefakten in der Praxis. Die Studie gibt Anregungen zu Herausforderungen, für die nähere Untersuchungen oder Lösungen hilfreich wären.

Für die vorliegende Arbeit stellen die Ergebnisse der Studie eine wichtige Grundlage dar. Sie zeigen, dass in der Praxis tatsächlich Anforderungslandschaften mit mehreren verschiedenen Artefakttypen im Einsatz sind und vor allem auch, dass die Abhängigkeiten zwischen Artefakttypen zu merkbaren Problemen führen. Die explizite Verknüpfung von Artefakten kann Projektteilnehmer unterstützen. Jedoch werden Tracing-Techniken nicht eingesetzt, da sie für viele Projekte als zu aufwändig und zu wenig hilfreich eingeschätzt werden.

Die weiteren Konzepte dieser Arbeit setzen genau hier – an den festgestellten Problemen mit Abhängigkeiten und Verknüpfungen – an. Im weiteren Verlauf der Arbeit wird herausgearbeitet, wie der Aufbau von Anforderungslandschaften basierend auf den berichteten Erfahrungen optimiert werden kann. Später, in Kapitel 8, werden Konzepte erarbeitet, welche die Erstellung und Verwendung von Verknüpfungen verbessern. Dazu liefern die Ergebnisse der Studie, genauer die identifizierten Aktivitäten und Erfordernisse, die *Anwendungsfälle*, auf welche die Lösungen ausgerichtet sind.

Die Interview-Studie hat allgemein Anforderungsartefakte und ihre Koexistenz in verschiedenen Projekten untersucht. *Hybride Anforderungslandschaften*, welche in dieser Arbeit im Speziellen betrachtet werden, besitzen die zusätzliche Eigenschaft, dass hier Artefakte aus der traditionellen und der agilen Entwicklung kombiniert werden. Die allgemeineren Ergebnisse der Studie treffen auch auf hybride Anforderungslandschaften zu. In hybriden Landschaften kommt es jedoch besonders häufig zur Verwendung mehrerer unterschiedlicher Artefakttypen, sodass hier die Probleme noch eher auftauchen. Da die Anforderungen in unterschiedlichen Sichten bereitgestellt werden müssen, kommt es hier eher zu Doppelungen von Informationen und dem Bedarf, beide Perspektiven leichtgewichtig einsehen zu können.

5 Ein Konzept zur Unterstützung der Koexistenz von agilen und traditionellen Anforderungsartefakten

5.1 Zielsetzung – Erstellung einer Methodik zur Unterstützung der Koexistenz

In der Interview-Studie in Kapitel 4 wurde gezeigt, dass (1) häufig mehrere Artefakttypen gemeinsam verwendet werden und dass es (2) häufig zu Schwierigkeiten kommt, weil die Artefakte zusammenhängen, diese Zusammenhänge jedoch nicht ausreichend unterstützt werden.

Um mit diesen Problemen besser umgehen zu können, hat das in dieser Arbeit vorgestellte Konzept das folgende Ziel:

Ziel des Konzeptteils dieser Arbeit

Ziel des Konzeptteils ist die Verbesserung der Unterstützung der Koexistenz von agilen und traditionellen Anforderungsartefakten

Eine Verbesserung lässt sich auf **zwei Ebenen** erreichen. Zum einen ist es auf der *methodischen Ebene* möglich, die Koexistenz von Anforderungsartefakten zu beeinflussen, indem eine explizite Analyse und Planung dazu durchgeführt wird, welche Typen von Anforderungsartefakten eingesetzt und welche Arten von Verknüpfungen besonders unterstützt werden. Der Aufbau mit verwendeten *Typen* von Artefakten und Verknüpfungen wird als Anforderungslandschaft bezeichnet. Die zweite Ebene betrifft die *konkrete Projektebene* mit den konkreten darin erstellten *Dokumenten*. Hier lässt sich die Koexistenz von Anforderungsartefakten beeinflussen, indem Verknüpfungen zwischen konkreten Anforderungen gefördert und auch besser ausgenutzt werden. Dies soll durch spezielle Tracing-Techniken, die als Trace-Operationen bezeichnet werden, geschehen. Drei konkrete Teilziele treiben das allgemeine Ziel auf den verschiedenen Ebenen voran:

Teilziel 1: Erstellung eines Ansatzes zur Modellierung von Anforderungslandschaften. Dabei soll eine Methodik erstellt werden, die Prozessingenieuren ermöglicht, für ein Projekt eine Anforderungslandschaft als Teil der Methode zu modellieren. Ein resultierendes Modell hebt dabei Aspekte, die für die Koexistenz relevant sind, hervor. Mithilfe eines solchen Modells kann die Koexistenz bereits verbessert werden, indem das Modell als Diskussionsgrundlage zum Umgang mit Anforderungen und Abhängigkeiten verwendet wird.

Teilziel 2: Erstellung eines Ansatzes zur inhaltlichen Optimierung von Anforderungslandschaften. Es soll ein Ansatz erstellt werden, der Prozessingeni-

5 Ein Konzept zur Unterstützung der Koexistenz von agilen und traditionellen Anforderungsartefakten

euren eine Hilfestellung bei der inhaltlichen Planung einer Anforderungslandschaft bietet. Damit kann der Prozessingenieur entscheiden, ob sich die ausgewählten Artefakt- und Verknüpfungstypen einer Anforderungslandschaft eignen, um den dokumentationsbedarf abzudecken und dabei eine gute Koexistenz zwischen Anforderungen zu erreichen.

Teilziel 3: Erstellung eines Ansatzes zur Verbesserung des täglichen Umgangs mit Artefakten und Abhängigkeiten durch Trace-Operationen. Dieser Ansatz soll die Arbeit mit Abhängigkeiten erleichtern und damit bewirken, dass Projektteilnehmer im alltäglichen Umgang mit Anforderungen abhängige Informationen oder Inkonsistenzen zu einer Anforderung schneller und vollständiger identifizieren können, als in einer Anforderungsmenge, die nicht verknüpft ist und keine Trace-Operationen besitzt.

Das Konzept teilt sich in zwei Teilkonzepte, welche die beiden Ebenen abdecken. Das Teilkonzept der Anforderungslandschaften erfüllt die Teilziele auf der methodischen Ebene, während das Teilkonzept der Trace-Operationen die Teilziele auf der konkreten Projektebene erfüllt. Abbildung 6 zeigt die beiden Teilkonzepte der Anforderungslandschaften und Trace-Operationen zusammen mit den Ansätzen, die in dieser Arbeit angegangen werden.

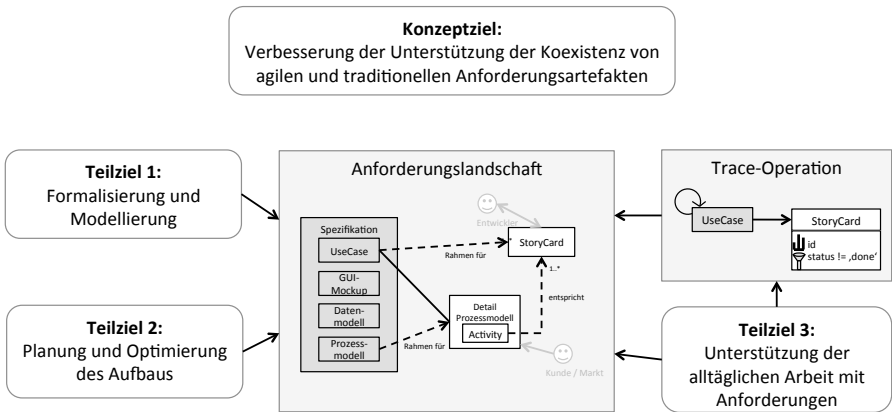


Abbildung 6: Überblick über Konzepte dieser Arbeit und deren Ziele

Die Teilziele werden jeweils in den folgenden Kapiteln aufgegriffen. Kapitel 6 befasst sich mit der Modellierung von Anforderungslandschaften. Kapitel 7 beschreibt die inhaltliche Optimierung von Anforderungslandschaften. In Kapitel 8 wird schließlich der Umgang mit konkreten Anforderungen im Projekt behandelt.

5.2 Zusammenhang zwischen Methodenebene und Projektebene

Wie im vorigen Abschnitt erwähnt, erarbeitet diese Dissertation Lösungen auf zwei Ebenen – der methodischen Ebene und der Projektebene. Auf den verschiedenen Ebenen arbeiten unterschiedliche Rollen im Projekt an unterschiedliche Arten von Artefakten. Dieser Abschnitt erklärt, die beiden Ebenen und zeigt den Zusammenhang zwischen ihnen auf. Zudem existiert eine dritte, übergeordnete, Ebene, die – wie auch in dieser Dissertation der Fall – Richtlinien zur Erstellung der Artefakte auf den jeweiligen Ebenen gibt.

Der Begriff *Methodik* bezeichnet die Sammlung an Methoden und Methodenwissen zu einem Fachgebiet. Eine Methodik in dieser Arbeit befasst sich demnach damit, die Definition und Auswahl von zu einem Projekt passenden Methoden zu ermöglichen. Eine konkrete Ausprägung in einem Projekt wird dann als eine Methode bezeichnet. Im Englischen wird die Methodik häufig als *methodology* bezeichnet. Dies kollidiert mit dem Begriff *Methodologie*, der im Deutschen als Meta-Wissenschaft allgemein für die Lehre von Methoden steht. In dieser Arbeit ist unter dem Begriff *methodology* stets die Methodik zu verstehen.

Der ISO-Standard 24744:2014 (*Software Engineering – Metamodel for Development Methodologies (SEMDM)*) [76] befasst sich mit der Definition von Methodiken im Software Engineering. Für die Definition einer Methodik sind nach der Herangehensweise des SEMDM prinzipiell drei Ebenen zu betrachten. Die Entwickler arbeiten in der *Endeavour Domain (Projektebene)*. Sie führen die Methode aus, die zu einem Projekt definiert ist. Die Methode schränkt damit auch das Handeln der Entwickler ein. Die *Definition der Methode* zu einem Projekt wird vom Methodeningenieur wiederum auf der *Methodology Domain (Methodik-Ebene)* vorgenommen. Was die Methodeningenieure bei einer Methode wiederum festlegen können, wird durch ein allgemeines Metamodell in der *Metamodel Domain (Metamodell-Ebene)* festgelegt.

Im SEMDM werden zwei Rollen definiert, die auf den verschiedenen Ebenen agieren. *Entwickler* sind diejenigen, welche eigentliche Work Products erzeugen, die der Methode unterliegen. *Method Engineers* sind die Personen, die diese Methoden festlegen. Im Requirements Engineering – und besonders in hybriden Vorgehensweisen – mischen sich Begrifflichkeiten bei dem Versuch, die Rollen des SEMDM mit konkreteren Rollen der jeweiligen Vorgehensweisen auszufüllen. Man kann dabei auf die folgenden Unklarheiten stoßen:

- (1) In traditionellen Projekten kann es gewünscht sein, statt allgemein von Entwicklern zu sprechen, zwischen Requirements Engineers, Architekten und Testern als Rollen in der Endeavour-Domain unterscheiden zu können.

5 Ein Konzept zur Unterstützung der Koexistenz von agilen und traditionellen Anforderungsartefakten

- (2) Andererseits können Requirements Engineers als weitere Aufgabe genau die Prozessdefinition des RE-Prozesses haben, womit sie sich dann auch auf der Methodik-Ebene bewegen. Auch dann können sie sich noch immer von weiteren Method-Engineers, die den gesamten Entwicklungsprozess im Blick haben, abgrenzen.
- (3) In agilen Vorgehensweisen sind hingegen häufig die Entwickler auch dafür verantwortlich, ihren Entwicklungsprozess kontinuierlich zu verbessern, zum Beispiel durch Retrospektiven in Scrum [38].
- (4) Zudem kann auch in einem agilen Projekt ein Requirements Engineer eingesetzt werden. Auch hier kann es dazu kommen, dass der Requirements Engineer die Methode festlegt und ebenfalls den Entwicklern dabei hilft, die konkreten Anforderungen aufzuschreiben.
- (5) In agilen (bzw. allgemeiner kundenorientierten) Methoden können auch Kunden für das Schreiben von Anforderungen verantwortlich (also auf der Endeavour-Ebene tätig) sein.

Um diesen Unklarheiten gerecht zu werden, werden die Rollenbezeichnungen angepasst. Statt Entwicklern werden Rollen, welche die eigentlichen Artefakte erzeugen, als Projektteilnehmer bezeichnet. Ein Methoden-Ingenieur, der sich konkret mit der Requirements Engineering-Methodik beschäftigt, wird als Requirements Methodiker bezeichnet.

Des Weiteren wird die Rolle des Requirements Engineers in dieser Arbeit als Rolle in der Projektebene gesetzt. Dies bedeutet, dass hier der Requirements Engineer für die Erstellung von Anforderungsartefakten verantwortlich ist. Ist ein Requirements Engineer auch an der Definition der Requirements Engineering-Methode beteiligt, so hat die Person zwei Rollen inne, nämlich die des Requirements Engineers und die des Requirements Methodikers.

Definition Projektteilnehmer:

Ein Projektteilnehmer ist jemand, der am Projekt (gemäß einer definierten Methode) beteiligt ist.

Definition Requirements Methodiker:

Ein Requirements Methodiker ist jemand, der eine Methode für ein Projekt (basierend auf einem Metamodell) definiert.

Anwendung des SEMDM-Ansatzes auf diese Arbeit

Angewendet auf diese Arbeit, ist es das Ziel, Koexistenz von agilen und traditionellen Artefakten zu unterstützen. Hierzu wird die SEMDM-Vorgehensweise auf Festlegung von Artefakten und Verknüpfungen angewendet. Die konkreten Modelle, die dabei zum Einsatz kommen, werden im folgenden erläutert. Ein Überblick ist in Abbildung 7 dargestellt.

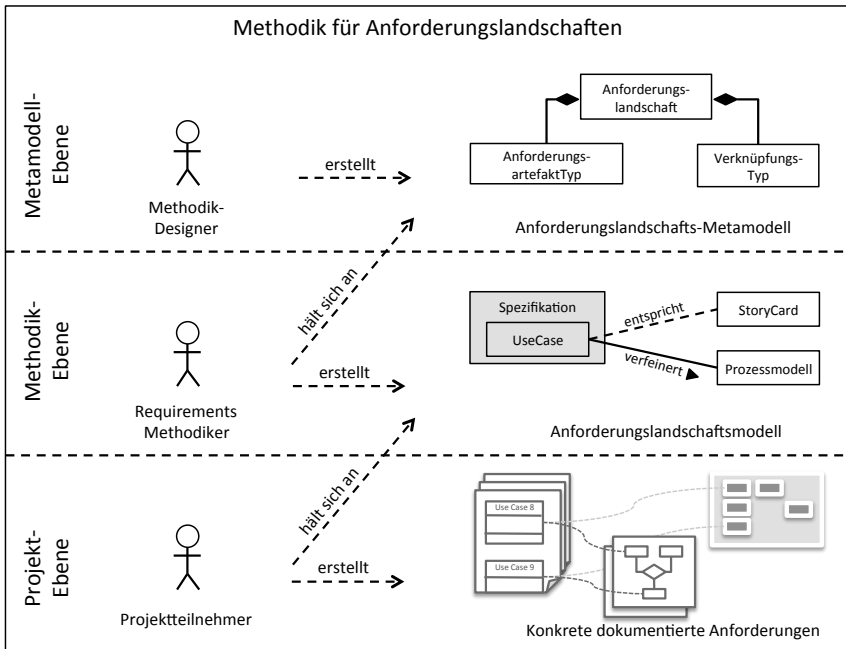


Abbildung 7: Übersicht über Anwendungsebenen der hier erarbeiteten Methodik

Projektebene (Endeavour Domain): Die einzelnen Projektbeteiligten gehen in ihrem Alltag mit einzelnen Instanzen um (in unserem Fall sind das einzelne Anforderungsartefakte). Zum Beispiel legen sie in der Iterationsplanung neue Stories an, passen bestehende Stories nach Kundenfeedback an und müssen zum Beispiel auch die entsprechenden Stellen in einem Prozessdiagramm, das den umzusetzenden Prozess zusammenfasst, anpassen. Um die richtigen Stellen schnell zu finden, haben sie einige der Stories mit Prozesseilen verknüpft.

Methodik-Ebene (Methodology Domain): Welche Arten von Artefakten sie dabei überhaupt verwenden und ob sie diese verknüpfen, wird dabei als Teil der Methode festgelegt. Dies kann unbewusst geschehen, indem ein Projekt sich an ein bekanntes Vorgehensmodell hält, das bestimmte Artefakttypen vordefiniert, zum Beispiel Epics und User Stories in Scrum-Projekten [30], [145]. Alternativ kann zu Projektbeginn auch bewusst festgelegt werden, dass bestimmte Artefakttypen verwendet werden sollen. Entweder wird hierzu eine eigene Methode (projekt- oder auch unternehmensweit) festgelegt oder eine bestehende Methode nach eigenen Bedürfnissen zugeschnitten, was auch unter dem Begriff Tailoring bekannt ist [5], [82].

Gerade bei hybriden Vorgehensweisen ist es wichtig, bewusst die Bedürfnisse eines Projektes zu analysieren und bewusst durch Tailoring die dem Projekt

5 Ein Konzept zur Unterstützung der Koexistenz von agilen und traditionellen Anforderungsartefakten

angemessene Form der Dokumentation festzulegen [5], [11], [16], [27]. Tut man dies nicht, kann es schnell passieren, dass unnötige Dokumente die Agilität behindern oder auch umgekehrt bei fehlender Dokumentation das Einhalten eines fest vorgegebenen Produktrahmens versäumt wird.

Hierbei ist die Festlegung und Optimierung der Methode jedoch nicht nur auf den Projektbeginn eingeschränkt. Zwar wird dort das grundsätzliche Vorgehen festgelegt, doch kommen – wie sich in der Interview-Studie gezeigt hat – oft weitere Artefakte erst später hinzu. Beispielsweise führen Entwickler auch im Verlauf der Entwicklung neue Artefakte ein, wenn sie feststellen, dass sie so mit der Kundenseite besser kommunizieren können. Besonders in Hinblick auf eine effiziente Einbettung nachträglich hinzugekommener Artefakte ist es dann wichtig, solche Änderungen auf Methodenebene zur Kenntnis zu nehmen.

Konkret werden im vorgestellten Konzept dieser Arbeit auf Methodik-Ebene die Artefakttypen und deren Verknüpfungsmöglichkeiten festgelegt, die in einem Projekt verwendet werden sollen. Diese Anforderungs-Konfigurationen werden in der Arbeit als *Anforderungslandschaften* bezeichnet und durch Anforderungslandschaftsmodelle visualisiert.

Metamodell-Ebene (Metamodel Domain): Was genau der Methodiker zu den einzelnen Typen (und auch zum Aufbau der Instanzen) festlegen muss und darf, wird wiederum in einem Metamodell beschrieben. Ähnliche Metamodelle, die Artefakte und deren Verknüpfungen aufgreifen, wurden von Espinoza [96], Adersberger [2], Méndez [111], Mäder [109] sowie im SEMDM-Standard selbst [76] vorgestellt. Ein relativ allgemeines Metamodell findet sich in Arbeiten von Ramesh und Jarke [132]. Diese können durchaus herangezogen werden, um zusätzliche Aspekte zu definieren.

Das vorliegende Metamodell unterscheidet sich vor allem dadurch von bisherigen Ansätzen, dass es neben expliziten Verknüpfungen auch prinzipielle Relationen darstellt. Außerdem wird ein spezieller Fokus auf das Verhalten von Anforderungstypen bei Änderungen (in dieser Arbeit als *Zähheit* bezeichnet) gelegt.

Um nun die konkreten Modelle für die verschiedenen Ebenen zu erarbeiten, werden zunächst die relevanten Elemente und Mechanismen auf Instanzebene betrachtet, welche die Koexistenz von agilen und traditionellen Anforderungsartefakten charakterisieren oder beeinflussen. Anschließend werden aus diesen Erkenntnissen die zu betrachtenden Eigenschaften auf Typebene abgeleitet.

6 Hybride Anforderungslandschaften

6.1 Anforderungsartefakte

Anforderungen werden in vielen Softwareprojekten auf Artefakten festgehalten. Dies können Dokumente sein, aber auch Teile innerhalb eines Dokumentes. Oft werden Anforderungsartefakte auch als Objekte in virtuellen Projektumgebungen, wie DOORS- oder JIRA-Projekten festgehalten und dann gar nicht mehr in Dokumenten abgelegt. Eine Anforderung ist dabei nach IEEE-Standard 24765:2010 [72] wie folgt definiert:

Eine **Anforderung** ist (nach [72] sowie [129]):

1. Eine Bedingung oder Eigenschaft, die ein System oder eine Person benötigt, um ein Problem zu lösen oder ein Ziel zu erreichen
2. Eine Bedingung oder Eigenschaft, die ein System oder eine Systemkomponente aufweisen muss, um einem Vertrag zu erfüllen oder einem Standard, einer Spezifikation oder einem andern formell auferlegten Dokument zu genügen
3. Eine dokumentierte Repräsentation einer Bedingung oder Eigenschaft wie in (1) oder (2) definiert

Wie am dritten Teil der Definition sichtbar, meint der Begriff Anforderung sowohl implizite als auch explizit dokumentierte Anforderungen [129]. Diese Arbeit beschäftigt sich damit, wie Anforderungen dokumentiert werden, und wie diese Repräsentationen dann in der Softwareentwicklung verwendet werden. Wird eine Anforderung dokumentiert, so wird diese in einem *Anforderungsartefakt* festgehalten.

Definition Anforderungsartefakt

Ein Anforderungsartefakt ist eine referenzierbare dokumentierte Einheit, die mindestens eine Anforderung enthält.

Anforderungsartefakte können sehr grobgranular sein. Ein grobes Anforderungsartefakt ist beispielsweise die gesamte Spezifikation, die eine ganze Reihe verschiedenartiger Anforderungen enthält oder ein Geschäftsprozessmodell, das den Prozess, den das System unterstützen soll, zeigt. Diese groben Artefakte lassen sich in feinere Artefakte unterteilen. Eine Spezifikation kann zum Beispiel Use Cases enthalten. Ein Use Case ist ein Anforderungsartefakt, das ein Szenario zu einem bestimmten Ziel beschreibt. Use Cases können sogar weiter in Use Case-Schritte als Anforderungsartefakte unterteilt werden, sofern die Schritte einzeln als *referenzierbare* Einheiten aufgeführt sind. Ist der Use Case im Casual-Style [29] verfasst – d.h. er enthält nur einen kurzen Fließtext, der

einen Ablauf beschreibt, ohne die einzelnen Schritte zu nummerieren – so ist der ganze Use Case bereits das feinste Anforderungsartefakt.

Die Referenzierbarkeit eines Anforderungsartefaktes ist in dieser Arbeit wichtig, weil jedes explizit aufgeführte Anforderungsartefakt prinzipiell mit einem anderen Anforderungsartefakt verknüpfbar sein soll. Eine Verknüpfung wiederum ist genau eine Referenz, die vom einen auf das andere Anforderungsartefakt verweist.

Die Verfeinerung ist nützlich, um die Beziehungen zwischen Anforderungsartefakten genauer darstellen zu können. Wird einer Story Card ein ganzes Geschäftsprozessmodell zugeordnet, so ist zwar klar, dass die Story Card irgendwo in diesem Prozessmodell zu finden sein wird. Unter Umständen muss das Prozessmodell jedoch lange durchsucht werden, weil sich die Story Card nur auf einen kleinen Teil im Modell bezieht. Wird der Story Card hingegen eine konkrete Prozessaktivität zugeordnet, so ist die relevante Stelle schneller gefunden und die Beziehung ist präziser dargestellt.

Ein Anforderungsartefakt grenzt sich von anderen Projektartefakten dadurch ab, dass im Anforderungsartefakt eine oder mehrere Anforderungen enthalten sein müssen. Andere Artefakte, wie der Feinentwurf oder Code sind auch Projektartefakte, jedoch keine Anforderungsartefakte. Dennoch ist die Grenze nicht immer klar zu ziehen. In agilen Projekten werden häufig Akzeptanztests als Konkretisierung einer Story Card mit zu den Anforderungen gezählt [8], [30]. Je nach Granularität können Schnittstellenbeschreibungen und Datenmodelle ebenfalls als Design- oder Anforderungsartefakte zählen. Das Volere Requirements Specification Template [137] sieht beispielsweise ein Datenmodell mit den wichtigsten Domänenobjekten und Schnittstellenbeschreibungen zu angrenzenden Systemen als Anforderungsartefakte vor. Im Rational Unified Process [87] hingegen werden Schnittstellenbeschreibungen zu einzelnen Klassen oder das Datenmodell als Design-Artefakte verwendet.

Auch hier ist zur Unterscheidung die Definition des Begriffes Anforderung heranzuziehen. Wenn ein Artefakt verändert werden kann ohne dass davon die Ziele eines Akteurs oder ein Standard berührt wird, so handelt es sich um ein anderes Projektartefakt. Wenn hingegen eine Änderung eines Artefaktes dazu führt, dass ein Ziel eines Akteurs, ein Standard oder eine Spezifikation nicht mehr korrekt dokumentiert ist, so handelt es sich um ein Anforderungsartefakt.

6.1.1 Zähheit von Anforderungsartefakten

Es ist wichtig, zu einem Artefakt zu betrachten, wie es sich bei Änderungen verhält. Wenn neue Anforderungen, Feedback oder auch einfach neue Erkenntnisse im Projekt entstehen, so ist es wichtig, auch diese zu dokumentieren, da sonst die Dokumentation veraltet. Jedoch sollten die Anforderungen auch nicht ständig beliebig geändert werden. Ein wichtiger Aspekt der Dokumentation –

besonders in traditionellen Projekten – ist auch, geplantes festzuhalten, um sicherzustellen, dass dieses auch eingehalten wird [141]. Ob die Anforderungen, die in einem Artefakt festgehalten sind, ohne weiteres geändert werden können oder nicht, soll im Rahmen dieser Arbeit durch die *Zähheit* festgehalten werden. Eine Story Card-Menge beispielsweise ist in der Regel jederzeit anpassbar [8], indem Story Cards hinzugefügt, geändert oder gelöscht werden können. So eine Menge ist sehr *flexibel*.

Eine Spezifikation hingegen wird in der Regel von Kunden- und Entwicklerseite unterzeichnet und ist ab diesem Zeitpunkt nicht ohne weiteres änderbar. Um eine Änderung zu bewirken, müssen beispielsweise erst beide Seiten zustimmen. Oft ist eine solche Änderung auch mit weiteren Kosten verbunden. Häufig werden für das Management von Änderungen zusätzliche Change Request-Artefakte verwendet und separat dokumentiert. Hierbei ist auch möglich, dass eine Änderung abgelehnt wird. Eine solche Spezifikation ist ein zäheres Anforderungsartefakt, als die eben erwähnte Story Card-Menge.

Die Zähheit eines Artefaktes wird jedoch nicht durch das Artefakt selbst, sondern eigentlich durch den dazu festgelegten Prozess bestimmt. Es ist durchaus möglich, eine Spezifikation frei änderbar zu gestalten.

Im Folgenden wird genauer festgelegt, wann ein Artefakt als zäh gilt. Grundsätzlich gibt es zwei Situationen, in denen die Änderung einer Anforderung in einem Anforderungsartefakt erschwert wird: (1) ein Projektteilnehmer oder eine andere projektbetroffene Person *will* nicht, dass sich eine Anforderung bzw. die damit zusammenhängende Funktionalität ändert, oder (2) ein Projektteilnehmer *kann* eine Änderung nicht effizient im Artefakt dokumentieren.

Im ersten Fall wird im Rahmen dieser Arbeit von *prozessbezogenen zähen Artefakten* gesprochen. Hier geht es darum, dass die Inhalte der Artefakte einen (Soll-)Zustand darstellen, der genauso gewünscht ist und auf den sich andere Parteien möglicherweise verlassen. Eine Änderung in einem solchen Artefakt ist ein Anzeichen dafür, dass sich die Funktionalität so weit ändern könnte, dass der gewünschte Soll-Zustand nicht mehr erreicht wird oder dass abhängige Parteien, wie benachbarte Projekte, größere Änderungen durchführen müssen, um sich auf den neuen Zustand einzustellen. Um solche Änderungen zu kontrollieren, wird im Prozess festgelegt, ob und wann diese stattfinden dürfen.

Definition „prozessbezogenes zähes Anforderungsartefakt“:

Ein Anforderungsartefakt gilt als *prozessbezogenes zähes Anforderungsartefakt*, wenn zu dem Artefakt eine Vereinbarung zwischen mindestens zwei Parteien aus dem Projekt oder dem Projektkontext darüber existiert, dass sich die Inhalte nur in Absprache ändern sollen. Diese Absprache kann eine formelle oder informelle Form besitzen, sie kann in der Regel jedoch nicht ad hoc stattfinden.

Ein prozessbezogenes zähes Artefakt wird häufig verwendet, um einen festen Rahmen für bestimmte Funktionalitäten zu definieren. Dabei können die Parteien *aus dem Projekt oder dem Projektkontext* [141] stammen. Die häufigste hier auftretende Situation ist, dass sich die Kunden- und Entwicklerseite als Projektparteien auf bestimmte Inhalte einigen wollen. Doch auch Parteien aus dem Projektkontext können ein Anliegen daran haben, bestimmte Teile des Produktes festzulegen. Solche Parteien können angrenzende Projekte oder angrenzende Systeme sein. Beispielsweise kann es ein projektübergreifendes Geschäftsprozess oder ein projektübergreifendes Datenmodell geben, das von mehreren Projekten, Systemen oder Abteilungen verwendet wird. Es macht Sinn, diese zu dokumentieren, damit die Projekte sich dort einordnen können und auch wissen, wovon sie ausgehen können oder wovon die anderen Projekte ausgehen. Wenn es angrenzende Systeme gibt, so kann es sinnvoll sein, eine Beschreibung der gemeinsamen Schnittstelle in die jeweiligen Anforderungslandschaften der beiden Systeme mit aufzunehmen.

In der Regel dient die Vereinbarung mindestens einem der folgenden Zwecke:

- Absicherung des Kunden, dass alle gewünschten Funktionen umgesetzt werden.
- Absicherung der Entwicklerseite, dass die Anforderungen sich nicht mehr von Kundenseite ändern. Dies ist besonders wichtig, wenn es mehrere Stakeholder mit Interessenskonflikten im Projekt gibt. Wird hier nicht ausreichend stark festgelegt in welchem Rahmen sich das Projekt bewegt, so kann es zu einer *Überanpassung der Anforderungen* kommen, da die Kundenseite immer wieder ihre Meinung zum Projektrahmen ändert (vgl. [16]).
- Absicherung weiterer Parteien aus dem Projekt oder Projektkontext, dass bestimmte Funktionalitäten oder Schnittstellen vorhanden sein werden.

Häufig wird die Vereinbarung formell dadurch getroffen, dass alle betroffenen Parteien das Anforderungsartefakt unterschreiben. Nachfolgende Änderungen am Artefakt sind dann nur mit einer neuen Unterschrift aller Parteien zu beschließen. Jedoch gibt es auch weniger formelle Absprachen, die dennoch dieselben Konsequenzen mit sich führen. Beispielsweise ist auch denkbar, dass der Prozess definiert, dass beide Parteien in einem bestimmten Meeting, wie einem Releaseplanungstreffen [94], mündlich zustimmen müssen, wenn eine Änderung durchgeführt wird.

Wirklich *zäh* ist ein Artefakt dabei aber nur, wenn die Absprache nicht sofort stattfinden kann. Nur dann verzögert sich die Änderung der Inhalte tatsächlich. Wenn im Gegensatz dazu zwar eine Absprache notwendig ist, diese aber größtenteils direkt stattfindet, so gilt das Artefakt dennoch als flexibel. Als Beispiel ist hier eine Story Card-Menge denkbar, bei der festgelegt wird, dass nur Story Cards eingefügt werden dürfen, wenn der Product Owner mit diesen einverstan-

den ist. Diese Menge würde nach der reinen Definition (ohne Betrachtung, dass die Absprache nicht ad hoc stattfinden kann) als zäh gelten. Diese Charakterisierung wäre unzutreffend, wenn der Product Owner die Entscheidung über eine Änderung stets direkt bei der Entstehung der neuen Story Card treffen kann. Dann verhält sich die Story Card-Menge nämlich wie eine flexible Story Card-Menge, bei der Änderungen direkt eingepflegt werden dürfen.

Ein Artefakt, das prozessbezogen zäh ist, wird erst mit der ersten Absprache (also zum Beispiel mit der ersten Unterschrift) zäh. Vorher befindet sich das Artefakt häufig in einer Erstellungsphase (bzw. Analysephase), bei der Änderungen häufig auftreten und flexibel eingearbeitet werden können. In dieser Arbeit wird die Unterscheidung dieser beiden Phasen nicht explizit angegangen. Für eine Anforderungslandschaft wird die Zähheit eines Artefaktes zum Zeitpunkt seiner Verwendung betrachtet. Damit wird beispielsweise eine Spezifikation, die zwar in der Analysephase flexibel geändert werden kann, dann während der Entwicklung jedoch zäh sein wird, in der Regel direkt als zähes Artefakt gekennzeichnet. Damit wird bezweckt, dass die typischen Herausforderungen zäher Artefakte direkt sichtbar sind und nicht von der aktuellen Projektphase abhängen.

Der zweite genannte Fall von Zähheit – dass ein Projektteilnehmer eine Änderung nicht effizient im Artefakt dokumentieren *kann* – wird *strukturelle Zähheit* genannt. Hier stammt die Verzögerung aus der Beschaffenheit des Artefaktes selbst. Wenn ein Artefakt beispielsweise sehr lang ist, wenig strukturiert ist oder viele abhängige Informationen besitzt, so ist das Einpflegen von Änderungen schwieriger, als bei einem simplen oder kurzen Artefakt.

Diese Hürde kann ähnliche Effekte haben, wie die von prozessbezogenen zähen Artefakten: wenn Änderungen in einem strukturell zähen Artefakt nachgepflegt werden müssen, so kann dies zu Verzögerungen der gesamten Dokumentation führen; aufgrund der höheren Komplexität kann sich erst nach einiger Zeit herausstellen, dass eine Änderung doch nicht möglich ist (beispielsweise weil man erst nach einigen Dokumentationsschritten ein Detail findet, das aufzeigt, warum die gewünschte Änderung gar nicht möglich ist), sodass diese revidiert werden muss; aufgrund des höheren Aufwandes versuchen Projektteilnehmer zu vermeiden, Änderungen zu dokumentieren.

Definition „strukturell zähes Anforderungsartefakt“:

Ein Anforderungsartefakt gilt als *strukturell zähes Anforderungsartefakt*, wenn mindestens ein Projektteilnehmer, der das Artefakt verwendet, empfindet, dass das Artefakt einen hohen Aufwand zum Einpflegen der Änderung erfordert.

Die strukturelle Zähheit eines Artefaktes ist subjektiv und kann sich von Projektteilnehmer zu Projektteilnehmer unterscheiden. Im Rahmen dieser Arbeit soll im Team über strukturell zähe Artefakte hauptsächlich diskutiert werden

und gegebenenfalls nach Artefakten gesucht werden, die eine änderungsfreundlichere Struktur besitzen. Dafür reicht die gegebene Definition zunächst aus.

Auch die strukturelle Zähheit eines Artefaktes kann sich mit der Zeit ändern. Meist geschieht dies unvorhergesehen, indem das Artefakt wächst und erst mit der Zeit so groß oder komplex wird, dass die Anwender es als schlecht handhabbar empfinden. Solche Änderungen der Zähheit eines Artefaktes – aber auch andere Veränderungen der Anforderungslandschaft – zu bemerken, kann sinnvoll sein, um darauf frühzeitig reagieren zu können. Um dies zu ermöglichen, sollte die Anforderungslandschaft in regelmäßigen Abständen evaluiert werden.

In dieser Arbeit sind beide Formen der Zähheit relevant, da beide Formen zu Verzögerungen oder Ausfällen der Dokumentation führen können. Daher werden beide Varianten in Anforderungslandschaften aufgeführt und unter dem Oberbegriff der Zähheit zusammengefasst. Ein Unterschied besteht darin, wie man auf die jeweilige Art von Zähheit einwirken kann. Ein prozessbezogen zähes Artefakt wird flexibler, indem im Prozess festgelegt wird, dass das Artefakt oder Teile des Artefaktes ohne besondere Zustimmung verändert werden dürfen. Ein strukturell zähes Artefakt wird flexibler, indem seine Struktur verändert wird oder gar ein anderer Artefakttyp zur Dokumentation der entsprechenden Information verwendet wird.

Sofern nicht anders gekennzeichnet, wird im folgenden Verlauf der Arbeit ein Artefakt als *zäh* bezeichnet, wenn es prozessbezogen zäh oder strukturell zäh oder beides ist.

Definition „zähes Anforderungsartefakt“:

Ein Anforderungsartefakt gilt als zähes Anforderungsartefakt, wenn mindestens eine der folgenden zwei Bedingungen erfüllt ist:

1. Das Artefakt ist prozessbezogen zäh
oder
2. Das Artefakt ist strukturell zäh.

Definition „flexibles Anforderungsartefakt“:

Ein Artefakt, das nicht zäh ist, wird als *flexibles Artefakt* bezeichnet.

So, wie die Zähheit hier definiert ist, besitzt ein Artefakt nur eine von zwei Ausprägungen: es kann nur zäh oder flexibel sein. Prinzipiell wäre es möglich, die Zähheit genauer festzulegen. Beispielsweise kann man zwischen einer bedingten Zähheit – die gilt, wenn über eine Änderung abgestimmt werden darf – und einer absoluten Zähheit – die besagt, dass ein Artefakt in jedem Fall nicht geändert werden darf – unterscheiden. Noch genauer könnte man die Zähheit als

Zahlenwert festlegen, der die durchschnittliche Dauer zur Entscheidung über eine Änderung oder deren tatsächlicher Dokumentation repräsentiert. Eine genauere Unterscheidung zwischen besonders zähen und weniger zähen Artefakten erlaubt es, beispielsweise besonders problematische Stellen klarer zu erkennen oder abgestufte Maßnahmen einzuführen. Im Gegenzug ist es aufwändiger und fehlerbehafteter, den richtigen Wert der Zähigkeit eines Artefaktes zu schätzen. Für die Maßnahmen, die in dieser Arbeit vorgestellt werden, ist eine Unterscheidung zwischen verschiedenen Stufen nicht ausschlaggebend. Daher wird im Rahmen dieser Arbeit die simpelste Variante der Charakterisierung – die binäre Unterscheidung zwischen zähen und flexiblen Artefakten – verwendet.

6.1.2 Agile und traditionelle Anforderungsartefakte

Es gibt ein allgemeines Verständnis dazu, was agile und traditionelle Anforderungsartefakte sind, welches in dieser Arbeit jedoch differenzierter betrachtet werden soll. Grundsätzlich lässt es sich nicht unbedingt am Artefakt selbst ausmachen, ob es agil oder traditionell ist. Es ist der gesamte Entwicklungsansatz, der als agil oder dokumentationsgetrieben bezeichnet wird. Entsprechend bestimmt erst die Verwendung eines Artefaktes und die darum herum angewendeten Werte, Prinzipien und Prozesse, ob tatsächlich ein agiler oder traditioneller Entwicklungsansatz unterstützt wird.

Beispielsweise können sehr detaillierte Use Cases starr in eine Spezifikation aufgenommen werden, die dann möglichst genauso umgesetzt werden sollen. Andererseits kann eine Use Case-Menge, wie in der agilen Methode Crystal Clear [27], nach und nach weiterentwickelt und auch immer wieder umpriorisiert oder abgeändert werden. Use Cases können also sowohl agil als auch dokumentationsgetrieben sein. Wie bereits an diesem Beispiel zu erkennen ist, spielen weitere Eigenschaften, wie die Zähigkeit des Artefaktes, eine wichtige Rolle.

Dennoch geben die Artefakte bereits einen wichtigen Hinweis darauf, ob sie eher agiles oder dokumentationsgetriebenes Vorgehen ermöglichen und unterstützen. Daher soll in dieser Arbeit zwischen agilen und traditionellen Artefakten unterschieden werden. Diese Arbeit nähert sich den Begriffen agiler und traditioneller Anforderungsartefakte, indem damit Artefakte, die besonders gut agile oder traditionelle Ansätze und Praktiken unterstützen, bezeichnet werden.

Agile Anforderungsartefakte

Allgemein werden agile Ansätze durch eine inkrementelle bzw. iterative Herangehensweise an die Exploration der Lösung getrieben. In der Literatur wird unter agilen Anforderungen häufig direkt die Verwendung von User Stories verstanden. Beispielsweise stellen Boehm und Turner die Gemeinsamkeiten für

agile Methoden auf und beschreiben dort „*Most agile methods express requirements in terms of adjustable informal stories.*“ bzw. „*Agile uses informal, user-prioritized stories as requirements.*“ ([16], S. 37). Auch Heeager, die untersucht hat, wie typische dokumentationsgetriebene und agile Projekte aussehen und wie sich beides *mischen* lässt, charakterisiert agile Artefakte als „*Agile methods rely on user stories written by the customer in a plain business-like language as the requirements.*“ ([66], S. 12).

Jedoch können auch weitere Artefakte zu agilen Anforderungsartefakten zählen. Wie bereits erwähnt, bauen die Crystal Methoden auf Use Cases auf [27]. Im Agile Modeling-Ansatz von Ambler können verschiedene stärker oder schwächer formale Formen von User Stories und Use Cases und zusätzlich UML-Modelle als Anforderungsartefakte verwendet werden [5]. Auch im Rational Unified Process, der unter anderem auf agile Art verwendet werden kann [5], wird eine Reihe weiterer Anforderungsartefakte genannt.

Die relevanten Eigenschaften, die agile Artefakte erfüllen müssen, werden bei der Betrachtung der agilen Werte und Prinzipien klar. Werte, die sich auf die Dokumentation von Anforderungen beziehen sind „*[We value] working software over comprehensive documentation*“ und „*[We value] responding to change over following a plan*“ [51]. Relevante Prinzipien, die sich auf einzelne Anforderungsartefakte anwenden lassen, sind: „*Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*“ und „*Welcome changing requirements, even late in development. [...]*“ [51].

Um einen agilen Ansatz zu unterstützen, muss ein Anforderungsartefakt es erlauben, gut mit Änderungen umzugehen. Hierzu muss es flexibel sein (nach oben genannter Definition). Das bedeutet, dass es vom Prozess her, aber auch durch leichtgewichtige Struktur erlaubt, Änderungen schnell und direkt vorzunehmen. Ebenfalls ist inkrementelles Vorgehen in agilen Entwicklungsansätzen wichtig. Für die Anforderungsartefakte bedeutet dies vor allem, dass sie nicht nur flexibel sind, also Änderungen erlauben, sondern dass sie adaptiv sind, also Änderungen auch tatsächlich vorkommen. Das bedeutet, dass die Artefakte von Anfang an nicht vollständig sind und mit der Zeit erweitert werden.

Definition agiles Anforderungsartefakt

Ein agiles Anforderungsartefakt ist ein Artefakt, das einen agilen Entwicklungsansatz unterstützt.

Um agil sein zu können, muss ein Anforderungsartefakt flexibel und adaptiv sein.

Typische agile Anforderungsartefakte sind User Stories, Epics, Use Cases, GUI-Mockups aber auch UML-Modelle.

Traditionelle Anforderungsartefakte

Traditionelle Artefakte sollen ein präskriptives Vorgehen unterstützen. Dies bedeutet, dass sie es erlauben, Aspekte eines Projektes im Voraus zu planen und dass diese Aspekte dann genau so umgesetzt werden sollen. Boehm und Turner charakterisieren traditionelle Artefakte durch „*plan-driven methods generally prefer formally baselined, complete, consistent, traceable and testable specifications*“ ([16], S. 37).

Ein wichtiger Aspekt für diese Arbeit ist, dass die Artefakte vollständig sind. Hiermit ist gemeint, dass es nach der Fertigstellung eines Artefaktes idealerweise keine weiteren Aspekte, wie zum Beispiel Anwendungsfälle, gibt, die unterstützt werden sollen, jedoch bisher nicht dokumentiert wurden. Solche zusätzliche Funktionalität würde *aus dem Rahmen* einer traditionellen Spezifikation *fallen*. In Zusammenhang mit der Vollständigkeit steht auch, dass traditionelle Anforderungsartefakte *zäh* (bzw. genauer *prozessbezogen zäh* gemäß der hier festgelegten Definition) sind. Wenn einmal mehrere Parteien bestätigt haben, dass die im Artefakt festgehaltenen Anforderungen vollständig sind, so sollte eine einzelne Partei nicht ohne weiteres beispielsweise zusätzliche Inhalte hinzufügen können.

Zähe Anforderungsartefakte sind für dokumentationsgetriebene Entwicklungsansätze wichtig, da sich hier viele Projektteilnehmer in ihren Aktivitäten auf die Inhalte in der Dokumentation verlassen. Treten Änderungen auf, so führt das hier verstärkt zu Nacharbeit und Kosten.

Definition traditionelles Anforderungsartefakt

Ein traditionelles Anforderungsartefakt ist ein Artefakt, das einen dokumentationsgetriebenen Entwicklungsansatz unterstützt.

Um dokumentgetriebene Entwicklung zu unterstützen, muss ein Anforderungsartefakt vollständig und prozessbezogen *zäh* sein.

Typische traditionelle Anforderungsartefakte sind natürlichsprachliche Spezifikationen, Use Cases, Geschäftsprozessmodelle, GUI-Mockups sowie UML-Modelle.

6.2 Beziehungen zwischen Anforderungsartefakten

Anforderungen beeinflussen sich häufig gegenseitig. Carlshamre et al. [23] haben die Releaseplanung – also die Planung, welche der Anforderungen in welchem Release umgesetzt werden – in mehreren Projekten untersucht und dabei festgestellt, dass oft weniger als 20 % der Anforderungen in einem Projekt unabhängig von anderen Anforderungen sind. Häufige Abhängigkeiten dort waren, dass eine Anforderung nur umgesetzt werden kann, wenn auch eine

andere umgesetzt wird, oder dass die Umsetzung einer Anforderung die Kosten oder den Kundenwert einer anderen Anforderung beeinflusst.

Besonders interessant sind auch Beziehungen zwischen Anforderungsartefakten unterschiedlicher Artefakttypen. Die parallele Verwendung verschiedener Typen von Anforderungsartefakten taucht an vielen Stellen auf:

In der Interview-Studie aus Abschnitt 4 wird festgestellt, dass in vielen Projekten mehrere Artefakttypen verwendet werden [97]. Méndez stellt einen artefaktorientierten Ansatz vor, in dem eine ganze Fülle an Artefakten auftaucht [111]. Rupp empfiehlt, textuelle mit grafischen Artefakten zu kombinieren [141]. Die Spezifikationsempfehlungen im Volere-Template, sowie im IEEE-Standard 29148:2011 sehen ebenfalls unterschiedliche textuelle oder grafische Artefakte zu unterschiedlichen Blickwinkeln vor, die in einer Spezifikation kombiniert werden sollten [73], [137]. Neben funktionalen und nichtfunktionalen Anforderungen können hier beispielsweise Geschäftsprozessmodelle, Oberflächenskizzen oder Datenmodelle verwendet werden. Im V-Modell XT oder im RUP werden verschiedene Dokumente erstellt, die in verschiedenen Phasen des Softwareentwicklungsprozesses eingesetzt werden [52], [87]. In agilen Methoden werden primär nur User Stories verwendet, um Anforderungen zu dokumentieren. Jedoch wird auch hier empfohlen, gegebenenfalls weitere Artefakte zu verwenden, wenn dies hilfreich ist [5], [8].

Die einzelnen Artefakttypen stellen selten isolierte Inhalte dar, sondern eher verschiedene Blickwinkel auf dieselbe gewünschte Funktionalität. Während beispielsweise Use Cases die nach außen hin sichtbaren Abläufe fokussieren, können Geschäftsprozessmodelle Abhängigkeiten zwischen allgemeinen Prozessschritten darstellen. GUI-Mockups wiederum zeigen, wie die einzelnen Funktionen auf der Oberfläche aufgerufen werden können. Neben der Darstellung aus verschiedenen Blickwinkeln kommt es auch häufig vor, dass mehrere Artefakttypen aufeinander aufbauen und sich gegenseitig konkretisieren oder formalisieren und auch so Abhängigkeiten besitzen. Ein weiteres Beispiel stellt die Beeinflussung funktionaler Anforderungen durch nichtfunktionale Anforderungen dar. Wie Glinz in [57] beschreibt, kann dabei eine nichtfunktionale Anforderung ein einzelnes Anforderungsartefakt, wie einen Use Case, betreffen. Sie kann sich aber auch auf das System als ganzes beziehen oder einen Einfluss auf eine Teilmenge der funktionalen Anforderungen haben. Gerade im letzten Fall ist es notwendig, die Abhängigkeiten durch explizite Verknüpfungen darzustellen, um die nichtfunktionalen Anforderungen effektiv zu repräsentieren.

Werden mehrere sich beeinflussende Artefakttypen in einem Projekt verwendet, so führt dies oft auch zu zusätzlichen Herausforderungen, wie den folgenden (vgl. [97]):

- Zusätzlicher Aufwand, da mehrere Artefakte mit ähnlichen Inhalten *erzeugt* werden müssen
- Unsicherheit, ob die Informationen aus einem Artefakttyp *vollständig* im anderen berücksichtigt sind
- Bei *Änderungen* müssen mehrere Artefakte nachgeführt werden
- Häufig treten auch trotz guten Änderungsmanagements *Inkonsistenzen* auf
- Es ist schwieriger, relevante Informationen in mehreren Artefakten zu *finden*

Dahlstedt und Persson [35] beschreiben, dass die Kenntnis der Abhängigkeiten zwischen Anforderungen in einer Fülle von Aktivitäten, wie Change Management, Impact Analyse, Releaseplanung, Wiederverwendung von Anforderungen, Design, Implementierung und Testen, hilfreich sein kann. Zu einer Anforderungslandschaft sollten also neben den verwendeten Artefakten auch stets ihre Beziehungen betrachtet sowie der Umgang der Projektteilnehmer mit den Beziehungen unterstützt werden.

In dieser Arbeit wird – im Gegensatz zu den meisten Ansätzen in der Literatur – insbesondere zwischen einer *Relation zwischen zwei Artefakten* und einer *expliziten Verknüpfung zwischen zwei Artefakten* unterschieden. In Anlehnung an Ramesh und Jarke [132] wird mit Verknüpfung ein dokumentierter Link bezeichnet.

Definition „Verknüpfung zwischen zwei Anforderungsartefakten“

Eine Verknüpfung zwischen zwei Anforderungsartefakten ist ein *dokumentierter Link* zwischen den beiden Artefakten.

Der dokumentierte Link wird häufig als *Traceability Link* bezeichnet [132]. Der Link kann beispielsweise in den Artefakten oder auch in einer Traceability-Matrix [160] hinterlegt sein. Wichtig im Rahmen dieser Arbeit ist, dass man über den Link von einem Artefakt zum anderen navigieren kann und somit das verknüpfte Artefakt schnell findet.

Zwei Artefakte, die in Abhängigkeit zueinander stehen, tun dies nun aber unabhängig davon, ob eine explizite Verknüpfung zwischen ihnen definiert ist oder nicht. Auch wenn zwei Artefakte nicht verknüpft sind, kann es beispielsweise gewünscht sein, nach der Änderung eines Artefaktes ein abhängiges anderes Artefakt anzupassen. Mit der *Relation* zwischen zwei Artefakten wird in dieser Arbeit diese grundsätzliche Abhängigkeit charakterisiert.

Definition „Relation zwischen zwei Anforderungsartefakten“

Eine Relation zwischen zwei Anforderungsartefakten gibt an, in welcher Abhängigkeits-Beziehung diese beiden Artefakte stehen.

Explizite Verknüpfungen unterstützen den Umgang mit Relationen, indem sie die Abhängigkeitsrelationen dokumentieren und somit deren Kenntnis bei den Projektteilnehmern potenziell steigern. Zudem ermöglichen die Links eine manuelle oder auch automatisierte Navigation zu abhängigen Artefakten sowie automatisierte Operationen darauf, was die Ausführung von anforderungsbezogenen Aktivitäten ebenfalls potenziell verbessert. Automatisierte Operationen, die Traces nutzen, werden in Abschnitt 8.4 aufgegriffen. In Abschnitt 9 wird evaluiert, ob solche Operationen Entwicklungs- bzw. genauer RE-Aktivitäten unterstützen.

6.2.1 Unterscheidung zwischen Relationen und Verknüpfungen

Die Literatur, die Abhängigkeiten zwischen Anforderungsartefakten aufgreift, ist häufig auf Modellierung und Unterstützung der expliziten Tracing-Links fokussiert, da diese ein wichtiges Mittel sind, um die Abhängigkeiten in den Griff zu bekommen [35], [60], [85], [110], [113], [132]. Oftmals unterscheiden deren Artefaktmodelle dann nicht klar zwischen Relationen und expliziten Tracing-Links.

In der Interview-Studie aus Abschnitt 4 haben sich jedoch Projekte gezeigt, die durch Probleme aus Artefakt-Abhängigkeiten betroffen sind, diese jedoch nicht mit expliziten Verknüpfungen unterstützen. Jedoch wollen die Projektteilnehmer bisher auch nicht unbedingt bestehende Tracing-Mechanismen (oder zumindest nicht für alle verwendeten Artefakttypen) einführen. Ein Auszug der Gründe hierfür ist:

- Hohe Kosten
- Hoher Zeitaufwand
- Projektteilnehmer wollen die eigentliche primäre Projektaktivität nicht für Tracing-Aktivitäten unterbrechen
- Heterogene Artefakt- bzw. Dokumentlandschaft führt dazu, dass nicht alle Artefakte im selben Tool dokumentiert sind und somit nicht immer leicht referenziert werden können
- Neue Artefakttypen werden erst nachträglich in die Landschaft eingefügt, ohne die dadurch entstandenen Relationen auch in Verknüpfungen zu dokumentieren

Dennoch können die Beteiligten in solchen Projekten Probleme durch bekannte Abhängigkeiten meiden, indem sie beispielsweise entsprechende (manuelle) Überprüfungsschritte in ihrer Methode verankern. Es macht also Sinn, die Ab-

hängigkeitsrelationen zu identifizieren, auch wenn keine Verknüpfung der Artefakte vorgesehen ist.

In dieser Arbeit werden sowohl Relationen als auch Verknüpfungen bei der Modellierung von Anforderungslandschaften berücksichtigt, um beide Situationen (komplette Nachverfolgbarkeit der Anforderungen und bewusster Verzicht auf einzelne Verknüpfungen) aufzugreifen.

Obwohl Verknüpfung und Relation konzeptuell unterschiedliche Dinge sind, hängen sie stark zusammen. Eine Verknüpfung dokumentiert eine bestehende Relation explizit. Es ist nicht sinnvoll, eine Tracing-Verknüpfung zwischen zwei Artefakten zu erstellen, welche nicht in einer Abhängigkeitsrelation stehen. Es kann sogar schädlich sein, da Softwareentwickler, die über Tracing-Verknüpfungen zu abhängigen Artefakten gelangen wollen, irrelevante Ergebnisse erhalten, die störend sind. Genauso sollte die Verknüpfung den Typ der zugehörigen Relation – zum Beispiel *Verfeinerung*, *Ableitung* oder *Erfüllung* – widerspiegeln.

6.2.2 Relevante Typen von Relationen und Verknüpfungen

Es gibt viele Relationstypen, die verschiedene Abhängigkeitsbeziehungen beschreiben. In der Literatur sind mehrere Ansätze zu finden, die diese Relationstypen beispielsweise in Fallstudien, Interviews oder durch Zusammenfassen weiterer Literatur ermitteln und kategorisieren [35], [46], [111], [128], [132], [153]. In den Grundlagen (Abschnitt 2.4.2) wurden bereits einige Kategorisierungen von Relationstypen vorgestellt, die in der Praxis sinnvoll sind.

Prinzipiell stellt diese Arbeit keine Einschränkungen bezüglich zugelassener oder möglicher Relationen in einer Anforderungslandschaft. Es ist jedoch sinnvoll, zwischen einigen grundsätzlichen Arten von Relationen zu unterscheiden. Genauer soll unterschieden werden zwischen den Relationstypen *verfeinert*, *enthält* sowie *entspricht*. Auch andere Relationstypen, wie *widerspricht*, *ist abhängig von*, *entwickelt sich aus*, etc., können in einer hybriden Anforderungslandschaft vorkommen. Diese werden jedoch in den Konzepten dieser Arbeit nicht gesondert hervorgehoben.

Die folgende Abbildung zeigt zunächst eine mögliche Strukturierung von Relationstypen zur Orientierung. Sie basiert auf der Strukturierung von Dahlstedt und Persson [35], die in Abschnitt 2.4.2 erläutert wurde, und wurde um die Relation *enthält* ergänzt. Diese Strukturierung wurde gewählt, weil sie die Unterscheidung der beiden relevanten Relationstypen *verfeinert* und *entspricht* gut unterstützt. Die drei relevanten Relationstypen sind hervorgehoben. Die angegebenen Relationstypen und deren Kategorien sind nicht abschließend. Sie dienen lediglich als Orientierung für jemanden, der eine Anforderungslandschaft modellieren möchte.

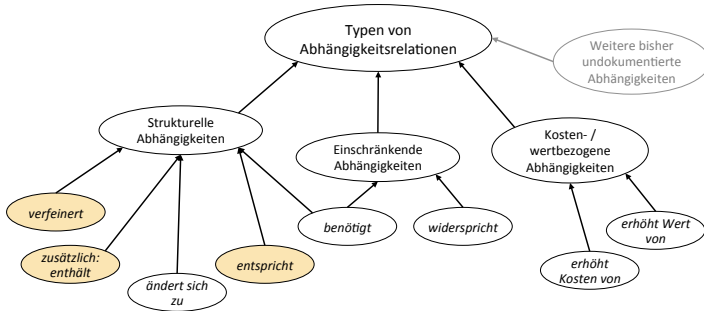


Abbildung 8: Typen von Abhängigkeitsrelationen (angelehnt an [35])

Es ist vor allem zu unterscheiden, ob ein Artefakt ein anderes *verfeinert* – also einen Teil des übergeordneten Artefaktes aufgreift und diesen um zusätzliche Informationen anreichert – oder ob sich zwei Artefakte *entsprechen* – also auf der gleichen Ebene liegen und die Informationen beispielsweise aus verschiedenen Blickwinkeln darstellen. Eine besondere Form der Verfeinerung ist die Relation *enthält*. Diese Relation wird verwendet, um zusammengesetzte Artefakte zu beschreiben.

Unterscheidung zwischen den Relationen *verfeinert* und *entspricht*

Die Unterscheidung zwischen sich verfeinernden und sich entsprechenden Artefakten ist deswegen relevant, weil sich die Artefakte je nach Relation bei Änderungen unterschiedlich verhalten können.

Beispielsweise können sich eine User Story und ein Use Case-Schritt entsprechen. In diesem Fall führt eine Änderung an einem der beiden Elemente häufig dazu, dass sich das andere Element ebenfalls ändern muss oder sonst eine Inkonsistenz entsteht. Abbildung 9 zeigt hierzu ein Beispiel.

Beispielsweise könnte die User Story SC2 so geändert werden, dass man eine Liste nicht filtert, sondern darin die Kredit-Restzeiten direkt anzeigt und diese Liste dann nur noch sortiert. In diesem Fall muss Use Case-Schritt 2 ebenfalls geändert werden. Wird eine neue User Story SC12 hinzugefügt, bei der nun Benachrichtigungen automatisch durch das System versendet werden, so fällt an dieser Stelle auf, dass es keinen entsprechenden Use Case-Schritt gibt. Um diese Änderung aufzunehmen, müsste ein neuer Use Case-Schritt oder gar ein neuer Use Case aufgenommen werden.

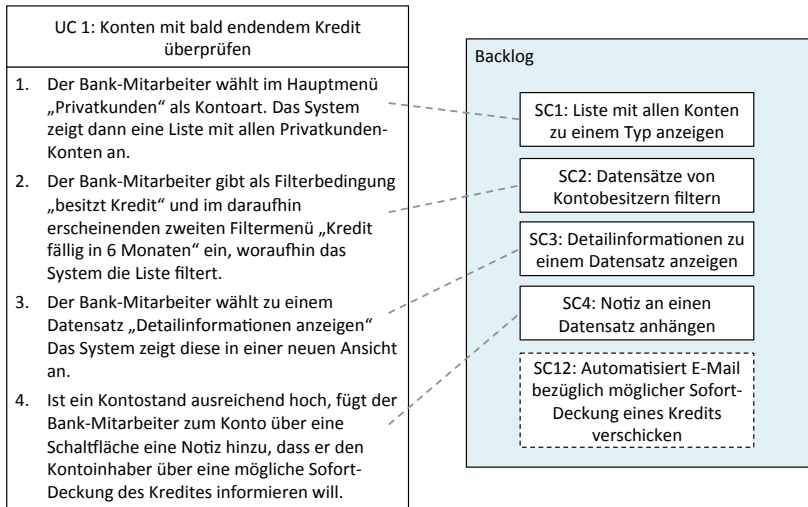


Abbildung 9: Beispiel für eine "entspricht"-Relation zwischen Use Case-Schritten und User Stories

Wird hingegen ein Artefakt durch ein anderes verfeinert, so verhalten sie sich bei Änderungen anders. Mehrere verfeinernde Elemente greifen jeweils Teile des übergeordneten Elementes auf und reichern diese mit zusätzlichen Details an. Zum Beispiel kann ein Use Case durch mehrere Story Cards verfeinert werden – also den Rahmen für diese bilden. Wird nun ein verfeinerndes Artefakt geändert, so kann diese Änderung unter Umständen nur die Detail-Information betreffen, den Rahmen aber weitestgehend unberührt lassen. In diesem Fall hat eine Änderung keine Auswirkung auf das übergeordnete Element. In anderen Fällen betrifft eine Änderung jedoch auch den Rahmen, sodass wieder eine Inkonsistenz entstehen kann, wenn der Rahmen im übergeordneten Artefakt nicht mitgeändert wird. Auch bei Änderungen des übergeordneten Artefaktes sind beide Fälle denkbar: Ein verfeinerndes Artefakt kann von einer Änderung des übergeordneten Artefaktes betroffen sein, muss dies aber nicht in jedem Fall sein. Wird in dem übergeordneten Artefakt ein Teil geändert, der vom verfeinernden Artefakt nämlich gar nicht aufgegriffen wird, so gibt es keine Auswirkung.

Abbildung 10 zeigt ein Beispiel für eine Rahmen-Relation zwischen Use Cases und User Stories. Der Use Case bildet den Rahmen und die Story Cards füllen diesen Rahmen aus. Die User Story SC2 (Datensätze filtern) unterstützt den Use Case, jedoch hat eine Änderung dieser Story keine Auswirkung darauf, ob das Ziel grundsätzlich erreicht wird oder nicht. Wenn sich in diesem Fall die User Story so ändert, dass die Liste sortiert statt gefiltert werden soll, so muss der Use Case nicht angepasst werden. Wird hingegen User Story SC1 so geändert, dass man sich keine Kontolisten anzeigen lassen kann, sondern nur einzelne

Konten nach Eingabe der Kontonummer abrufen kann, so ist der Use Case in seiner bisherigen Form nicht ausführbar und müsste geändert werden. Käme jetzt eine Story Card SC12 (System verschickt automatisch Benachrichtigungen) hinzu, so würde diese ganz aus dem Rahmen fallen, da diese Funktionalität zunächst ganz neu ist und im Rahmen des Use Cases bisher nicht enthalten war. In diesem Fall muss also auch hier der Use Case verändert oder ein neuer Use Case angelegt werden.

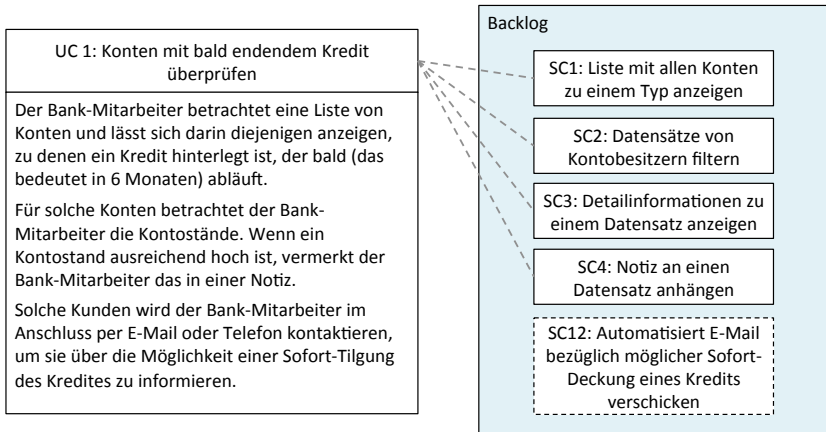


Abbildung 10: Beispiel für eine „verfeinert“-Relation zwischen Use Cases und User Stories

Das Änderungsverhalten ist also je nach Relationstyp unterschiedlich. Dies ist zusammenfassend in Abbildung 11 dargestellt. Wenn sich zwei Artefakte entsprechen, so ist bei einer Änderung eines Artefaktes sehr häufig das andere Artefakt ebenfalls zu ändern. Dafür fällt es bei sich entsprechenden Artefakten leichter, die genaue Stelle für die Änderung zu identifizieren und die Änderung durchzuführen. Wenn zwei Artefakte in einer Verfeinerungs-Relation stehen, so müssen diese nicht bei jeder Änderung mit angepasst werden. Jedoch ist es hier schwieriger zu beurteilen, ob eine Änderung eines Artefaktes auch ein anderes Artefakt betrifft, weil diese auf unterschiedlichen Ebenen sind und weil häufig auch mehrere verfeinernde Artefakte zusammen betrachtet werden müssen, um zu erkennen, ob ein übergeordnetes Artefakt noch erfüllt ist.

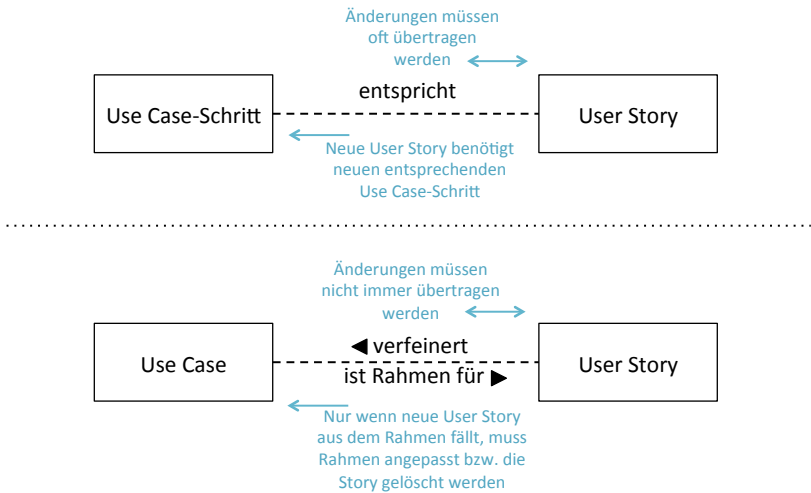


Abbildung 11: Unterschied zwischen den Relationen *entspricht* und *verfeinert*

Die Relation *enthält*

Die dritte besondere Relation ist die *enthält*-Relation. Diese Relation stellt eine besondere Form der Dokumentation dar: Das enthaltene Artefakt ist Teil der physischen Dokumentation des übergeordneten enthaltenden Artefaktes. In der obigen Abbildung 9 steht der Use Case in einer *enthält*-Relation mit seinen Use Case-Schritten, da diese direkt im Use Case festgehalten werden.

Das besondere an dieser Relation ist, dass hier Abhängigkeiten leichter behandelt werden können. Da beide Artefakte zusammen aufgeführt sind, ist es einfacher, beispielsweise bei einer Änderung direkt zu prüfen, ob die Elemente noch konsistent sind oder angepasst werden müssen. Außerdem sind hier die Abhängigkeiten allein durch die gemeinsame Darstellung bereits sichtbar, sodass explizite Verknüpfungen hier in der Regel nicht notwendig sind.

6.3 Anforderungslandschaften als Modell der Typen von Anforderungen und Verknüpfungen

Soeben wurde beschrieben, wie einzelne Artefaktinstanzen und Verknüpfungen in einem Projekt aussehen. In diesem Abschnitt wird nun darauf eingegangen, wie zu einem Projekt genereller die darin zu verwendenden Typen von Artefakten und Verknüpfungen festgelegt werden. Dieser Aufbau eines Projektes in Hinblick auf seine Anforderungsartefakte und deren Verknüpfungen wird in dieser Arbeit *Anforderungslandschaft* des Projektes genannt. Abbildung 12 zeigt ein Beispiel für eine Anforderungslandschaft.

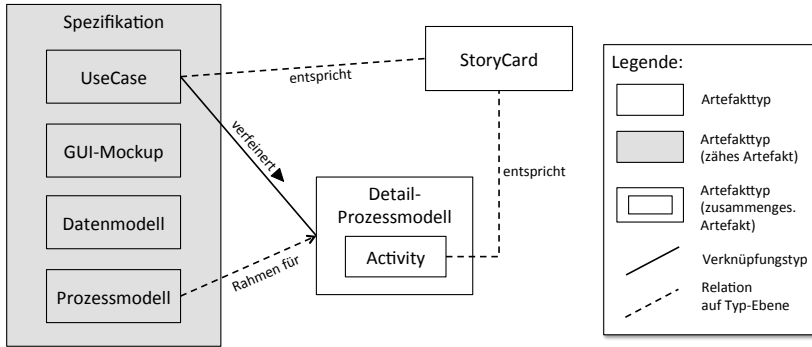


Abbildung 12: Beispiel für ein Anforderungslandschaftsmodell

In Anlehnung an die drei SEMDM-Ebenen (siehe Abschnitt 5.2) ist ein Anforderungslandschaftsmodell auf der Methodik-Ebene angesiedelt. Dieses Modell hilft dem Requirements Methodiker dabei, ein Projekt auf seine Anforderungen hin zu planen oder zu analysieren. Es stellt damit für ein bestimmtes Projekt einen Teil der konkreten Methode³ dar. Die Projektbeteiligten handeln dann nach der Methode, indem sie die vorgegebenen Artefakt- und Verknüpfungstypen verwenden und entsprechend Artefakte und Verknüpfungen instanziierten.

Zum generellen Aufbau der in einem Projekt verwendeten Anforderungen ist zu klären, welche Arten von Anforderungsartefakten verwendet werden – beispielsweise ob es eine Spezifikation gibt, ob diese Spezifikation auch Geschäftsprozessmodelle enthält, ob zudem GUI-Mockups verwendet werden. Nun wird der Umgang mit Anforderungen aber auch durch deren potenzielle Abhängigkeiten und Verknüpfungen untereinander bestimmt. Entsprechend müssen auch die erlaubten Verknüpfungstypen und die potenziellen Abhängigkeitstypen modelliert werden.

Wird zum Beispiel auf der methodischen Ebene festgelegt, dass die Artefakttypen *Use Case* und *Detail-Prozessmodell* verknüpft werden sollen, so hat das auf der Projektebene die Auswirkung, dass beim Anlegen eines konkreten Use Cases *uc2* eine Verknüpfung zum verwandten Detail-Prozessmodell *dpl* erstellt werden darf.

Neben den erlaubten Verknüpfungstypen sind auch potenzielle Arten von Abhängigkeiten auf methodischer Ebene zu betrachten. Beispielsweise bedeutet eine Abhängigkeit zwischen den Artefakttypen *Use Case* und *User Story*, dass

³ Wie in Abschnitt 2.2 beschrieben wurde, wird hier eine Methode als Gruppierung der Teile Prozess, Artefakte, Aktivitäten und Rollen aufgefasst. Das Anforderungslandschaftsmodell stellt hier den Teil der Artefakte dar.

6.3 Anforderungslandschaften als Modell der Typen von Anforderungen und Verknüpfungen

die einzelnen Instanzen von Use Cases potenziell Abhängigkeiten zu User Stories besitzen werden. Gibt es keine Verknüpfungen zwischen Use Cases und User Stories, so wird es dennoch oft notwendig sein, bei der Änderung an einer User Story, z.B. *sc6*, manuell den entsprechenden Use Case, z.B. *uc4*, anzupassen.

Im folgenden wird näher darauf eingegangen, wie die Bestandteile der Anforderungslandschaft definiert werden und wie sie sich auf die Anforderungen im Projekt – also auf die Instanzen in der Projektebene – auswirken. Danach wird auf die Modellierung von Anforderungslandschaften eingegangen, die durch ein Metamodell und eine Notation unterstützt wird.

6.3.1 Bestandteile von Anforderungslandschaften

Eine Anforderungslandschaft kapselt zu einem Projekt die verwendeten Anforderungstypen (AT) sowie deren Beziehungen. Bei Beziehungen zwischen Anforderungsartefakten ist weiterhin zwischen den auftretenden Abhängigkeitsrelationen (AR_{typ}) und den Verknüpfungstypen von explizit dokumentierten Verknüpfungen (VT) zu unterscheiden.

Definition Anforderungslandschaft:

Eine Anforderungslandschaft zu einem *Projekt p* ist damit ein 3-Tupel

$$Landschaft(p) = (AT, VT, AR_{typ})$$

mit

AT = Menge der Artefakttypen von in p verwendeten Artefakten,

$AR_{typ} \subseteq AT \times AT$ = Menge der beachtenswerten Relationen

$VT \subseteq AR_{typ}$ = Menge der Verknüpfungstypen

Es werden nur Artefakte als *im Projekt p verwendet* berücksichtigt, wenn sie von allen Projektbeteiligten abgerufen werden können. Damit zählen persönliche Notizen nicht zu den in p verwendeten Anforderungsartefakten. E-Mails, Protokolle, Interviewaufzeichnungen und ähnliche Artefakte zählen nur, wenn sie beispielsweise in einem Projekt-Repository abgelegt und damit für alle zugänglich sind.

Zur Veranschaulichung der nun folgenden Formalisierung der Bestandteile dient das Beispiel in Abbildung 13.

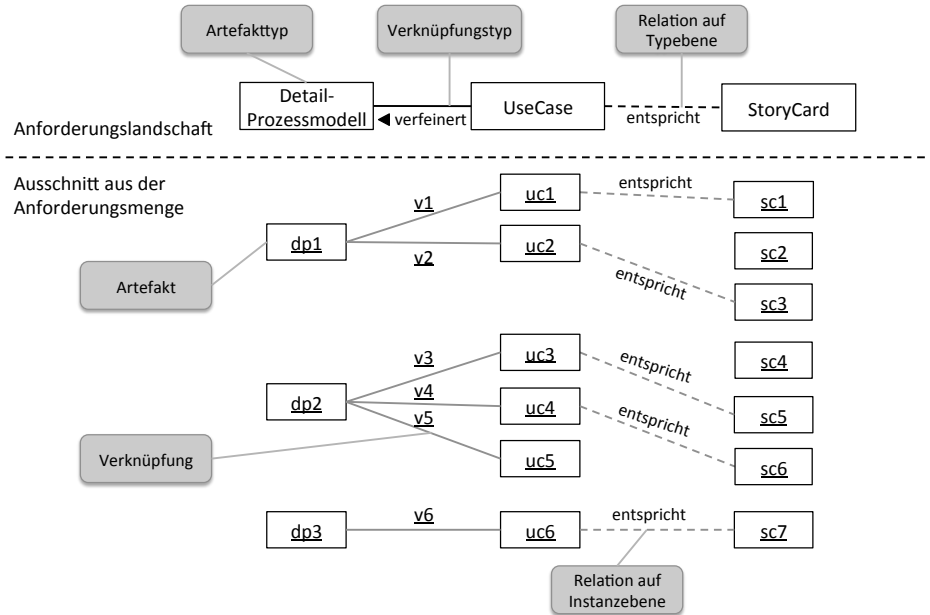


Abbildung 13: Beispiel für Bestandteile von Anforderungslandschaften und Anforderungsmengen

Artefakttypen

Der Artefakttyp klassifiziert zunächst die verschiedenen im Projekt erzeugten und verwendeten Artefakte. Unter dem Artefakttyp werden solche Typen, wie *natürlichsprachliche funktionale Anforderung, GUI-Mockup, Geschäftsprozessmodell, User Story, Domain Model, etc.* verstanden.

Definition Anforderungsartefakttyp:
 Ein Anforderungsartefakttyp (kurz Artefakttyp) bezeichnet eine Klasse von Anforderungsartefakten, die gemeinsame Eigenschaften besitzen.

Alle konkreten Artefakte eines Artefakttyps werden (in Anlehnung an Adersberger [2]) als *Artefakt-Instanzmenge* bezeichnet.

Definition Artefakt-Instanzmenge:
 Die Instanzmenge zu einem Anforderungsartefakttyp $at \in AT$ ist die Menge

$$AI(at) = \{a \in A \mid a \text{ ist eine Instanz von } at\}$$

mit A = Menge aller Anforderungsartefakte im Projekt.

Bedeutung für Projektebene:

Wenn ein Artefakttyp zu einer Anforderungslandschaft gehört, so dürfen auf der Projektebene Instanzen von diesem Artefakttyp erstellt werden.

Beispiel. Im obigen Beispiel ist $AT = \{Detail\text{-Prozessmodell}, Use\ Case, Story\ Card\}$ die Menge der Artefakttypen der Landschaft und z.B. $AI(Use\ Case) = \{uc1, uc2, uc3, uc4, uc5, uc6\}$.

In Abschnitt 6.1.1 ist die Zähheit eines Anforderungsartefaktes als wichtige Eigenschaft beschrieben worden. In der Regel wird in einem Projekt festgelegt, dass alle Artefakte eines bestimmten Typs, beispielsweise alle *Use Cases*, entweder zäh oder flexibel sein sollen. Das bedeutet, dass in der Regel nicht einzelne *Use Cases* flexibel und einzelne *Use Cases* zäh sind. Die Zähheit von Artefakten wird daher auch in Anforderungslandschaften als Typ-Attribut, statt als Instanz-Attribut, festgelegt.

Verknüpfungstypen:

Ähnlich wie die Artefakttypen, sind auch die Verknüpfungstypen zur Klassifikation von Verknüpfungen zwischen Artefakten da. Eine Verknüpfung wird nach der Abhängigkeitsrelation, die sie dokumentiert benannt. Zum Beispiel wurden bereits in Abschnitt 6.2.2 die Relationstypen *verfeinert* und *entspricht* hervorgehoben. Zum Typ zählen nun jedoch neben der Bezeichnung der Relation auch die beiden Artefakttypen, zwischen denen die Verknüpfungen erstellt werden. Der Verknüpfungstyp $\langle Use\ Case\text{-Schritt} \rangle$ entspricht $\langle User\ Story \rangle$ ist demnach ein anderer Verknüpfungstyp als $\langle Detail\text{-Prozessmodell} \rangle$ entspricht $\langle Use\ Case \rangle$.

Definition Verknüpfungstyp:

Ein Verknüpfungstyp $vt = (at1, at2) \subseteq AT \times AT$ bezeichnet eine Klasse von Verknüpfungen die zwischen Artefakten der Artefakttypen $at1$ und $at2$ bestehen.

Ein Verknüpfungstyp vt ist ein 2-Tupel, das aus zwei Artefakttypen besteht.

$$vt \in VT \subseteq AT \times AT$$

Auch für Verknüpfungstypen können Instanzmengen definiert werden:

Definition Verknüpfungs-Instanzmenge:

Die Instanzmenge zu einem Verknüpfungstyp $vt \in VT$ ist die Menge

$$VI(vt) = \{v \in V \mid v \text{ ist eine Instanz von } vt\}$$

mit V = Menge aller Verknüpfungen im Projekt.

Bedeutung für Projektebene:

Wenn ein Verknüpfungstyp $vt = (at1, at2)$ in einer Anforderungslandschaft enthalten ist, so bedeutet dies, dass zwischen zwei Artefakten der Typen $at1$ und $at2$ Verknüpfungen $v = (a1, a2)$ mit $a1 \in AI(at1)$ und $a2 \in AI(at2)$ erstellt werden dürfen.

Beispiel. Im obigen Beispiel ist

$\langle Use\ Case \rangle$ verfeinert $\langle Detail\text{-}Prozessmodell \rangle$
 $= (Use\ Case, Detail\text{-}Prozessmodell)$

der einzige definierte Verknüpfungstyp. Die Menge der Verknüpfungen ist

$VI(Use\ Case, Detail\text{-}Prozessmodell) = \{v1, v2, v3, v4, v5, v6\}$
 $= \{(dp1, uc1), (dp1, uc2), (dp2, uc3), (dp2, uc4), (dp2, uc5), (dp3, uc6)\}$

Relationen auf Typ-Ebene:

Neben möglichen Verknüpfungstypen sollen in einer Anforderungslandschaft auch Relationen zwischen Artefakttypen dargestellt werden können. In Abschnitt 6.2.1 wurde bereits erläutert, aus welchen Gründen es sinnvoll sein kann, zwischen Relationen und Verknüpfungen zu unterscheiden. Während Verknüpfungen dokumentierte Abhängigkeiten darstellen, können auch nicht dokumentierte Abhängigkeiten zwischen Anforderungsartefakten auftreten. Diese führen beispielsweise dazu, dass zwei Artefakte inkonsistent werden können, wenn sich eines davon ändert. Eine Relation in der Anforderungslandschaft hat vor allem den Zweck, anzuzeigen, dass bei Änderungen oder auch beim Lesen von Anforderungen Abhängigkeiten berücksichtigt werden sollten.

Prinzipiell ist eine Relation auf Typ-Ebene, ähnlich wie beim oben definierten Verknüpfungstyp, ein Tupel von zwei Artefakttypen. Allerdings ist es im Fall der Relationen nicht ganz so einfach, festzulegen, welche Bedeutung eine Relation auf Typ-Ebene für die einzelnen Elemente der Anforderungsmenge haben soll.

Die Bedeutung eines Verknüpfungstyps war, dass in der Anforderungsmenge Instanzen dieses Verknüpfungstyps erzeugt werden dürfen. Das ist bei der Relation nicht möglich, da es nicht sinnvoll ist, von Instanziierung von Relationen zu sprechen. Wenn eine Relation zwischen den Typen Use Case und Story Card besteht, so ist es nicht sinnvoll, festzulegen, dass nun Relations-Instanzen zwischen Artefakten der beiden Typen erschaffen werden dürfen. Die Relationen bestehen allein bedingt durch den Inhalt von zwei Artefakten.

Deswegen werden hier auch nicht die Bezeichnungen Relationstyp und Relation verwendet, sondern die Bezeichnungen *Relation auf Typ-Ebene* und *Relation auf Instanz-Ebene*. Im Beispiel in Abbildung 13 sind entsprechend die Relationen auf Instanzebene auch nicht wie Objekte mit $r1$, $r2$, usw. bezeichnet, sondern nur mit einem Relationsbezeichner, wie *entspricht*.

6.3 Anforderungslandschaften als Modell der Typen von Anforderungen und Verknüpfungen

Zur Festlegung der Bedeutung gibt es nun mehrere Möglichkeiten. Beispielsweise könnte die Aussage „der Typ *Use Case* entspricht dem Typ *Story Card*“ bedeuten, dass jede User Story einem Use Case entspricht, oder alternativ, dass prinzipiell eine User Story einem Use Case entsprechen kann. Die gewünschte Bedeutung liegt sogar dazwischen: Mit der Aussage „der Typ *Use Case* entspricht dem Typ *Story Card*“ möchte der Requirements Methodiker ausdrücken, dass „eine *Story Card* einem *Use Case* entsprechen kann, aber nicht muss, und dass sie dies auch so häufig oder schwerwiegend tut, dass man das stets beachten sollte wenn man Inkonsistenzen vermeiden möchte“.

Es ist nicht sinnvoll, zu fordern, dass bei einer Relation auf Typ-Ebene $ar_{Typ} = (at1, at2)$ jedes Artefakt in der gleichnamigen Relation auf Instanz-Ebene ar steht, sodass gelte:

$$\forall a1 \in AI(at1) \exists a2 \in AI(at2) \text{ mit } (a1, a2) \in ar$$

Für die Ziele der Anforderungslandschaftsmodelle ist es auch wünschenswert, eine Abhängigkeit auf Typ-Ebene zu kennzeichnen, wenn beispielsweise nur die Hälfte der Artefakt-Instanzen in dieser Relation stehen.

Genauso ist es aber auch nicht sinnvoll festzulegen, dass bei einer Relation auf Typ-Ebene $ar_{Typ} = (at1, at2)$ die Artefakte potenziell in dieser Relation stehen dürfen. Alle prinzipiell erlaubten Abhängigkeiten einzuzichnen ist (1) sehr aufwändig und führt (2) dazu, dass das Modell an Aussagekraft verliert, da alles mit allem verbunden ist und man so den Überblick über besonders kritische Stellen verliert.

Um das Problem zu lösen, muss eine Definition herangezogen werden, die berücksichtigt, dass nur „bedeutende“ Relationen in einer Anforderungslandschaft aufgeführt werden.

Definition: Relation auf Typ-Ebene:

Eine Relation auf Typ-Ebene ist ein Tupel $ar_{Typ} = (at1, at2) \in AT \times AT$ mit der folgenden Eigenschaft:

Sei ar die zu ar_{Typ} gleichnamige Relation auf Instanz-Ebene. Dann gilt:

Die Menge der Artefaktpaare in der Relation ar , $\{(a1, a2) \in AI(at1) \times AI(at2) \text{ mit } (a1, a2) \in ar\}$, ist beachtenswert.⁴

⁴ Es ist zu beachten, dass zwar die Relation auf Typ-Ebene ar_{Typ} genau ein Tupel von Artefakttypen darstellt, aber die gleichnamige Relation auf Instanz-Ebene ar eine Menge von Tupeln ist, da hier mehrere Artefaktpaare in derselben Relation stehen können.

6 Hybride Anforderungslandschaften

Ein Schwachpunkt dieser Definition ist, dass sie subjektiv ist. Verschiedene Requirements Methodiker betrachten unterschiedliche Mengen von abhängigen Artefakten als beachtenswert.

Als Richtlinie ist daher hinzuzufügen, dass die Menge der Tupel als potenziell beachtenswert gilt, wenn sie sehr groß ist, es also viele Abhängigkeiten zwischen den Artefakten gibt, oder aber wenn die Tupel wichtige Artefakte enthalten. Auch wenn es in der zweiten Situation nicht viele Abhängigkeiten gibt, so sind allerdings die wenigen, die vorhanden sind, sehr wichtig, weil sie wichtige Artefakte betreffen.

Beispiel. Im obigen Beispiel ist „*<Use Case> entspricht <User Story>*“ = (*Use Case, User Story*) eine Relation auf Typ-Ebene. Die Menge der Artefaktpaare, die in der gleichnamigen Relation auf Instanz-Ebene stehen, ist $\{(uc1,sc1), (uc2,sc3), (uc3,sc5), (uc4,sc6), (uc6,sc7)\}$.

Eigentlich ist auch „*<Use Case> verfeinert <Detail-Prozessmodell>*“ = (*Use Case, Detail-Prozessmodell*) eine Relation auf Typ-Ebene. Da diese Relation aber durch eine Verknüpfung dokumentiert wird, ist sie im Landschaftsmodell nicht explizit sichtbar. Dieser Sachverhalt wird im Folgenden erklärt.

Beziehung zwischen Verknüpfungstypen und Relationen auf Typ-Ebene

Eine wichtige Einschränkung der Anforderungslandschaft ist, dass ein Verknüpfungstyp zwischen zwei Artefakten nur dann definiert werden sollte, wenn es eine gleichnamige beachtenswerte Relation zwischen diesen beiden Artefakttypen gibt. Mit Verknüpfungen sollen Abhängigkeiten dokumentiert werden können. Es dürfen daher keine Verknüpfungen zwischen Artefakten gesetzt werden, deren Typen in keiner beachtenswerten Relation stehen. Daher die folgende Festlegung:

Festlegung:

In einer Anforderungslandschaft muss zusätzlich $VT \subseteq AR_{typ}$ gelten.

Dabei ist

VT = Menge aller Verknüpfungstypen

AR_{typ} = Menge aller Relationen auf Typebene

6.3.2 Hybride Anforderungslandschaften

Wie in Abschnitt 2.1 beschrieben wurde, werden in hybriden Entwicklungsansätzen agile und traditionelle Elemente vereint. Daran angelehnt ist eine hybride Anforderungslandschaft nun eine Landschaft, die agile und traditionelle Anforderungsartefakttypen vereint.

Definition „hybride Anforderungslandschaft“

Eine hybride Anforderungslandschaft ist eine Anforderungslandschaft, die mindestens einen agilen Artefakttyp sowie mindestens einen traditionellen Artefakttyp enthält.

In dieser Arbeit werden hybride Entwicklungsansätze betrachtet, die hybride Anforderungslandschaften verwenden. Jedoch muss nicht jeder hybride Entwicklungsansatz eine solche hybride Anforderungslandschaft besitzen.

Wie in den letzten Abschnitten außerdem klar geworden ist, spielen in Anforderungslandschaften die Relationen zwischen Anforderungsartefakten bzw. Artefakttypen eine wichtige Rolle. Dabei tauchen Herausforderungen besonders dann auf, wenn zähe und flexible bzw. adaptive Anforderungsartefakte zusammenkommen. Gerade in hybriden Anforderungslandschaften ist dies oft der Fall, da agile Anforderungsartefakte flexibel sind und traditionelle Artefakte häufig zäh sind.

Dieser Aspekt wird hier noch einmal besonders herausgestellt und später immer wieder verwendet. In den Anforderungslandschaften wird nun auch allgemeiner auf die Unterscheidung zwischen zähen und flexiblen Artefakttypen – statt agilen und traditionellen Artefakttypen im Speziellen – eingegangen.

Charakterisierung „hybride Anforderungslandschaft“

Charakteristisch für hybride Anforderungslandschaften ist, dass sie sowohl zähe als auch flexible Anforderungsartefakte enthalten.

6.4 Ein Metamodell für Anforderungslandschaften

Zur Planung und Analyse von Anforderungslandschaften ist es sinnvoll, eine Landschaft auch grafisch darzustellen. Im vorigen Abschnitt wurde die Anforderungslandschaft als 3-Tupel aus Anforderungsartefakttypen, beachtenswerten Relationen und darauf basierenden Verknüpfungstypen vorgestellt ($Landschaft = (AT, VT, AR_{typ})$). Sie ist damit als Graph interpretierbar, bei dem die Artefakttypen die Knoten und die Relationen und Verknüpfungstypen die Kanten darstellen.

Das folgende Metamodell greift die Zusammenhänge aus dem vorigen Abschnitt auf und stellt dar, wie eine Anforderungslandschaft modelliert wird.

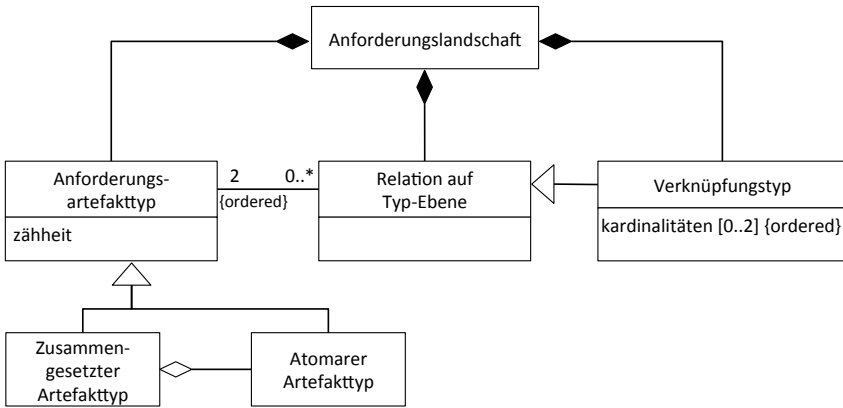


Abbildung 14: Metamodell für Anforderungslandschaftsmodelle

Als Besonderheit ist hier hervorzuheben, dass der Verknüpfungstyp als Spezialisierung der beachtenswerten Relation modelliert ist. Hiermit wird erreicht, dass es Verknüpfungstypen nur zwischen solchen Artefakttypen geben darf, die in einer beachtenswerten Relation stehen (also $VT \subseteq AR_{typ}$). Diese Modellierung passt mit der Bedeutung zusammen, dass eine Verknüpfung eine dokumentierte – also eine spezielle – Relation ist. Gleichzeitig kann man so erreichen, dass im Anforderungslandschaftsmodell nicht für Verknüpfungstyp und Relation jeweils eine gesonderte Kante eingezeichnet werden muss, sondern die Verknüpfung nur anders vermerkt wird als die Relation. Dies erhöht die Übersichtlichkeit der Modelle.

Den gleichen Effekt erzielt man, indem man über ein Attribut in der Relation kennzeichnet, ob es sich um eine *implizite oder dokumentierte* Relation handelt. Jedoch kann der Verknüpfungstyp als eigene Spezialisierung weitere eigene Attribute erhalten, die bei der Relation keinen Sinn machen.

So kann beim Verknüpfungstyp zusätzlich über optionale Kardinalitäten festgelegt werden, dass Artefakte eine bestimmte Anzahl an Verknüpfungen besitzen müssen. Da Relation und Verknüpfung zunächst nicht gerichtet sind, werden Kardinalitäten hier als eine zweistellige Liste modelliert.

Eine weitere Besonderheit ist, dass zwischen zusammengesetzten Anforderungsartefakttypen und atomaren Anforderungsartefakttypen unterschieden wird. Dies hat den Grund, dass zusammengesetzte Artefakte auf besondere Weise dargestellt werden sollen.

6.5 Eine Notation für Anforderungslandschaften

Kühne [88] beschreibt, dass es typischerweise zwei Arten von Modellen gibt. Die *Tokenmodelle* zeigen einzelne Objekte der echten Welt, während die *Typmodelle* mit einem Modellelement ganze Typen bzw. Klassen von Objekten darstellen. In der UML wäre beispielsweise – bezogen auf Java-Code – ein Objektdiagramm ein Tokenmodell, während ein Klassendiagramm ein Typmodell ist. Dabei ist laut Kühne nicht am bloßen Modell festlegbar, ob es sich um ein Token- oder ein Typ-Modell handelt, sondern es hängt vom Bezugspunkt ab. Ein Anforderungslandschaftsmodell ist in Bezug auf die Landschaft eines Projektes ein Tokenmodell, denn die Landschaft besteht ja aus Anforderungsartefakttypen und Verknüpfungstypen. In Bezug auf das Projekt selbst, stellt das Landschaftsmodell nun aber ein Typ-Modell dar, denn es drückt aus, von welchen Typen die instanziierten Artefakte und Verknüpfungen sein dürfen [85].

Der Zweck des Anforderungslandschaftsmodells ist, zu zeigen, wie Anforderungen im Projekt dokumentiert werden können und sollen. Daher ist es naheliegend, eine Notationsform zu wählen, die auf UML-Klassendiagrammen basiert. Auch in der Literatur wird häufig die UML-Klassendiagramm-Notation für Artefaktmodelle verwendet (vgl. Traceability Information Models von Mäder et al. [109], [110]; Artefaktmodelle in Case Study von Konrad und Degen [85]; Traceability-Modelle in TraceML von Adersberger [2]).

Auch in dieser Arbeit orientiert sich die Notation für Anforderungslandschaftsmodelle an UML-Klassendiagrammen, wird aber an einigen Stellen abgewandelt. Damit wird den besonderen Erfordernissen, genauer (1) einer guten Eignung zur gemeinsamen Diskussion in Workshops und (2) einer hohen Sichtbarkeit von Eigenschaften, die für Koexistenz von Anforderungsartefakten besonders förderlich oder hinderlich sind, Rechnung getragen.

Bei Verzicht auf die zusätzlichen Notationsmittel entspricht ein Anforderungslandschaftsmodell einem regulären UML-Diagramm. Die Verwendung der regulären UML-Notation hat den Vorteil, dass die Modellierung bereits durch eine Fülle von Werkzeugen unterstützt wird. Diese Variante der Darstellung einer Anforderungslandschaft wird beispielsweise in Kapitel 8 verwendet, wo zusätzliche Werkzeugunterstützung für OCL-Ausdrücke benötigt wird.

6.5.1 Notation für Anforderungsartefakttypen

Ein *einzelner Anforderungsartefakttyp* repräsentiert eine Klasse von Artefakten mit denselben Eigenschaften. Dieser wird auch mit dem UML-Klassen-Symbol dargestellt, wie in Abbildung 15 gezeigt.

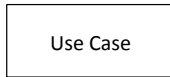


Abbildung 15: Notation für Anforderungsartefakttypen

Typen für *zusammengesetzte Artefakte* werden nun jedoch nicht, wie in der UML und der oben genannten Literatur ([2], [85], [110]) als zwei Klassen mit Kompositionsbeziehung dargestellt. Stattdessen werden diese als geschachtelte Klassensymbole dargestellt, wie in Abbildung 16 gezeigt. Ein ähnliches Vorgehen wird auch von Gotel und Finkelstein angewendet, die in [59] eine Notation vorstellen, die Anforderungsartefakte und die Mitwirkung von Personen zu diesen Artefakten (*Contribution Structures*) darstellt. Auch sie stellen zusammengesetzte Artefakte als geschachtelte Elemente dar und signalisieren mit Assoziationen zum äußeren Element, dass das gesamte Element in einer Relation steht, während Assoziationen zum inneren Element zeigen, dass nur das innere Element in der Relation steht.

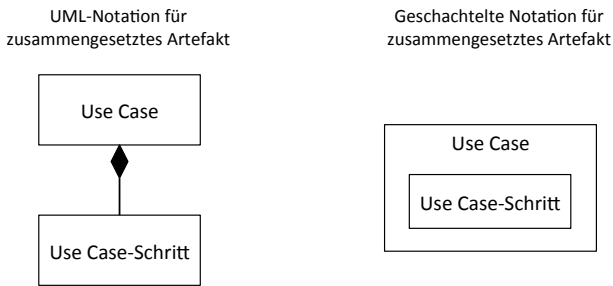


Abbildung 16: Notation für geschachtelte Anforderungsartefakte

Diese Abweichung von der UML-Notation macht den Unterschied zwischen der *enthält*-Relation und anderen Relationen deutlicher, ohne dabei auf die Komplexität mehrerer Pfeiltypen zurückzugreifen. Die Besonderheit der *enthält*-Relation wurde am Ende von Abschnitt 6.2.2 erläutert.

Eine Relation, dargestellt als Kante zwischen Artefakttypen, suggeriert, dass durch die Abhängigkeit zusätzlicher Aufwand zu erwarten ist, zum Beispiel durch zusätzliche Anpassungsschritte oder Kontrollen auf Konsistenz der Elemente. Bei zusammengesetzten Artefakten treffen diese Charakteristika jedoch nicht zu. Zwar müssen auch hier Abhängigkeiten beachtet werden, doch fällt die Identifikation von abhängigen Elementen leichter, da diese bereits über die gemeinsame Darstellung möglich ist.

Anders wäre es beispielsweise bei Nutzerzielen und verfeinernden Use Cases, die an zwei verschiedenen Stellen dokumentiert sind. Beide Artefakttypen ste-

hen zwar potenziell ebenfalls in einer hierarchischen Relation. Jedoch kann hier bei der Änderung eines Use Cases schneller vergessen werden, zu prüfen, ob der neue Use Case noch zum Nutzerziel passt, da das Nutzerziel an einer anderen Stelle steht. Diese beiden Artefakttypen möchte man nicht als zusammengesetzten Artefakttyp darstellen, sondern als zwei Artefakttypen mit Abhängigkeitsrelation.

Weiterhin werden **zähe Artefakte** gegenüber flexiblen Artefakten (siehe Abschnitt 6.1.1) hervorgehoben. Dazu werden sie grau ausgefüllt dargestellt, wie in Abbildung 17 gezeigt.

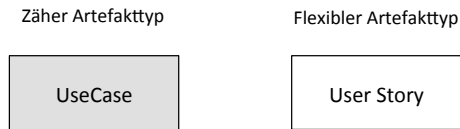


Abbildung 17: Notation für zähe Artefakttypen

In hybriden Anforderungslandschaften ist es hilfreich, zähe von flexiblen Artefakten zu unterscheiden, weil so schnell sichtbar ist, wo Änderungen – als Teil agiler Elemente des Entwicklungsansatzes – eingearbeitet werden können. Insbesondere können so aber schnell Stellen erkannt werden, an denen flexible und zähe Artefakte in einer Abhängigkeitsrelation stehen. Diese Stellen führen zu zusätzlichen Herausforderungen, die in den Abschnitten 6.7 und 6.8.3 erklärt werden.

Es ist durchaus möglich, zähe und flexible Artefakttypen gemeinsam zu verwenden, wie auch in Abschnitt 6.7 beschrieben wird. Ihr Einsatz sollte jedoch geplant und mit den Projektteilnehmern abgestimmt sein. Mithilfe des Anforderungslandschaftsmodells lässt sich die Zähheit von Anforderungsartefakten mit den Projektbeteiligten diskutieren. Es lassen sich Artefakttypen identifizieren, bei denen Zähheit eher hinderlich für den Projektablauf ist. Abhängigkeiten zwischen zähen und flexiblen Artefakten lassen sich schnell aufdecken und frühzeitig durch Planung von expliziten Verknüpfungen unterstützen.

6.5.2 Darstellung von Relationen und Verknüpfungen

Grundsätzlich werden Relationen und Verknüpfungen zwischen Artefakttypen in Anlehnung an die UML-Notation und auch die oben erwähnte Literatur zu Artefaktmodellen ([2], [85], [110], [113]) als Kanten zwischen den Artefakttypen dargestellt.

Ein wichtiges Prinzip der Anforderungslandschaften ist, dass darin zwischen Relationen und Verknüpfungen unterschieden wird. Dies soll auch in Anforderungslandschaftsmodellen sichtbar sein, indem verschiedene Kantenarten verwendet werden.

In UML-Klassendiagrammen gibt es zwei Arten, Beziehungen darzustellen, die gut zu den Konzepten der Anforderungslandschaften passen: Assoziationen und Dependencies. Assoziationen werden in der UML 2.5 Spezifikation beschrieben als: „*An Association classifies a set of tuples representing links between typed instances. [...] An Association declares that there can be links between instances whose types conform to or implement the associated types.*“ ([121], Kap. 11.5., S.197).

Verknüpfungstypen, die durch Links dokumentierte Relationen darstellen, können also als UML-Assoziation modelliert werden. Sie können mit einer Kardinalität und bei Bedarf mit einer Richtung versehen werden. Zur Vereinfachung der Diagramme, verzichtet diese Arbeit auf die explizite Nutzung spezieller Assoziationstypen, wie Aggregationen und Kompositionen.

Dependencies sind in der UML 2.5 Spezifikation definiert durch „*A Dependency signifies a supplier/client relationship between model elements where the modification of a supplier may impact the client model elements.*“ ([121], Kap. 7.7, S.38) Dies entspricht genau dem Verständnis der Abhängigkeitsrelation in dieser Arbeit. Weiterhin beschreibt die Spezifikation: „*The presence of Dependency relationships in a model does not have any runtime semantic implications. The semantics are all given in terms of the NamedElements that participate in the relationship, not in terms of their instances.*“ ([121], Kap. 7.7, S. 38) Dependencies bestehen also auf Klassen- statt Instanzebene. Auch das passt zur gewünschten Semantik, dass Relationen auf Typ-Ebene nicht instanziiert werden können (siehe Abschnitt 6.3.1).

Bei der genaueren Bedeutung des Begriffes *Abhängigkeit* muss jedoch differenziert werden. In der Spezifikation wird ausgeführt: „*A Dependency implies that the semantics of the clients are not complete without the suppliers.*“ ([121], Kap. 7.7, S.38) Dies trifft auf Relationen vom Typ *verfeinert* zu. Es muss jedoch nicht auf Relationen vom Typ *entspricht* zutreffen.⁵ Hier ist es möglich, dass die Bedeutung der Artefakte eines Artefakttyps auch ohne Artefakte des *entsprechenden* Artefakttyps komplett sind. Sie können bereits allein für sich stehen. Dennoch beeinflussen sich die Artefakte. Bei der Änderung eines der Elemente muss das andere unter Umständen mit geändert werden, um Inkonsistenzen zu vermeiden. In dieser Arbeit werden auch solche Situationen als Abhängigkeit betrachtet, bei denen zwei Elemente für sich stehen, in Kombination aber eine

⁵ Zur Erläuterung des Fokus auf beiden Relationstypen *verfeinert* und *entspricht* siehe Abschnitt 6.2.2.

Inkonsistenz auslösen können. Es werden daher auch *entspricht*-Relationen als Abhängigkeiten behandelt und mittels UML-Dependencies dargestellt.

Die UML-Spezifikation definiert im Standard Profile ([121], Kap. 22, S. 677 ff.) bereits sinnvolle Arten von Dependencies, die hier verwendet werden können. Darunter sind die Unterart *Abstraction* und deren Stereotypen «refine», «derive» und «trace» besonders interessant für Anforderungslandschaften, da sie die *verfeinert*-Relation abbilden.

Eine Dependency ist stets eine gerichtete Relation. Für die *entspricht*-Relation ist die Abhängigkeit jedoch symmetrisch und damit immer in beide Richtungen definiert. In Anforderungslandschaftsmodellen werden *entspricht*-Relationen daher ohne explizite Richtung eingezeichnet. Auch in der UML-Spezifikation des «trace»-Stereotyps ist erwähnt, dass hier die Richtung keine große Rolle spielt und weggelassen werden kann, da die Abhängigkeit meist in beide Richtungen definiert ist.

Verknüpfungen sind in der Regel in beide Richtungen navigierbar, sodass Verknüpfungstypen grundsätzlich ungerichtet dargestellt werden. Ein Verknüpfungstyp der Art *verfeinert* kann durch Angabe der Beziehung mit Leserichtung, angegeben durch ein ausgefülltes Dreieck in Leserichtung, versehen werden.

Anders als im UML-Metamodell, ist im Anforderungslandschafts-Metamodell der Verknüpfungstyp eine Spezialisierung der beachtenswerten Relation. Für eine Relation, die durch einen Verknüpfungstyp dokumentiert werden kann, wird also immer nur eine Kante im Modell eingezeichnet. Die Kante trägt immer den Namen der Abhängigkeitsrelation. Abbildung 18 zeigt ein Beispiel für die Notation beider Beziehungs-Arten.

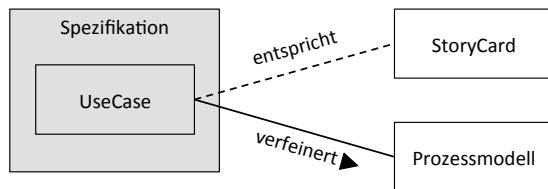


Abbildung 18: Notation für Relationen (obere Kante) und Verknüpfungen (untere Kante)

6.6 **Verwendung von Anforderungslandschaften**

Das Anforderungslandschaftsmodell liegt laut SEMDM in der methodischen Ebene. Es wird also vom Requirements Methodiker verwendet, um für ein Projekt festzulegen, welche Artefakte erstellt werden sollen und auf welche Abhängigkeiten dabei zu achten ist. Der Requirements Methodiker legt damit für ein Projekt eine Methode fest, an die sich die Projektteilnehmer dann im Projekt halten. Zusätzlich können zur Methode auch Rollen, Prozesse, Aktivitäten sowie Standards und Tools festgelegt werden [89].

Neben dem Hauptzweck, einen Teil der Methode zu modellieren, kann das Anforderungslandschaftsmodell aber auch für weitere Aktivitäten verwendet werden. Diese werden im Folgenden aufgelistet.

Verwendung als Kommunikationsmittel

Die Darstellung kann aktiv als Kommunikationsmittel mit Projektteilnehmern eingesetzt werden. Mäder et al. [110] berichten, wie die Kommunikation über Traceability Information Models, die den Anforderungslandschaften ähneln, zu folgenden Aktivitäten beitragen kann:

- Stelle Richtlinien für Projektmitglieder bereit, die zeigen, wie Artefakte aufeinander aufbauen und welche Verknüpfungen gefordert sind
- Schaffe gemeinsames Verständnis für Aufbau und Zweck einzelner Artefakte und gemeinsame Terminologie für Artefakttypen ([85])
- Erhöhe damit auch Konformität der Anforderungen und Verknüpfungen bei mehreren involvierten Projektteilnehmern
- Erhöhe Verständnis für geforderte Verknüpfungen bei Projektmitgliedern, die nicht an der Erstellung von Verknüpfungs-Regeln beteiligt waren, aber von diesen betroffen sind
- Erlaube den Projektmitgliedern so auch eine bessere Einschätzung der Tracing-Möglichkeiten im Projekt
- Schaffe eine Basis für Vollständigkeit bei der Verknüpfung von Artefakten
- Schaffe eine Basis für automatisiertes Tracing

Verwendung als Analysewerkzeug

Weiterhin können Anforderungslandschaften als Analysewerkzeug eingesetzt werden. Sie bieten einen Gesamtüberblick, an dem sich Probleme und Unstimmigkeiten klarer feststellen lassen. Am Anforderungslandschaftsmodell können Projektteilnehmer ihre Erfordernisse an Artefakte und Verknüpfungen diskutieren und mitteilen, welche Aktivitäten gut unterstützt werden und welche bisher nicht. Die Analyse kann folgende Probleme aufdecken:

- Sichtweisen der Projektteilnehmer weichen von Sichtweisen des Requirements Methodikers ab
 - Auch wenn der Requirements Methodiker festgelegt hat, welche Anforderungsartefakte eingesetzt werden sollen, kann es passieren, dass die Projektteilnehmer weitere Artefakte erstellen wollen, wenn sie das Gefühl haben, dass diese die Anforderungskommunikation besser unterstützen können.
 - Welche Projektbeteiligten welche Werte in Artefakten sehen, kann der Requirements Methodiker schwer identifizieren. Auch hier ist es hilfreich, die Artefaktwerte in einem gemeinsamen Workshop zu diskutieren.
- Feststellung struktureller Probleme, wie
 - Artefakte haben starke Abhängigkeiten zueinander, sind aber bisher nicht verknüpft (siehe Abschnitt 6.8)
 - Zähle und flexible Artefakte haben starke Abhängigkeiten zueinander (siehe Abschnitt 6.8.3)
 - Es werden zwei ähnliche Artefakte verwendet, die sich zu einem zusammenfassen lassen
- Feststellung von Anwendungsproblemen, wie
 - Verschiedene Projektteilnehmer verwenden unterschiedliche Artefakte, die nicht gut verknüpft sind (z.B. Kunden notieren ihre Änderungen im Geschäftsprozessmodell, während Entwickler User Stories für die Planung nutzen. Manuelle Übersetzung führt zu Mehraufwand und Fehlern.)
 - Die verwendeten Anforderungsartefakte unterstützen die Aktivitäten bestimmter Projektteilnehmer nicht gut
 - Bestimmte Anforderungsartefakte werden nicht mehr verwendet. Sie erhöhen die Komplexität der Anforderungsmenge und führen unter Umständen zu Zusatzaufwand, wenn sie noch aktuell gehalten werden.

Während mithilfe von Anforderungslandschaften Struktur- und Anwendungsprobleme entdeckt werden können, ist es jedoch nicht möglich, aus den Modellen allein Rückschlüsse auf die grundsätzliche Qualität der Entwicklungsmethode zu ziehen. In Bezug auf die Qualität der Entwicklungsmethode stellen sich vor allem Fragen, wie, ob die Methode zur Projektgröße und –Kritikalität passt, ob die Methode erlaubt, effektiv zu kommunizieren, ob die Methode Projektumstände, wie häufige externe Änderungen abfangen kann [18], [28].

Vor allem bei einer hybriden Entwicklungsmethode stellt sich die Frage, ob die Methode zusätzlichen Nutzen schafft oder ob sie möglicherweise schadet, weil sie gegen bewährte Methoden der plangetriebenen und der agilen Entwicklung verstößt. Ein mögliches Problem ist zum Beispiel, dass das Weglassen von Dokumentation mit dem Streben nach Agilität begründet wird. Ohne die richtigen

Kommunikationsmechanismen, die nun anstelle der Dokumente zur Informationsverteilung verwendet werden müssen, ist so eine Vorgehensweise jedoch ungeeignet.

Wie aber auch bereits am Beispiel sichtbar ist, kann aus dem bloßen Aufbau der Anforderungslandschaft nicht auf die Qualität des Entwicklungsansatzes geschlossen werden. Zum einen werden zusätzliche Informationen bezüglich der Prozesse und auch der Informationsflüsse im Projekt benötigt, um Aussagen über die Qualität des Entwicklungsansatzes zu treffen. Zum anderen hängt die Eignung eines Entwicklungsansatzes auch von den Projektzielen und – umständen ab. Im nächsten Abschnitt (6.7) werden beispielsweise fünf unterschiedliche Situationen mit unterschiedlichen Bedürfnissen und zwei unterschiedliche Repräsentanten von Anforderungslandschaften vorgestellt.

Solche Aspekte, wie Informationsflüsse und Projektbedürfnisse sind im bloßen Anforderungslandschaftsmodell nicht sichtbar. Wie jedoch im vorigen Punkt beschrieben, können Anforderungslandschaften dennoch als Kommunikationsmittel verwendet werden und eignen sich zumindest als Hilfsmittel bei der Beurteilung der Entwicklungsmethode.

Verwendung zur Feststellung von Veränderungen

Es ist sinnvoll, den Ist-Zustand von Zeit zu Zeit neu zu erheben und so auch Veränderungen festzustellen. So können zum Beispiel Artefakte, die von Projektteilnehmern neu hinzu genommen wurden, weil sie förderlich für die Anforderungskommunikation waren, identifiziert und eingebettet werden. In der Interview-Studie (Abschnitt4) wurde festgestellt, dass sich die Anforderungslandschaft eines Projektes mit der Zeit ändern kann. Besonders um der Kommunikation mit Stakeholdern gerecht zu werden, werden häufig verschiedene Darstellungsformen von Anforderungen ausprobiert [85], [97]. Auch Nguyen und Swatman haben in [117] festgestellt, dass Anforderungen nicht nur linear weiterentwickelt, also beispielsweise nach einem bestimmten Schema weiter detailliert oder formalisiert, sondern auch immer wieder komplett umorganisiert wurden. Durch unerwartete Einsichten kam es hier immer wieder dazu, dass die Anforderungen und ihr Aufbau umgestaltet wurden. Um auch nach solchen Veränderungen noch passende Artefakte zu verwenden, die keine unnötigen Abhängigkeiten besitzen und gut zu den Projektaktivitäten passen, sollte die Landschaft von Zeit zu Zeit mit allen Projektbeteiligten analysiert werden.

Verwendung als Planungs- / Optimierungswerkzeug

Neben der Darstellung des Ist-Zustandes eines Projektes können Anforderungslandschaftsmodelle den Soll-Zustand zeigen und damit als Planungs- / Optimie-

rungswerkzeug dienen. Hierauf wird in Kapitel 7 näher eingegangen. Beispielsweise können die folgenden Aktivitäten durchgeführt werden:

- Sammle und strukturiere alle notwendigen Artefakte
- Stelle sicher, dass die Artefakte in Einklang mit den Projektteilnehmern und den Projektaktivitäten stehen
- Stelle sicher, dass zu jedem Artefakt klar ist, ob es aus anderen Artefakten abgeleitet wird
- Lege zu festgestellten Abhängigkeiten Mechanismen zur Berücksichtigung der Abhängigkeiten fest (z.B. dass bei Änderung eines Use Cases manuell überprüft werden muss, ob der neue Use Case noch zu allen angrenzenden GUI-Mockups passt)
- Plane Tracing-Verknüpfungen. Stelle sicher, dass alle gewünschten Trace-Pfade, auch über mehrere Artefakte, möglich sind
- Plane Trace-Operationen. Dies sind Operationen, welche die Einhaltung und Verwendung von Tracing-Verknüpfungen unterstützen. Hierauf wird in Abschnitt 8.4 näher eingegangen.

Wiederverwendung von Anforderungslandschaftsmodellen

Anforderungslandschaftsmodelle repräsentieren Teile einer Methode, wie in Abschnitt 5.2 erklärt. Durch Wiederverwendung bewährter Modelle lassen sich Erfahrungen zu Requirements Engineering-Methoden weitergeben.

Dies können relativ allgemeine Modelle sein, die unter anderem als Referenzmodelle mit Anpassungsmöglichkeit angegeben werden. Beispielsweise gibt Méndez ein Referenzmodell [111] und auch domänenspezifische Artefaktmodelle [113] an, die in vielen artefaktorientierten RE-Ansätzen verwendet werden können.

Auch innerhalb eines Unternehmens kann es gewünscht sein, spezifische Artefaktkonstellationen festzulegen und wiederzuverwenden. Dies kann wiederum eine individuelle Anpassung eines Referenzmodells gewesen sein, die sich für bestimmte Projekte oder Unternehmen bewährt hat. Konrad und Degen [85] berichten über Projekte, in denen Artefaktmodelle als Ausgangsbasis in ähnlichen Projekten wiederverwendet wurden, um die Kosten für die Dokumentation des Artefaktmodells zu reduzieren.

6.7 Typische hybride Landschaften und deren Entstehung

Hybride Anforderungslandschaften kombinieren agile und traditionelle Anforderungsartefakte und ermöglichen damit auch, ihre jeweiligen Vorzüge zu nutzen. Ziel dieses Abschnittes ist es, sinnvolle Kombinationen zusammen mit den gewünschten Effekten zu diskutieren.

6 Hybride Anforderungslandschaften

Agile Artefakte ermöglichen leichtgewichtiges Vorgehen und eine höhere Flexibilität in Bezug auf Anforderungsänderungen. Die Anforderungen sind auf funktionale Anforderungen und deren Kundennutzen fokussiert. User Stories ermöglichen eine transparente Planung und Schätzung von gewünschter Funktionalität. Zudem regen sie frühe Integration und frühes Testen an. Traditionelle Artefakte helfen dabei, eine Absicherung für Kunden und Management zu schaffen und Schnittstellen zu angrenzenden Projekten bereitzustellen. Sie können einen Schutz gegen Überanpassung bieten, indem sie einen festen Bezugspunkt darstellen. Zudem können sie verwendet werden, um einen umfassenden Gesamtüberblick darzustellen.

Je nach gewünschten Effekten werden unterschiedliche Varianten der Kombination von agilen und traditionellen Artefakten gewählt. Dann stehen unterschiedliche Artefakte im Hauptfokus der Entwicklungsmethode, d.h. sie werden als primäres Artefakt zur Diskussion der Anforderungen verwendet, und es werden unterschiedliche weitere Artefakte hinzugefügt.

Prinzipiell können hybride Landschaften auf zwei Arten aufgebaut sein. Entweder steht die reguläre agile oder traditionelle Vorgehensweise im Mittelpunkt und wird um zusätzliche Artefakte oder auch Methoden der anderen Vorgehensweise erweitert. Alternativ werden agile und traditionelle Artefakte gleichermaßen verwendet und es lässt sich kein primäres Artefakt ausmachen.

Bisheriger Stand der Praxis, wie er informell in der Interview-Studie vermittelt wurde, ist, dass sich die Projektteilnehmer zu Projektbeginn für eine grundsätzlich agile oder plangetriebene Herangehensweise oder für ein primäres Artefakt entscheiden. Die Variante des gleichwertigen Einsatzes mehrerer agiler und traditioneller Artefakte, wurde in der Interview-Studie nicht explizit berichtet.

Steht eine agile oder traditionelle Vorgehensweise im Fokus der Entwicklungsmethode, so führen verschiedene gewünschte Effekte dazu, dass eine Landschaft gewählt wird, die hybrid ist. Diese Effekte sind in Abbildung 19 illustriert und werden im Folgenden erläutert. Dabei ist auch eine Kombination an beabsichtigten Effekten möglich und führt zu einer Kombination der unten genannten Strategien.

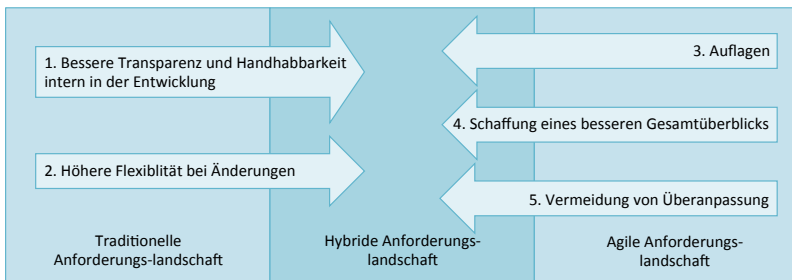


Abbildung 19: Beabsichtigte Effekte, die zu hybriden Landschaften führen

Die einzelnen Beweggründe, die den Einsatz einer hybriden Anforderungslandschaft veranlassen können, werden im Folgenden diskutiert. Dazu wird gezeigt, welche Defizite die regulären agilen oder traditionellen Vorgehensweisen besitzen und wie Zusammensetzungen zu hybriden Landschaften diese Defizite mindern können. Im Anschluss daran werden die besonderen Herausforderungen der einzelnen Zusammensetzungsvarianten diskutiert.

1. Traditionelle Projekte werden intern mithilfe agiler Methoden unterstützt

Ausgleichende Defizite: In traditionellen Projekten wird häufig auf Basis vollständiger und detaillierter Anforderungs- oder Entwurfsartefakte entwickelt. Dabei wird die Entwicklung häufig nicht an der schnellen Lieferung anwenderorientierter Teile ausgerichtet, sondern eher nach technischen Komponenten aufgeteilt. Beispielsweise setzen Entwickler die gesamte Architekturgrundlage oder das gesamte Datenmodell auf, bevor sie Möglichkeiten zur Interaktion implementieren und damit erste Anwendungsszenarien durchführbar machen. Dies hat oft späte Integration, späte Tests des Gesamtsystems und den späten Abgleich mit den Nutzerzielen zur Folge. Gibt es hier kritische Abweichungen oder Probleme, so kann das Produkt möglicherweise sogar wertlos für den Kunden werden, da es dessen Ziele nicht erfüllt. Zudem ist im Anschluss nicht gut sichtbar, in welche Funktionalität Aufwand geflossen ist, was relevant wird, wenn sich das Projekt verspätet.

Erzielbare Effekte: Zur Behebung dieser Probleme kann es sinnvoll sein, die interne Entwicklung auf User Stories auszurichten. Dann werden zu Beginn weiterhin vollständige Anforderungen erarbeitet, was auch zunächst zu einer langen Analyse führt. Anschließend werden aus diesen Anforderungen aber User Stories abgeleitet und die interne Entwicklung damit durchgeführt. Durch explizite Schätzung und vollständige Umsetzung der kleinen nutzerorientierten Features kann sich die Vorhersagekraft in Bezug auf Fertigstellung von für den Nutzer sichtbaren Features verbessern. Insbesondere kann so transparenter nachvollzogen werden, welcher Aufwand mit den einzelnen Features verbunden ist. Auf diese Art kann die gesamte Entwicklung besser auf Nutzerziele und Kundenwünsche ausgerichtet werden. Eine iterative Vorgehensweise mit kontinuierlicher Integration und Tests wird ebenfalls unterstützt.

Komplikationen: Die Ableitung von User Stories aus der Spezifikation muss möglichst vollständig sein. Werden hier Funktionen vergessen, so fällt dies möglicherweise erst spät auf. Durch ihre Flexibilität und Aktualität geraten die User Stories schnell in den Gesamtfokus der Entwicklung, sodass die Spezifikation nicht mehr oft verwendet wird. Dabei kann es vorkommen, dass auch Änderungen in die User Stories eingearbeitet werden, die nicht mehr in Einklang mit der Spezifikation sind, weil die Projektbeteiligten vergessen, die Änderun-

gen mit der Spezifikation abzugleichen. Auch wenn die Änderungen gerechtfertigt sind, kann es am Ende des Projektes ein Problem darstellen, wenn das System nicht der Spezifikation entspricht. Weiterhin sind User Stories eher auf funktionale Anforderungen ausgerichtet, sodass nichtfunktionale Anforderungen, Constraints, Stakeholderbeschreibungen und weitere Informationen möglicherweise gar nicht übersetzt werden können.

2. Traditionelle Projekte werden mithilfe agiler Anforderungsartefakte flexibler in Bezug auf Anforderungsänderungen

Auszugleichende Defizite: In traditionellen Projekten werden die Anforderungen zunächst vollständig und detailliert für das Projekt oder ein Inkrement erhoben. Jedoch kommt es häufig vor, dass sich Anforderungen noch ändern. Beispielsweise kann sich der Projektkontext ändern, sodass neue Einschränkungen oder Anforderungen an das System entstehen [167]. Häufig können auch die Kunden Anforderungsdetails erst einschätzen, wenn sie die umgesetzte Software sehen [13]. Ähnlich stellen Entwickler manchmal fest, dass eine spezifizierte Umsetzung so nicht möglich ist. Treten Änderungen auf, so ist der Aufwand, detaillierte Spezifikationen anzupassen, hoch. Die richtigen Abschnitte müssen in einem umfangreichen Dokument identifiziert und angepasst werden. Dabei können wieder Inkonsistenzen mit anderen Artefakten entstehen. Oft schreiben Projektteilnehmer dann Änderungswünsche schnell in eine E-Mail oder ein Besprechungsprotokoll und vergessen aufgrund des hohen Aufwandes – oder verzichten gar bewusst darauf – diese im Anforderungsartefakt anzupassen.

Erzielbare Effekte: Der Aufwand, auf Änderungen einzugehen, kann reduziert werden, indem User Stories verwendet werden. Zum einen hilft die Fokussierung auf User Stories dabei, Inhalte von vornherein nicht zu detailliert aufzunehmen. Details werden dann erst zu einem Zeitpunkt spezifiziert, an dem der Informationsstand bei Kunden und Entwicklern bereits höher ist. Zusätzlich erlauben User Stories wegen ihrer simplen Art und Anordnung, Änderungen grundsätzlich schnell einzupflegen. Ein Projekt kann die gesamte Anforderungsdokumentation auf User Stories umstellen oder alternativ auch eine grobe Spezifikation verfassen und dann im Detail durch User Stories ausfüllen. Auch bei Durchführung eines plangetriebenen Prozesses, bei dem in einer Analysephase alle zu unterstützenden Nutzerziele und -interaktionen zu Beginn identifiziert werden, ist es möglich, diese auf User Stories zu dokumentieren und gar vom Kunden gegenzeichnen zu lassen.

Komplikationen: Wenn es sehr leicht ist, Änderungen einzufügen, so kann es schnell passieren, dass immer mehr neue Anforderungen aufgenommen werden, was zu einer Erhöhung des gesamten umzusetzenden Umfangs führt (*Scope Creep* [167]). Diese Situation endet häufig in einer Überschreitung von Zeit oder Budget. Wird ein agiler Prozess eingesetzt, bei dem für jede Iteration Anforderungen priorisiert werden – und folglich auch unwichtige Anforderungen

aus dem Projektumfang herausfallen– so lässt sich gut mit neuen Anforderungen umgehen. Wird jedoch, wie im plangetriebenen Vorgehen, verlangt, dass der anfängliche Rahmen vollständig umgesetzt wird, so führt zu schnelle Einbeziehung nachträglicher Änderungen zu unerwünschten Kosten. In diesem Fall sollten Änderungen mit Change Requests assoziiert werden, um den Zusatzaufwand zu berücksichtigen [87].

3. Agiles Projekt wird um weitere Artefakte erweitert, um Auflagen zu erfüllen

Ausgleichende Defizite: Richtlinien, Management, Kunden oder angrenzende Projekte können ein Projekt dazu zwingen, zumindest an der Schnittstelle nach außen hin, bestimmte Artefakte zu verwenden oder bereitzustellen. Beispielsweise kann es ein Wunsch der Kundenseite sein, gewisse umzusetzende Features vertraglich per unterzeichneter Spezifikation zugesichert zu bekommen. Ist das Projekt in ein größeres System eingebettet, so teilt es sich oft Schnittstellen, Datenmodell-Festlegungen oder Systemoberflächen mit angrenzenden Systemen beziehungsweise Projekten. Diese verwenden möglicherweise bereits bestimmte Artefakte, um ihre Schnittstellen darzustellen oder Anforderungen zwischen Projekten auszutauschen. Wenn ein agiles Projekt Schnittstellen zu solchen Systemen oder Projekten besitzt, so muss es zusätzliche geforderte Artefakte anzufertigen.

Erzielbare Effekte: In dieser Situation können verlangte Artefakte als zusätzliche Dokumentation hinzugefügt werden. Wie beispielsweise Beck [8] und Ambler [5] vorschlagen, wird diese dann neben den User Stories mit gewartet. Häufig wird die Wartung in die Erfüllungskriterien (*Definition of Done* [93]) jeder Story hinzugefügt. So wird mit Fertigstellung des Codes zu einer User Story gegebenenfalls auch die zusätzliche Dokumentation gepflegt.

Komplikationen: Sind die zusätzlichen Artefakte zäh, so kann es zu Schwierigkeiten bei Änderungen kommen. Agile Projekte sind darauf ausgelegt, Änderungen leichtgewichtig und schnell aufzunehmen und umzusetzen. Hat eine Änderung nun jedoch Auswirkungen auf ein angrenzendes Projekt, so kann diese unerwünscht sein. Beispielsweise könnte die Änderung einer Schnittstelle zu einem zentralen System viele angrenzende Projekte auf einmal betreffen. Genauso können Änderungen – ähnlich wie in Situation 1 – gegen einen vertraglich festgelegten Rahmen verstoßen und müssen dann vorher abgestimmt werden. Grundsätzlich entsteht zunächst ein Konflikt zur agilen Herangehensweise.

4. Agiles Projekt mit großem Umfang oder vielen Beteiligten verwendet zusätzliche Artefakte, um einen Bezugspunkt und Gesamtüberblick herzustellen

Ausgleichende Defizite: In agilen Projekten mit umfangreichen Anforderungen, aber vor allem auch in solchen, an denen viele Personen beteiligt sind, kann die User Story-Menge unübersichtlich für alle Projektbeteiligten werden [27]. Nicht alle Personen bekommen dann noch alle Änderungen mit. Doppelungen und Abhängigkeiten zwischen User Stories werden eher übersehen. Außerdem ist es dann schwierig, im Blick zu behalten, was alles noch umgesetzt werden soll. So stellt man möglicherweise erst spät fest, wenn ein Feature doch wichtiger war, als die Priorisierung in den einzelnen Iterationen es widerspiegelt hat.

Erzielbare Effekte: Zusätzliche Artefakte, wie beispielsweise ein Use Case-Diagramm oder ein Geschäftsprozessmodell stellen ein Gesamtbild des Projektes dar. So helfen sie, den gesamten Rahmen im Blick zu behalten und einzelne User Stories gedanklich in diesen einzuordnen. Eine grobe Spezifikation hilft zudem, einen Bezugspunkt mit den wichtigsten Anforderungen für alle Projektbeteiligten bereitzustellen. Sie stellt in Projekten mit vielen Beteiligten einen Rahmen dar, auf den sich alle Projektbeteiligten beziehen können.

Komplikationen: Ein zusätzliches Überblicksartefakt erfordert zusätzlichen Wartungsaufwand. Besonders in Projekten mit vielen Beteiligten kann es dazu kommen, dass am Überblicksartefakt häufige und teilweise auch konkurrierende Änderungen vorgenommen werden. Doch auch das Gegenteil kann eintreten. Es ist auch möglich, dass das Überblicksartefakt nicht gewartet wird und veraltet, da es nicht so wie die User Stories das primäre Anforderungsartefakt ist.

5. Agiles Projekt verwendet zähes Artefakt, um Überanpassung zu vermeiden.

Ausgleichende Defizite: Änderungen sind in agilen Projekten etwas positives, weil sie dazu beitragen, sich schrittweise einer besseren Lösung zu nähern. Jedoch haben „zu viele Änderungen“ auch negative Auswirkungen. Sind an einem Projekt viele Stakeholder beteiligt oder stehen die Stakeholder im Interessenskonflikt bezüglich des Produktes, so bringen sie oft auch immer wieder widersprüchliche Änderungen ein. Als Resultat wird immer wieder hin- und zurück-geändert, ohne dass das Produkt dabei gut vorankommt.

Erzielbare Effekte: In dieser Situation kann es sinnvoll sein, mit einem zähen Artefakt, zu dessen Inhalt alle Stakeholder zustimmen, einen festen Rahmen zu schaffen. Diese Festlegung schafft eine gültige Umsetzungsrichtung. Sie ermöglicht den Entwicklern, innerhalb des festgelegten Rahmens agil vorzugehen, dabei aber nur auf Änderungen zu reagieren, die das Projekt weiter in die vorgegebene Grundrichtung bringen.

Komplikationen: Es ist ein häufiges Problem, dass Projektteilnehmer zu Beginn eines Projektes nicht gut einschätzen können, ob eine Spezifikation umsetzbar sein und die Wünsche der Nutzer treffen wird. Diesen Effekt beschreibt Boehm mittels des Akronyms IKIWISI – „I know it when I see it“, [13]. Gerade in einer Situation, in der ein Kompromiss zwischen mehreren Parteien möglichst früh festgelegt werden muss, können schnell ungeeignete Lösungen festgelegt werden. Das Beharren auf dieser früh festgelegten Grundrichtung kann dazu führen, dass einige der Stakeholder am Ende komplett unzufrieden sind. Es wird also eine Möglichkeit benötigt, von der festgelegten Grundrichtung im Notfall doch abweichen zu können. Außerdem verlängert sich die Analysephase, da die Abstimmung einer Lösung, die für alle Parteien zufriedenstellend ist, Zeit in Anspruch nimmt.

Unterschiedliche Komplexitäten von Anforderungslandschaften

Die obigen Beispiele deuten auch an, dass sich Anforderungslandschaften je nach beabsichtigten Effekten sehr in ihrer Größe und Komplexität unterscheiden können. Im Folgenden werden zwei konkrete Landschaften gezeigt, die für ihren jeweiligen Zweck einen sinnvollen Aufbau darstellen, sich aber sehr in ihrer Größe unterscheiden.

Zunächst zeigt Abbildung 20 eine sehr simple Landschaft. Diese Landschaft unterstützt ein vorwiegend agiles Vorgehen, welches – wie im Fall 4 (Schaffung eines besseren Gesamtüberblicks) oder Fall 5 (Vermeidung von Überanpassung) der obigen Konstellationen – durch einen festen Rahmen um traditionelle Elemente ergänzt wird.

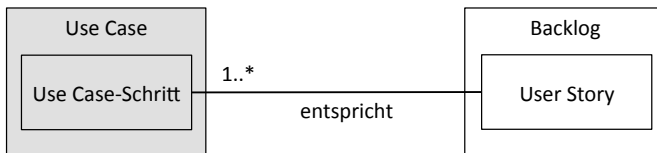


Abbildung 20: Beispiel für eine simple Anforderungslandschaft

Im Kontrast dazu stellt Abbildung 21 eine komplexe Landschaft dar. Diese kann in Projekten zum Tragen kommen, die – wie in Fall 1 (bessere Transparenz und Handhabbarkeit) und Fall 2 (höhere Flexibilität bei Änderungen) der obigen Konstellationen – plangetrieben durchgeführt werden, aber intern eine höhere Flexibilität und Handhabbarkeit der Anforderungen anstreben. Das angegebene Modell orientiert sich im traditionellen Teil am Referenzmodell des artefaktorientierten Requirements Engineering-Ansatzes von Méndez und Penzenstadler [113]. Der agile Teil ist angelehnt an User Story Maps von Patton [125]

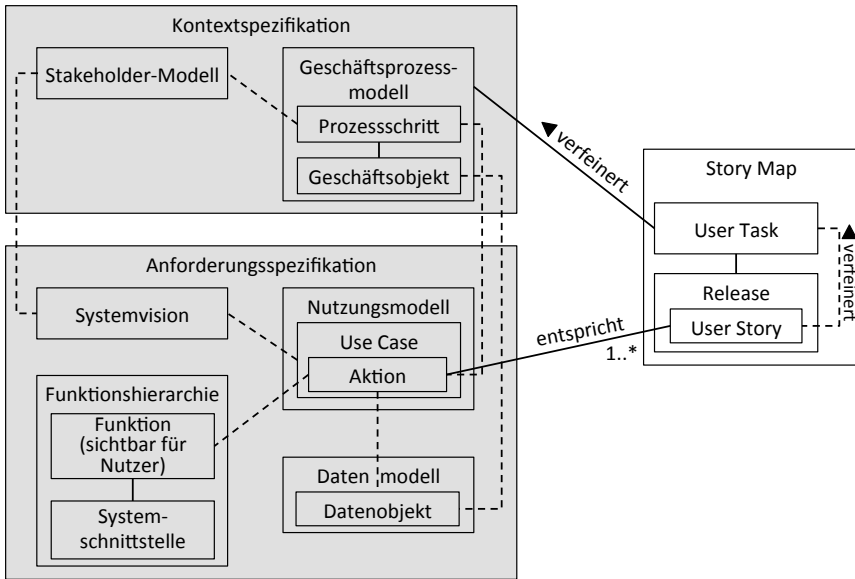


Abbildung 21: Beispiel für eine komplexe Anforderungslandschaft

6.8 Herausforderungen aus der Koexistenz von Anforderungsartefakten

Anforderungslandschaften zeichnen sich zum einen durch die Eigenschaften der einzelnen darin verwendeten Artefakttypen aus. Beispielsweise veranschaulichen GUI-Mockups den Umgang mit der gewünschten Software; User Stories spalten Funktionalität in handhabbare Teile auf und verlagern Detaildiskussionen auf später; Geschäftsprozessmodelle stellen einen Gesamtüberblick her. Andererseits charakterisieren aber auch Effekte, die durch das Zusammenwirken all dieser Artefakte entstehen, eine Anforderungslandschaft.

Das Verständnis vom Zusammenwirken der Artefakte in einer Landschaft ist wichtig, um detailliert absehen zu können, welche Auswirkungen die Hinzunahme und Verknüpfung von Artefakttypen hat und welche Kosten oder auch Synergien durch Relationen mehrerer Artefakte zu erwarten sind. Dazu erarbeiten die folgenden Abschnitte Effekte, die auf das Zusammenwirken von Anforderungsartefakten zurückzuführen sind, und umreißen auf dieser Basis Herausforderungen, die aus der Koexistenz von Artefakten entstehen.

Hierzu wird im nächsten Abschnitt mit Koexistenz-Aktivitäten zunächst ein theoretisches Konstrukt aufgestellt, welches hilft, die Zusatzkosten von Abhän-

gigkeiten zu extrahieren. Dieses wird auch später in Kapitel 8 als Ansatzpunkt zur Unterstützung von Projektteilnehmern durch Senkung dieser zusätzlichen Kosten verwendet.

6.8.1 Koexistenz-Aktivitäten als Kostentreiber von Abhängigkeiten

Eine besondere Rolle für Koexistenz-Effekte spielen Abhängigkeiten zwischen Artefakten. Abhängige Artefakte besitzen überlappende Informationen. Häufig schlagen sie sich letztlich in gemeinsamen Codeteilen nieder, wie die Abbildung 22, in Anlehnung an [44], zeigt. Dies führt dazu, dass Information in einem Artefakt nicht immer isoliert betrachtet werden kann, sondern in Kontext mit anderen Artefakten zu setzen ist.

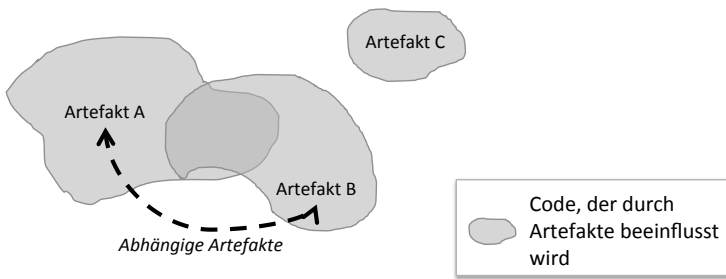


Abbildung 22: Abhängige Artefakte beeinflussen oft überlappende Code-Stellen (angelehnt an [44])

Wenn Projektteilnehmer mit Anforderungsartefakten umgehen, die in Abhängigkeitsrelation stehen, so verursacht die Berücksichtigung der Abhängigkeiten zusätzlichen Aufwand. Um während der Handhabung von Anforderungen auf Abhängigkeiten einzugehen, müssen die regulären Aktivitäten um zusätzliche Schritte erweitert werden. Beispielsweise wäre eine reguläre Aktivität das Einpflegen einer neuen Anforderung als neue User Story. In einer Anforderungslandschaft, die zusätzlich Use Cases besitzt, müsste neben dieser Aktivität noch geprüft werden, ob die neue Anforderung auch mit den Use Cases in Einklang steht und gegebenenfalls die Use Cases ergänzt werden.

Dabei müssen nicht unbedingt schon konkrete Abhängigkeiten herrschen oder gar bekannt sein. Es reicht bereits aus, dass die Anforderungen potenziell zueinander in Abhängigkeitsrelation stehen können, um Zusatzaufwand dadurch zu erzeugen, dass die Projektteilnehmer die Anforderungen auf konkrete Abhängigkeiten überprüfen müssen.

Die zusätzlichen Schritte, die das Aufdecken und Berücksichtigen von Abhängigkeiten darstellen, werden hier separat ausgewiesen und untersucht, um so die

speziellen Effekte der Koexistenz mehrerer Anforderungsartefakte zu beleuchten. Da diese zusätzlichen Aktivitäten speziell bei Koexistenz mehrerer Anforderungsartefakte auftreten, werden sie *Koexistenz-Aktivitäten* genannt. Zur Unterscheidung werden reguläre Aktivitäten, welche die Handhabung der Anforderungen selbst betreffen, *anforderungsbezogene Aktivitäten* genannt.

Beide Arten von Aktivitäten sind verwoben. Es lässt sich nicht genau trennen, wann eine anforderungsbezogene Aktivität, z.B. Änderung einer User Story, aufhört und eine Koexistenz-Aktivität, z.B. entsprechende Anpassung des Geschäftsprozessmodells, beginnt. Die Koexistenz-Aktivitäten sind eher als theoretisches Konstrukt zu betrachten, deren Aufwandsabschätzung hilft, den Aufwand von Abhängigkeiten greifbar zu machen.

Definition anforderungsbezogene Aktivität:

Eine anforderungsbezogene Aktivität ist eine Aktivität, bei der eine Anforderung gehandhabt – das bedeutet erstellt, verwendet oder gewartet – wird.

Definition Koexistenz-Aktivität:

Eine Koexistenz-Aktivität, ist eine Aktivität, um die eine anforderungsbezogene Aktivität ergänzt wird, um Abhängigkeiten der Anforderung zu weiteren Anforderungen zu berücksichtigen.

Die Handhabung einer Anforderung umfasst, angelehnt an die CRUD-Systematik⁶, eine oder mehrere von drei Aktivitäten: Erstellung, Verwendung und Wartung. Dabei fasst die Wartung einer Anforderung hier das Bearbeiten oder Löschen einer Anforderung zusammen. Prinzipiell wird bei der Wartung eine Anforderung auch gelesen, jedoch liegt hier das Augenmerk auf der anschließenden Bearbeitung. Für die Erörterung von Koexistenz-Aktivitäten sind also solche Aktivitäten zu untersuchen, die als Ergänzung bei diesen drei anforderungsbezogenen Aktivitäten vorkommen. Zusätzlich kommen weitere Aktivitäten hinzu, wenn die Abhängigkeiten im Projekt durch Verknüpfungen dokumentiert werden. Im Folgenden werden mögliche Koexistenz-Aktivitäten aufgelistet:

⁶ Das Akronym CRUD steht im Datenbankwesen oder auch im Bereich der RESTful Webservices [135] für die vier Grundoperationen, die mit einem Datensatz bzw. einer Ressource durchgeführt werden können: create, read, update, delete. Alle Transaktionen in einer relationalen Datenbank setzen sich aus diesen Operationen zusammen. Genauso lässt sich auffassen, dass alle Operationen, die mit dokumentierten Anforderungsartefakten als Ressourcen durchgeführt werden, letztlich auf eine der vier Grundoperationen zurückzuführen sind.

Koexistenz-Aktivitäten

Zur Erstellung von Anforderungen:

- Abgleich, ob neue Anforderung im Einklang mit bisher dokumentierten Anforderungen steht
- Einpflegen der neuen Anforderung in allen relevanten Artefakten

Zur Wartung von Anforderungen:

- Abgleich, ob Änderung im Einklang mit bisher dokumentierten Anforderungen steht
- Übertragung der Änderung auf weitere abhängige Artefakte

Zur Verwendung von Anforderungen:

- Ermitteln aller relevanten Informationen zu einer Anforderung, z.B.:
 - Kontext (zusammenhängende Nutzerziele; weitere Funktionen, die vor/nach der Funktion in der Anforderung aufgerufen werden)
 - Konkretisierung (wie konkret wird Anforderung umgesetzt)
 - Constraints (Einschränkungen der Umsetzung)
 - Bisherige Umsetzung
- Aufdecken von Inkonsistenzen zwischen Anforderungen
 - Widersprüche
 - Unvollständige Verfeinerung / Übersetzung in weitere Artefakttypen
- Übersetzung einer Anforderung in anderen Blickwinkel (z.B. gehe Use Case-Ablauf am GUI-Mockup durch, um zu prüfen, ob alle Funktionen in der Oberfläche berücksichtigt sind)
- Übertragung und Aggregation der Eigenschaften eines Artefakttyps auf anderen Artefakttyp (z.B. übertrage Schätzwerte von User Stories auf Geschäftsprozessmodell, um die Umsetzung eines Teilprozesses zu schätzen).
- Übertragung anforderungsbezogener Aktivitäten auf abhängige Artefakte (z.B. stufe ganzen Use Case auf höchste Priorität hoch und daraufhin auch alle damit zusammenhängenden User Stories)
- Tracing (z.B. verfolge Entstehung und Verarbeitung einer Anforderung)

Zur Dokumentation von Abhängigkeiten durch Verknüpfungen:

- Verknüpfung anlegen
- Verknüpfung anpassen

Zusätzliche Kosten durch Koexistenz-Aktivitäten.

Koexistenz-Aktivitäten verursachen zusätzliche Kosten beim Umgang mit Anforderungen. Im Folgenden wird ein Beispiel für konkrete Koexistenz-Aktivitäten innerhalb einer Anforderungsmenge und der daraus heranzuziehenden Koexistenzkosten gezeigt. Abbildung 23 fasst dazu drei Koexistenz-Aktivitäten zusammen, die im Anschluss erläutert werden.

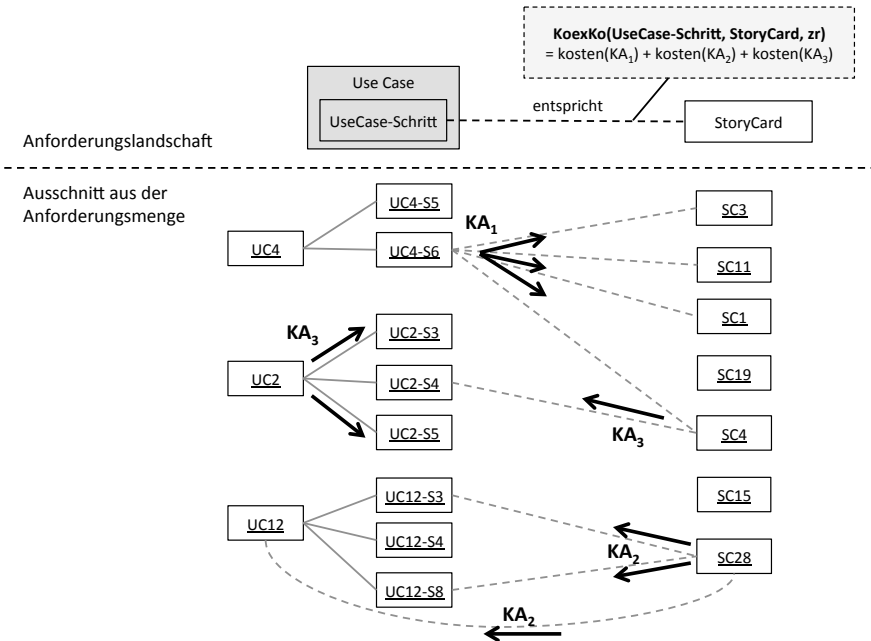


Abbildung 23: Beispiel für Koexistenzkosten in einer Anforderungsmenge

Angenommen, es existiert eine Abhängigkeitsrelation zwischen den beiden Artefakttypen *Use Case-Schritt* und *Story Card* im Projekt. Nehmen wir nun an, dass im Zeitraum *zr* drei Koexistenz-Aktivitäten (KA_x) durchgeführt werden.

1. Zum einen wird der Use Case-Schritt *UC4-S6* geändert. Das beeinflusst zwei (von insgesamt vier abhängigen) Story Cards, die daraufhin umgeschrieben werden (KA_1).
2. Dann wird eine neue Story Card *SC28* eingefügt. Dabei muss geprüft werden, ob die Story Card in Einklang mit dem Use Case *UC12* ist. Das ist sie, weil es eine Detaillierung einer anderen Story Card ist. *SC28* muss mit zwei Use Case-Schritten aus *UC12* verknüpft werden (KA_2).
3. Die Umsetzung der Story Card *SC4* wird begonnen. Dazu sieht sich der Entwickler an, welches Nutzerziel damit verfolgt wird (Use Case *UC2*)

und mit welchen weiteren Schritten zusammen diese Story Card innerhalb des Use Cases ausgeführt wird (Schritte *S3*, *S4*, *S5*), um den Gesamtprozess testen zu können (*KA₃*).

Wir ordnen den drei Koexistenz-Aktivitäten jeweils Kosten *kosten(KA1)*, *kosten(KA2)*, *kosten(KA3)* zu. Auf die Zuordnung von Kosten zu einer ganzen Abhängigkeit, wie auch in der Abbildung rechts oben angegeben, geht der nächste Abschnitt näher ein.

Allein die Notwendigkeit, diese vielen zusätzlichen Aktivitäten während der Softwareentwicklung und der Anforderungskommunikation durchführen zu müssen, stellt viele Projektteilnehmer vor Herausforderungen, wie auch die Interview-Studie gezeigt hat. Ansätze für den Umgang mit diesen Herausforderungen präsentiert diese Arbeit im späteren Verlauf. In Kapitel 7 wird auf methodischer Ebene der Aufbau einer Landschaft so beeinflusst, dass keine unnötigen Koexistenz-Aktivitäten notwendig werden. Die Unterstützung der konkreten Koexistenz-Aktivitäten erfolgt dann auf der Projektebene. Hierzu sei an dieser Stelle auf den Ansatz zur werkzeuggestützten Unterstützung in Kapitel 8 verwiesen.

6.8.2 Rechenmodell zur Bestimmung von Koexistenzkosten einer Abhängigkeitsrelation

Abhängigkeiten – beziehungsweise genauer die Koexistenz-Aktivitäten zur Berücksichtigung von Abhängigkeiten – erzeugen Zusatzaufwand. Das ist ein wichtiges Kriterium, um zu entscheiden, ob zusätzliche Artefakte in eine Landschaft aufgenommen werden oder ob die Landschaft vereinfacht werden sollte. Da die Kosten von Abhängigkeiten zudem von Verknüpfungen beeinflusst werden, können diese auch zur Entscheidung über die Einführung von Verknüpfungen herangezogen werden.

In diesem Abschnitt wird ein vereinfachtes Modell aufgestellt, das es erlaubt, Kosten abzuschätzen und auch unabhängig von der genauen Kostenabschätzung zu bestimmen, welchen Einfluss Verknüpfungen auf die Kosten haben können.

Wie im vorigen Abschnitt vorgestellt, müssen bei Abhängigkeiten zwischen zwei Artefakttypen zu den regulären anforderungsbezogenen Aktivitäten zusätzlich Koexistenz-Aktivitäten durchgeführt werden, um Inkonsistenzen zu vermeiden. Zur Bestimmung der Kosten einer Abhängigkeit werden genau diese Kosten von Koexistenz-Aktivitäten herangezogen.

Definition Koexistenzkosten einer Abhängigkeit innerhalb eines Zeitraumes:

Die Koexistenzkosten $KoexKo(A, B, ZR)$ einer Abhängigkeit zwischen Artefakttyp A und Artefakttyp B innerhalb eines Zeitraumes ZR sind die Summe der Kosten aller Koexistenz-Aktivitäten zwischen Artefakten der Typen A und B , die in diesem Zeitraum stattfinden.

Im Beispiel aus dem vorigen Abschnitt (Abbildung 23) sind die Koexistenzkosten der Abhängigkeit zwischen Use Case-Schritten und Story Cards im Zeitraum zr definiert als:

$$KoexKo(UseCase-Schritt, StoryCard, zr) = kosten(KA_1) + kosten(KA_2) + kosten(KA_3).$$

Aufgrund der hohen Anzahl an Koexistenz-Aktivitäten, die in einer Zeiteinheit, wie einer Iteration, auftreten, ist es nicht praktikabel, die Kosten für jede Koexistenz-Aktivität wie im obigen Beispiel einzeln zu ermitteln und zu summieren. Stattdessen werden die Koexistenz-Aktivitäten alle gleich behandelt und die Einzelkosten über deren Durchschnittswert angenähert. Dies vereinfacht das Rechenmodell so, dass sich die Koexistenzkosten einer Abhängigkeit für eine Zeiteinheit ⁷ als Produkt *Durchschnittskosten* \times *Auftrittsfrequenz* von Koexistenz-Aktivitäten errechnen lassen.

Eine Art von Koexistenz-Aktivitäten wird allerdings gesondert behandelt: die Erstellung neuer Verknüpfungen weisen wir als einzelnen Summand aus. Diese Aktivität tritt nur auf, wenn Verknüpfungen eingesetzt werden, was nicht bei jeder Abhängigkeit der Fall ist. Zudem ist diese Aktivität eine, die Zusatzkosten verursacht, während alle anderen Koexistenz-Aktivitäten durch Verknüpfungen – unter bestimmten Umständen, auf die noch eingegangen wird – günstiger werden. Um später die Effekte der Einführung von Verknüpfungen zu bestimmen, stellen wir die Erstellung neuer Verknüpfungen einzeln dar. Im weiteren Verlauf werden unter dem Sammelbegriff Koexistenz-Aktivitäten (KA) alle Koexistenz-Aktivitäten aus obiger Liste mit Ausnahme der Aktivität „Erstellung neuer Verknüpfungen“ verstanden. Die Aktivität „Erstellung neuer Verknüpfungen“ (*Verkn*) wird gesondert herausgestellt.

Die geschätzten Koexistenzkosten einer Abhängigkeit berechnen sich nun wie folgt:

⁷ Bei der Addition der Kosten einzelner Koexistenz-Aktivitäten, musste der Zeitraum angegeben werden, in dem die konkreten Aktivitäten stattfanden. Da nun Durchschnittskosten geschätzt werden, sind diese für jeden Zeitraum mit derselben Dauer gleich. Daher werden die geschätzten Koexistenzkosten in Bezug auf eine Zeiteinheit, wie einen Tag, eine Woche oder eine Iteration, angegeben.

Definition Geschätzte Koexistenzkosten einer Abhängigkeit innerhalb einer Zeiteinheit:

$$KoexKo(A, B, ZE) = freq_{KA} \cdot kosten_{KA} + freq_{Verkn} \cdot kosten_{Verkn}$$

mit:

A, B Artefakttypen

ZE Zeiteinheit (z.B. ein Tag, ein Monat, eine Iteration)

$freq_{KA}$ = Auftrittsfrequenz von Koexistenz-Aktivitäten

$kosten_{KA}$ = Durchschnittskosten von Koexistenz-Aktivitäten

$freq_{Verkn}$ = Auftrittsfrequenz der Koexistenz-Aktivität „neue Verknüpfung erstellen“

$kosten_{Verkn}$ = Durchschnittskosten der Koexistenz-Aktivität „neue Verknüpfung erstellen“

Beispiel. Im obigen Beispiel betragen die Koexistenzkosten der Relation zwischen Use Case-Schritten und Story Cards

$$KoexKo(UseCase-Schritt, StoryCard, zr) = kosten(KA_1) + kosten(KA_2) + kosten(KA_3).$$

Bei Verwendung der Durchschnittskosten, hier mit $kosten_{KA}$ dargestellt, ergibt sich nun (unter der Annahme, dass $zr = 1$ Tag ist):

$$KoexKo(UseCase-Schritt, StoryCard, 1Tag) = 3 * kosten_{KA}.$$

Praktische Bestimmung der Koexistenzkosten.

Die Verallgemeinerung der geschätzten Koexistenzkosten von einem konkreten Zeitraum auf eine Zeiteinheit (wie eine Iteration, Woche oder Tag) bedeutet, dass diese geschätzten Kosten auf einen beliebigen Zeitraum der Dauer der Zeiteinheit zutreffen. Die Einführung von *geschätzten Koexistenzkosten für eine Iteration* macht demnach nur Sinn, wenn diese beispielsweise sowohl in *Iteration 3* als auch in *Iteration 14* in etwa zutreffend sein wird. Diese Verallgemeinerung auf eine allgemeine Zeiteinheit ist also nur sinnvoll, wenn sich die Koexistenzkosten auch tatsächlich nicht grundlegend von Zeitraum zu Zeitraum ändern.

Tatsächlich können sich die Koexistenzkosten einer Abhängigkeit jedoch mit der Zeit ändern: Wenn mit der Zeit die Zahl der Anforderungen und damit auch die Zahl der Abhängigkeiten steigt, so werden die Durchschnittskosten einer einzelnen Koexistenz-Aktivität, $kosten_{KA}$, höher, weil zum Auffinden von Abhängigkeiten mehr Elemente durchsucht und gefiltert werden müssen. Hierzu kann das Rechenmodell so erweitert werden, dass es den Zeitverlauf berücksich-

tigt. Beispielsweise wird an jedem Zeitpunkt t_1, t_2, \dots, t_n ein Anstieg der Koexistenzkosten um p Prozent angesetzt. Dann ergeben sich die Durchschnittskosten einer Koexistenz-Aktivität zum Zeitpunkt t durch

$$\text{kosten}_t = \text{kosten}_1 * \left(1 + \frac{p}{100}\right)^t.$$

Im weiteren Verlauf reicht jedoch das simple Rechenmodell aus, da die Koexistenzkosten immer im selben Zeitraum betrachtet und nicht zwischen Zeiträumen gemischt werden.

Einfluss von Verknüpfungen auf die Koexistenzkosten von Abhängigkeiten.

Die Koexistenzkosten hängen unter anderem davon ab, ob die Abhängigkeiten mithilfe von Verknüpfungen dokumentiert werden. Verknüpfungen helfen, die Suche nach abhängigen Elementen zu erleichtern. Im Folgenden konkretisieren wir das Rechenmodell so, dass wir den Einfluss von Verknüpfungen auf die Koexistenzkosten ermitteln können.

Zunächst beeinflusst die Einführung von Verknüpfungen vor allem die *individuellen Kosten der einzelnen Koexistenz-Aktivitäten* – und damit auch die durchschnittlichen Kosten einer Koexistenz-Aktivität. Für die jeweiligen Kosten definieren wir zwei Variablen k_m und k_v :

Definition Durchschnittskosten für manuelle / verknüpfungsgestützte Koexistenz-Aktivität

k_m = Durchschnittskosten einer manuell durchgeführten Koexistenz-Aktivität

k_v = Durchschnittskosten einer verknüpfungsgestützten Koexistenz-Aktivität

Weiterhin müssen die *Kosten zur Erstellung von Verknüpfungen* ausgewiesen werden. Im Fall der manuellen Durchführung von Koexistenz-Aktivitäten betragen diese 0, da keine Verknüpfungen erstellt werden müssen. Im Fall der Verwendung von Verknüpfungen fallen diese Kosten jedoch an. Zur Vereinfachung nehmen wir an, dass diese Kosten in etwa genauso hoch sind, wie die Durchschnittskosten zur manuellen Durchführung einer normalen Koexistenz-Aktivität, da in beiden Fällen abhängige Elemente manuell gefunden werden müssen.

Annahme: Die durchschnittlichen Kosten der Koexistenz-Aktivität „Erstellung von Verknüpfungen zu einem Artefakt“ sind in etwa gleichzusetzen mit den durchschnittlichen Kosten für die manuelle Durchführung einer Koexistenz-Aktivität (k_m).

Begründung: In beiden Fällen müssen zunächst abhängige Elemente manuell gefunden werden. Im Fall einer normalen Koexistenz-Aktivität, um dann diese Elemente beispielsweise anzupassen. Im Fall der Erstellung von Verknüpfungen, um eine Verknüpfung zum neuen Element zu erzeugen.

Bei Verzicht auf die Annahme: Die Kosten müssen dann gesondert, beispielsweise in einer Variablen k_l ausgewiesen werden. Für die anschließenden Rechnungen ist es notwendig, k_l in Verhältnis zu den Kosten c_m zu setzen, also $k_l = a \cdot k_m$ anzusetzen. a lässt sich dann experimentell bestimmen.

Eine weitere Vereinfachung des Rechenmodells ist die Annahme, dass die Auftrittsfrequenz von Koexistenz-Aktivitäten nicht davon abhängt, ob Abhängigkeiten manuell überprüft werden müssen oder ob dazu Verknüpfungen verwendet werden können. Dies ist deswegen eine Vereinfachung, weil prinzipiell denkbar ist, dass explizite Verknüpfungen die Hürde für Koexistenz-Aktivitäten senken und somit deren Anwendungsfrequenz erhöhen.

Annahme: Die Auftrittsfrequenz von Koexistenz-Aktivitäten $freq_{KA}$ wird nicht davon beeinflusst, ob Verknüpfungen im Projekt verwendet werden oder nicht.

Begründung: Geht man davon aus, dass Koexistenz-Aktivitäten immer dann ausgeführt werden, wenn dies zur Vermeidung von Inkonsistenzen notwendig ist, so ist dies immer dann der Fall, wenn eine anforderungsbezogene Aktivität ausgeführt wird. Das Auftreten dieser Aktivitäten hängt nun nicht davon ab, ob Verknüpfungen verwendet werden.

Bei Verzicht auf die Annahme: Wenn die Annahme nicht zutrifft, können die Auftrittsfrequenzen nicht gleichgesetzt werden. Beispielsweise ist denkbar, dass das Vorhandensein von Verknüpfungen die Identifikation abhängiger Elemente erleichtert und damit dazu führt, dass auch häufiger nach Abhängigkeiten gesucht wird. In diesem Fall sind – wie schon bei den Kosten c_v und c_m – für die Auftrittsfrequenzen getrennte Variablen, $freq_{KA,v}$ und $freq_{KA,m}$ aufzuführen. Sie können jedoch in Bezug zueinander gesetzt werden, beispielsweise $freq_{KA,v} = 1,5 \cdot freq_{KA,m}$.

Die gesamten Koexistenzkosten lassen sich nun wie folgt konkretisieren: Für die Kosten zur Durchführung von Koexistenz-Aktivitäten ($kosten_{KA}$) werden k_m respektive k_v eingesetzt. Für die Kosten zur Erstellung neuer Verknüpfungen ($kosten_{Verkn}$) werden die Kosten k_m angesetzt. Die Auftrittsfrequenz $freq_{KA}$ ist aufgrund der obigen Annahme in beiden Situationen gleich. Die gesamten Koexistenzkosten ergeben sich dann als:

$$\begin{aligned} KoexKo_m(A, B, ZE) &= freq_{KA} \cdot k_m \\ KoexKo_v(A, B, ZE) &= freq_{KA} \cdot k_v + freq_{Verkn} \cdot k_m \end{aligned}$$

Für die bessere Lesbarkeit werden nun außerdem die Schreibweisen vereinfacht: $KoexKo_x(A,B,ZE)$ wird abgekürzt durch $KoexKo_x$; $freq_x$ wird abgekürzt durch f_x

$$\begin{aligned} KoexKo_m &= f_{KA} \cdot k_m \\ KoexKo_v &= f_{KA} \cdot k_v + f_{Verkn} \cdot k_m \end{aligned}$$

Wann lohnen sich Verknüpfungen?

Wir machen den Einfluss von Verknüpfungen an einer Änderung der Koexistenzkosten aus. Verknüpfungen sollten genau dann eingesetzt werden, wenn die Kostenänderung eine Ersparnis darstellt. Verknüpfungen können die Koexistenzkosten senken, indem sie die Identifikation von abhängigen Artefakten vereinfachen. Jedoch verursachen die Verknüpfungen auch selbst Kosten, denn sie müssen angelegt werden. Im Folgenden wägen wir ab, wann Verknüpfungen die Koexistenzkosten senken und wann sie diese möglicherweise eher erhöhen. Dazu betrachten wir die Ersparnis der Koexistenzkosten, die sich als Kostendifferenz der beiden Alternativen *manuelle Koexistenz-Aktivitäten* und *verknüpfungsgestützte Koexistenz-Aktivitäten* ergibt.

Ersparnis durch Verknüpfungen (E) :

$$\begin{aligned} E &= KoexKo_m - KoexKo_v \\ &= f_{KA} \cdot k_m - (f_{KA} \cdot k_v + f_{Verkn} \cdot k_m) \\ &= f_{KA} \cdot (k_m - k_v) - f_{Verkn} \cdot k_m \end{aligned}$$

Wir gehen davon aus, dass die Einzelkosten k_m und k_v nicht negativ sein können, also immer $k_m, k_v \geq 0$ gilt.

Reflexion: Falls $k_m < k_v$ gilt, also die Durchführung einer Koexistenz-Aktivität mit Verknüpfungen teurer ist als ohne Verknüpfungen, so ist die Ersparnis immer negativ (solange $k_m, k_v \geq 0$ gilt).

Es ist davon auszugehen, dass eine Koexistenz-Aktivität mithilfe von Verknüpfungen günstiger wird, da die Verknüpfungen die Identifikation abhängiger Elemente erleichtern. Dennoch könnte der Fall $k_m < k_v$ beispielsweise auftreten, wenn die Verknüpfungen den Abhängigkeiten nicht gut entsprechen. Enthalten die Verknüpfungen viele falsch-positive Ergebnisse (*false positives*) indem irrelevante Elemente verknüpft werden oder viele falsch-negative Ergebnisse (*false negatives*) indem relevante Elemente nicht verknüpft werden, so erhöhen sich die Kosten für Koexistenz-Aktivitäten. Sind viele *false positives* vorhanden, so müssen die Projektteilnehmer die verknüpften Elemente noch einmal manuell filtern, um die tatsächlich relevanten zu finden. Sind viele *false*

negatives vorhanden, so müssen die Projektteilnehmer zusätzliche manuelle Suchen durchführen, um alle relevanten Elemente zu identifizieren.

Betrachten wir nun den Fall, dass $k_m > k_v$ (bzw. $k_m - k_v > 0$) gilt, so hängt die Ersparnis von der Größe der Differenz $k_m - k_v$ und von der Relation zwischen f_{KA} , was der potenziellen *Verwendung von Verknüpfungen* entspricht (und eine Ersparnis bringt) und f_{Verkn} , also der *Erstellung von Verknüpfungen* (die zusätzliche Kosten bringt), ab:

Reflexion: Grundsätzlich wird die Ersparnis größer, je größer die Differenz der durchschnittlichen Einzelkosten, also die Differenz $(k_m - k_v)$, ist. Je mehr in jeder Koexistenz-Aktivität durch die Verwendung von Verknüpfungen eingespart werden kann, desto geringer fallen dann auch die Gesamtkosten aus.

Diese Feststellung ist interessant, weil die Differenz durchaus klein werden kann. Ist k_m grundsätzlich niedrig, so kann die Differenz $(k_m - k_v)$ auch niedrig ausfallen. Dies kann beispielsweise dann der Fall sein, wenn nur wenige Artefakte vorhanden sind oder die Struktur der Anforderungen gut ist, sodass passende Elemente schnell gefunden werden können.

Auch hier (im Fall $k_m - k_v > 0$) ist es möglich, dass der Einsatz von Verknüpfungen sich nicht lohnt: Wenn Verknüpfungen zu oft erstellt werden müssen und zu selten verwendet werden, so können die Zusatzkosten für die Erstellung die Ersparnis übersteigen. Um eine positive Ersparnis zu erhalten, muss folgendes gelten:

$$\begin{aligned}
 & 0 < E \\
 \Leftrightarrow & 0 < f_{Verkn} \cdot (k_m - k_v) - f_{KA} \cdot k_m \\
 \Leftrightarrow & f_{Verkn} \cdot k_m < f_{KA} \cdot (k_m - k_v) \\
 \xleftrightarrow{k_m - k_v > 0, f_{Verkn} > 0} & \frac{k_m}{k_m - k_v} < \frac{f_{KA}}{f_{Verkn}} \\
 \Leftrightarrow & \frac{1}{\left(\frac{k_m - k_v}{k_m}\right)} < \frac{f_{KA}}{f_{Verkn}}
 \end{aligned}$$

$\left(\frac{k_m - k_v}{k_m}\right)$ stellt die relative Ersparnis der Durchschnittskosten bei einzelnen Koexistenz-Aktivitäten dar. Der **break-even-Punkt**, an dem die Einführung von Verknüpfungen vorteilhaft bzw. nachteilhaft wird, hängt umgekehrt proportio-

nal davon ab, wie hoch die relative Ersparnis ist und wie häufig sie zum Tragen kommt. Fassen wir $(\frac{k_m - k_v}{k_m})$ als unabhängige Variable auf so lässt sich die Situation $\frac{f_{KA}}{f_{Verkn}} > \frac{1}{(\frac{k_m - k_v}{k_m})}$ mithilfe der Hyperbelfunktion $y = \frac{1}{x}$ illustrieren, wie in Abbildung 24 gezeigt.

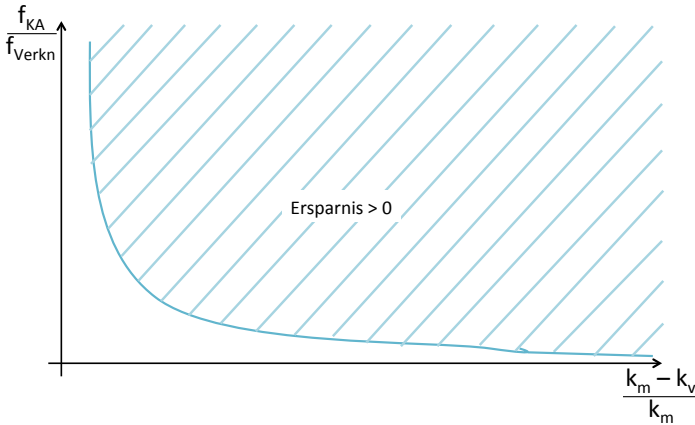


Abbildung 24: Verlauf des break-even-Punktes in Abhängigkeit der relativen Ersparnis und der Verwendungshäufigkeit von Verknüpfungen

Ist also beispielsweise $\frac{k_m - k_v}{k_m} = 0,25$ ($\Leftrightarrow k_v = (1 - 0,25) \cdot k_m = 0,75 \cdot k_m$), d.h. mit Verknüpfungen können Aktivitäten 25% schneller durchgeführt werden, so lohnt sich die Verknüpfung, sobald $\frac{f_{KA}}{f_{Verkn}} > \frac{1}{0,25} \Leftrightarrow f_{KA} > 4 \cdot f_{Verkn}$ gilt, also die Verknüpfungen an der Relation mehr als viermal so häufig verwendet werden, wie sie erstellt werden müssen.

Im Experiment, das ausführlicher in Kapitel 9 vorgestellt wird, hat sich gezeigt, dass für komplexe Aufgaben⁸ das Kostenverhältnis bei $\frac{k_m - k_v}{k_m} = 0,33$ bzw. $k_v = 0,66 \cdot k_m$ liegt. Dort wurden die Kosten als (umgekehrte) Performance für die ganze Aufgabe gemessen. Diese setzen sich aus der Dauer und dem Fehleranteil zusammen, zu dem auch die vergessene Berücksichtigung von Abhängigkeiten zählt. Hier müssen Verknüpfungen dreimal so häufig verwendet werden, um eine positive Ersparnis zu erhalten.

⁸ Unter *komplexer* Aufgabe wird im Experiment eine Aufgabe verstanden, bei der Artefakte von mehr als einem Artefakttyp herangezogen werden müssen, um die Aufgabe zu lösen. Im Gegensatz dazu ist eine *simple* Aufgabe auch bereits mit Artefakten von nur einem Artefakttyp lösbar.

Reflexion: In hybriden Landschaften wird ein solcher Wert durchaus entscheidend. Einzelne agile Artefakte, wie eine einzelne User Story wird für sich relativ wenige Male verwendet, sodass es möglich ist, unter einem Grenzwert, wie 3 Verwendungen zu liegen. Auch können Verknüpfungen hier schneller obsolet werden, als in traditionellen Projekten, da sie nach der Fertigstellung einer User Story möglicherweise überhaupt nicht mehr verwendet werden.

In einem typischen agilen Projekt wird eine User Story bzw. ihre Verknüpfung voraussichtlich einmal in der Releaseplanung, einmal in der Iterationsplanung, einmal bei ihrer Implementierung und dann einmal zur Abnahme betrachtet. Das sind bis zu vier Verwendungen, sodass die erwartete Ersparnis hier durchaus gering ausfällt. Erst wenn die Verknüpfungen beispielsweise auch nach der Fertigstellung einer User Story zur Nachverfolgung genutzt werden oder wenn die Verknüpfungen verwendet werden, um von einem anderen Artefakttyp zu User Stories zu navigieren, kann man mit einer signifikanten Ersparnis rechnen.

Das Rechenmodell gibt Requirements Methodikern einen ersten Anhaltspunkt zur Entscheidung, eine Abhängigkeit durch eine Verknüpfung zu unterstützen. Wie man sieht, hängt es nicht nur von der Art der Abhängigkeit ab, sondern auch davon, wie die entsprechenden Artefakte verwendet werden.

Dass das Verhältnis von Erstellung zur Verwendung von Verknüpfungen relevant ist, wird auch im risikobasierten Optimierungsansatz (Kapitel 7) in Form jeweils eines Risikos der Verwendung und des Weglassens von Verknüpfungen aufgegriffen.

Weiterhin zeigt das Rechenmodell, an welchen Aspekten Lösungen ansetzen können, um die Verwendung von Traceability-Ansätzen lohnenswerter zu machen. Beispielsweise können Trace-Operationen, die in Kapitel 8 vorgestellt werden, die Kosten zur Verwendung, aber auch die Kosten zur Erstellung von Verknüpfungen reduzieren und so dazu beitragen, dass Verknüpfungen sich eher lohnen. Traceability-Operationen verwenden Verknüpfungen automatisiert und präsentieren Nutzern direkt die Resultate. Damit senken sie die Kosten für einige Koexistenz-Aktivitäten, wie die Identifikation abhängiger Artefakte. Es kommt hinzu, dass einige Trace-Operationen automatisiert ausgelöst werden. Gerade in Situationen, in denen Projektteilnehmer vergessen, abhängige Artefakte zu betrachten und anzupassen, können Trace-Operationen sie daran erinnern. Trace-Operationen steigern damit den Nutzen von Verknüpfungen und tragen so zu einem besseren Kosten-Nutzen-Verhältnis bei.

6.8.3 Zusätzliche Herausforderungen in hybriden Anforderungslandschaften

In hybriden Anforderungslandschaften treffen zähe und flexible Anforderungsartefakte aufeinander. Neben den üblichen Kosten, die entstehen, weil Koexistenz-Aktivitäten durchgeführt werden müssen, kommen hier zusätzliche Herausforderungen hinzu, die der Requirements Methodiker ebenfalls bei der Planung des Ansatzes betrachten sollte. In diesem Abschnitt wird angegeben, welche Herausforderungen dies sind und durch welche Mechanismen diese berücksichtigt werden können.

1. Anpassung zäher Artefakte wird nach schneller Änderung eines flexiblen Artefaktes schnell vergessen: In hybriden Landschaften treffen flexible und zähe Artefakte – beziehungsweise die Konzepte schneller Änderungen und eines festen groben Rahmens – aufeinander. In dieser Situation ist besonders darauf zu achten, dass Änderungen, die beispielsweise schnell in Story Cards festgehalten werden, bei Bedarf auch auf zähe Artefakte übertragen werden. Dies wird jedoch oft bei der Verwendung eines flexiblen Artefaktes als primäres Artefakt vernachlässigt, was zur Folge hat, dass sich das Projekt unbemerkt vom Rahmen der Spezifikation entfernt.

2. Änderungen, die ein zähes Artefakt betreffen, müssen zusätzlich explizit und transparent klar gemacht werden: Zähe Artefakte stellen einen festen Rahmen dar, gegen den man nicht ohne weiteres verstoßen kann. Muss bei einer Koexistenz-Aktivität ein zähes Artefakt angepasst werden, so muss diese Änderung zusätzlich abgesprochen werden. Änderungen an dem Rahmen sind natürlich möglich. Werden sie jedoch nicht explizit abgestimmt und transparent dokumentiert, kann dies dazu führen, dass sich das Projekt unbemerkt vom Rahmen entfernt und die Stakeholder am Projektende unzufrieden sind, weil sie noch immer am Rahmen festhalten.

3. Verzögerungen von Änderungen durch Abstimmungsprozesse: Enthält eine Anforderungslandschaft ein zähes Artefakt, so ist zudem mit Verzögerungen bei Anforderungs-Änderungen zu rechnen, da bestimmte Parteien der Änderung zustimmen müssen. In dieser Situation sollten zusätzliche Mechanismen eingesetzt werden, eine *ausstehende Änderung* direkt in die Anforderungen einzupflegen, um beispielsweise mit Stakeholdern die Konsequenzen der geplanten Änderungen im Vorfeld durchgehen zu können.

4. Zähe Artefakte erhöhen Dokumentationshürde: Wird hingegen ein zähes Artefakt allein eingesetzt, so kann dies dazu führen, dass Änderungen an den geplanten Funktionalitäten vorgenommen, jedoch wegen des hohen Aufwandes nicht dokumentiert werden. Auch dies kann dazu führen, dass sich das Projekt unbemerkt vom festgelegten Rahmen entfernt und dass zusätzlich keine Möglichkeit besteht, nachzuverfolgen wann eine Abweichung überhaupt stattgefunden hat. In einer solchen Situation kann es Sinn machen, ein flexibles Artefakt

6.8 Herausforderungen aus der Koexistenz von Anforderungsartefakten

in die Anforderungslandschaft mit aufzunehmen, da so durch die niedrigere Dokumentationshürde potenziell die Dokumentation von Änderungen verbessert werden kann.

Mechanismen, die mit obigen Herausforderungen helfen können, werden in den folgenden Kapiteln vorgestellt. Die Erinnerung an und explizite Dokumentation von Anpassungen wird durch werkzeuggestützte Mechanismen, wie in Kapitel 8 vorgestellt, unterstützt. Entscheidungen, ob – wie im vierten Aspekt der Fall – zusätzliche Artefakte hinzugefügt werden, werden in Kapitel 7 aufgegriffen.

7 Risikobasierte Optimierung von Anforderungslandschaften

Der Aufbau einer Anforderungslandschaft bestimmt, wie gut die Projektteilnehmer mit den Anforderungen umgehen können. Dies beeinflusst wiederum viele abhängige Projektaktivitäten. Wie sich bereits in der Interview-Studie gezeigt hat, kann eine unpassend gewählte Landschaft die Anforderungskommunikation verschlechtern oder den Aufwand für Dokumentation und auch das Finden und Verstehen von Anforderungen unnötig erhöhen. Dies verursacht hohe Kosten für das Requirements Engineering und führt zu Fehlern durch schlechte Kommunikation und Inkonsistenzen.

Wenn Artefakte fehlen, so können Projektteilnehmer möglicherweise bestimmte Aspekte der Anforderungen nicht verstehen. Sind jedoch zu viele verschiedene oder zu schwergewichtige Artefakttypen eingesetzt, so wird der Dokumentationsaufwand zu hoch und verzögert das gesamte Projekt. Ähnlich verhält es sich mit Verknüpfungen zwischen Anforderungsartefakten. Diese dokumentieren Abhängigkeiten zwischen Artefakten und helfen, abhängige Artefakte schnell zu identifizieren. Gleichzeitig ist ihre Erstellung und Wartung aufwändig.

Ziel ist es daher – sowohl bei der Erstellung von Anforderungslandschaften, aber auch im Lauf eines Projektes⁹ – genau die richtige Menge an dokumentierten Anforderungen und dokumentierten Abhängigkeiten zu besitzen. Dazu stellt diese Arbeit eine risikobasierte Methode zur lokalen Optimierung von Anforderungslandschaften vor. Optimierungsdimensionen und relevante Risikofaktoren dienen der Bewertung von Anforderungslandschaften. Diese werden aus der Interview-Studie abgeleitet. Die Methode ermöglicht einem Requirements Methodiker so, systematisch auf Erfahrungen zu Anforderungslandschaften zuzugreifen.

7.1 Risk-based Balancing als Optimierungsmethode

Die Schwierigkeit bei der Optimierung von Anforderungslandschaften ist, dass hierbei viele Einflussaspekte zu berücksichtigen sind. Es muss festgelegt werden, welche Artefakttypen in der Landschaft enthalten sind, ob sie sich ändern dürfen oder ob sie mit anderen Artefakten verknüpft werden sollen. Zu jedem dieser Einflussaspekte gibt es zudem eine Reihe von Vor- und Nachteilen, die teilweise gegensätzlich sind. Beispielsweise hat die Verknüpfung von Use Cases mit Story Cards den Vorteil, dass zu einer Story Card schnell das entsprechende Nutzerziel gefunden werden kann. Die Verknüpfung hat aber den Nachteil, dass beim Anlegen einer neuen Story diese mit allen passenden Use Case-Teilen verknüpft werden muss, was zusätzlichen Aufwand erzeugt und Aktivitäten wie die Iterationsplanung behindert.

⁹ Wie in der Interview-Studie festgestellt wurde, ändern sich die verwendeten Anforderungsartefakttypen oder ihre Verwendung oft im Laufe eines Projektes.

Aufgrund der vielen Einflussmöglichkeiten lässt sich nicht einfach ein optimales Vorgehen finden, das in allen Situationen erfolgreich ist. Gerade in hybriden Anforderungslandschaften gibt es, wie in Abschnitt 6.7 gezeigt, unterschiedliche Ziele, die mit der Aufstellung verfolgt werden. Welche Einstellung die wichtigeren Vor- und Nachteile bietet, ist zudem vom Projekt selbst abhängig. Die Größe des Projektes, Anzahl und Verfügbarkeit der Stakeholder sowie die Beständigkeit der Anforderungen spielen hier zum Beispiel eine wichtige Rolle [16], [167]. Es wird eine Möglichkeit benötigt, die hilft, Vor- und Nachteile gegenseitig abzuwägen und sie in Beziehung zum Projektkontext zu bewerten.

In dieser Arbeit wird hierzu ein risikobasiertes Verfahren vorgestellt, das ermöglicht, die verschiedenen Aspekte in Bezug auf ihr Risiko vergleichbar zu machen. Es baut auf dem *risikobasierten Ansatz zum Ausbalancieren von agilen und plangetriebenen Methoden* von Boehm und Turner auf [17]. Dort wird abgewogen, wann die grundsätzliche Herangehensweise eher agil oder plange-trieben sein sollte und wann Mischformen verwendet werden sollten.

Der Ansatz zielt – ähnlich wie das Spiralmodell von Boehm [14] – darauf ab, die Projektrisiken, die mit einer Vorgehensweise verbunden sind, zu minimieren. Das Grundprinzip besteht darin, dass beide Alternativen, also eine agile oder plangetriebene Vorgehensweise, Risiken bergen, die im eigenen Projektkontext bewertet werden müssen. Hat eine der Alternativen ein geringeres Risiko, so wird das Projekt auf diese Alternative ausgerichtet. Die größten identifizierten Risiken werden zusätzlich abgedämpft, was meist durch gezielte Hinzu-nahme bestimmter agiler und plangetriebener Praktiken geschieht. Abbildung 25 zeigt einen Ausschnitt des Vorgehens, wie es in [17] beschrieben wird. In Ab-bildung 26 ist ein Teil der Liste von Risikofaktoren abgebildet, die dort heran-gezogen werden, um die Alternativen abzuwägen.

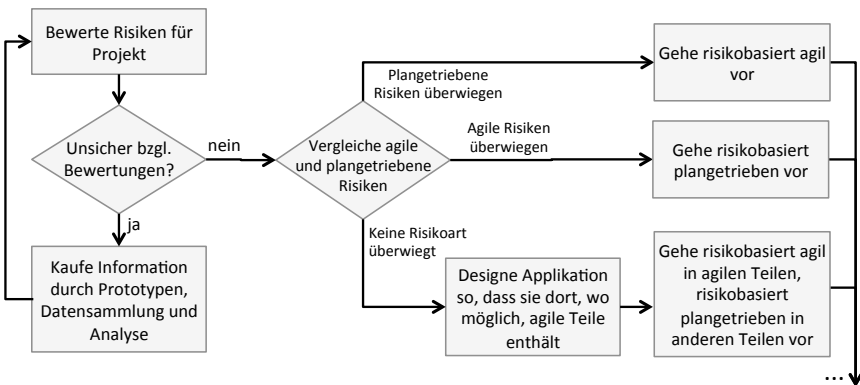


Abbildung 25: Vorgehen bei der Erstellung einer risikobasierten Methode (nach Boehm und Turner [17])

Risikofaktor	Risikobewertung für Beispielprojekt „Event Managers“
Umweltbedingte Risikofaktoren	
Unsicherheiten bezüglich der Technologie	■
Viele Stakeholder	□
...	...
Risikofaktoren der agilen Herangehensweise	
Skalierbarkeit und Kritikalität	□
Personelle Veränderungen	■ ■
...	...
Risikofaktoren der plangetriebenen Herangehensweise	
Schneller Wandel (Rapid Change)	■ ■ ■ ■
Notwendigkeit schneller Resultate	■ ■ ■ ■
...	...

□ Minimales Risiko
 ■ Moderates Risiko
 ■ ■ Ernstes aber handhabbares Risiko
 ■ ■ ■ Sehr ernstes aber handhabbares Risiko
 ■ ■ ■ ■ Inakzeptables Risiko

Abbildung 26: Risikotabelle für Abwägung zwischen agiler und plangetriebener Vorgehensweise (Ausschnitt aus [17])

7.1.1 Anwendung auf Optimierung von Anforderungslandschaften

Zur Optimierung von Anforderungslandschaften werden nun, genau wie bei Boehm und Turner, ebenfalls Alternativen basierend auf deren Risiken abgewogen. Dabei gibt es in Anforderungslandschaften mehrere relevante Aspekte, bezüglich derer eine Landschaft optimiert werden kann. Beispielsweise muss entschieden werden, ob *Artefakttypen* in die Landschaft aufgenommen werden und ob Artefakttyppaare miteinander *verknüpft* werden sollen oder nicht. Diese relevanten Aspekte werden **Optimierungsdimensionen** von Anforderungslandschaften genannt. Sie beziehen sich auf ein konkretes Element in einer Anforderungslandschaft und besitzen mehrere Alternativen, die gegeneinander abgewogen werden. In dieser Arbeit wird für jede solche Dimension eine Tabelle mit Risikofaktoren, wie in Abbildung 26 zu Boehm und Turners Ansatz gezeigt, erstellt. Zu jeder Dimension lässt sich damit zwischen verschiedenen Alternativen abwägen.

Das hier vorgestellte Verfahren basiert somit vor allem auf einer Reihe von Risikobewertungen, die vom Requirements Methodiker im jeweiligen Projektkontext durchgeführt werden. Abbildung 27 zeigt hierzu den Überblick. Die konkreten Elemente, wie Optimierungsdimensionen, Alternativen und Risikofaktoren werden aus Aussagen der Interview-Studie abgeleitet und basieren damit auf Praxiserfahrungen. Die Methode zur Ableitung wird in Abschnitt 7.1.2 beschrieben. Die Begriffe Risiko und Risikofaktor werden hier synonym verwendet.

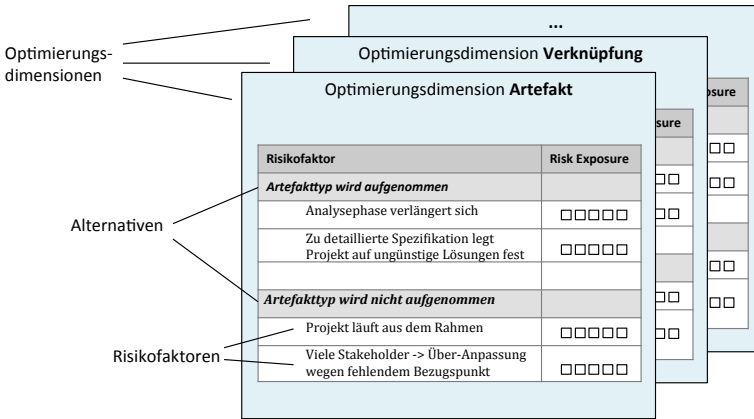


Abbildung 27: Risikotabellen für die Optimierung verschiedener Dimensionen einer Anforderungslandschaft

Bei der Optimierung müssen nicht alle Optimierungsdimensionen durchlaufen werden. Es ist möglich, die Landschaft *lokal* an einer bestimmten Stelle bezüglich eines bestimmten Aspektes zu optimieren, ohne dabei auf andere Aspekte eingehen zu müssen. Man kann beispielsweise abwägen, ob ein bestimmter Artefakttyp in die Landschaft aufgenommen werden sollte. Man kann aber auch direkt zu zwei vorhandenen Artefakten abwägen, ob diese durch eine Verknüpfung verbunden werden sollen – und das unabhängig davon, ob bereits die Existenz der Artefakttypen möglicherweise nicht risikogünstig ist. Abbildung 28 veranschaulicht dies an einer beispielhaften Landschaft. Hier ist die Spezifikation auf Kundenwunsch bereits fest eingeplant. Optimiert wird die Landschaft nun in Bezug auf die Frage, ob im Projekt Story Cards eingesetzt werden sollen oder nicht. Eine weitere lokale Optimierung trifft die Abwägung, ob Use Cases mit Aktivitäten des Geschäftsprozessmodells explizit verknüpft werden.

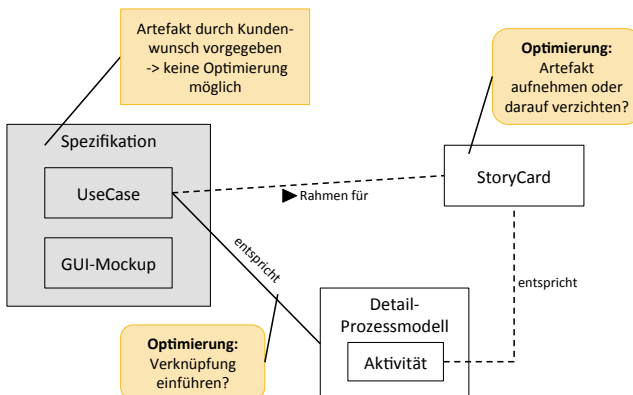


Abbildung 28: Lokale Optimierung von Anforderungslandschaften

7.1.2 Methode zur Ableitung von Optimierungsdimensionen und Risikofaktoren

Die Optimierungsdimensionen und dazugehörigen Risikofaktoren werden aus der Interview-Studie ermittelt. In den Interviews haben die Teilnehmer erklärt, auf welche Aspekte beim Umgang mit Anforderungslandschaften zu achten ist. Sie haben ihre Erfahrungen mit Anforderungsartefakten oder deren Abhängigkeiten beschrieben und dabei Vor- und Nachteile der Aspekte erwähnt. Diese Vor- und Nachteile wurden durch die Autorin nach dem im Folgenden beschriebenen Verfahren interpretiert und zu Risiken weiterentwickelt. Anhang A zeigt zu den resultierenden Risiken die Interview-Aussagen, aus denen sie abgeleitet wurden.

Die Ergebnisse sind als Entscheidungshilfe zu verstehen. Sie zeigen mögliche Risikofaktoren, die sich im Zuge der Interview-Studie als relevant herausgestellt haben. Sie sind jedoch nicht vollständig und sollten vom Requirements Methodiker auf individuelle Ergänzungen geprüft werden. Hierzu kann der Requirements Methodiker beispielsweise Risiko-Identifikations-Methoden, wie in [15] oder [19], nutzen. Wie in Abschnitt 4.6 zur *Validität der Ergebnisse* der Interview-Studie erwähnt, wurde für die Studie eine Vielzahl unterschiedlicher Rollen und Projekte herangezogen und theoretische Sättigung der Aussagen erreicht. Dies trägt dazu bei, dass die Auswahl von Risikofaktoren so repräsentativ wie möglich ist.

Vor- und Nachteile zu einem Aspekt – bzw. genauer zu einer gewählten Alternative bezüglich des Aspektes (z.B. Einführung einer Verknüpfung) – können als Risiko aufgefasst werden, wenn diese mit Schäden bzw. Kosten verbunden sind. Wie Abbildung 29 zeigt, kann ein Problem für ein Projekt entstehen, wenn eine Alternative hohe Kosten verursacht oder wenn eine Alternative einen wichtigen Nutzen nicht erbringt.

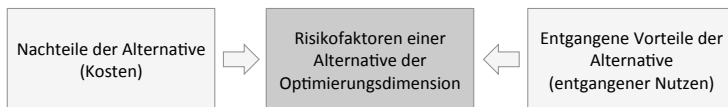


Abbildung 29: Quellen für Risikofaktoren zu einer Alternative

Ableitung von Risikofaktoren aus Nachteilen

Zum einen werden *Nachteile*, die in der Interview-Studie zu einer Alternative genannt wurden, als Risikofaktoren der Alternative aufgefasst. Hierzu wird das konkrete Problem, das der Nachteil in Bezug auf das Projekt bietet, herausgearbeitet. Auf diese Art kann der Requirements Methodiker das Risiko bzw. dessen Schaden direkter bewerten. Beispielsweise kann ein umfangreiches formales

Dokument den Nachteil haben, dass es schwerer handzuhaben ist und meist mehr Aufwand zur Erstellung benötigt. Das eigentliche Problem für das gesamte Projekt entsteht dann aber erst, wenn dies auch die Analysephase verzögert und dieses zu Unzufriedenheit der Kunden oder des Managements führt.

Ableitung von Risikofaktoren aus entgangenen Vorteilen

Oftmals wurden in den Interviews auch *Vorteile* zu einer Alternative genannt, welche die Projektteilnehmer dazu bewegten, diese Alternative zu wählen. Beispielsweise wurden GUI-Mockups zu einem Projekt hinzugefügt, um anhand der bildlichen Darstellung besser mit dem Kunden über Abläufe kommunizieren und Missverständnisse aufdecken zu können. Auch Vorteile sollten in der risikobasierten Bewertung berücksichtigt werden, da der *Verzicht auf einen Vorteil* ein Risiko darstellen kann. Die Entscheidung, GUI-Mockups nicht in die Anforderungslandschaft aufzunehmen kann dann beispielsweise dazu führen, dass die Kommunikation mit dem Kunden nicht gut läuft und mehr Missverständnisse entstehen.

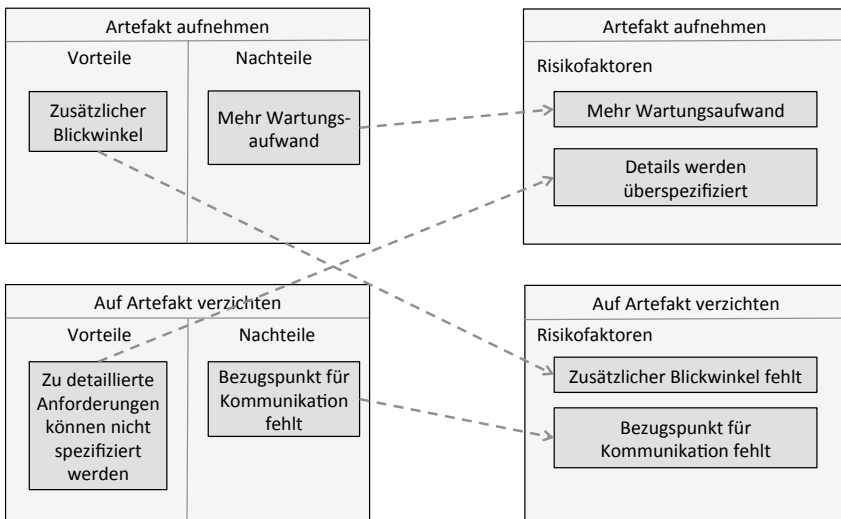


Abbildung 30: Zuordnung von Vor- und Nachteilen zu Risikofaktoren

Nun gibt es zwei Situationen, für die aus entgangenen Vorteilen Risiken abgeleitet werden können:

1. *Durch den entgangenen Vorteil entsteht potenziell ein realer Schaden.* Beispielsweise kann es durch den Verzicht auf GUI-Mockups und dem damit entgangenen Vorteil der besseren Kommunikation zu Missverständnissen bezüglich der Anforderungen und damit zu falscher Funktionalität der Software kommen. Diese müssen gegen zusätzliche Kosten korrigiert werden, was einen Schaden darstellt.

2. *Durch den entgangenen Vorteil entsteht kein realer Projektschaden.* D.h. dem Projekt entstehen keine echten Kosten, aber der entgangene Vorteil hätte eine zusätzliche Ersparnis oder einen zusätzlichen Wert bringen können. Beispielsweise kann es in einem anderen Projekt vorkommen, dass beim Verzicht auf GUI-Mockups keine Missverständnisse auftreten, weil die Stakeholder gut mit anderen abstrakten Anforderungsartefakten klarkommen. Dennoch hätte der Vorteil der verbesserten Kommunikation mittels GUI-Mockups zu zusätzlichen wertvollen Einsichten führen oder den RE-Aufwand reduzieren können. In dieser Situation entstehen durch den Verzicht auf GUI-Mockups keine realen Kosten, jedoch kann man Opportunitätskosten gegenüber der Hinzunahme von GUI-Mockups, ansetzen, die beispielsweise in einer höheren Kundenzufriedenheit oder Einsparungen von Aufwand liegen.

Auch bei den Risikofaktoren, die aus Vorteilen abgeleitet sind, ist es also wichtig, das genaue Problem herauszustellen, das durch den entgangenen Vorteil hervorgerufen wird und klarzumachen, ob es sich dabei um einen Schaden durch reale Kosten oder durch Opportunitätskosten handelt.

7.2 Optimierungsdimensionen

Bei der Zusammenstellung einer Anforderungslandschaft sind grundsätzlich drei Faktoren relevant. Einerseits lassen sich die verwendeten *Artefakttypen* selbst und ihre Beschaffenheit wählen. Zudem können die *Abhängigkeiten* zwischen den Artefakttypen beeinflusst werden. Als dritte Möglichkeit kann auch die *Verwendung* der Anforderungsartefakte durch Vorgaben an die Vorgehensweise beeinflusst werden. Um relevante **Optimierungsdimensionen** auszuwählen, werden die Aussagen aus der Interview-Studie herangezogen. Eigenschaften, die in Zusammenhang mit Vor- oder Nachteilen von Artefakttypen, Abhängigkeiten oder der Verwendung von Artefakten genannt wurden, werden hier aufgenommen. Zum Beispiel ist die *Zähheit* eines Artefaktes eine relevante Optimierungsdimension. Diese wurde unter anderem aus folgender im Interview geschilderten Situation abgeleitet: Ein Softwareentwickler versuchte, die Anpassung von zähen Artefakten zu meiden, weil dies für ihn zu umständlich war. Auch wenn er den Code selbst änderte, dokumentierte er dies nicht mehr. Im späteren Verlauf des Projektes waren solche unvollständigen Spezifikationen dann hinderlich, da die Entwickler so immer wieder Abhängigkeiten übersahen.

Im Folgenden werden die einzelnen Optimierungsdimensionen erläutert. Abbildung 31 zeigt an einem Beispiel, auf welche Elemente in einer Landschaft die Dimensionen sich beziehen.

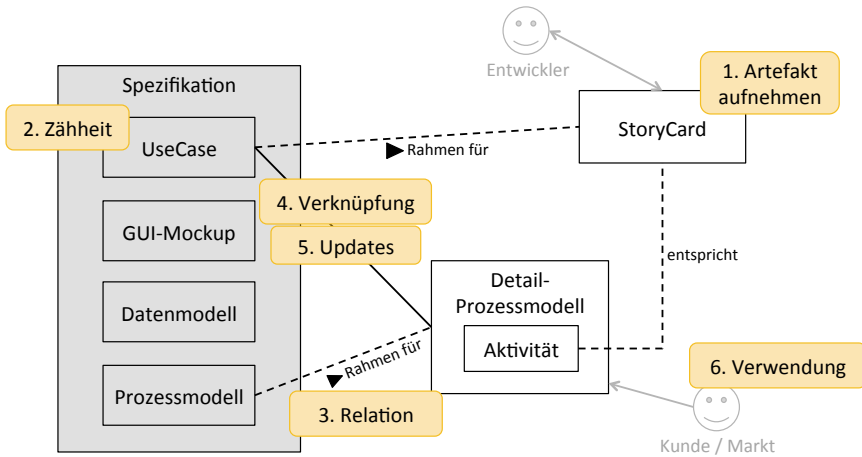


Abbildung 31: Mögliche Optimierungsfaktoren, dargestellt an einer beispielhaften Anforderungslandschaft

Artefaktbezogene Optimierungsfaktoren

- (1) **Verwendung eines bestimmten Artefakttypen:** Zunächst lässt sich grundsätzlich abwägen, ob ein Artefakttyp überhaupt im Projekt verwendet werden soll. Mit der Hinzunahme wird zum einen festgelegt, dass Informationen, die dieses Artefakt repräsentiert, überhaupt spezifiziert, also nicht offen gelassen, werden [16]. Zum anderen geht es oftmals auch darum, mit einem zusätzlichen Artefakt, wie beispielsweise einem Geschäftsprozessmodell, einen *zusätzlichen Blickwinkel* auf die Anforderungen abzubilden. Durch die verwendeten Artefakte wird vor allem beeinflusst, wie gut das Verständnis bestimmter Projektteilnehmergruppen unterstützt wird, da unterschiedliche Projektteilnehmer mit unterschiedlichen Artefakten besser umgehen können. Auf der anderen Seite kann es auch sinnvoll sein, auf Artefakte bewusst zu verzichten. Wie Glinz und Fricker schreiben [58], erzeugt die explizite Spezifikation von Anforderungen zusätzliche Kosten. Kann ein gemeinsames Verständnis auch implizit erreicht werden, werden durch den Verzicht auf den entsprechenden Artefakttyp Kosten gespart.
- (2) **Zähheit:** Zu einem Artefakttyp wird im Prozess bzw. mittels Abprache festgelegt, ob Projektteilnehmer es einfach ändern können, oder ob dazu die Zustimmung anderer notwendig ist. Damit lässt sich die prozessbezogene Zähheit durch Festlegung der Vorgehensweise beeinflussen. Dies ist hilfreich, um beispielsweise schnelle Änderungen zu unterstützen oder umgekehrt zu starke Änderungen, die kontraproduktiv sind, zu verhindern. Die strukturelle Zähheit lässt sich hingegen nicht direkt festlegen, da diese vom Artefakt selbst vorgegeben ist. Die Projektteilnehmer können allenfalls eine neue Strukturierung oder einen neuen Formalitätsgrad der Anforderungen festlegen.

Abhängigkeitsbezogene Optimierungsfaktoren:

- (3) **Abhängigkeiten bei sich entsprechenden Artefakttypen:** Abhängigkeiten zwischen Artefakttypen verursachen zusätzlichen Aufwand bei der Erstellung, Verwendung und Wartung von Anforderungen (siehe Abschnitt 6.2). Nun lässt sich die Abhängigkeit zwischen Artefakttypen jedoch prinzipiell nicht einfach beeinflussen. Die Abhängigkeiten kommen durch überlappende Inhalte zustande und sind dadurch schon durch die verwendeten Artefakttypen festgelegt. Jedoch lassen sich einige Abhängigkeiten indirekt verringern, indem die entsprechenden Artefakte zusammengefasst werden. Dies macht am meisten Sinn bei Artefakttypen, die in einer *entspricht*-Relation stehen. Diese beschreiben oft ähnliche Informationen aus verschiedenen Blickwinkeln.
- (4) **Verknüpfungen:** Hier wird festgelegt, ob die Artefakte zweier Artefakttypen über explizite Verknüpfungen verknüpft werden sollen. Verknüpfungen ermöglichen beispielsweise die Nachverfolgbarkeit der Entwicklung sowie ein leichteres Nachpflegen von Änderungen. Die Erstellung und Wartung von Verknüpfungen verursacht jedoch Kosten. Zu viele Verknüpfungen können außerdem dazu führen, dass die Präzision von verknüpften Elementen sinkt. Das bedeutet, dass beim Betrachten aller verknüpften Elemente zu einem Artefakt nicht alle Elemente relevant sind. Es sollten daher nur Verknüpfungen eingeplant werden, wenn sie tatsächlich benötigt werden [41], [109].
- (5) **Updates weiterer Artefakte bei Änderungen:** In vielen Fällen werden Änderungen, die in ein Anforderungsartefakt eingepflegt werden, auch an alle weiteren betroffenen Stellen, wie Designartefakte, Tests oder eben weitere Anforderungsartefakte, propagiert. Dies führt zu hohem Wartungsaufwand, der nicht immer gewinnbringend ist. Nur wenn ein Anforderungsartefakt noch häufig im Projekt verwendet wird, lohnt es sich, das Artefakt aktuell zu halten. Wird es hingegen selten verwendet, so kann sich eine Strategie lohnen, bei der ein Artefakt grundsätzlich nicht mehr bei jeder Änderung eines verknüpften Artefaktes aktuell gehalten wird. Stattdessen werden die Inhalte des Artefaktes erst bei einer Verwendung des Artefaktes wieder mit den aktuellsten Inhalten überschrieben.

Verwendungsbezogene Optimierungsfaktoren:

- (6) **Nutzung festgelegter Anforderungsartefakte:** Über Festlegungen im Prozess oder Absprachen mit den Projektteilnehmern wird festgelegt, dass beispielsweise neue Anforderungen vorzugsweise in ein bestimmtes Artefakt aufgenommen werden sollen. Dies kann sinnvoll sein, um Indirektionen zu vermeiden. Wenn beispielsweise die Entwickler ihre Arbeit hauptsächlich mittels User Stories planen, sollte auch die Kundenseite neue An-

forderungen in Form von User Stories einpflegen. So wird Übersetzungsaufwand und damit mögliche Missverständnisse vermieden.

Dieser Faktor ist ein wichtiger Optimierungsfaktor und sollte bei der Analyse von Anforderungslandschaften berücksichtigt werden. Jedoch lässt sich hier keine risikobasierte Entscheidungshilfe aufstellen, sodass dieser Aspekt in den folgenden Kapiteln nicht weiter genannt wird.

7.3 Umgang mit mehreren Optimierungsdimensionen

Einzelne Optimierungsschritte können unabhängig voneinander durchgeführt werden. Die angegebene Reihenfolge der Optimierungsdimensionen ist jedoch dann wichtig, wenn zu einem Element, wie einem Artefakttyp, mehrere Dimensionen betrachtet werden sollen. Die Optimierungsdimensionen sind insofern voneinander abhängig, als dass die Entscheidung bezüglich einer früheren Dimension – welche weiter oben in oben aufgeführter Liste steht – die Landschaft so verändert, dass die späteren Dimensionen möglicherweise neu evaluiert werden müssen. Zum Beispiel macht es keinen Sinn, zuerst abzuwägen, ob eine Verknüpfung zwischen zwei Artefakttypen definiert werden soll und danach erst zu entscheiden, ob möglicherweise einer der Artefakttypen ganz aus der Landschaft herausgenommen werden soll. Abbildung 32 zeigt die vorgesehene Reihenfolge. Die Abbildung zeigt noch eine weitere Optimierungsdimension, die den anderen vorgelagert ist: In Optimierungsdimension 0 wird basierend auf dem Vorgehen von Boehm und Turner zunächst die grundsätzliche Vorgehensweise, also eine agile oder plangetriebene Vorgehensweise, ausgewählt.

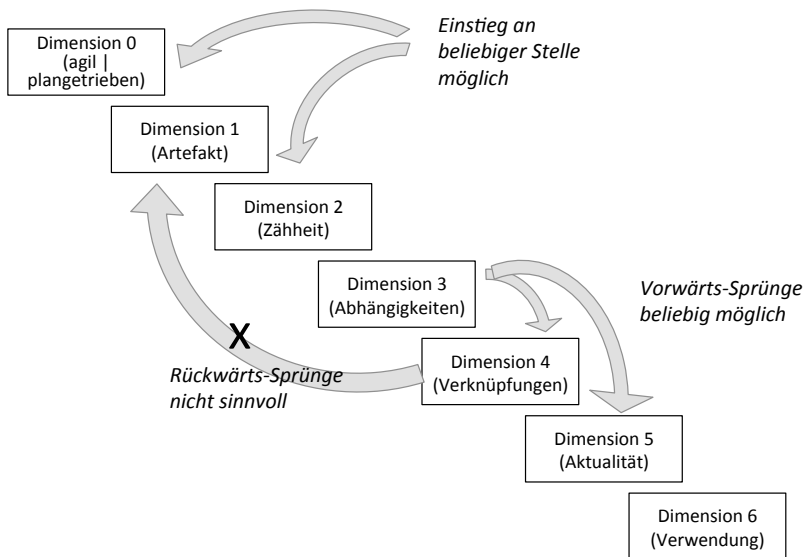


Abbildung 32: Optimierungsdimensionen und ihre Bearbeitungsreihenfolge

Berücksichtigung nachgelagerter Optimierungsdimensionen. Wie bereits erwähnt, sind die Optimierungsdimensionen jeweils abhängig von vorgelagerten Optimierungsdimensionen. Doch auch Entscheidungen aus nachgelagerten Optimierungsdimensionen können eine Optimierung beeinflussen. So kann es sein, dass es zunächst risikogünstiger erscheint, auf ein Artefakt ganz zu verzichten, wenn es nicht mit anderen Artefakten verknüpft wird. Jedoch können die Kosten *unter der Bedingung* dass man das Artefakt verknüpft, weiter sinken, sodass es insgesamt am günstigsten ist, das Artefakt doch hinzuzufügen und zu verknüpfen. Abbildung 33 veranschaulicht die beiden Richtungen der Abhängigkeiten an einem Beispiel.

Um diese gegenläufigen Abhängigkeiten zu nachgelagerten Optimierungsdimensionen zu berücksichtigen, werden diese in die Risikofaktoren aufgenommen. Der Requirements Methodiker kann so Alternativen abwerten, wenn er schätzt, dass in nachgelagerten Optimierungsdimensionen Entscheidungen getroffen werden, die für die Alternative ungünstig sind. Beispielsweise kann so bei der Einführung eines Artefakttyps direkt berücksichtigt werden, wenn im Projekt prinzipiell kein Tracing und somit keine Verknüpfungen eingesetzt werden.

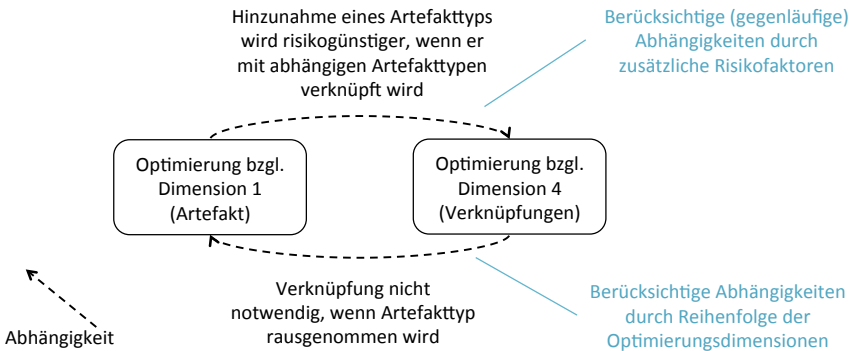


Abbildung 33: Umgang mit Abhängigkeiten zwischen Optimierungsfaktoren

7.4 Systematischer Aufbau einer Anforderungslandschaft

Neben der lokalen Optimierung bestehender Landschaften, kann die vorgestellte Methode auch dazu verwendet werden, eine Anforderungslandschaft von Beginn an komplett zu aufzubauen. Auch zum Aufbau einer Anforderungslandschaft empfiehlt es sich, die Dimensionen in der vorgegebenen Reihenfolge abzuarbeiten und die Risikoabschätzung als Entscheidungshilfe zu verwenden. Auch hier gilt, dass nicht in allen Optimierungsdimensionen komplette Ent-

scheidungsfreiheit bestehen muss. Dies bedeutet, dass Aspekte der Landschaft auch unabhängig von der Optimierung festgelegt werden können.

Eine komplette Optimierung jedes Elementes, also jedes Artefakttyps mitsamt seiner Eigenschaften und jeder möglichen Beziehung zwischen Artefakttyppaaren, wäre sehr umfangreich. Für jedes Artefakt und für jede potenzielle Beziehung zwischen Artefakten müssten mehrere Optimierungsschritte für die jeweiligen Optimierungsdimensionen durchgeführt werden. Da potenziell jeder Artefakttyp mit jedem anderen Artefakttyp in einer Abhängigkeitsrelation stehen könnte, steigt die Zahl der möglichen Abhängigkeitsrelationen mit der Zahl der Artefakttypen quadratisch an¹⁰. Für das gewünschte Ziel, eine Anforderungslandschaft zu planen, mit der die Projektteilnehmer ausreichend gut umgehen können, ist es nicht praktikabel, eine so umfangreiche Analyse durchzuführen. Daher werden zuerst Aspekte mit größerem Optimierungspotenzial ausgewählt.

Dazu ermittelt der Requirements Methodiker über Analyseschritte zwischen den Optimierungsschritten, welche Aspekte am ehesten optimiert werden sollten. Dazu kann die Landschaft auf problematische Muster hin untersucht werden. Alternativ kann der Requirements Methodiker problematische Aspekte der Anforderungslandschaft in Workshops mit den Projektteilnehmern identifizieren. Abbildung 34 zeigt, wie Analyse und Optimierung aufeinander abgestimmt werden, um eine Anforderungslandschaft systematisch zu erstellen.

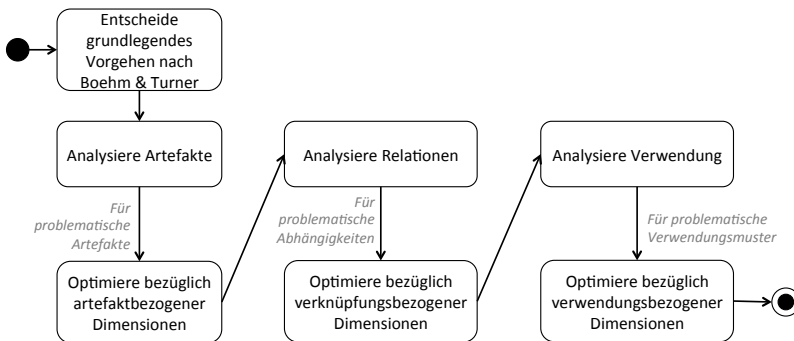


Abbildung 34: Vorgehen zur systematischen Erstellung von Anforderungslandschaften

Zunächst wird das grundlegende Vorgehen festgelegt. Hierzu wird nach dem Verfahren von Boehm und Turner [16] grundsätzlich entschieden, ob die Vorgehensweise primär an agilen oder plangetriebenen Methoden ausgerichtet wird.

¹⁰ Da jeder Artefakttyp mit jedem anderen potenziell in Relation stehen kann, müsste man jedes Artefakttyppaar prüfen. Die Zahl der nötigen Überprüfungen würde der Kantenzahl in einem vollständigen Graph entsprechen. Ein vollständiger Graph mit n Knoten besitzt nun $n*(n-1)/2$ Kanten, welche in etwa quadratisch mit n wachsen.

Je nach Ergebnis werden agile oder plangetriebene Artefakttypen als Haupt-Artefakttypen in die Anforderungslandschaft eingeplant. Nun können weitere Artefakttypen hinzugenommen werden, die weitere Funktionen im Projekt ausüben oder bei speziellen Aktivitäten helfen. Insbesondere können nun zu agilen Artefakten zusätzliche plangetriebene Artefakttypen hinzugenommen werden und umgekehrt, um beide Ansätze stärker zu vereinen. Als Orientierung können hier die Beispiel-Konstellationen aus Abschnitt 6.7 (Beispiele für hybride Landschaften) herangezogen werden.

Für problematische Artefakte wird nun jeweils eine Optimierung in Hinblick auf die artefaktbezogenen Optimierungsdimensionen durchgeführt, um Artefakte möglicherweise zu entfernen oder ihre grundsätzliche Zähheit anzupassen. Sind nun die Artefakte festgelegt, so werden als nächstes die Abhängigkeiten zwischen diesen dokumentiert und im Anschluss optimiert. Auch hier wird zunächst nur eine Auswahl der problematischsten Abhängigkeiten, beispielsweise auf Grundlage von in Workshops genannten Problemen, getroffen. Zum Schluss wird dokumentiert, welche Rollen welche Artefakte verwenden. Wenn sich hier Probleme zeigen, so können diese über Anpassung der Verwendung von Artefakten verbessert werden.

7.5 Vorgehen zur Bewertung der Risikofaktoren

Um nun bezüglich der einzelnen Optimierungsdimensionen zwischen den konkreten Alternativen abzuwägen, werden deren Vor- und Nachteile verglichen. In unterschiedlichen Projektkontexten wirken sich bestimmte Vor- und Nachteile unterschiedlich stark aus, sodass die Faktoren in jedem Projekt individuell bewertet werden müssen. Beispielsweise kann die Fokussierung auf User Stories den Nachteil haben, dass Detail-Informationen nur implizit vorhanden sind und nicht nachgeschlagen werden können. Dieser Nachteil kommt jedoch vor allem in Projekten zum Tragen, in denen die Projektteilnehmer auf Kundenseite wenig Zeit für Besprechungen haben. In einem Projekt, in dem ein Kundenvertreter wie der *OnSite-Customer* in Extreme Programming-Projekten [8] über die gesamte Zeit zur Verfügung steht, ergeben sich durch den Einsatz von User Stories diese Nachteile nicht.

Nun reicht die bloße Identifikation von Vor- oder Nachteilen der Alternativen jedoch nicht aus, um systematisch entscheiden zu können, welche Alternative die bessere ist. Die Vor- und Nachteile sind per se nicht direkt vergleichbar. Häufig haben alle Alternativen sowohl mehrere Vor- als auch Nachteile. Es wird daher eine Möglichkeit benötigt, zu ermitteln, welcher von zwei Vorteilen oder Nachteilen *schwerwiegender* ist. Genauso sollte es möglich sein, einen Vorteil gegen einen Nachteil abzuwiegen.

Um Vergleichbarkeit von Aspekten zu ermöglichen, müssen diese mit messbaren Größen versehen werden [150]. Damit werden die Aspekte bewertet und anschließend deren Werte verglichen. Im hier vorliegenden Fall sollen *Vorgehensweisen* bewertet und verglichen werden. Dazu eignen sich Größen, wie der Wert, die Kosten, das Risiko oder Kombinationen, wie Return on Investment, der Vorgehensweisen [12]. In dieser Arbeit wird die Bewertung durch Risiken gewählt. Dies hat den Vorteil, dass sowohl der Schaden, als auch die Eintrittswahrscheinlichkeit in die Bewertung einbezogen werden können.

Aus den genannten Vor- und Nachteilen werden Risikofaktoren abgeleitet, die den möglichen Alternativen zugeordnet werden. Durch eine Bewertung der Risikofaktoren kann der Requirements Methodiker sich schließlich ein Bild verschaffen, ob eine der Alternativen risikogünstiger ist und dies als Entscheidungshilfe bei der Optimierung nutzen. Abbildung 35 zeigt hierzu ein Beispiel als Überblick – die detaillierte Bewertung wird in den folgenden Abschnitten erklärt. Im Beispiel wird der zweite Risikofaktor in der Tabelle mit einer 10%igen Eintrittswahrscheinlichkeit und einem Schaden von 50 eingeschätzt. Das Gesamtrisiko der Alternative, den Artefakttyp aufzunehmen wird mit einer gesamten Risk Exposure von 8 bewertet. Das Gesamtrisiko der Alternative, den Artefakttyp nicht aufzunehmen beträgt 26, es ist also deutlich höher. Das Artefakt sollte daher in die Landschaft aufgenommen werden.

Optimierungsdimension Artefakt			
Konkretes bewertetes Artefakt: User Stories			
Risikofaktor	Eintrittswahrscheinlichkeit (0-1)	Schaden (0-100)	Risk Exposure
Artefakttyp wird aufgenommen			Summe: 8
Analysephase verlängert sich	0,1	30	3
Zu detaillierte Spezifikation legt Projekt auf ungünstige Lösungen fest	0,1	50	5
Artefakttyp wird nicht aufgenommen			Summe: 26
Projekt läuft aus dem Rahmen	0,6	30	18
Viele Stakeholder -> Über-Anpassung wegen fehlendem Bezugspunkt	0,2	40	8

Bewertung eines einzelnen Risikofaktors

Bewertung des Gesamtrisikos einer Alternative

Abbildung 35: Entscheidungshilfe zur Nutzung von User Stories durch Bewertungen der mit beiden Alternativen verbundenen Risikofaktoren

7.5.1 Bewertung der Risikofaktoren mittels Risk Exposure

Ein *Risiko* ist laut IEEE Standard 1540:2001 [74] ein potenzielles Problem, bestehend aus der Wahrscheinlichkeit des Eintritts eines Ereignisses oder einer Gefahr zusammen mit dessen unerwünschten Konsequenzen. Laut Boehm kön-

nen sich die unerwünschten Konsequenzen auf verschiedene Projektteilnehmer, wie die Kunden, Nutzer, Wartungsverantwortlichen oder Entwickler einer Software, beziehen [15]. Die unerwünschten Konsequenzen stellen einen Schaden für das Projekt dar. Laut Boehm kann sich der Schaden beispielsweise auf finanzielle Gewinne, die Reputation, Lebensqualität oder weitere wertverwandte Attribute beziehen [11].

Im IEEE Standard 1540:2001 ([74], Abschnitt 5.1.3.2., S. 10) ist zudem aufgeführt, dass Wahrscheinlichkeiten und Schäden sowohl quantitativ als auch qualitativ bewertet werden können. Eine quantitative Bewertung kann bei der Wahrscheinlichkeit durch Angabe eines Wahrscheinlichkeitswertes zwischen 0 und 1 vorgenommen werden. Der entstehende Schaden kann durch die Angabe eines Geldwertes quantitativ charakterisiert werden. Eine qualitative Bewertung von Wahrscheinlichkeit und Schaden kann beispielsweise durch die Einordnung in *geringe*, *mittlere* oder *hohe* Eintrittswahrscheinlichkeit bzw. Schaden erfolgen.

Eine wichtige Kenngröße zur Einschätzung von Risiken ist die **Risk Exposure**. Sie stellt laut IEEE Standard 1540:2001 [74] den potenziellen Verlust dar, den ein Risiko gegenüber einer Person, einem Projekt oder einer Organisation birgt. Die Risk Exposure erlaubt es nun, verschiedene Risiken miteinander zu vergleichen. Auch hier kann laut IEEE Standard die Risk Exposure quantitativ oder qualitativ ausgedrückt werden ([74], Abschnitt 3.8).

Die häufigste Variante zur Ermittlung der Risk Exposure ist, wie auch z.B. von Boehm verwendet, durch Multiplikation der Eintrittswahrscheinlichkeit eines potenziellen Problems mit dessen Schadensgröße [17], [74]. Sind Wahrscheinlichkeit und Schaden quantitativ bewertet, so entspricht die Risk Exposure dem Erwartungswert (im stochastischen Sinne) des Schadens zum potenziellen Problem.

Eintrittswahrscheinlichkeit und Schaden sind Schätzwerte, die ein Risikomanager, ein Projektleiter oder – wie im Falle dieser Arbeit – ein Requirements Methodiker ermittelt. Je detaillierter die Skalen zur Bewertung von Wahrscheinlichkeit und Schaden nun sind, desto genauer lassen sich Risiken einschätzen und vergleichen. Jedoch wird die Schätzung zeitgleich mit zunehmender Genauigkeit auch aufwändiger und fehleranfälliger ([15], S. 6).

Das Ziel der Risikoschätzung liegt meist darin, die wichtigsten Risiken zu priorisieren [19], [74] oder mehrere Alternativen abzuwägen [16]. Dieses kann oft bereits mit groben Skalen erreicht werden. Beispielsweise stellt Boehm in [15] heraus, dass zur Priorisierung von Risiken jeweils eine relative Skala mit ganzzahligen Werten zwischen 1 und 10 ausreicht. Häufig werden auch relative Skalen mit nur drei qualitativen Werten für niedrig, mittel und hoch verwendet. In dieser Arbeit wird die Risikoanalyse als Entscheidungshilfe herangezogen, um beim Aufbau einer Anforderungslandschaft zwischen mehreren Alternativen

zu wählen. Detaillierte Einschätzungen der Risk Exposure sind hierfür nicht entscheidend. Vielmehr ist wichtig, zu erkennen, wenn eine Alternative deutlich risikogünstiger als eine andere ist – oder auch wenn beide Alternativen in etwa gleichviel Risikopotenzial besitzen. Für die Bewertung eines einzelnen Risikofaktors wird daher eine möglichst simple Vorgehensweise zur Einschätzung der Risk Exposure gesucht.

Die simpelste Variante wäre eine qualitative Bewertung der gesamten Risk Exposure eines Risikos. Hierzu würden ähnlich wie in Boehms Methode, die beispielsweise in Abbildung 26 dargestellt wurde, Risiken als *minimales, moderates, inakzeptables Risiko*, etc. eingestuft. Jedoch ist bei den für unsere Methode zugrundeliegenden Risiken damit zu rechnen, dass diese Einschätzung sehr ungenau wird und vor allem, dass die meisten Risiken nur als minimale oder moderate Risiken eingeschätzt werden. Für eine höhere Genauigkeit wird daher die Risk Exposure präziser als Produkt von Eintrittswahrscheinlichkeit und Schaden berechnet.

Eine Voraussetzung dafür, dass Risk Exposure-Werte miteinander verglichen werden können, ist, dass die beiden Faktoren Wahrscheinlichkeit und Schaden auf einer Verhältnisskala (auch Rationalskala genannt) vorliegen. Für eine Verhältnisskala gilt: (1) Die Werte besitzen eine Ordnung, (2) die Intervalle zwischen zwei Werten lassen sich exakt bestimmen und vergleichen und (3) die Skala besitzt einen absoluten Nullpunkt. Nur unter diesen Voraussetzungen kann eine sinnvolle Multiplikation erfolgen.

Abbildung 36 veranschaulicht an einem Beispiel, welche Probleme auftreten, wenn sich die Werte nicht korrekt ins Verhältnis setzen lassen. Hier stellen die Werte für s eine Einordnung in eine von drei Schadensklassen dar (*Klasse 1*: 0-20k € Schaden; *Klasse 2*: 20k-30k €; *Klasse 3*: 30k-50k €). Die Position auf der Achse s stellt dabei – in grau markiert – den tatsächlichen Schaden dar. Risiko A besitzt beispielsweise einen Schaden von 20000 € und wird in die Schadensklasse 1 einsortiert. Risiko B besitzt einen Schaden von 30000 € und wird in die Schadensklasse 2 einsortiert. Berechnet man nun einfach die Risk Exposure als Produkt aus Wahrscheinlichkeit und Schaden (im Sinne der Schadensklasse), so erhalten beide Risiken dieselbe Risk Exposure, nämlich 0,8. Die tatsächlich korrekten Werte weichen jedoch voneinander ab. Hier besitzt Risiko A eine korrekte Risk Exposure von 16000, Risiko B aber nur eine Risk Exposure von 12000. Das Problem entsteht, weil die Werte der Schadensklassen nicht äquidistant sind.

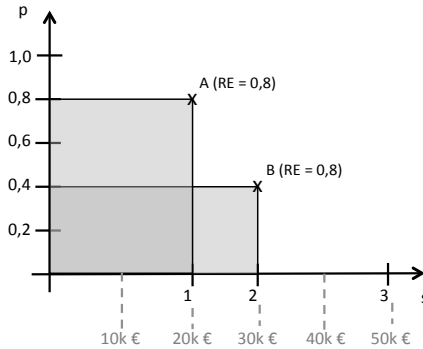


Abbildung 36: Fehler bei der Multiplikation auf einer Skala mit nicht äquidistanten Werten

In der hier vorgestellten Methode wird die Eintrittswahrscheinlichkeit, wie oft üblich, auf einer relativen Skala von 0 bis 1 eingeschätzt. Für den Schaden wird eine Skala von 0 bis 100 verwendet. Bei der Festlegung der Bedeutung der Werte muss der Requirements Methodiker beachten, dass es sich um eine Verhältnisskala handeln muss. Das bedeutet, dass ein Schaden von 20 dem doppelten eines Schadens von 10 entsprechen sollte. Hierzu kann für den Wert 100 ein maximaler Schaden festgelegt und die anderen Werte als Anteile davon gerechnet werden. Wir setzen den Wert 100 auf den Wert des initialen Projektbudgets. So entspricht ein Schaden von 20 dann einer Budgetüberschreitung um 20%. Eine Voraussetzung hierfür ist, dass es keine Risiken gibt, die das initiale Projektbudget übersteigen (sonst muss der maximale Schaden höher angesetzt werden). Bei den vorliegenden Risiken ist jedoch eher damit zu rechnen, dass viele kleine Schäden auftreten, sodass es auch sinnvoll sein kann, den maximalen Schaden auf einen geringeren Wert zu setzen, um die Skala besser auszuschöpfen.

Die Risk Exposure wird dann durch Multiplikation beider Werte ermittelt. Abbildung 37 zeigt hierzu ein Beispiel.

Definition Risk Exposure eines Risikofaktors r :

$$RE(r) = P(r) \cdot S(r), \text{ mit}$$

$P(r) \in [0,1]$: Eintrittswahrscheinlichkeit des Risikofaktors r und
 $S(r) \in [0,100]$: Schadensgröße des Risikofaktors r .

Bewertung eines Risikofaktors mittels Risk Exposure

Risikofaktor	Eintrittswahrscheinlichkeit (0-1)	Schaden (0-100)	Risk Exposure
Artefakttyp wird aufgenommen			Summe: 8
Analysephase verlängert sich	0,1	30	3
Zu detaillierte Spezifikation legt Projekt auf ungünstige Lösungen fest	0,1	50	5
Artefakttyp wird nicht aufgenommen			Summe: 26
Projekt läuft aus dem Rahmen	0,6	30	18
Viele Stakeholder -> Über-Anpassung wegen fehlendem Bezugspunkt	0,2	40	8

Abbildung 37: Bewertung eines einzelnen Risikofaktors

In einer traditionellen Risikoanalyse würde ein Risiko nicht mit einer Wahrscheinlichkeit oder einem Schaden von 0 bewertet werden können. Ein Problem, das sicher nicht eintreten kann oder keinen Schaden erzeugt, ist per Definition kein potenzielles Problem, also kein Risiko. Die Risikolisten zu einer Optimierungsdimension enthalten jedoch zunächst nur Kandidaten für Risiken, sodass der Requirements Methodiker hier auch eine Möglichkeit benötigt, zu kennzeichnen, wenn ein Risikofaktor im individuellen Projekt überhaupt kein Risiko darstellt.

7.5.2 Vergleich von Alternativen mittels Risikobewertung

Einzelne Risikofaktoren können nun mittels ihrer Risk Exposure-Werte verglichen werden. Eine *Alternative*, die in der Optimierung von Anforderungslandschaften betrachtet wird, setzt sich nun aus mehreren Risiken zusammen. Es stellt sich die Frage, wie nun solche Alternativen bewertet bzw. verglichen werden können.

Existieren mehrere Risiken zu einer Alternative, so können alle diese Risiken eintreten und Schäden verursachen. Zu einer Alternative mit mehreren Risiken müssen wir das zusammengesetzte Ereignis aus der Kombination der einzelnen Risiko-Ereignisse betrachten. Dies wird hier am Beispiel von zwei Risiken, die zu einer Alternative gehören, veranschaulicht. Die Berechnungen lassen sich später auf mehr als zwei Risiken verallgemeinern.

Zu einem Risiko gehören, wie im IEEE Standard 1540:2001 [74] angegeben, eine Wahrscheinlichkeit zum Eintritt eines Ereignisses und der zum Ereignis gehörende Schaden (siehe Abschnitt 7.5.1). In diesem Abschnitt fassen wir r_1

und r_2 als die Ereignisse auf, die zu zwei gleichnamigen Risiken r_1 und r_2 gehören¹¹.

Tabelle 8 zeigt, die möglichen Ereignisse für eine Alternative, welche die Risiken r_1 und r_2 besitzt sowie deren Wahrscheinlichkeiten und Schäden.

Tabelle 8: Mögliche Kombinationen von Risiken zu einer Alternative

Ereignis E	Wahrscheinlichkeit P(E)	Schaden S(E)
$r_1 \wedge r_2$	$P(r_1 \wedge r_2)$	$S(r_1 \wedge r_2)$
$r_1 \wedge \bar{r}_2$	$P(r_1 \wedge \bar{r}_2)$	$S(r_1)$
$\bar{r}_1 \wedge r_2$	$P(\bar{r}_1 \wedge r_2)$	$S(r_2)$
$\bar{r}_1 \wedge \bar{r}_2$	$P(\bar{r}_1 \wedge \bar{r}_2)$	0

Um den potenziellen Gesamtschaden einer Alternative zu bewerten, muss betrachtet werden, wie sich die Risiken bzw. die Schäden verhalten, wenn sie gemeinsam auftreten. Nachdem im vorigen Abschnitt Schätzwerte für $P(r_1)$, $P(r_2)$ und $S(r_1)$, $S(r_2)$ ermittelt wurden, muss nun noch ermittelt werden, was $P(r_1 \wedge r_2)$ und $S(r_1 \wedge r_2)$ ist.

Zur einfacheren Berechnung der Risk Exposure einer Alternative werden zwei Annahmen getroffen. Diese werden im Folgenden erläutert.

Annahmen an das Verhalten von Risiko-Ereignissen und deren Schäden

Annahme 1: Die Ereignisse der Risiken sind voneinander unabhängig.

Annahme 2: Die Schäden der Risiken sind voneinander unabhängig.

Beide Annahmen stellen eine Vereinfachung dar. Durch die Vereinfachungen können Fehler in der errechneten Risk Exposure einer Alternative entstehen. Entsprechend sollte, wenn der Risk Exposure-Wert einer Alternative nur geringfügig kleiner ist als der Wert einer anderen Alternative, nicht darauf geschlossen werden, dass die Alternative in jedem Fall überlegen ist.

Requirements Methodiker sollten zudem stets prüfen, ob ihre Risikolisten die Annahmen erfüllen und zwei Risiken mit abhängigen Ereignissen oder Schäden beispielsweise zu einem Risiko zusammenfassen. Am Ende des Abschnittes ist außerdem eine Erläuterung dazu angegeben, wie mit überlappenden Schäden umzugehen wäre, wenn diese doch auftreten.

¹¹ Die Bezeichnung des Risikos mit dem Ereignisbezeichner r_1 ist eine Abkürzung. Eigentlich ist das Risiko zum Ereignis r_1 ein Tupel $(P(r_1), S(r_1))$. Zur besseren Lesbarkeit wird jedoch auch das Risiko nur mit r_1 bezeichnet. Dies folgt auch dem Sprachgebrauch, in dem ein Risiko häufig auch nur mit seinem Ereignis bezeichnet wird (z.B. „Analysephase verlängert sich“).

Annahme 1: Die Ereignisse der Risiken sind voneinander unabhängig.

Wenn die beiden Ereignisse r_1 und r_2 unabhängig sind, so bedeutet dies, dass die Eintrittswahrscheinlichkeit für r_1 bzw. r_2 nicht vom Eintreten des jeweils anderen Ereignisses abhängt. Für zwei unabhängige Ereignisse r_1 und r_2 gilt stets $P(r_1 \wedge r_2) = P(r_1) \cdot P(r_2)$.

Die beiden Ereignisse sind hingegen voneinander abhängig, wenn das Eintreten eines Risikos r_1 die Wahrscheinlichkeit dafür, dass nun auch r_2 eintritt, verändert oder umgekehrt.

Beispiel: Unabhängige Ereignisse: Die beiden Ereignisse r_1 =*„Ein Mitarbeiter wird krank“* und r_2 =*„Die Computerfestplatte wird beschädigt“* sind voneinander unabhängig.

Abhängige Ereignisse: Die Ereignisse r_1 =*„Es befindet sich ein kritischer Fehler in der textuellen Spezifikation“* und r_2 =*„Es befindet sich ein kritischer Fehler im BPMN-Geschäftsprozessmodell“* sind abhängig, wenn beide Dokumente aufeinander aufbauen. Wenn sich ein kritischer Fehler in der textuellen Spezifikation befindet, so ist die Wahrscheinlichkeit dafür, dass im BPMN-Modell ebenfalls ein kritischer Fehler ist, höher, als wenn es in der textuellen Spezifikation keinen kritischen Fehler gäbe. Das liegt daran, dass die Möglichkeit besteht, dass derselbe falsche Sachverhalt in beiden Dokumenten gleich dokumentiert wurde.

Für den Ansatz wird vereinfachend angenommen, dass die Ereignisse der Risiken unabhängig voneinander sind. Dies ist dadurch zu erreichen, dass der Requirements Methodiker bei der Erstellung von Risikofaktoren darauf achtet, dass diese möglichst unabhängig voneinander sind. Im Fall offensichtlich abhängiger Ereignisse sollte der Requirements Methodiker diese zu einem Ereignis zusammenfassen, um Rechenfehler durch die Abhängigkeiten zu vermeiden. Im obigen Beispiel würde man die Risiken r_1 und r_2 zu einem Risiko r = *„Es befindet sich ein kritischer Fehler in der Anforderungsdokumentation“* zusammenfassen und dann Wahrscheinlichkeit und Schaden dieses Risikos r schätzen.

Folge der Annahme 1: Unter der Annahme, dass die Ereignisse der Risiken voneinander unabhängig sind, gilt:

$$P(r_1 \wedge r_2) = P(r_1) \cdot P(r_2)$$

Annahme 2: Die Schäden der Risiken sind voneinander unabhängig.

Die Schäden der Risiken r_1 und r_2 sind voneinander *unabhängig*, wenn der Gesamtschaden bei Eintreten beider Ereignisse auch der Summe der Einzelschäden entspricht, also $S(r_1 \wedge r_2) = S(r_1) + S(r_2)$ gilt.

Beispiel: Zur Anschaulichkeit entspreche hier ein Schaden von 1 einer Zeitüberschreitung des Projektes um einen Tag.

Sei nun $r_1 =$ „Ein Mitarbeiter wird krank“ mit einem Schaden von 10 und $r_2 =$ „Es befindet sich ein kritischer Fehler in der Spezifikation“ mit einem Schaden von 20. Wenn nun beide Risiken eintreten, so ergibt sich ein Schaden von $S(r_1 \wedge r_2) = 30$. Das Projekt verspätet sich um insgesamt 30 Tage, weil es einmal zu einer Verzögerung von 10 Tagen und dann zu einer weiteren Verzögerung von 20 Tagen kommt.

Der gegenteilige Fall, dass Schäden voneinander abhängig sind, tritt auf, wenn sich Schäden *verstärken* oder *überlappen*.

Eine **Verstärkung** von Schäden liegt vor wenn der Gesamtschaden größer wird, wenn mehrere Risiken zusammen auftreten. Es gilt $S(r_1 \wedge r_2) \geq S(r_1) + S(r_2)$.

Beispiel: Eine Verstärkung von Schäden ist beispielsweise bei der Kundenzufriedenheit denkbar. Treten mehrere Schäden in Form von „Punktabzug“ bei der Kundenzufriedenheit auf, so kann es sein, dass der Punktabzug für einen Aspekt höher ist, wenn dem Kunden vorher schon andere negative Aspekte aufgefallen sind.

Sei beispielsweise $r_1 =$ „Die Analysephase verlängert sich“ mit einem Schaden von 30 und $r_2 =$ „Ein Mitarbeiter wird krank“ mit einem Schaden von 10. Nun ist denkbar, dass sich beim Kunden aber eine Unzufriedenheit von $S(r_1 \wedge r_2) = 50$ einstellt, weil „ständig etwas schief läuft“.

Für die hier vorliegende Methode wird davon ausgegangen, dass Schäden in Form von Zeit- und Budget-Überschreitungen auftreten und es zu keiner Verstärkung der Schäden kommt.

Wenn sich umgekehrt Schäden *überlappen*, so fällt der Gesamtschaden geringer aus, als die Summe der Einzelschäden. Es gilt $S(r_1 \wedge r_2) \leq S(r_1) + S(r_2)$.

Beispiel:

Sei ein Schaden von 1 wieder gleich einer Zeitüberschreitung des Projektes um einen Tag.

Sei $r_1 =$ „Häufige externe Änderungen von Anforderungen“ mit einem Schaden von 30 und $r_2 =$ „Änderungen der Anforderungen durch hohe Zahl an internen Stakeholdern mit unterschiedlichen Interessen“ mit einem Schaden von 20.

Nun ist es möglich, dass sich eine konkrete Anforderung zuerst aufgrund des Eintretens von r_1 ändert und sich danach – noch vor ihrer Implementierung – nochmals ändert aufgrund des Eintretens von r_2 . Da die Anforderung dann aber nur einmal implementiert werden muss, ist der Schaden geringer als die Summe der Einzelschäden. Für die Summe aller Anforderungen kann dann z.B. $S(r_1 \wedge r_2) = 40$ gelten.

In dieser Arbeit wird angenommen, dass sich Risiken *nicht überlappen*. Bei den der Methode vorliegenden Risiken kann diese vereinfachende Annahme getrof-

fen werden, da die Schäden in der Regel unterschiedliche Aspekte abdecken. Eine Erläuterung am Ende dieses Abschnittes zeigt, wie vorzugehen wäre, wenn die Annahme nicht zutrifft und sich die Schäden doch überlappen.

Folge der Annahme 2: Unter der Annahme, dass die Schäden der Risiken voneinander unabhängig sind, gilt:

$$S(r_1 \wedge r_2) = S(r_1) + S(r_2)$$

Risk Exposure der gesamten Alternative

Die Risk Exposure der gesamten Alternative soll – wie auch schon bei den Einzelrisiken – den Erwartungswert des Gesamtschadens darstellen. Hierfür sind die möglichen Ausprägungen des aus r_1 und r_2 zusammengesetzten Ereignisses zu betrachten. Wie bereits oben angegeben, sind diese:

Ereignis E	Wahrscheinlichkeit P(E)	Schaden S(E)
$r_1 \wedge r_2$	$P(r_1 \wedge r_2)$	$S(r_1 \wedge r_2)$
$r_1 \wedge \bar{r}_2$	$P(r_1 \wedge \bar{r}_2)$	$S(r_1)$
$\bar{r}_1 \wedge r_2$	$P(\bar{r}_1 \wedge r_2)$	$S(r_2)$
$\bar{r}_1 \wedge \bar{r}_2$	$P(\bar{r}_1 \wedge \bar{r}_2)$	0

Als Erwartungswert summiert die Risk Exposure die möglichen Ausprägungen. Damit definieren wir die Risk Exposure einer Alternative A als

$$RE(A) = P(r_1 \wedge r_2) \cdot S(r_1 \wedge r_2) + P(r_1 \wedge \bar{r}_2) \cdot S(r_1) + P(\bar{r}_1 \wedge r_2) \cdot S(r_2) + P(\bar{r}_1 \wedge \bar{r}_2) \cdot 0$$

Unter den beiden Annahmen $P(r_1 \wedge r_2) = P(r_1) \cdot P(r_2)$ und $S(r_1 \wedge r_2) = S(r_1) + S(r_2)$ ergibt sich:

$$\begin{aligned} RE(A) &= P(r_1 \wedge r_2) \cdot S(r_1 \wedge r_2) + P(r_1 \wedge \bar{r}_2) \cdot S(r_1) + P(\bar{r}_1 \wedge r_2) \cdot S(r_2) \\ &\quad + P(\bar{r}_1 \wedge \bar{r}_2) \cdot 0 \\ &= P(r_1 \wedge r_2) \cdot (S(r_1) + S(r_2)) + P(r_1 \wedge \bar{r}_2) \cdot S(r_1) + P(\bar{r}_1 \wedge r_2) \cdot S(r_2) \\ &= P(r_1 \wedge r_2) \cdot S(r_1) + P(r_1 \wedge r_2) \cdot S(r_2) + P(r_1 \wedge \bar{r}_2) \cdot S(r_1) \\ &\quad + P(\bar{r}_1 \wedge r_2) \cdot S(r_2) \\ &= (P(r_1 \wedge r_2) + P(r_1 \wedge \bar{r}_2)) \cdot S(r_1) + (P(r_1 \wedge r_2) + P(\bar{r}_1 \wedge r_2)) \cdot S(r_2) \\ &= P(r_1) \cdot S(r_1) + P(r_2) \cdot S(r_2) \\ &= RE(r_1) + RE(r_2) \end{aligned}$$

Die Risk Exposure ergibt sich also als Summe der Risk Exposures der beiden Einzelrisiken. Dies passt zu der Betrachtungsweise, dass die Risk Exposure den Erwartungswert darstellt und der Erwartungswert linear ist [81]. Für eine mathematisch korrekte Erläuterung müssen statt der Ereignisse r_1 und r_2 eigentlich deren Zufallsvariablen betrachtet werden, da der Erwartungswert nur auf Zufallsvariablen definiert ist. Dies wird in Anhang B aufgegriffen.

Auch allgemein, d.h. im Fall von mehr als zwei Risiken lassen sich die Berechnungsweisen anwenden. Daher definieren wir die Risk Exposure einer Alternative wie folgt. Abbildung 38 zeigt ein Beispiel zur praktischen Berechnung.

Definition: Risk Exposure der Alternative A:

Unter der Annahme, dass für alle Risiken $r \in R(A)$ die Ereignisse ihres Eintritts und ihre Schäden unabhängig voneinander sind, ist:

$$RE(A) = \sum_{r \in R(A)} RE(r), \text{ mit}$$

$R(A) = \text{Menge aller Risikofaktoren zu } A$

$RE(r) = \text{Risk Exposure des Risikofaktors } r$

Risikofaktor	Eintrittswahrscheinlichkeit (0-1)	Schaden (0-100)	Risk Exposure
Artefakttyp wird aufgenommen			Summe: 8
Analysephase verlängert sich	0,1	30	3
Zu detaillierte Spezifikation legt Projekt auf ungünstige Lösungen fest	0,1	50	5
Artefakttyp wird nicht aufgenommen			Summe: 26
Projekt läuft aus dem Rahmen	0,6	30	18
Viele Stakeholder -> Über-Anpassung wegen fehlendem Bezugspunkt	0,2	40	8

Bewertung des Gesamtrisikos der einzelnen Alternativen

Abbildung 38: Bewertung des Gesamtrisikos einer Alternative

Auf diese Weise können nun Alternativen prinzipiell verglichen werden. Jedoch muss beachtet werden, dass es bei der Addition von Risiken zur Bewertung von Alternativen auch zu Verfälschungen kommen kann. Die Anzahl der herangezogenen Risiken beeinflusst das Gesamtrisiko einer Vorgehensweise. Nun werden in den Risikolisten jedoch nur *bekannte* Risiken aufgeführt. Falls also eine der Vorgehensweisen zusätzliche Risiken besitzt, die nicht aufgeführt sind, so kann es passieren, dass sie weniger riskant wirkt, weil eben weniger Risiken aufgeführt sind. Um hier eine Verfälschung zu vermeiden, müsste der Requirements Methodiker die Risikolisten zusätzlich auf Vollständigkeit prüfen und gegebenenfalls um weitere Elemente ergänzen.

Erläuterung zur Berechnung unter Berücksichtigung überlappender Schäden

Wenn sich die Schäden von zwei Risiken überlappen, so kann der Schaden bei Eintreten beider Risiken nicht einfach als Summe der beiden einzelnen Schäden bestimmt werden. Dann ist $S(r_1 \wedge r_2) \neq S(r_1) + S(r_2)$. Folglich muss der Requirements Methodiker $S(r_1 \wedge r_2)$ separat schätzen.

Dann lässt sich aber wieder die angegebene Formel verwenden, dieses Mal jedoch nicht vereinfachen:

$$RE(A) = P(r_1 \wedge r_2) \cdot S(r_1 \wedge r_2) + P(r_1 \wedge \bar{r}_2) \cdot S(r_1) + P(\bar{r}_1 \wedge r_2) \cdot S(r_2)$$

Da die Ereignisse nach wie vor als unabhängig angenommen werden, können die Wahrscheinlichkeiten aber mittels $P(r_1 \wedge r_2) = P(r_1) \cdot P(r_2)$ über die bereits geschätzten Einzelwahrscheinlichkeiten berechnet werden:

$$\begin{aligned} RE(A) &= P(r_1) \cdot P(r_2) \cdot S(r_1 \wedge r_2) + P(r_1) \cdot (1 - P(r_2)) \cdot S(r_1) + (1 - P(r_1)) \cdot P(r_2) \\ &\quad \cdot S(r_2) \end{aligned}$$

Werden mehr als zwei Risiken kombiniert, so sind alle möglichen Kombinationen des Eintretens und Nicht-Eintretens der Risiken zu betrachten. Dies führt allerdings zu einem schnellen Anstieg der Komplexität bei der Bewertung der Schäden.

7.6 Ergebnis: Ermittelte Risikofaktoren

Im Folgenden werden die Risikotabellen zu den einzelnen Optimierungsdimensionen aufgeführt. In Anlehnung an die Risikodefinition wird in den Erläuterungen der Tabellenelemente einzeln ausgewiesen, was die Eintrittswahrscheinlichkeit beeinflusst („Wie entsteht das Risiko“) und was die Schadensgröße beeinflusst („Warum ist es ein Risiko“). Zudem werden Abhängigkeiten zu nachgelagerten Optimierungsdimensionen genannt, sofern welche existieren.

Die Risikofaktoren werden durch Interpretation und Weiterentwicklung von Aussagen aus der Interviewstudie abgeleitet. Die Autorin hat aus den Aussagen auf mögliche Eintrittsbedingungen oder Folgen von Risiken geschlossen. Zur Nachvollziehbarkeit sind die konkreten Aussagen in Anhang A dargestellt.

7.6.1 Risikofaktoren für die grundsätzliche Abwägung zwischen agiler oder plangetriebener Herangehensweise

Zunächst werden hier die Risikofaktoren zur grundsätzlichen Abwägung zwischen agiler oder plangetriebener Herangehensweise aufgeführt. Diese stammen aus dem Ansatz zur risikobasierten Ausbalancierung von Agilität und Disziplin von Boehm und Turner [16]. Die grundsätzliche Herangehensweise kann als eine Optimierungsdimension aufgefasst werden, die den anderen Optimierungsschritten vorgelagert ist (wie in Abschnitt 7.4 gezeigt).

Boehm und Turner unterscheiden zwischen Risiken einer agilen Herangehensweise, denen einer plangetriebenen Herangehensweise und zusätzlich allgemeinen umgebungsbedingten Risiken. Boehm und Turner führen diese auf, um sie bei der Gestaltung von Mitigation-Strategien zu berücksichtigen. Mitigation-Strategien sind Strategien zur Abwendung von Risiken. Tabelle 9 zeigt die in [16] aufgeführten Faktoren.

Tabelle 9: Risikofaktoren für agile bzw. traditionelle Herangehensweise

Risikofaktor
<i>Umweltbedingte Risikofaktoren</i>
Unsicherheiten bezüglich der Technologie
Viele Stakeholder
Komplexes System von Systemen
<i>Risikofaktoren der agilen Herangehensweise</i>
Skalierbarkeit und Kritikalität
Anwendung von <i>Simple Design</i>
Personelle Veränderungen
Nicht ausreichend Personen mit Erfahrung in agilen Methoden
<i>Risikofaktoren der plangetriebenen Herangehensweise</i>
Schneller Wandel (<i>Rapid Change</i>)
Notwendigkeit schneller Resultate
Neu auftkommende Anforderungen (<i>Emergent Requirements</i>)
Nicht ausreichend Personen mit Erfahrung in plangetriebenen Methoden

Für die Erklärung der einzelnen Faktoren sei an dieser Stelle an [16] verwiesen.

7.6.2 Risikofaktoren für die Aufnahme von Artefakten

In diesem Optimierungsschritt wird entschieden, ob ein Anforderungsartefakttyp in eine Anforderungslandschaft aufgenommen wird oder nicht. Die Aufnahme von oder der Verzicht auf Artefakttypen zu einer Anforderungslandschaft beeinflusst vor allem, welche Information der Analyse erarbeitet wird. Außerdem beeinflussen die vorhandenen Artefakttypen, von welchen Rollen und für welche Aktivitäten die Anforderungen später besonders gut genutzt werden können. Die folgende Tabelle zeigt hierzu die konkreten Risikofaktoren.

Tabelle 10: Auflistung relevanter Risikofaktoren bei Aufnahme bzw. Verzicht eines Artefaktes

Risikofaktor	
<i>Risikofaktoren bei Aufnahme eines Artefaktes</i>	
R-1.1	Verlängerung der Analysephase
R-1.2	Zu detaillierte Spezifikation legt Projekt auf ungeeignete / ungünstige Lösungen fest
R-1.3	Viele Relationen zu anderen Artefakten; häufige Anpassungen der Anforderungen -> Hoher Maintainance-Aufwand
R-1.4	Zähheit -> lähmt Projekt
<i>Risikofaktoren bei Verzicht auf ein Artefakt</i>	
R-1.5	Projekt läuft aus dem Rahmen
R-1.6	Häufige Anforderungswechsel oder viele Stakeholder -> Überanpassung wegen fehlendem Bezugspunkt
R-1.7	Fehlendes kundenverständliches Artefakt -> falsche Spezifikationen oder falsche Priorisierung

R-1.1 – Verlängerung der Analysephase:

Wie entsteht das Risiko? Die Anforderungsanalysephase eines Projektes kann sich durch die Zunahme eines Artefakttyps verlängern, wenn zur Erstellung der Artefakte dieses Typs eine umfangreiche und aufwändige Ausarbeitung der Anforderungen notwendig ist, bevor sie verwendet werden können. Beispielsweise kann die Erstellung eines kompletten Geschäftsprozessmodells, das von mehreren Stakeholdern gegengezeichnet wird, bevor es verwendet werden kann, viel Zeit in Anspruch nehmen. Im Gegensatz dazu kann die Aufnahme der wichtigsten Use Cases, vor allem wenn diese im Casual-Style [29] als kurze natürlichsprachliche Absätze verfasst sind, schneller erfolgen und führt zu keiner so großen Verlängerung der Analysephase.

Warum ist es ein Risiko? Die Verlängerung der Analysephase kann einem Projekt schaden, wenn dieses auf schnelle Resultate angewiesen ist. Dies ist zum Beispiel der Fall, wenn die Kunden früh erste Ergebnisse sehen wollen, wenn frühes Kundenfeedback wegen Unsicherheiten benötigt wird oder wenn ein frühes Release wichtig ist, um Marktanteile gegenüber der Konkurrenz zu gewinnen [16].

Abhängigkeiten zu anderen Optimierungsdimensionen: Zähheit des Artefakttyps: ist das Artefakt flexibel, so kann es leichter iterativ entwickelt und verwendet werden. Dadurch kann die Wahrscheinlichkeit der Verlängerung der Analysephase reduziert werden.

R-1.2 – Zu detaillierte Spezifikation legt Projekt auf ungeeignete / ungünstige Lösungen fest:

Wie entsteht das Risiko? Zum einen begünstigen bestimmte Artefakttypen die frühe Festlegung von Lösungen. Beispielsweise ist es in einem umfangreichen

Spezifikationsdokument, welches das Festhalten beliebiger Informationen erlaubt, leichter, bereits detaillierte Lösungsansätze, wie das genaue Aussehen von Oberflächen, zu spezifizieren, als in einer User Story-Menge, in der zunächst nur die Nutzerziele festgehalten und Lösungsdetails erst in der entsprechenden Iteration besprochen werden.

Zudem hängt es von der Erfahrung der Entwickler und Kunden ab, ob sie gut einschätzen können, welche Informationen bereits festgelegt und welche zunächst offen gelassen werden sollten.

Besonders bei zähen Artefakten kann es passieren, dass eine Lösung so umgesetzt werden muss, wie spezifiziert, auch wenn im Verlauf des Projektes klar wird, dass es bessere Lösungen gibt.

Warum ist es ein Risiko? Oftmals ergeben sich im Verlauf des Projektes – mit zunehmendem Kenntnisstand bei Entwicklern und Kunden – Änderungen in Bezug auf Lösungsansätze. Beispielsweise werden neue Einschränkungen entdeckt oder Stakeholder verstehen die technischen Möglichkeiten und deren Auswirkungen besser [13]. In einem unsicheren Projektumfeld kommen von außen Änderungen der grundsätzlichen Anforderungen hinzu. Ist bereits eine Lösung festgelegt, so kann sich diese als ungeeignet herausstellen, muss jedoch unter Umständen trotzdem umgesetzt werden, da es so vereinbart war. Die ungeeignete Lösung kann teurer, schlechter wartbar und schwerer erweiterbar sein als eine Alternative. Im schlimmsten Fall ist eine Lösung gar nicht realisierbar und Teile der Software müssen neu geschrieben werden.

Abhängigkeiten zu anderen Optimierungsdimensionen: Zähheit: ist das Artefakt flexibel, so können ungünstige Lösungen leichter geändert werden.

R-1.3 – Viele Abhängigkeiten zu anderen Artefakten; häufige Änderungen der Anforderungen -> Hoher Maintenance-Aufwand:

Wie entsteht das Risiko? Häufig kommt es im Lauf eines Projektes zu Änderungen der Anforderungen [13], [167]. Um diese Änderungen später auch noch nachvollziehen zu können, sollten Anforderungsartefakte angepasst werden, wenn Teile davon durch Änderungen nicht mehr aktuell sind. Dies erzeugt zusätzlichen Aufwand. Ändern sich im Projekt die Anforderungen häufig, so ist es auch wahrscheinlicher, dass bei Hinzunahme eines neuen Artefakttyps dieser zusätzliche Wartungskosten bei der Änderung von Requirements verursachen wird. Häufige Änderungen kommen beispielsweise durch besseres Verständnis und Wissen der Kunden, das sich mit der Zeit entwickelt, oder durch eine sich ändernde Projektumgebung zustande.

Des Weiteren werden Maintenance-Kosten durch Abhängigkeiten zwischen Anforderungsartefakten erhöht, wie bereits in Abschnitt 6.8.1 (*Koexistenz-Aktivitäten*) vorgestellt. Wenn ein Artefakt geändert wird, so muss dann auch geprüft werden, ob weitere abhängige Artefakte von der Änderung betroffen sind und diese gegebenenfalls angepasst werden. Wenn der hinzuzufügende Artefakttyp viele solche Abhängigkeiten zu anderen Artefakttypen besitzt, so

können entsprechend die Maintenance-Kosten hoch ausfallen. Dies ist beispielsweise dann der Fall, wenn das Artefakt inhaltlich vielen anderen Artefakten entspricht, nur die Inhalte aus einem neuen Blickwinkel darstellt. Genauso kann ein zentraler Artefakttyp, wie die Spezifikation oder die User Stories viele Abhängigkeiten besitzen, da sie praktisch immer etwas mit den anderen Blickwinkeln zu tun haben.

Warum ist es ein Risiko? Die Anpassung der Anforderungsartefakte bei Änderungen verursacht von sich aus bereits zusätzliche Kosten. Je mehr Abhängigkeiten dabei zwischen den Artefakten existieren, desto höher ist auch deren Komplexität und damit auch die Kosten. Es kommt aber auch hinzu, dass die Änderung andere Aktivitäten unterbricht und damit stört. Beispielsweise kann ein Änderungswunsch aufkommen, während ein Stakeholder die Software gerade reviewt. Wenn nun diese Änderung zuerst in mehreren Artefakten eingepflegt werden muss, so kommen die Beteiligten nicht mehr dazu, weiteres Feedback beim Review durchzusprechen. Häufig wird daher ein Änderungswunsch zunächst in einem Artefakt notiert, die Wartung der weiteren Anforderungsartefakte aber auf später verschoben. Hierbei kann es wiederum vorkommen, dass die Anpassungen vergessen werden und so Inkonsistenzen zustande kommen. Diese führen möglicherweise zu weiteren Fehlern und damit auch Kosten. Je weniger der Prozess die Einarbeitung solcher Änderungen berücksichtigt, desto eher kann es zu Inkonsistenzen oder inaktuellen Artefakten kommen.

Abhängigkeiten zu anderen Optimierungsdimensionen: Wie bereits oben erwähnt, beeinflussen Abhängigkeiten die Maintenance-Kosten. Diese können reduziert werden, indem konkrete **Abhängigkeiten** reduziert werden oder indem diese durch Einführung von **Verknüpfungen** leichter sichtbar gemacht werden. Die Verwendung von Verknüpfungen wiederum kann jedoch weiteren Maintenance-Aufwand erzeugen, da durch Änderungen auch Verknüpfungen selbst ungültig werden können [107].

Ist das einzuführende Artefakt **zäh**, so ist mit weniger Änderungen, die vom Artefakt selbst ausgehen, zu rechnen.

R-1.4 – Zähheit -> lähmt Projekt:

Wie entsteht das Risiko? Ist der hinzuzufügende Artefakttyp **zäh**, so sind Änderungen meist nur mit Verzögerung in die Artefakte einzupflegen. Besonders bei einem prozessbezogenen zähen Artefakt muss bei einer Änderung der Anforderungen zuerst auf die Zustimmung mehrerer Parteien gewartet werden, bevor diese umgesetzt und weiterverwendet werden kann. Ist so ein Abstimmungsprozess langwierig, so kann es passieren, dass ein Teil der Entwicklung stillsteht, bis die Entscheidung getroffen ist.

Vor allem wenn der Artefakttyp sehr detailliert ist, ist die Gefahr noch größer, dass bei häufigen kleinen Detail-Änderungen solche Wartezeiten oft entstehen.

Warum ist es ein Risiko? Wird das Projekt unnötig verzögert, so kann es zu einer Überschreitung der Fertigstellungsdeadline kommen. Das ist besonders

negativ für Projekte, die hohe vertragliche Strafen für solche Verzögerungen besitzen.

Abhängigkeiten zu anderen Optimierungsdimensionen: Wenn die **Zähheit** des Artefakttyps auf flexibel gesetzt wird, so tritt eine Verzögerung mit geringerer Wahrscheinlichkeit auf. Absprachen können noch immer nötig sein, doch sind sie nicht mehr starr für jedes Detail vorgegeben.

R-1.5 – Projekt läuft aus dem Rahmen („Scope Creep“):

Wie entsteht das Risiko? Der Projektrahmen (auch Projektscope genannt) stellt dar, welche Probleme von der Software gelöst oder welche Funktionen umgesetzt werden sollen und welche nicht. Wird Feedback vom Kunden zu Anforderungen oder zur Software aufgenommen, so muss auch darauf geachtet werden, ob die damit verbundene Funktionalität noch im vereinbarten Rahmen ist oder bereits über diesen hinausgeht. Genauso können Änderungen im Lauf eines Projektes dazu führen, dass zuvor vereinbarte Funktionen nicht mehr implementiert werden. Beispielsweise werden statt der fehlenden Funktionen eher Verbesserungen an bestehender Funktionalität implementiert, weil einige Stakeholder das wünschen. In beiden Fällen wird dann ein zuvor vereinbarter Rahmen nicht eingehalten.

Um auch während der Entwicklung zu überprüfen, ob eine Änderung vereinbar mit dem Projektscope ist, sind Artefakte hilfreich, die den Scope gut abbilden können. Beispielsweise können das Use Cases in einer Spezifikation sein. Wird auf ein solches Artefakt verzichtet [und gibt es im Projekt auch sonst keine Artefakte, die den Rahmen darstellen] (indem zum Beispiel nur User Stories verwendet werden), so kann dies zu dem Problem führen, dass die Entwicklung aus dem Rahmen läuft, ohne dass dies bemerkt wird.

Warum ist es ein Risiko? Erfüllt die Software nicht alle vereinbarten Funktionen, so kann dies zu Unzufriedenheit von Stakeholdern führen, die sich darauf verlassen haben. Besonders Stakeholder, die nicht die primäre Rolle bei der Anforderungskommunikation spielen oder nicht das ganze Projekt über verfügbar sind, verlassen sich möglicherweise stärker auf schriftlich vereinbarte Funktionen. Wenn dann erst spät kommuniziert werden kann, dass Funktionen fehlen werden, sind die Änderungen für diese Stakeholder möglicherweise nicht mehr nachvollziehbar.

Zudem kann die Nicht-Erfüllung des Rahmens zum Problem werden, wenn dieser vertraglich vereinbart wurde. Selbst wenn die Nutzer der Software mit der Funktionalität zufrieden sind, kann es zu formalen Vertragsstrafen kommen, wenn beispielsweise Abnahmetestfälle nicht erfüllt sind.

Grundsätzlich wurde auch ein statistisch signifikanter Zusammenhang zwischen Änderungen des Rahmens, bzw. *Anforderungsvolatilität*, und Zeit- sowie Budget-Überschreitungen eines Projekts festgestellt [167].

Abhängigkeiten zu anderen Optimierungsdimensionen: Keine. Wenn auf ein Artefakt verzichtet wird, so gibt es keine zutreffenden nachgelagerten Optimierungsdimensionen und damit auch keine weiteren Abhängigkeiten.

R-1.6 – Häufige Anforderungswechsel oder viele Stakeholder -> Überanpassung wegen fehlendem Bezugspunkt:

Wie entsteht das Risiko? Anpassungen von Anforderungen oder Software im Laufe eines Projektes sind häufig hilfreich [8], [51]. Treten jedoch zu viele Änderungen in der Umgebung auf oder sind am Projekt zu viele Stakeholder mit widersprüchlichen Interessen vertreten, so kann es zu zu starken Anpassungen der Software kommen, die sich immer wieder widersprechen.

Dokumentation kann den Nutzen haben, dass sie einen Bezugspunkt darstellt, auf den sich Projektteilnehmer berufen können. Ist ein zähes Artefakt im Projekt vorhanden, zu dem alle relevanten Parteien zugestimmt haben, so können Änderungen, die den Anforderungen in diesem Artefakt widersprechen, abgewiesen werden, wenn erforderlich. Damit wird zu starke Änderung der Anforderungen vermieden. Fehlt ein solches Artefakt hingegen, ist Überanpassung wahrscheinlicher. Beispielsweise ist es dann auch schwieriger nachzuvollziehen, wenn eine beschlossene Anpassung noch mehrmals umgekehrt wird, weil es keinen Bezugspunkt gibt.

Warum ist es ein Risiko? Änderungen erzeugen zusätzlichen Aufwand. Werden zudem Änderungen immer wieder rückgängig gemacht, so werden die investierten Ressourcen aufgelöst. Zu starke oder häufige Anpassungen an Änderungen können so dazu führen, dass das Projekt „stillsteht“, da alle Entwicklungen immer wieder rückgängig gemacht werden. Letztlich führt dies zu einer Überschreitung von Budget und Zeit oder einer Stilllegung des Projektes aufgrund mangelnder Zwischenresultate.

Abhängigkeiten zu weiteren Optimierungsdimensionen: Keine

R-1.7 – Fehlendes kundenverständliches Artefakt -> falsche Spezifikationen oder falsche Priorisierung:

Wie entsteht das Risiko? Artefakttypen sind für verschiedene Personen unterschiedlich verständlich. In der Interview-Studie hat sich herausgestellt, dass Personen auf Kundenseite oft Schwierigkeiten haben, Anforderungsartefakte zu verstehen, wenn diese zu technisch oder zu abstrakt sind. Konkrete Beschreibungen von Abläufen, unterstützt durch GUI-Mockups können das Kundenverständnis hingegen verbessern. Fehlen im Projekt nun solche kundenverständlichen Artefakte und soll bei der Optimierung auf ein kundenverständliches Artefakt verzichtet werden, so kann es passieren, dass Stakeholder aufgrund mangelnden Verständnisses Fehler in der Spezifikation nicht entdecken. Genauso kann es aufgrund des mangelnden Verständnisses dazu kommen, dass Stakehol-

der die Bedeutung von Anforderungen falsch einschätzen und sie nicht richtig priorisieren.

Warum ist es ein Risiko? Falsche Anforderungen sind vor allem dann ein Problem, wenn die Entwicklungsmethode mit nachträglichen Änderungen nicht gut umgehen kann. Dies ist besonders bei plangetriebenen Entwicklungsmethoden der Fall. Im Kontext von agilen Entwicklungsmethoden, die auf leichte Änderung der Software ausgerichtet sind, kann sich das schlechte Kundenverständnis jedoch ebenfalls negativ auswirken, wenn die Anforderungen ungünstig priorisiert werden. Dies kann nämlich dazu führen, dass das Produkt am Ende nicht den erwarteten Wert liefert, sodass die Stakeholder unzufrieden sind.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

7.6.3 Risikofaktoren für verschiedene Ausprägungen der Zähheit eines Anforderungsartefaktes

Nicht immer lässt sich die Zähheit eines Artefaktes beeinflussen. Durch prozessbezogene Festlegungen kann in einigen Fällen dennoch festgelegt werden, welche Änderungen durch wen vorgenommen werden können. Um diesen Aspekt festzulegen, lassen sich die Faktoren aus Tabelle 11 heranziehen.

Tabelle 11: Risikofaktoren für verschiedene Ausprägungen der Zähheit eines Anforderungsartefaktes

Risikofaktor	
<i>Risikofaktoren eines zähen Artefaktes</i>	
R-2.1	Abhängigkeiten zu flexiblem Artefakt -> ungewünschte Änderungen treten an anderer Stelle auf
R-2.2	Unnötige Verzögerung durch Abstimmungsprozess
R-2.3	Artefakt wird nicht gepflegt
<i>Risikofaktoren eines flexiblen Artefaktes</i>	
R-2.4	Überanpassung
R-2.5	Projektteilnehmer oder angrenzende Projekte bekommen Änderungen nicht rechtzeitig mit
R-2.6	Änderungen an Anforderungen sind nicht mehr nachvollziehbar

R-2.1 – Abhängigkeiten zu flexiblem Artefakt -> ungewünschte Änderungen treten an anderer Stelle auf:

Wie entsteht das Risiko? Entscheidet man sich dazu, ein Artefakt zäh zu machen, so wird es schwieriger, dessen Inhalte zu ändern. Wenn jetzt dieselben oder verwandte Inhalte aber in anderen flexiblen Artefakten vorhanden sind – was also eine Abhängigkeit zu einem flexiblen Artefakt darstellt – so kann es

passieren, dass diese Anforderungen doch durch Personen geändert werden, die hauptsächlich mit den flexiblen Artefakten arbeiten. Dann muss entweder das zähe Artefakt ebenfalls geändert werden oder es entstehen Inkonsistenzen.

Je mehr der Inhalte des zähen Artefaktes auch in anderen Artefakten vorhanden sind desto eher tritt dieses Problem auf. Außerdem spielt es eine Rolle, wie häufig generell Änderungen im Projekt auftreten.

Warum ist es ein Risiko? Es muss viel Aufwand in die Zusatzabstimmungen gesteckt werden, die mit Anpassungen der Anforderungen in vorliegenden Artefakttyp verbunden sind. Wird dies nicht getan, so treten Inkonsistenzen auf, was auf lange Sicht ebenfalls Kosten durch Fehler oder Nacharbeit verursacht.

Abhängigkeiten zu weiteren Optimierungsdimensionen: Manche Abhängigkeiten können aufgelöst werden, indem überlappende Inhalte aus verschiedenen Artefakten zusammengefasst werden. Ist gewünscht, dass sich die Inhalte wenig ändern, so sollten diese nur im zähen Artefakt auftauchen. Handelt es sich bei den überlappenden Teilen um Details, die tatsächlich frei geändert werden können, so sollten sie aus dem zähen Artefakt herausgenommen werden.

R-2.2 – Unnötige Verzögerung durch Abstimmungsprozess:

Wie entsteht das Risiko? Bei einem zähen Artefakt müssen mehrere Parteien zu Änderungen zustimmen. Dies ist häufig mit Verzögerungen verbunden, weil die Prüfung der Änderungen Zeit in Anspruch nimmt und die Projektteilnehmer auch nicht immer kurzfristig Zeit haben, sich darum zu kümmern. Ein Problem kann entstehen, wenn die Inhalte nun nicht gut auf die abstimmenden Parteien zugeschnitten sind. Beispielsweise können mehrere Arten oder Granularitätsebenen von Anforderungen in einem Artefakt, wie einer Spezifikation, gemischt und einige der abstimmenden Parteien an den detaillierten Teilen gar nicht mehr interessiert sein. Wird dennoch aus Gründen der Formalität des Prozesses eine Zustimmung von solchen Projektteilnehmern zu Inhalten gefordert, die diese eigentlich nicht mehr betreffen, kommt es zu unnötigen Verzögerungen im Projekt.

Warum ist es ein Risiko? Durch einzelne Verzögerungen wird schließlich auch das Gesamtprojekt verzögert, was dazu führen kann, dass Zeitvorgaben nicht eingehalten werden können. Sind außerdem viele Abhängigkeiten zwischen Teilen der Software vorhanden, so wirkt sich das Problem stärker aus. Dann müssen bei Verzögerungen möglicherweise mehrere Entscheidungs- oder Implementierungsaktivitäten warten.

Abhängigkeiten zu weiteren Optimierungsdimensionen: Keine

R-2.3 – Artefakt wird nicht gepflegt:

Wie entsteht das Risiko? Die Hürde, ein zähes Artefakt zu ändern ist größer als bei einem flexiblen Artefakt. Je nach dem, wie komplex der Prozess ist, können schon für kleinste Änderungen viele Aktivitäten auf die Projektteilnehmer zu-

kommen. Beispielsweise müssen sie ein Dokument auschecken, sich als Editor eintragen, es wieder einchecken. Für zu kleine Änderungen will man dann die abstimmenden Parteien auch nicht unbedingt stören. Die Nachfrage wird dann beispielsweise auf später verschoben. Außerdem besteht die Gefahr, dass die Projektteilnehmer bei kleinen Änderungen bewusst darauf verzichten, die Artefakte zu pflegen, um sich den Zusatzaufwand zu sparen.

Warum ist es ein Risiko? Änderungen werden dann nur im Code durchgeführt, aber nicht mehr ins Artefakt eingetragen. Das führt zu Artefakten, die nicht mehr aktuell sind. Werden diese Artefakte verwendet, um beispielsweise einen Teil der Software zu warten oder eine Erweiterung zu planen, so kann es passieren, dass Entwickler oder Kunden Abhängigkeiten übersehen oder falsche Annahmen über das System treffen.

Nicht aktuelle Artefakte verlieren außerdem schnell an Vertrauen, wie in der Interview-Studie beschrieben. Eine Folge ist also, dass das Artefakt dann gar nicht mehr verwendet wird und nicht mehr zur Anforderungskommunikation beitragen kann.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

R-2.4 - Überanpassung:

Wie entsteht das Risiko? Dieses Risiko wurde bereits in R-1.7 diskutiert. Dort kommt das Problem der Überanpassung dadurch zustande, dass Inhalte gar nicht dokumentiert werden. Wenn dann zu viele Stakeholder im Projekt widersprüchliche Interessen haben, kann es passieren, dass diese die Entwicklungsrichtung hin- und zurückändern und wichtige Entscheidungen immer wieder rückgängig machen. In der Situation, die nun in R-2.4 betrachtet wird, werden Inhalte im Artefakttyp festgehalten – und damit prinzipiell grundsätzlich ein Bezugspunkt für Projektteilnehmer geschaffen – jedoch kann dieser nun unbeständig sein, wenn das Artefakt flexibel ist und potenziell alle Änderungen zulässt.

Warum ist es ein Risiko? Wie bereits in R-1.7 erläutert, führen die Änderungen zu zusätzlichem Aufwand und potenziell zu einem „Stillstand“ des Projektes.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

R-2.5 – Projektteilnehmer oder angrenzende Projekte bekommen Änderungen nicht rechtzeitig mit:

Wie entsteht das Risiko? Projektteilnehmer oder angrenzende Projekte verlassen sich möglicherweise darauf, dass spezifizierte Funktionen in der Zukunft bereitgestellt werden. Werden nun Änderungen an den Anforderungen vorgenommen, so kann dies dazu führen, dass solche Funktionen nicht mehr implementiert werden oder sich ändern.

Ein zähes Artefakt, bei dem die abhängigen Parteien zu Änderungen zustimmen, könnte die Awareness bezüglich solcher steigern. Ist das Artefakt jedoch flexibel und sind keine anderen Mechanismen für Awareness vorhanden, so ist es

möglich dass Änderungen schnell durchgeführt werden, ohne die anderen Parteien davon zu informieren.

Warum ist es ein Risiko? Wenn sich Entwickler oder andere Projekte auf spezifizierte Funktionen verlassen, jedoch zu spät bemerken, dass sich diese geändert haben, so kann das zu Mehraufwand in der Entwicklung führen. Die Software muss umgeschrieben oder gar in ihrer Architektur verändert werden, um den Änderungen gerecht zu werden. Außerdem können Änderungen, beispielsweise an der Schnittstelle, bestehende Funktionalität beschädigen. Wird dies zu spät bemerkt, so entsteht auch hier Mehraufwand durch nachträgliche Korrekturen.

Auch Projektteilnehmer auf Kundenseite können Änderungen an Anforderungen verpassen, wenn sie nicht täglich mit den Anforderungen in Berührung sind. Auch wenn eine Änderung mit anderen Stakeholdern abgestimmt war, kann es passieren, dass die Kunden unzufrieden sind, weil sie sich auf die spezifizierte Funktionalität verlassen haben.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

R-2.6 – Änderungen an Anforderungen sind nicht mehr nachvollziehbar:

Wie entsteht das Risiko? Häufig ist mit zähen Artefakten auch ein Änderungsprozess verbunden. Dieser kann auch die Dokumentation von Änderungen, zum Beispiel mithilfe von Change Requests oder Versionierung der Anforderungen, beinhalten. In flexiblen Artefakten hingegen können Anforderungen schnell und leichtgewichtig geändert werden, sodass eine Dokumentation von Änderungen schnell vergessen werden kann. Dann ist es eine Tool- und Konfigurationsfrage, ob die Historie von Änderungen gespeichert wird oder nicht. Im Fall von generischen Dokumenten, wie Word- oder Excel-Dateien gibt es beispielsweise keine toolunterstützte Historie bis auf einige rudimentäre Funktionen des Änderungsmodus.

Warum ist es ein Risiko? Dieses Problem tritt in Projekten auf, in denen es notwendig ist, Änderungen nachvollziehen zu können. Dies sind zum Beispiel Projekte, die vertraglich an eine Spezifikation gebunden sind. Sind am Ende aufgrund von Änderungen einige der initialen Anforderungen nicht erfüllt, so muss ggf. belegbar sein, wann, warum und durch wen es zur Änderung kam.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

7.6.4 Risikofaktionen für verschiedene Ausprägungen von Abhängigkeiten bei sich entsprechenden Artefakttypen

Meist können Abhängigkeiten nicht beeinflusst werden. Werden zwei Artefakttypen verwendet, die ähnliche Informationen aus unterschiedlichen Blickwinkeln beschreiben, so wird immer eine Abhängigkeit zwischen Artefakten herrschen. In einigen Fällen lassen sich die zusammenhängenden Informationen

jedoch zusammenfassen. Dadurch wird vermieden, dass die Information über mehrere Artefakte verstreut ist, wodurch es einfacher wird, die Abhängigkeiten zu berücksichtigen. Abbildung 39 zeigt hierzu ein Beispiel mit zwei Anforderungslandschaften. Im oberen Fall sind die Use Cases und Geschäftsprozessmodelle in jeweils unterschiedlichen Artefakten untergebracht. Im unteren Fall werden die Use Cases und Prozessmodelle, die zu einem Ablauf gehören, zusammengefasst und auch gemeinsam dargestellt. So sind alle relevanten Informationen zu einem bestimmten Ablauf an einem Ort vorhanden, was die Anpassung erleichtert, wenn beispielsweise an einem Ablauf etwas verändert wird. Beide Varianten haben Vor- und Nachteile, die in Tabelle 12 aufgegriffen werden.

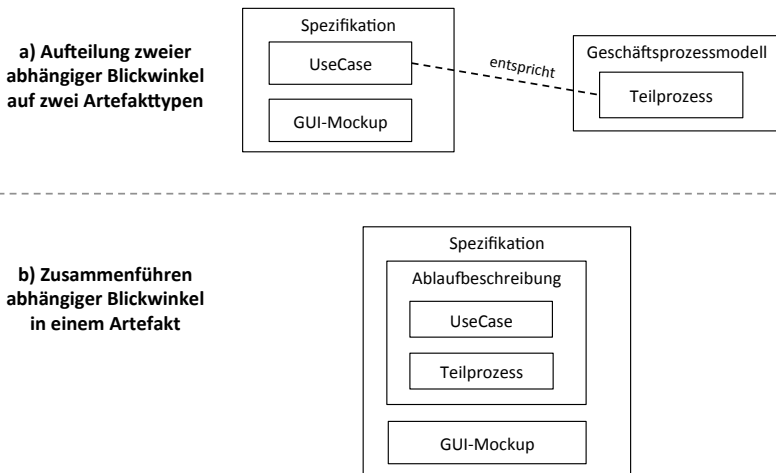


Abbildung 39: Beispiel für unterschiedlichen Umgang mit Abhängigkeiten

Tabelle 12: Risikofaktoren für verschiedene Ausprägungen von Abhängigkeiten zwischen Artefakttypen

Risikofaktor	
<i>Risikofaktoren bei Halten zweier sich entsprechender Artefakttypen</i>	
R-3.1	Bei Anpassungen eines Artefaktes wird das entsprechende andere Artefakt nicht angepasst
R-3.2	Informationen müssen aus mehreren Artefakten zusammengetragen werden
R-3.3	Viele Änderungen -> hoher Maintenance-Aufwand
<i>Risikofaktoren bei Zusammenfassen der Informationen zu einem Artefakttyp</i>	
R-3.4	Artefakt wird komplexer und weniger lesbar
R-3.5	Komplexe Beziehungen zwischen Artefakten -> Doppelung von Informationen

R-3.1 – Bei Anpassungen eines Artefaktes wird das entsprechende andere Artefakt nicht angepasst:

Wie entsteht das Risiko? Vor allem, wenn für die unterschiedlichen Aktivitäten jeweils nur ein einzelner Artefakttyp verwendet wird, können die Projektteilnehmer alle Änderungen in diesem Artefakttyp vornehmen, jedoch vergessen, abhängige Anforderungen in anderen Artefakttypen anzupassen. Beispielsweise werden bei der Iterationsplanung oft primär User Stories verwendet. Diese werden bei der Iterationsplanung schnell angepasst. So kann es passieren, dass ein Überblicksmodell über die Geschäftsprozesse oder ein Überblicks-Use Case, der den Gesamtablauf beschreibt, nicht betrachtet und auch nicht angepasst wird.

Warum ist es ein Risiko? Auch Artefakte, die in der Entwicklung nicht als primäre Artefakte verwendet werden, können dennoch eine wichtige Rolle spielen. Sie helfen, Widersprüche festzustellen, nachzuvollziehen, ob alle Ziele erreicht wurden oder dienen als Einstieg für neue Projektteilnehmer, die sich in das Projekt einarbeiten wollen. Sind solche Artefakte jedoch nicht aktuell, so werden sie unbrauchbar.

Abhängigkeiten zu weiteren Optimierungsdimensionen: Verknüpfungen. Werden die abhängigen Artefakte miteinander verknüpft, so wird es einfacher, diese Abhängigkeiten beim Anpassen von Anforderungen zu identifizieren. Werden außerdem die Verknüpfungen sichtbar gemacht, so können die Projektteilnehmer auch eher daran erinnert werden, auch die abhängigen Artefakte zu berücksichtigen.

R-3.2 – Informationen müssen aus mehreren Artefakten zusammengetragen werden:

Wie entsteht das Risiko? Dieses Risiko besteht, wenn die Artefakte der beiden Artefakttypen viele disjunkte Informationen enthalten, also Informationen, zu denen es keine direkte Entsprechung im jeweils anderen Artefakttyp gibt. Dann müssen sowohl die Informationen aus dem einen als auch aus dem anderen Artefakttyp herangezogen werden, um ein umfassendes Bild beispielsweise zu einem gewünschten Feature zu bekommen.

Dies kann der Fall sein, wenn die einzelnen Artefakttypen viele Informationen enthalten, die in andere Artefakttypen nicht hineinpassen. Beispielsweise finden nichtfunktionale Anforderungen nur schwer Platz in einer User Story-Menge. Ähnlich lässt sich die Information über die Reihenfolge von Aktivitäten, wie im Geschäftsprozess üblich, nur schwer mit User Stories abbilden, da diese nach Priorität statt Ausführungsreihenfolge sortiert werden. Wenn viele Informationen zu Anforderungen erst nachträglich hinzukommen, wie in der agilen Entwicklung üblich, kann es ebenfalls passieren, dass die Informationen, wie in R-3.1 beschrieben, nur in einem Artefakt festgehalten und nicht auf die anderen Artefakte übertragen werden.

Warum ist es ein Risiko? Wichtige Informationen können schnell übersehen werden, was zu Fehlern führt. Da meist auch nicht bekannt ist, ob Informationen zu einer Anforderung oder einem Feature nur in einem oder in beiden Artefakten enthalten sind, ist oft auch nicht klar, wann alle Informationen gefunden wurden bzw. wie lange gesucht werden sollte. Wenn relevante Stellen in mehreren Artefakten identifiziert werden müssen, dauert die Suche meist auch länger. Dies führt wieder zu einer Unterbrechung der eigentlichen Aktivitäten, wie der Priorisierung von Anforderungen oder der Analyse von Auswirkungen einer Änderung für einen Change Request.

Abhängigkeiten zu weiteren Optimierungsdimensionen: Verknüpfungen. Werden die abhängigen Artefakte miteinander verknüpft, so wird es einfacher, diese zusätzlichen Informationen beim Suchen von Anforderungen zu identifizieren. Werden außerdem die Verknüpfungen sichtbar gemacht, so können die Projektteilnehmer auch eher daran erinnert werden, auch die abhängigen Artefakte zu berücksichtigen.

R-3.3 – Viele Änderungen -> hoher Maintainance-Aufwand:

Wie entsteht das Risiko? Wie auch in R-1.3 beschrieben, kann das Hinzunehmen eines zusätzlichen Artefakttypen zu zusätzlichem Aufwand für die Wartung der entsprechenden Artefakte führen. Werden Anforderungen in einem der beiden abhängigen Artefakttypen häufig geändert, so können hohe Kosten für die Anpassung des jeweils anderen Artefakttyps entstehen.

Warum ist es ein Risiko? Wie bereits in R-1.3 beschrieben, verursacht die zusätzliche Maintainance sich entsprechender Artefakte zusätzliche Kosten und kann dazu führen, dass eine anforderungsbezogene Aktivität dadurch gestört wird.

Abhängigkeiten zu weiteren Optimierungsdimensionen: Verknüpfungen reduzieren den Aufwand, abhängige Artefakte zu identifizieren und können so dazu beitragen, die Maintainance-Kosten zu verringern.

R-3.4 – Artefakt wird komplexer und weniger lesbar:

Wie entsteht das Risiko? Durch Zusammenfassen von Blickwinkeln in einen Artefakttyp mischen sich verschiedenartige Informationen. Dadurch wird es auch schwieriger, die richtigen Informationen herauszufiltern. Je umfangreicher dabei die einzelnen Artefakte sind, desto schlechter lesbar wird ein Artefakt, in dem die relevanten Informationen gefiltert werden müssen. Vor allem, wenn in vielen der Aktivitäten nur einer der beiden Blickwinkel interessant ist, kann die zusätzliche Information als störend empfunden werden.

Warum ist es ein Risiko? Wenn die Projektteilnehmer länger brauchen, benötigte Informationen zu finden oder den Vorgang als schwieriger empfinden, kann dies dazu führen, dass sie nicht alle Informationen in den Anforderungsartefakten nachschlagen. Stattdessen können sie versuchen, das Artefakt zu meiden und

Informationen aus dem Gedächtnis abzurufen oder erneut mündlich abzusprechen. Beide Situationen können zu Fehlern und Inkonsistenzen führen.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

R-3.5 – Komplexe Beziehungen zwischen Artefakten -> Doppelung von Informationen:

Wie entsteht das Risiko? Wenn die konkreten Artefakte der abhängigen Artefakttypen nicht in einer eins-zu-eins-Beziehung stehen, so kann es passieren, dass Inhalte doppelt dargestellt werden müssen, wenn die Artefakte zu einem Artefakt zusammengefasst werden. Wenn beispielsweise in einer Situation, wie in Abbildung 39 (b) dargestellt, zwei der Use Cases zu einem Teilprozess passen, so kann es – je nach Aufbau der Ablaufbeschreibung - notwendig sein, den Teilprozess zweimal abzubilden. Wird aus jedem Use Case genau eine Ablaufbeschreibung erzeugt, dann muss der Teilprozess bei jedem Use Case jeweils einmal aufgeführt werden. Vor allem bei n-zu-m-Beziehungen ließen sich Doppelungen so nicht mehr vermeiden.

Warum ist es ein Risiko? Ist dieselbe Information an mehreren Stellen doppelt vorhanden, so wird die Wartung solcher Informationen inkonsistenz- bzw. fehleranfälliger. Wird ein Artefakt an einer Stelle geändert, so muss es auch an den weiteren Stellen, an denen es auftaucht, angepasst werden.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

7.6.5 Risikofaktoren für die Verknüpfung von Artefakten

Die Verknüpfung von Artefakten ist grundsätzlich positiv, weil Verknüpfungen Abhängigkeiten zwischen den Anforderungen visualisieren und damit helfen, diese besser zu berücksichtigen. Jedoch ist die Erstellung von Verknüpfungen auch aufwendig. Zudem müssen für zwei Artefakttypen klare Richtlinien gelten, um die Verknüpfungen später auch effektiv nutzen zu können. Beispielsweise sollte man bei der gewünschten Verknüpfung zwischen Geschäftsprozessen und User Stories verlangen, dass möglichst alle Abhängigkeiten durch eine Verknüpfung abgedeckt werden. Ist hingegen nicht klar, ob die mit einer User Story verknüpften Prozessaktivitäten wirklich alle abhängigen Prozesse abdecken, verlieren die Verknüpfungen ihre Wirkung, da die Projektteilnehmer die Anforderungsmenge dennoch immer wieder manuell durchsuchen müssen, um eventuelle weitere Elemente zu identifizieren. Trotz solcher Richtlinien kann es dazu kommen, dass nicht alle abhängigen Elemente verknüpft wurden. Wenn sich die Projektteilnehmer jedoch auf die Vollständigkeit der Verknüpfungen verlassen, birgt dies neue Risiken, Abhängigkeiten zu übersehen. Tabelle 13 zeigt diese und weitere Risiken, die eine Entscheidung für oder gegen die Einführung von Verknüpfungen beeinflussen.

Tabelle 13: Risikofaktoren für Verknüpfung von abhängigen Artefakten

Risikofaktor	
<i>Risikofaktoren bei Verknüpfung zweier abhängiger Artefakttypen</i>	
R-4.1	Viele <i>False Positives</i> -> Verknüpfte Elemente müssen zusätzlich gefiltert werden
R-4.2	Viele <i>False Negatives</i> -> Relevante Informationen werden missachtet oder nicht angepasst
R-4.3	Hohe Kosten für Erstellung von Verknüpfungen
<i>Risikofaktoren bei Verzicht auf explizite Verknüpfung zweier Artefakttypen</i>	
R-4.4	Relevante abhängige Informationen werden missachtet oder nicht angepasst
R-4.5	Suche nach abhängigen Anforderungen nimmt viel Zeit in Anspruch und unterbricht Aktivitäten

R-4.1 – Viele *False Positives* -> Verknüpfte Elemente müssen zusätzlich gefiltert werden:

Wie entsteht das Risiko? *False Positives* sind Artefakte, die zu einem bestimmten Element als verknüpfte Artefakte zurückgegeben werden, jedoch nicht relevant sind, also keine relevante zusätzliche Information enthalten. Sie können entstehen, wenn Elemente fälschlicherweise verknüpft werden oder wenn Verknüpfungen ungültig werden nachdem sich Anforderungen geändert haben. Dies ist beispielsweise der Fall, wenn viele Änderungen an den Anforderungen oder auch ihrer Struktur auftreten.

Wird die Anforderungsmenge für sehr unterschiedliche Aktivitäten verwendet, so kommt hinzu, dass für die verschiedenen Aktivitäten unterschiedliche Artefakte als relevant gelten. Während beispielsweise für einen Entwickler zu einem Use Case-Schritt nicht relevant ist, welche bereits abgeschlossenen User Stories damit verknüpft sind, können für den Projektleiter durchaus auch die abgeschlossenen User Stories interessant sein, um die Entwicklung des Use Case-Schrittes im Projektverlauf nachvollziehen zu können. So kann es also sein, dass ein-und-dieselbe Verknüpfung für eine Person zu *True Positives* führt – also eine Daseinsberechtigung besitzt – aber gleichzeitig für eine andere Person zu *False Positives* führt.

Warum ist es ein Risiko? Wie auch schon bei einigen der anderen Risiken, besteht hier das Problem, dass die anforderungsbezogenen Aktivitäten erschwert oder verzögert werden. Werden als verknüpfte Artefakte auch solche zurückgegeben, die für eine Aktivität nicht relevant sind, so muss ein Projektteilnehmer zunächst alle zurückgegebenen Artefakte betrachten und aus ihnen solche filtern, die tatsächlich angepasst werden müssen oder zusätzliche relevante Informationen enthalten. Wird diese Aufgabe als zu schwierig empfunden, so besteht die Gefahr, dass die Projektteilnehmer verknüpfte Artefakte gar nicht mehr betrachten oder anpassen, wenn sie mit Anforderungen umgehen. Dieses Problem tritt vor allem in umfangreichen Projekten auf.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

R-4.2 – Viele *False Negatives* -> Relevante Informationen werden missachtet oder nicht angepasst:

Wie entsteht das Risiko? False Negatives zu einem Element sind Artefakte, die eine Abhängigkeit zu diesem Element besitzen, jedoch nicht mit diesem verknüpft sind. Werden alle verknüpften Elemente betrachtet, um beispielsweise bei einer Änderung alle abhängigen Anforderungen anzupassen, so taucht ein falsch negatives Artefakt nicht in dieser Menge auf und wird möglicherweise nicht angepasst. False Negatives entstehen, wenn zu einem Artefakt nicht alle abhängigen Artefakte verknüpft werden, beispielsweise weil einige Abhängigkeiten übersehen werden. Dies wird besonders in umfangreichen Anforderungsmengen begünstigt oder in solchen, in denen viele Artefakte in komplexen m-zu-n-Beziehungen – mit hohen Werten für m und n – stehen. Gestaltet sich der Verknüpfungsprozess als aufwändig, kann es auch dazu kommen, dass ein neues Artefakt erstellt und gar nicht mit vorhandenen Artefakten verknüpft wird, wenn die Prozessteilnehmer die anderen Artefakttypen als irrelevant ansehen.

Warum ist es ein Risiko? Die Missachtung relevanter Informationen führt zu Fehlern und Inkonsistenzen. Solche Fehler können dazu führen, dass Abhängigkeiten oder Einschränkungen erst spät bemerkt werden und zu nachträglichen Änderungen führen.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

R-4.3 – Hohe Kosten für Erstellung von Verknüpfungen:

Wie entsteht das Risiko? Zunächst führt die Erstellung der Verknüpfungen selbst zu Kosten. Wie auch in Abschnitt 6.8.2 beschrieben, ist die Erstellung von Verknüpfungen eine Koexistenz-Aktivität, die Zeit in Anspruch nimmt. Vor allem, wenn Verknüpfungen nachträglich angelegt werden müssen, müssen die Anforderungen immer wieder auf der Suche nach Abhängigkeiten durchgegangen werden, was langwierig und schwierig sein kann.

Doch auch bei der direkten Erstellung von Verknüpfungen zu einem neuen Artefakten können zusätzliche Kosten dadurch entstehen, dass die eigentliche Aktivität, wie das Festhalten von neuen User Stories in einem Sprint-Review-Treffen, unterbrochen wird, weil nach Abhängigkeiten gesucht werden muss.

Insbesondere wenn die Anforderungen allgemein viele Abhängigkeiten besitzen und in unterschiedlichen Artefakttypen auftreten, werden die Kosten für die Verknüpfungen sehr hoch. Dann kann es dazu kommen, dass bei der bloßen Dokumentation einer neuen User Story diese mit beispielsweise zehn Use Case-Schritten und zehn Prozessaktivitäten verknüpft werden muss. Diese Aktivität verletzt dann schnell die Leichtgewichtigkeit, die in der agilen Entwicklung angestrebt ist.

In Abschnitt 6.8.2 ist näher erläutert, wann die Kosten für die Erstellung von Verknüpfungen ihren Nutzen übersteigen können.

Warum ist es ein Risiko? Die hohen Kosten für die Erstellung und Maintenance von Verknüpfungen manifestieren sich vor allem in Zeitaufwand und

können das Projekt verlangsamen. Einzelne anforderungsbezogene Aktivitäten können ebenfalls gestört werden, was wiederum dazu führen kann, dass die Verknüpfung von Artefakten auf später verschoben oder ganz gemieden wird. Dies wiederum kann zu *False Negatives* führen, die in R-4.2 aufgeführt sind.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

R-4.4 – Relevante abhängige Informationen werden missachtet oder nicht angepasst:

Wie entsteht das Risiko? Wenn abhängige Artefakte nicht explizit miteinander verknüpft werden, so sind die Abhängigkeiten, die ein Artefakt besitzt, auch nicht direkt sichtbar. Entsprechend kann es passieren, dass Projektteilnehmer bei der Suche nach abhängigen Elementen einige davon übersehen. Weiterhin arbeiten Projektteilnehmer oft primär mit Anforderungen von nur einem Artefakttyp auf einmal. Wenn zu einem Artefakt keine abhängigen Elemente explizit durch Verknüpfungen angezeigt werden, so kann es passieren, dass die Projektteilnehmer während ihrer Aktivitäten nicht daran denken, auch Artefakte von anderen Artefakttypen anzupassen.

Warum ist es ein Risiko? Die Missachtung relevanter Informationen führt zu Fehlern und Inkonsistenzen. Solche Fehler können dazu führen, dass Abhängigkeiten oder Einschränkungen erst spät bemerkt werden und zu nachträglichen Änderungen führen.

Die Missachtung abhängiger Informationen spielt bei zwei typischen Aspekten eine wichtige Rolle. Bei der Abschätzung des Aufwandes, den eine Änderung mit sich bringt, (Impact Analyse) müssen häufig genau die Abhängigkeiten in Betracht gezogen werden, um zu einer realistischen Schätzung zu gelangen. Fehler führen hier häufig zu Mehraufwand. Der zweite Aspekt ist die Erfüllung eines vorgegebenen Projektrahmens. Damit ist gemeint, dass im Vorfeld definiert wird, welche Features oder Anforderungen die Software später erfüllen muss. In der täglichen Arbeit – insbesondere bei agilem Vorgehen – werden dann Anforderungen jedoch häufig geändert. Werden dabei die Abhängigkeiten zu den festgelegten Nutzerzielen oder Features nicht beachtet, so kann es schnell passieren, dass das Ergebnis sich vom vereinbarten Rahmen entfernt.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

R-4.5 – Suche nach abhängigen Anforderungen nimmt viel Zeit in Anspruch und unterbricht Aktivitäten:

Wie entsteht das Risiko? Sind abhängige Artefakte nicht explizit miteinander verknüpft, so müssen abhängige Elemente zu einem Artefakt manuell aus der Anforderungsmenge herausgesucht werden. Abhängigkeiten können von komplexer Natur sein, beispielsweise wenn erst mehrere User Stories zusammen ein Nutzerziel ermöglichen. Um solche Abhängigkeiten zu entdecken, reicht es nicht aus, die Anforderungen zu überfliegen. Sie müssen auch verstanden wer-

den. Ist die Anforderungsmenge umfangreich, so kann der Suchprozess daher viel Zeit in Anspruch nehmen.

Je länger die Suche dauert, desto eher kann es auch passieren, dass Projektteilnehmer die Bearbeitung abhängiger Elemente auf später verschieben, weil sie ihre derzeitige Aktivität, wie die Priorisierung oder das Review von Anforderungen, nicht unterbrechen wollen.

Warum ist es ein Risiko? Wenn die Aktivität, abhängige Elemente zu identifizieren, als zu schwierig oder zeitintensiv empfunden wird, kann dies dazu führen, dass die Projektteilnehmer es meiden, die Anforderungen aus der Dokumentation herauszusuchen. Stattdessen verlassen sie sich beispielsweise auf ihre Erinnerung oder sprechen Inhalte von Neuem ab. Beides kann zu Fehlern und Inkonsistenzen führen.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

7.6.6 Risikofaktoren für verschiedene Ausprägungen der gewünschten Aktualität eines Artefaktes

Zwar ist es erstrebenswert, dass alle Anforderungsartefakte stets auf dem neuesten Stand der Anforderungen sind. Jedoch erfordert dies auch Aufwand, um bei Änderungen alle Dokumente nachzupflegen. Dieser Aufwand lohnt sich nicht immer. Insbesondere wenn ein Artefakt nur noch sehr selten verwendet wird, kann es sinnvoller sein, die Informationen im Artefakt nur dann zu aktualisieren, wenn sie gerade benötigt werden. Insbesondere kann es passieren, dass Artefakte im Lauf eines Projektes an Relevanz verlieren. Dies ist dann der Fall, wenn die Aktivitäten, die dieses Artefakt unterstützt hat, abgeschlossen sind. Tabelle 14 zeigt Risikofaktoren, die auftreten können, wenn man sich entscheidet, ein Artefakt aktuell zu halten oder nicht.

Tabelle 14: Risikofaktoren für verschiedene Ausprägungen der gewünschten Aktualität eines Artefaktes

Risikofaktor	
<i>Risikofaktoren wenn Artefakttyp stets aktuell gehalten wird</i>	
R-5.1	Hohe Anpassungskosten -> Entwickler vermeiden Dokumentation von Anforderungen
R-5.2	Unterbrechung von Aktivitäten, die Erstellung und Änderung von Artefakten beinhalten
<i>Risikofaktoren wenn Artefakttyp vernachlässigt wird (Lazy Updates)</i>	
R-5.3	Hohe Kosten für Identifikation und Anpassung veralteter Artefakte -> Projektteilnehmer verzichten ganz auf die Verwendung des Artefakttyps
R-5.4	Unterbrechung von Aktivitäten, die Identifikation von Informationen beinhalten

R-5.1 – Hohe Anpassungskosten -> Entwickler vermeiden Dokumentation von Anforderungen:

Wie entsteht das Risiko? Wenn die Artefakte eines Artefakttyps aktuell gehalten werden sollen, so entstehen Anpassungskosten, sobald sich an den darin enthaltenen oder davon abhängigen Anforderungen Änderungen ergeben. Treten insgesamt viele Änderungen auf oder ist es schwierig, eine Änderung in die Sprache eines anderen Artefakttyps zu übersetzen, so müssen die Projektteilnehmer auch einen hohen Aufwand für die Wartung der Artefakte treiben. Insbesondere, wenn die Artefakte häufiger geändert werden müssen als sie gelesen werden, übersteigen die Kosten den Nutzen.

In diesen Fällen besteht das Risiko, dass die Entwickler aufgrund der hohen Kosten, die durch die Wartung aller abhängigen Artefakte entstehen, versuchen, die Dokumentation von Änderungen in einigen Artefakttypen ganz zu meiden.

Warum ist es ein Risiko? Sind die Anforderungsartefakte nicht aktuell, so kann es schnell zu Fehlern kommen, in denen falsche Annahmen über die Software getroffen oder falsche Funktionen implementiert werden. Vor allem in Projekten für langlebige Software sind die Entwickler auf Dokumentation angewiesen, um sich einen Überblick über Teile der Software zu verschaffen. Sind einige Teile der Dokumentation nicht aktuell, so kann es schnell zu Unklarheiten kommen.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

R-5.2 – Unterbrechung von Aktivitäten, die Erstellung und Änderung von Artefakten beinhalten:

Wie entsteht das Risiko? Wenn die Artefakte eines Artefakttyps aktuell gehalten werden sollen, so bedeutet dies, dass bei Änderungen an abhängigen Artefakttypen zusätzlich Anpassungen an diesem Artefakt durchgeführt werden müssen. Entsprechend kann es dazu kommen, dass solche Aktivitäten unterbrochen werden, um die zusätzlichen Anpassungen vorzunehmen. Beispielsweise kann es gewünscht sein, eine Use Case-Menge stets aktuell zu halten. Entsprechend müssen dann bei Änderungen an einer User Story-Menge – wie der Erstellung neuer Stories im Review-Meeting oder der Anpassung bestehender Stories während der Iterationsplanung – solche Aktivitäten unterbrochen werden, um zu überprüfen, ob die Änderungen in Einklang mit den Use Cases sind und um diese anzupassen. Prinzipiell ist es auch möglich, solche Anpassungen auf später zu verschieben. Allerdings besteht dann die Gefahr, dass die Anpassungen vergessen werden oder dass zu spät bemerkt wird, wenn eine Anpassung nicht mit der bestehenden Use Case-Menge zusammenpasst.

Warum ist es ein Risiko? Die Besprechung von Änderungen der Anforderungen können Schlüsselaktivitäten sein. Insbesondere können solche Treffen, wie Review- oder Iterationsplanungstreffen die einzigen Möglichkeiten zur direkten Kommunikation mit Stakeholdern sein. Wenn diese Aktivitäten nicht effizient durchgeführt werden, so kann es zu Verzögerungen aber auch zu allgemeinen Kommunikationsproblemen, wie nicht geklärten Missverständnissen kommen.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

**R-5.3 – Hohe Kosten für Identifikation und Anpassung veralteter Artefakte
-> Projektteilnehmer verzichten ganz auf die Verwendung des Artefakttyps:**

Wie entsteht das Risiko? Wenn nicht klar ist, welche Stellen bzw. Artefakte eines Artefakttyps gerade aktuell und welche veraltet sind, muss potenziell bei jeder Verwendung eines Artefaktes zuerst geprüft werden, ob es aktuell ist. Dies kann zu hohen Kosten führen. Insbesondere, wenn Artefakte häufiger gelesen als geändert werden, übersteigen die Kosten den Nutzen.

Auch hier kann es dazu kommen, dass die Projektteilnehmer die Überprüfung auf aktuellere Inhalten vermeiden oder ganz auf die Verwendung des Artefaktes verzichten, wenn sie den Aufwand für zu hoch halten.

Warum ist es ein Risiko? Ein Verzicht auf die Verwendung eines Anforderungsartefaktes kann dazu führen, dass Entscheidungen, die in der Vergangenheit abgesprochen und dort dokumentiert wurden, nicht mehr berücksichtigt werden. Dies kann zu Fehlern führen oder dazu, dass beispielsweise Einschränkungen erst zu spät während der Entwicklung bemerkt werden.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

R-5.4 – Unterbrechung von Aktivitäten, die Identifikation von Informationen beinhalten:

Wie entsteht das Risiko? In vielen Aktivitäten ist es notwendig, Informationen aus den Anforderungsartefakten heranzuziehen. Vor allem in Wartungsprojekten müssen die unterschiedlichen Blickwinkel der verschiedenen Artefakttypen betrachtet werden, um eine Änderungen planen und auch deren Kosten abschätzen zu können. Aber auch in regulären Projekten können Informationen aus unterschiedlichen Artefakten relevant sein, wenn es beispielsweise darum geht, den Kontext oder das Ziel einer Anforderung zu verstehen, um entsprechende Systemtests zu erstellen oder um Abnahmen durchzuführen. Beispielsweise kann es relevant sein, wie eine externe Änderungen in einen Geschäftsprozess integriert werden kann. Dazu wäre dann aber auch – beispielsweise anhand der implementierten User Stories – zu prüfen, ob der Geschäftsprozess auch so realisiert wurde, wie ursprünglich geplant.

Warum ist es ein Risiko? Die Projektteilnehmer können die oben genannten Aktivitäten nicht effektiv durchführen, da sie viel Zeit mit der Suche nach Informationen verbringen. Als Folge kann es dazu kommen, dass sie darauf verzichten, alle Informationen aus der Dokumentation zu identifizieren und stattdessen nicht alle Abhängigkeiten berücksichtigen und raten. Dies kann wiederum zu Fehlern führen.

Abhängigkeiten zu weiteren Optimierungsdimensionen: keine

8 Werkzeugbasierte Unterstützung anforderungsbezogener Aktivitäten: der IRE-Ansatz

Modellierung (Kapitel 6) sowie Planung und Optimierung des Aufbaus von Anforderungslandschaften (Kapitel 7) sind wichtige erste Schritte zum effektiven Umgang mit Anforderungen und damit zu guter Anforderungskommunikation. Dennoch ist es für eine gute Koexistenz genauso wichtig, auch die Projektteilnehmer in ihren alltäglichen Aufgaben im Umgang mit Anforderungen zu unterstützen. Wie die Interview-Studie (Kapitel 4) gezeigt hat, verursacht die Koexistenz von Anforderungsartefakten Abhängigkeiten, die den alltäglichen Umgang mit Anforderungen erschweren. Auch wenn sich durch gute Planung die Zahl der Abhängigkeiten reduzieren lässt, so sind sie dennoch in vielen Fällen unvermeidbar, da es wichtiger ist, mehrere Repräsentationen der Anforderungen, also mehrere Artefakttypen, zu besitzen. Besonders in hybriden Anforderungslandschaften treffen mehrere Artefakttypen aufeinander, die Abhängigkeiten besitzen, weil sie sich auf dieselben Anforderungen beziehen.

Inkonsistenzen sowie die Verteilung von Informationen auf mehrere Artefakte, führen dazu, dass viele *Aktivitäten im täglichen Umgang mit Anforderungen*, wie die Klärung von Anforderungsdetails oder die Priorisierung oder Änderung von Anforderungen, länger dauern und fehleranfälliger sind. Es ist wichtig, auch diesen täglichen Umgang mit den konkreten Anforderungen so gut wie möglich zu unterstützen, um eine effektive und effiziente Anforderungskommunikation zu schaffen.

Abbildung 40 zeigt ein Beispiel zur Veranschaulichung. Werden im Projekt zum Beispiel User Stories für die Steuerung der Implementierung und Use Cases für den Gesamtüberblick der gewünschten Funktionalität verwendet, so führen Abhängigkeiten zwischen User Stories und Use Cases zu zusätzlichen Kosten. Die Projektteilnehmer pflegen Änderungen mithilfe von User Stories ein, müssen jedoch bei manchen – aber auch nicht allen – Änderungen auch die Use Cases anpassen. Dazu müssen sie im richtigen Moment an die abhängigen Use Cases denken, die richtigen Stellen in den Use Cases identifizieren und sie korrekt anpassen. Tun die Entwickler das alles beispielsweise im Review-Meeting, während sie gerade Feedback zur Iteration mit dem Kunden diskutieren, so wird das gesamte Meeting verzögert. Es ist wichtig, diese Aktivitäten zu unterstützen. Beispielsweise kann hier ein System bei der Änderung einer User Story, die mit Use Cases verknüpft ist, die Projektteilnehmer direkt auf die Abhängigkeiten hinweisen. Es kann ihnen direkt alle relevanten Use Case-Stellen anzeigen, um die Suche zu beschleunigen. Es kann ihnen erlauben, eine Verknüpfung als inkonsistent zu markieren, um eine Anpassung auf später zu vertagen, sie aber nicht aus den Augen zu verlieren.

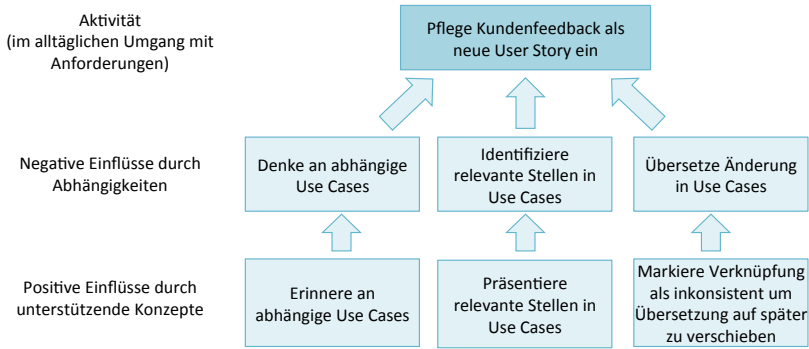


Abbildung 40: Beispiel für Aktivitäten im alltäglichen Umgang mit Anforderungen

Mithilfe zusätzlicher Hilfs-Operationen – im Verlauf der Arbeit *Trace-Operationen* genannt – können Aktivitäten, wie die obigen, effizienter durchgeführt werden. Weiterhin helfen spezielle integrierte Sichten dabei, die Aktivitäten zu vereinfachen, indem zusammenhängende Artefakttypen gemeinsam angezeigt und so Kontextwechsel vermieden werden. Letztendlich führt die Unterstützung der einzelnen Aktivitäten zu einem effektiveren und effizienteren Umgang mit Anforderungen. Der IRE-Ansatz (*Integrated Requirements Environment-Ansatz*) bündelt diese verschiedenen Mechanismen für Werkzeugunterstützung, die den Umgang mit Abhängigkeiten verbessern. In Abschnitt 8.2.2 wird erläutert, wie der IRE-Ansatz zu den bisherigen Ansätzen steht, die auf Anforderungslandschaften basieren.

8.1 Leitprinzipien für die Lösungserstellung

Die vorliegende Lösung baut auf drei Leitprinzipien auf, die im Folgenden erläutert werden. Das Ziel ist, die Projektteilnehmer im täglichen Umgang mit Anforderungsartefakten und deren Abhängigkeiten zu unterstützen. Zur Konkretisierung einer Lösung, die dieses Ziel erreicht, werden die folgenden drei Prinzipien herangezogen: die Unterstützung erfolgt werkzeuggestützt, die Unterstützung fokussiert sich auf anforderungsbezogene Aktivitäten (statt Tracing-Aktivitäten) und die Unterstützung ist für jedes Projekt individuell einstellbar.

Unterstützung der alltäglichen Arbeit mithilfe von CASE-Tools.

Zur Unterstützung des täglichen Umgangs mit Anforderungsartefakten, die Abhängigkeiten aufweisen, bedarf es der Verwendung von Werkzeugen. Die Menge an Anforderungen und die Komplexität, die durch Abhängigkeiten hervorgerufen wird, sind in der Regel nur schwierig manuell handhabbar.

Die Softwareentwicklung wird an vielen Stellen durch Werkzeuge unterstützt. Sogenannte CASE-Tools (*Computer Aided Software Engineering Tools*) helfen,

Praktiken und Prozesse umzusetzen, Code und Designs zu analysieren, die Kommunikation im Team zu stärken und Arbeit durch Automatisierung zu vereinfachen. CASE-Tools werden für viele Bereiche, wie Versionskontrolle, Testautomatisierung, Programmanalyse, Dokumentation, Projektmanagement sowie dem Änderungsmanagement und der Methodenunterstützung, die hier relevant sind, eingesetzt ([151], S. 116). Für die Verwaltung von Anforderungen gibt es eine Schar spezialisierter Tools, der *Requirements Management Tools* (RM/RE-Tools) oder auch *Application Lifecycle Management Tools* (ALM-Tools).

Für eine effektive Unterstützung des alltäglichen Umgangs mit Anforderungen und ihrer Abhängigkeiten wird ein Werkzeug benötigt, das die folgenden zwei Anforderungen erfüllt:

1. Das Werkzeug erlaubt, einzelne Anforderungsartefakte zu verwalten.
Können einzelne Anforderungen nicht als einzelne Elemente verwaltet werden, so ist es beispielsweise nicht möglich, Änderungen an einzelnen Artefakten (automatisiert) festzustellen und an diesen Stellen unterstützend einzugreifen.
2. Das Werkzeug repräsentiert Abhängigkeiten durch Verknüpfungen und kann auf die Verknüpfungen zugreifen.
Sind Abhängigkeiten nicht durch Link-Objekte oder zumindest Referenzen auf andere Elemente dargestellt, so kann ein Werkzeug nicht automatisiert feststellen, welche Elemente mit einem anderen Element verknüpft sind und diese Information automatisiert an den Nutzer weitergeben oder sie für weitere Auswertungen, wie Konsistenzanalysen, nutzen.

Für die folgenden Konzepte wird davon ausgegangen, dass Projektanforderungen mit einem Werkzeug, das die beiden oben genannten Anforderungen erfüllt, verwaltet werden. Die meisten Requirements Management-Tools, wie DOORS, JIRA, HP ALM, Enterprise Architect [71], [7], [70], [152], verfügen bereits über die notwendige Funktionalität. Die vorgestellten Konzepte sind dabei nicht abhängig von einem einzelnen Tool. Vielmehr werden allgemeine Konzepte erarbeitet, die in verschiedene CASE- bzw. RE-Tools integriert werden können.

In vielen Softwareprojekten werden statt spezialisierter Requirements Engineering-Werkzeuge, nur generische Dokumente (Word- oder Excel-Dokumente) oder Papierkarten (User Stories) verwendet. Für solche Anforderungsumgebungen sind die folgenden Konzepte zwar prinzipiell anwendbar, indem die beschriebenen Operationen manuell durchgeführt werden, jedoch aufgrund des hohen Aufwandes nicht praktikabel.

Fokus auf Unterstützung anforderungsbezogener Aktivitäten statt Tracing-Aktivitäten.

Zur Unterstützung von Abhängigkeiten und Verknüpfungen zwischen Anforderungsartefakten gibt es bereits viele Ansätze im Bereich des Tracings. Beispielsweise existieren viele Ansätze zur automatisierten Erstellung oder Wartung von Verknüpfungen zwischen abhängigen Elementen [25], [36], [43], [65], [107], [116], [147]. Die aktive Abfrage bestimmter miteinander verknüpfter Artefakte (sog. *Trace-Queries*) kann durch Query-Sprachen [109], [146] oder Natural-Language-Interpreter [131] vereinfacht werden. Potenzielle Inkonsistenzen können automatisiert entdeckt, angezeigt und teilweise korrigiert werden [165]. Diese Ansätze sind besonders in solchen Projekten hilfreich, in denen Traceability einen hohen Stellenwert hat. Beispielsweise sind das besonders komplexe oder kritische Projekte oder solche, die aufgrund von Regulationen, Normen oder Reifegradmodell-Vorgaben vollständige Traceability gewährleisten müssen.

Die Tracing-Konzepte sind jedoch darauf fokussiert, explizite **Tracing-Aktivitäten** zu unterstützen. Damit sind Aktivitäten gemeint, bei denen das Tracing selbst im Vordergrund steht. Beispielsweise sind das Aktivitäten, die auf die Herstellung guter Traceability im Projekt ausgelegt sind, wie das Finden aller nicht verknüpften Artefakte oder die Suche und Verknüpfung von verwandten Anforderungen. Weiterhin zählen Aktivitäten dazu, die hauptsächlich das Folgen eines Traces als Ziel haben, wie das Finden aller User Stories, deren Akzeptanztests fehlschlagen.

In der Softwareentwicklung treten jedoch viele anforderungsbezogene Aktivitäten auf, bei denen nicht das Tracing im Vordergrund steht. Um mit den Abhängigkeiten umgehen zu können, wären Methoden des Tracing hilfreich – primär haben die Projektteilnehmer jedoch ein anderes Ziel. Beispielsweise wollen Entwickler als primäres Ziel schnell Feedback des Kunden auf einer User Story festhalten. Zu überprüfen, ob sie auch das Geschäftsprozessmodell anpassen müssen, ist in dieser Situation sekundär.

In Abschnitt 6.8.1 wurden zur Abgrenzung zwei Arten von Aktivitäten eingeführt, die *anforderungsbezogenen Aktivitäten* und die *Koexistenz-Aktivitäten*. Dabei beinhalten anforderungsbezogene Aktivitäten hauptsächlich die Handhabung – also Erstellung, Anpassung oder Wartung – von Anforderungen. Koexistenz-Aktivitäten sind Aktivitäten, die zusätzlich durchgeführt werden müssen, um Abhängigkeiten zwischen Anforderungen zu berücksichtigen. Im Beispiel zu Beginn dieses Abschnittes wäre das Einarbeiten von Kundenfeedback eine anforderungsbezogene Aktivität, die durch die Koexistenz-Aktivitäten *Identifikation* und *Anpassung abhängiger Use Case-Stellen* komplettiert werden muss. Anforderungsbezogene Aktivitäten sind damit genau die primären Aktivitäten, die Projektteilnehmer eigentlich durchführen wollen. Koexistenz-Aktivitäten

sind die sekundären Aktivitäten, welche Projektteilnehmer stören und oft vernachlässigt werden.

Die Lösungsidee, die durch das hier vorgestellte Konzept verkörpert wird, liegt nun darin, Methodenunterstützung auf *anforderungsbezogene Aktivitäten zu fokussieren* und die Durchführung von *Koexistenz-Aktivitäten* weitestgehend zu *vereinfachen*.

Individuelle Festlegung der Unterstützung für relevante anforderungsbezogene Aktivitäten.

Welche anforderungsbezogenen Aktivitäten innerhalb eines Projektes durchgeführt werden und welcher Unterstützung diese bedürfen, ist von Projekt zu Projekt und sogar von Person zu Person unterschiedlich [166]. In umfangreichen Projekten ist es wichtiger, zu einer Anforderung abhängige Elemente hervorzuheben, als in simplen Projekten, wo die Projektteilnehmer die Abhängigkeiten überblicken. Genauso haben einige Personen alle Abhängigkeiten im Kopf und finden es störend, wenn das System sie auffordert, die Anforderungen nochmals nach abhängigen Elementen zu überprüfen, während andere Personen es schwierig finden, abhängige Elemente in der umfangreichen Anforderungsmenge zu identifizieren.

Entsprechend können nur einige Hilfen in dieser Arbeit allgemein angegeben werden, weil sie für alle Abhängigkeiten zutreffen. Andere müssen für jedes Projekt individuell konfiguriert werden, da sie vom Prozess und der Anforderungslandschaft abhängen. Dies ist wieder eine Aufgabe, die vom Requirements Methodiker durchgeführt wird, da sie Teil der im Projekt angewendeten Requirements Engineering-Methode ist.

In Anlehnung an die bisherigen Ansätze dieser Arbeit, erhält der Requirements Methodiker so die Möglichkeit, nach der Festlegung einer Anforderungslandschaft, nun weiter die Methode zu stärken, indem er auf die konkreten anforderungsbezogenen Aktivitäten in dieser konkreten Landschaft eingeht. Dies wird in Abschnitt 8.2.2 näher erläutert.

8.2 Überblick über den IRE-Ansatz

Der IRE-Ansatz unterstützt anforderungsbezogene Aktivitäten und vereinfacht Koexistenz-Aktivitäten. Er setzt sich aus mehreren Teilkonzepten zusammen, die in ein CASE-Tool integriert werden. Im Folgenden werden die Teilkonzepte vorgestellt und in Zusammenhang mit anforderungsbezogenen Aktivitäten und Koexistenz-Aktivitäten gesetzt. Abbildung 41 zeigt einen Überblick über die Teilkonzepte. Zu jedem Teilkonzept wird anschließend vorgestellt, in welchen Situationen es benötigt wird (*Bedarf*) und wie das Teilkonzept in diesen Situationen helfen kann (*Lösung*).

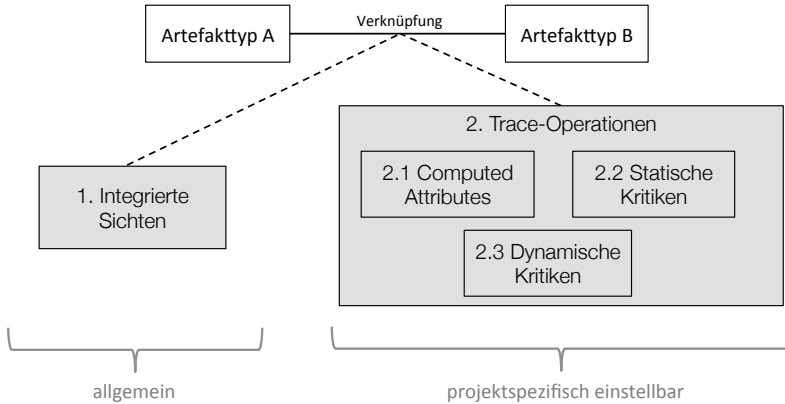


Abbildung 41: Teilkonzepte für CASE-Tool-Unterstützung von Koexistenz-Aktivitäten

8.2.1 Teilkonzepte des IRE-Ansatzes

1. Integrierte Sichten

Bedarf: Das Erkennen von Abhängigkeiten ist unter anderem deswegen schwierig, weil die unterschiedlichen Artefakttypen häufig in verschiedenen einzelnen Sichten dargestellt werden. Wenn Projektteilnehmer zu einem Artefakt abhängige oder verknüpfte Artefakte eines anderen Typs betrachten wollen, müssen sie häufig eine zweite Ansicht öffnen, die unabhängig von der ersten ist. Die so entstehenden Kontextwechsel stören den Ablauf der meisten Koexistenz-Aktivitäten. Verknüpfungen sind außerdem nur indirekt sichtbar, indem bei Öffnen eines Links das verknüpfte Artefakt geöffnet wird. Ein Überblick über alle Artefakte, die mit einem bestimmten Artefakt verknüpft sind, ist so nur schwierig zu bekommen.

Lösung: Integrierte Sichten können die Sichten zweier (oder mehrerer) Artefakttypen vereinen, indem sie sie nebeneinander in derselben Sicht darstellen. So können Projektteilnehmer immer parallel Artefakte beider Typen sehen und so schneller Abhängigkeiten auswerten oder Anpassungen vornehmen. Über spezielle Links ist es möglich, zu einem Artefakt direkt parallel verknüpfte Artefakte anzuzeigen. Sind beide Artefakttypen in einer Sicht vereint, können Links zusätzlich visualisiert werden, indem alle verknüpften Elemente optisch hervorgehoben werden oder indem Links sogar als explizite Linien zwischen Elementen dargestellt werden.

2. Trace-Operationen

Wann immer Projektteilnehmer mit Artefakten umgehen, die potenziell Abhängigkeiten besitzen, müssen sie diese Abhängigkeiten berücksichtigen. Überhaupt im richtigen Moment an die abhängigen Elemente zu denken und die Koexistenz-Aktivitäten durchzuführen, ist (laut Interview-Studie, Abschnitt

4.4.6) oftmals bereits problematisch, da die Projektteilnehmer in dem Moment oft nur auf die eigentliche primäre Aktivität fokussiert sind.

Mithilfe von Verknüpfungen (also *Traces*) können verschiedene Aspekte der Koexistenz-Aktivitäten teil-automatisiert werden. Ein System kann anhand von Verknüpfungen feststellen, wann überhaupt Koexistenz-Aktivitäten notwendig sind und Teile davon, wie die Identifikation und Präsentation von abhängigen Elementen, bereits übernehmen. Das System kann eine Reihe solcher Operationen mithilfe von *Traces* durchführen, die im Folgenden erläutert sind. Zusammenfassend wird das Teilkonzept *Trace-Operationen* genannt, weil alle Operationen die automatisierte Verwendung und Auswertung von *Traces* innehaben.

Die Automatisierung der Übersetzung selbst, bei der Informationen aus einem Artefakt automatisiert auf Artefakte eines anderen Typs übertragen werden, ist jedoch nicht im Rahmen dieser Arbeit enthalten. Es werden die Schritte zur Vorbereitung der Übersetzung (z.B. Identifikation der abhängigen Elemente) automatisiert, während die Projektteilnehmer manuell entscheiden müssen, ob sie tatsächlich etwas an den abhängigen Elementen ändern müssen. Die Übersetzung ist durchaus eine weitere mögliche Art von *Trace-Operation*, die in der Zukunft erforscht und integriert werden kann. Jedoch verlangt die Automatisierung der Übersetzung einen hohen Grad an Formalität der Anforderungen, beispielsweise indem Anforderungen spezielle Attribute erhalten, einer Satzschablone (wie von Rupp in [141] vorgeschlagen) folgen oder indem Modellelemente nach einem festen Schema benannt werden. Da sich diese Arbeit auf beliebige Arten von Anforderungen bezieht, überschreitet die automatisierte Übersetzung den Rahmen dieser Arbeit.

2.a Computed Attributes

Bedarf: Attribute, die in einem Artefakttyp vorhanden sind und dort manipuliert werden, können auch für andere Artefakttypen interessant sein. Beispielsweise kann die Aufwandsschätzung anhand von User Stories durchgeführt werden. Dennoch ist es auch interessant, den geschätzten Gesamtaufwand eines Use Cases anhand der damit verknüpften Stories zu bestimmen und so ggf. festzustellen, dass ein Use Case voraussichtlich aufwändiger sein wird, als in einer groben initialen Planung angenommen. Normalerweise müssen Projektteilnehmer dazu verknüpfte Artefakte erst öffnen und die Attribute mehrerer Artefakte aggregieren, was ebenfalls Aufwand erzeugt.

Lösung: Dieser Aufwand kann gespart werden, indem solche Attribute automatisiert aus verknüpften Artefakten berechnet und direkt angezeigt werden. Im obigen Beispiel könnte so der Requirements Methodiker für Use Cases ein zusätzliches Attribut *summierter Aufwand* definieren, das sich als Summe der Aufwände der verknüpften Stories berechnet. Die Darstellung direkt im verknüpften Artefakt (Use Case) erlaubt es, direkt mit den Werten zu arbeiten, ohne unter Zusatzaufwand eigens dafür in den Kontext des anderen Artefakttyps

(User Story) wechseln zu müssen. Die Anzeige kann dauerhaft erfolgen oder bei Bedarf über eine Schaltfläche angestoßen werden.

Tabelle 15: Beispiel-Regeln für Computed Attributes

ID	Kurzbezeichnung	Berechnung
CA1	Verknüpfte Stories	Zeige am Use Case-Schritt Anzahl der verknüpften User Stories an
CA2	Verknüpfte Stories	Zeige am Use Case-Schritt per Schnellschaltfläche eine Liste mit Links zu allen verknüpften User Stories an
CA3	Aufwandsumme	Zeige am Use Case die Summe der Aufwandschätzungen aller (disjunkten) verknüpften User Stories an
CA4	Früheste Fertigstellung	Zeige am Use Case die späteste Iteration, der eine verknüpfte User Story zugeordnet ist (inkl. Backlog)
CA5	Ausführbarer Use Case	Markiere Use Case, wenn für alle Schritte mindestens eine User Story fertig implementiert ist, der Ablauf also potenziell ausführbar ist

2.b Statische Kritiken für die Unterstützung von Richtlinien

Bedarf: Für einige Abhängigkeitsrelationen ist es sinnvoll, über Grundregeln bzw. Richtlinien festzulegen, welche grundsätzlichen Eigenschaften diese haben sollen. Beispielsweise kann es gewünscht sein, zu jeder User Story mindestens ein Nutzerziel zuzuordnen, um sicherzustellen, dass nicht User Stories aufgenommen werden, die keinem der vorgesehenen Nutzerziele dienen.

Auch hier müssen die Projektteilnehmer zunächst im richtigen Moment an die Richtlinien denken, sodass sie beispielsweise beim Anlegen einer neuen User Story nicht vergessen, diese mit einem passenden Nutzerziel zu verknüpfen. Wenn es kein passendes Nutzerziel gibt, müssen die Projektteilnehmer möglicherweise diskutieren, was das entsprechende Nutzerziel wäre und ob dieses Ziel relevant ist – sodass ein neues Nutzerziel aufgenommen wird – oder irrelevant ist – sodass die User Story wieder gelöscht werden sollte. Oft wollen die Projektteilnehmer in solchen Situationen nur schnell Informationen festhalten sodass sie beim Anlegen neuer Elemente möglicherweise gar keine Zeit haben, diese mit abhängigen Elementen zu verknüpfen.

Lösung: Der Aufwand für die Sicherstellung von Richtlinien kann reduziert werden, indem der Requirements Methodiker Richtlinien mittels *statischer Kritiken* festlegt, die dann vom System teilautomatisiert durchgeführt werden. Statische Kritiken wenden im Grunde heuristische Regeln auf *Computed Attributes* (vorige Trace-Operation) an. Das bedeutet, dass auch hier Informationen über verknüpfte Artefakte eingesammelt und ggf. aggregiert werden, dann aber das Ergebnis zusätzlich ausgewertet wird, um zu überprüfen, ob eine Guideline eingehalten wird. Ist dies nicht der Fall, wird eine Kritik in Form einer Warnung ausgegeben. Beispielsweise kann eine statische Kritik eine Warnung auslösen,

wenn die Anzahl der mit einer User Story verknüpften Nutzerziele gleich null ist.

Neben der Festlegung der einzelnen relevanten Richtlinien muss der Requirements Methodiker auch entscheiden, wie wichtig die Einhaltung einer konkreten Guideline ist. Basierend darauf legt er fest, wie stark das System bei den Nutzern die Einhaltung einer Guideline forcieren soll. Beispielsweise kann das System nur einen Hinweis einblenden oder im Extremfall weitere Aktionen stoppen, bis das Problem behoben ist.

Da zur Überprüfung nur der Zustand der Anforderungs- und Verknüpfungsmenge herangezogen wird, ist die Überprüfung jederzeit möglich und wiederholbar. Damit bleiben Probleme auch nach ihrem Auftreten weiterhin sichtbar und können nachträglich leicht wiedergefunden werden. Diese Art der Überprüfung wird daher *statisch* genannt.

Tabelle 16: Beispiel-Regeln für Statische Kritiken

ID	Kurzbezeichnung	Berechnung
StK1	Use Case-Schritt ohne Implementierung	Warne im Use Case-Schritt, wenn ein Element keine direkt verknüpften User Stories besitzt
StK2	Use Case-Schritt ohne Testfälle	Weise hin/ biete passiv an, wenn ein Element keine (indirekt) verknüpften Testfälle besitzt
StK3	Unterschätzter Use Case	Warne, wenn Use-Case-Schätzung geringer als die Summe der verknüpften Story Card-Schätzungen ist
StK4	Use-Case-Schritt fortsetzen	Warne, wenn Use Case-Schritt nicht fertig ist, aber die Story Cards schon. (Evtl. muss neue Story (als <i>vertical Slice</i>) angelegt werden.)
StK5	Wertbasierte Story-Verknüpfungen	Warne in der Story, wenn sie keinen verknüpften Use Case-Schritt besitzt, aber nur wenn die Story eine Priorität höher als 2 besitzt

2.c Dynamische Kritiken für die Unterstützung von aktionsbezogenen Richtlinien

Bedarf: Einige sinnvolle Richtlinien im Umgang mit Anforderungsartefakten beziehen sich nicht auf den Zustand der Anforderungen, sondern auf durchgeführte Aktionen, wie dem Ändern oder Löschen eines Artefaktes. Wird beispielsweise ein Anforderungsartefakt, wie eine User Story, geändert, so sollte stets auch geprüft werden, ob es nach wie vor zu abhängigen Artefakten, wie Nutzerzielen, passt oder ob eine Inkonsistenz entstanden ist. Auch hier müssen die Projektteilnehmer im richtigen Moment – beispielsweise direkt bei der Änderung – daran denken, die Überprüfung durchzuführen. Für die Überprüfung müssen sie die abhängigen Artefakte identifizieren, sie durchgehen und die Inhalte abgleichen.

Bei aktionsbezogenen Richtlinien gilt die *Aktion*, also das Löschen oder Ändern eines Artefaktes, als Hinweis auf ein potenzielles Problem. Es gibt jedoch keinen leicht auswertbaren Indikator am *Zustand* der Artefakte, der als Hinweis herangezogen werden kann, wie es bei den statischen Kritiken der Fall war. Für eine Feststellung von Inkonsistenzen in beliebigen Arten von Anforderungen ist eine umfangreiche inhaltliche Auswertung notwendig, die manuell durchgeführt werden muss.

Nachdem die Aktion vorbei ist, gibt es damit nachträglich keine guten Hinweise mehr auf die potenziellen Probleme. Diese können dann nur noch festgestellt werden, indem die Anforderungen explizit nach Inkonsistenzen durchsucht werden, was sehr aufwändig ist.

Tabelle 17: Beispiel-Regeln für Dynamische Kritiken

ID	Kurzbezeichnung	Berechnung
DyK1	Potenzielle Inkonsistenz durch Änderung des Elementes	Wenn User Story geändert wird und verknüpfte Use Case-Schritte besitzt, zeige Hinweis über mögliche Inkonsistenz (und die verknüpften Elemente)
DyK2	Potenzielle Inkonsistenz durch Löschen des Elementes	Wenn Use Case-Schritt gelöscht wird und verknüpfte User Stories besitzt, die nicht bereits erledigt sind, zeige Hinweis über mögliche Inkonsistenz (und die verknüpften Elemente)
DyK3	Potenzielle Inkonsistenz durch Verschieben eines Elementes	Wird eine Aktivität, die verknüpfte Use Case-Schritte besitzt, im Geschäftsprozessmodell verschoben, zeige Warnung, dass das Modell evtl. nicht mehr konsistent ist mit dem Use Case.
DyK4	Potenzielle Inkonsistenz durch geändertes Attribut	Wird die Priorität einer nichtfunktionalen Anforderung geändert, zeige Warnung, dass verknüpfte Use Cases evtl. nicht mehr konsistent mit der Priorität der nichtfunktionalen Anforderung sind.

Lösung: Ähnlich wie statische Kritiken, können für die aktionsbezogenen Richtlinien Warnungen eingesetzt werden, die Verknüpfungen bereits im Vorfeld automatisiert auswerten. Diese sogenannten *dynamische Kritiken* werden durch Nutzeraktionen ausgelöst, und können Projektteilnehmer so im richtigen Moment an potenzielle Probleme erinnern. Verknüpfte Elemente werden direkt zusammen mit der Kritik angezeigt, um auch die Suche nach diesen verknüpften Elementen zu beschleunigen. Außerdem kann die Ergebnismenge der Kritik, also die verknüpften Elemente, als *potenziell inkonsistent* markiert werden, um die Überprüfung der potenziell problematischen Stellen auf später zu verschieben.

8.2.2 Zusammenhang mit dem Konzept der Anforderungslandschaften

Die Konzepte der Trace-Operationen und der Anforderungslandschaften können unabhängig von einander angewendet werden. Werden sie jedoch zusammen verwendet, so ergänzen sie sich gegenseitig. Der IRE-Ansatz stellt Mechanismen bereit, die auf einen Verknüpfungstyp angewendet werden und diesem zusätzliche Eigenschaften geben. Zu einem Verknüpfungstyp wird so festgelegt, welche Anwendungen er erlaubt und welche Richtlinien er – durch statische und dynamische Kritiken – unterstützt. Das stellt eine weitere Detaillierung des Anforderungslandschaftsmodells dar, wie in Abbildung 42 angedeutet. Gleichzeitig bieten Anforderungslandschaften einen Gesamtüberblick über die Artefakte im Projekt und helfen damit auch, zu beurteilen, welche Trace-Operationen für einen Verknüpfungstyp angemessen sind.

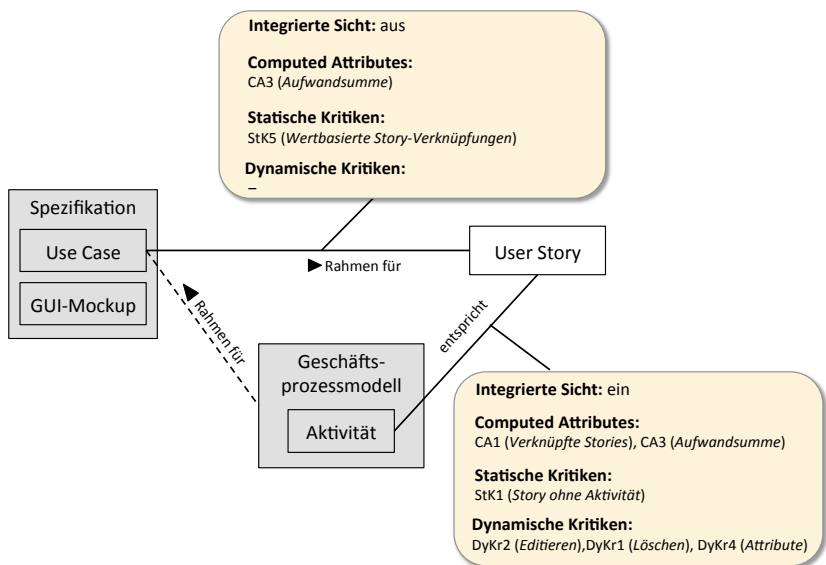


Abbildung 42: Zusammenhang zwischen IRE-Teilkonzepten und Anforderungslandschaften

Die neue Detaillierungsebene, auf welcher der IRE-Ansatz wirkt, bezieht sich auf die *Verwendung* der Anforderungen. Für Requirements Methodiker schlägt dies eine Brücke zwischen der statischen Sicht auf die Artefakte und der dynamischen Sicht auf die Aktivitäten, Prozess-Richtlinien und Informationsflüsse in einem Projekt. Der Zusammenhang erlaubt auch genau die gewünschte Integration von Tracing-Techniken in anforderungsbezogene Aktivitäten.

In Zukunft kann dieser Zusammenhang zwischen statischer Artefaktsicht und dynamischer Verwendungssicht noch weiter ausgebaut werden. Beispielsweise können Verknüpfungstypen einer Anforderungslandschaft neue Attribute, wie

eine gewünschte „Verknüpfungsstärke“ erhalten. Dann könnten die Mechanismen des IRE-Ansatzes automatisch basierend auf der gewünschten Verknüpfungsstärke konfiguriert werden. Beispielsweise könnten zu Verknüpfungen der Stärke 4 (sehr stark) automatisch dynamische Kritiken hinzugefügt werden, die bei jedem Änderungs- und Löschvorgang den Nutzer auffordern, abhängige Anforderungen zu prüfen. Für Verknüpfungen der Stärke 2 (mittel) werden hingegen nur Kritiken angezeigt, wenn das Ausgangsartefakt eine hohe Priorität besitzt, was dem Ansatz des Value-based Tracing entspricht [44], [68]. Die Verbindung von Eigenschaften der Anforderungslandschaft und Trace-Operationen erleichtert wiederum die Arbeit des Requirements Methodikers beim Festlegen der gewünschten Richtlinien und derer Unterstützung. Abbildung 43 zeigt ein Beispiel dazu, wie die Planung der Anforderungslandschaft inklusive der IRE-Mechanismen dann aussehen kann.

Auch anders herum kann man den Umgang mit den Anforderungen im RE-Tool beobachten – beispielsweise mittels neuartiger Trace-Operationen – und diesen in der Anforderungslandschaft darstellen. Werden beispielsweise die Artefakte zweier nicht verknüpfter Artefakttypen immer wieder gemeinsam im Werkzeug verwendet, kann man die Abhängigkeit als besonders kritische Abhängigkeit in der Anforderungslandschaft markieren und so die Aufmerksamkeit des Requirements Methodikers und der Projektteilnehmer auf diese Abhängigkeit richten.

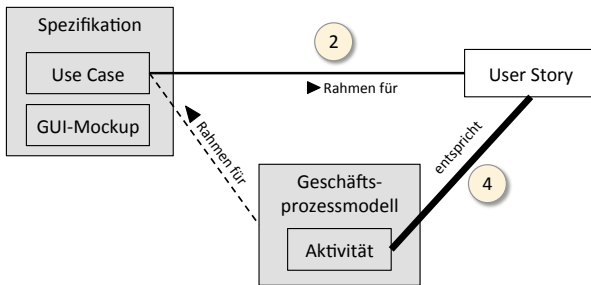


Abbildung 43: Automatisierte Konfiguration des IRE-Konzeptes über Verknüpfungsstärke

8.3 Integrierte Sichten

Die Anzeige von Anforderungsartefakten im RE-Tool beeinflusst, wie gut darin Abhängigkeiten und Verknüpfungen zu anderen Artefakten sichtbar und verwendbar sind. Häufig zeigen RE-Tools in einer Ansicht genau einen Artefakttyp, wie zum Beispiel nur Use Cases. Dies macht es jedoch schwierig, Abhängigkeiten zu anderen Artefakttypen, wie zum Beispiel User Stories, zu sehen und Verknüpfungen zu nutzen. Sollen Nutzer leicht erkennen können, welche Artefakte mit welchen anderen Artefakten eines anderen Typs zusammenhängen, so sollten auch beide Artefakttypen gemeinsam angezeigt werden können. Abbildung 44 und Abbildung 45 veranschaulichen den Unterschied zwischen isolierten Sichten und integrierten Sichten.

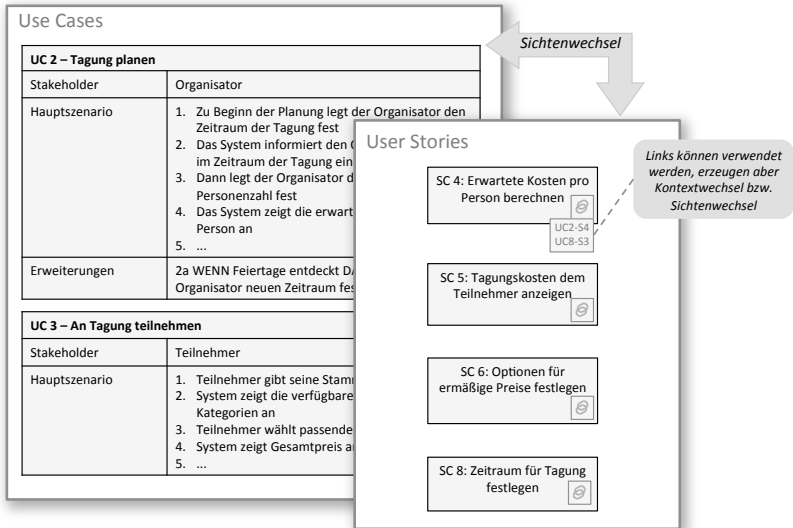


Abbildung 44: Zwei isolierte Sichten für Use Cases und User Stories

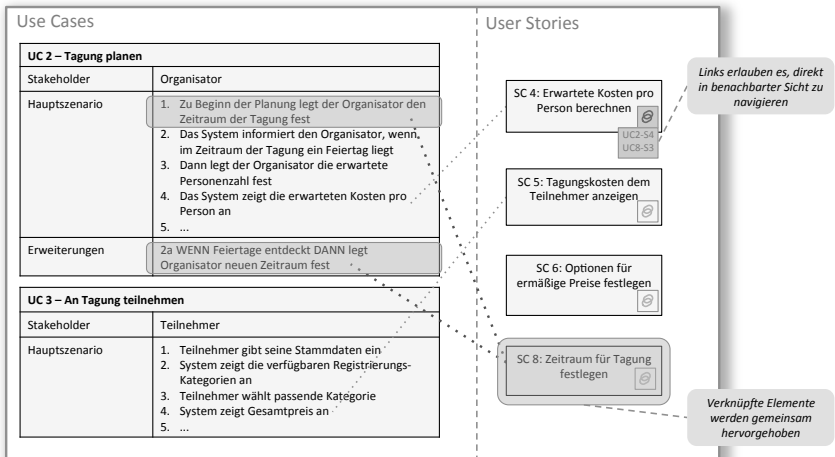


Abbildung 45: Eine integrierte Sicht für Use Cases und User Stories

Wenn ein Artefakttyp, wie eine Story Card-Menge, isoliert dargestellt wird, so müssen die Nutzer immer wieder manuell eine neue Sicht öffnen, um zugehörige andere Artefakte, wie beispielsweise die Use Case-Menge, zu untersuchen. Sie müssen dazu den Kontext wechseln und immer wieder die einzelnen Elemente abgleichen, um die Verknüpfungen zu erkennen. Will ein Nutzer am

Beispiel der obigen Abbildung 44 entscheiden, ob User Story *SC 4* oder User Story *SC 8* in die nächste Iteration aufgenommen werden soll, so müsste er zuerst zu *SC 4* manuell die passenden Stellen in der Use Case-Menge nachschlagen und dann genauso für *SC 8* die passenden Stellen in der Use Case-Menge herausuchen. Erst dann würde er erkennen, dass *SC 8* in einem der Use Case-Abläufe in der Reihenfolge vor *SC 4* vorkommt und daraufhin beispielsweise entscheiden, *SC 8* zuerst zu implementieren.

Integrierte Sichten, wie in Abbildung 45 dargestellt, zeigen hingegen die verschiedenen Artefakte in einer Sicht an. Werden zwei Artefakttypen in einer Sicht vereint, so wird vor allem eine neuartige Navigation ermöglicht. Durch Auswählen eines Elementes auf der einen Seite wird es möglich, direkt zu zugehörigen Elementen auf der anderen Seite zu navigieren. So kann der Nutzer während der Arbeit mit Artefakten des einen Typs direkt nachvollziehen, welche Abhängigkeiten es dazu im anderen Artefakttyp gibt. Dies beschleunigt Koexistenz-Aktivitäten und damit auch anforderungsbezogene Aktivitäten. Der Nutzer kann nun beispielsweise beim Priorisieren der User Stories direkt die Abhängigkeiten zu zeitlich vorgelagerter Funktionalität anhand der Use Case-Abläufe erkennen und seine Priorisierungs-Aufgabe so schneller erledigen.

Prinzipiell wird die Sicht bei Integration von mehr als zwei Teilsichten schnell unübersichtlich. Eine flexible Kombination, bei der ein Nutzer die jeweiligen beiden anzuzeigenden Artefakttypen auswählt, ist dann notwendig. Ein weiteres hilfreiches Konzept ist die Verwendung von Visualisierungen, die spezielle Zoom-Mechanismen anbieten. Beispielsweise stellt Ghazi [54], [55] eine Visualisierung verknüpfter Artefakte vor, bei der aktuell relevante Inhalte fokussiert und detailliert dargestellt werden, während die anderen Inhalte verkleinert und an den Rand der Gesamtsicht gedrückt werden, so aber noch im Sichtfeld des Nutzers bleiben. Dieses Konzept kann helfen, alle Teilsichten auf einmal zu integrieren und dem Nutzer die relevanten Informationen übersichtlich anzuzeigen.

Ein weiteres Prinzip, das hilft, Artefakttypen besser zu integrieren, besteht darin, Informationen des einen Artefakttyps, die häufig in Zusammenhang mit einem zweiten Artefakttyp verwendet werden, dort auch direkt – als echten Teil des zweiten Artefakttyps – anzuzeigen. Die Integration von Informationen direkt in abhängige Artefakte fördert ihre Verständlichkeit und Handhabbarkeit [100]. Wie Abbildung 46 zeigt, können beispielsweise in einer User Story direkt die Stakeholder der Use Cases, an denen die User Story beteiligt ist, angezeigt werden. Diese Information kann dann bei der Priorisierung von User Stories herangezogen werden, ohne dass zu jeder User Story immer wieder die verknüpften Use Cases geprüft werden müssen und ohne dass die Stakeholder direkt in der User Story gespeichert werden, was anfälliger für Inkonsistenzen wäre.

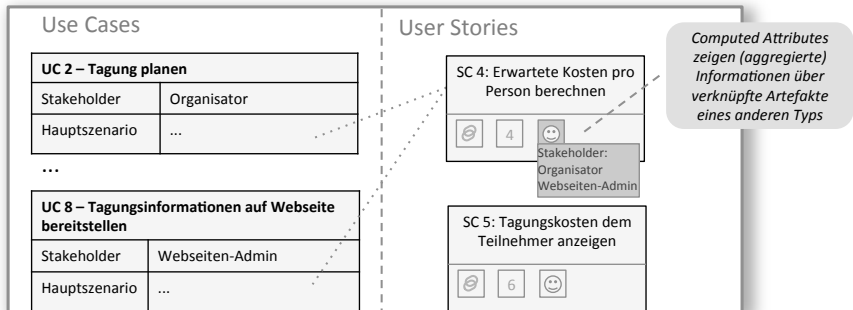


Abbildung 46: Integration zusätzlicher Informationen über verknüpfte Artefakte eines anderen Typs

Welche Informationen die wichtigsten und am häufigsten verwendeten sind, ist projektspezifisch. Die projektspezifisch einstellbaren Trace-Operationen des Typs *Computed Attributes* verkörpern genau solche Informationen. Sie werden im nächsten Abschnitt vorgestellt. Die Anzeige solcher *Computed Attributes* führt zu angereicherten Sichten, die helfen, Abhängigkeiten und Verknüpfungen zwischen Artefakten besser zu erkennen und zu verwenden. Genauso können auch die Ergebnisse der weiteren Trace-Operationen (statische und dynamische Kritiken) in die Sichten integriert werden, um den Nutzern weitere Informationen über die Verknüpfungen und deren Zustand zu geben.

Zusammenfassend ergeben sich die beiden folgenden Prinzipien, die helfen, abhängige Artefakte stärker zu integrieren.

Prinzipien für integrierte Sichten

1. Ansichten von Artefakten abhängiger Artefakttypen sollten nebeneinander in einer gemeinsamen Sicht dargestellt werden. Über Links in den jeweiligen Teil-Sichten sollte es möglich sein, zu verknüpften Elementen der anderen Teil-Sicht zu navigieren.
2. Zu einem Artefakt sollten wichtige Informationen über verknüpfte Artefakte eines abhängigen Artefakttyps direkt als Teil seiner eigenen Sicht angezeigt werden.

8.4 Trace-Operationen

Trace-Operationen sind projektspezifisch definierbare Tool-Operationen, die helfen, Verknüpfungen zu folgen und diese auszuwerten. Je nach dem, welche Artefakttypen in einem Projekt verwendet werden und welche Abhängigkeiten besonders wichtig beziehungsweise schwierig sind, legt der Requirements Methodiker unterschiedliche Trace-Operationen fest. Es gibt verschiedene Arten von Trace-Operationen, die auf verschiedene Art beschrieben werden und unterschiedliche Ergebnisse liefern.

Im Folgenden wird erarbeitet, welche genauen Informationen zum Festlegen der Trace-Operationen benötigt werden, wie diese formalisiert werden können und wie ein Computersystem diese für die Unterstützung im Umgang mit Anforderungen verarbeiten kann.

In der obigen Übersicht wurden drei Typen von Trace-Operationen vorgestellt: Computed Attributes sowie Statische und Dynamische Kritiken. Die dort vorgestellten Beispiel-Operationen werden hier als Referenz-Operationen herangezogen, um festzulegen, welche Eigenschaften Trace-Operationen besitzen müssen und wie diese aufgebaut werden können. Dazu werden diese hier noch einmal gesammelt und im weiteren Verlauf weiter analysiert und formalisiert. Abbildung 47 zeigt ein Referenz-Modell für eine mögliche Anforderungslandschaft, in der dann die Trace-Operationen aus Tabelle 18 gelten.

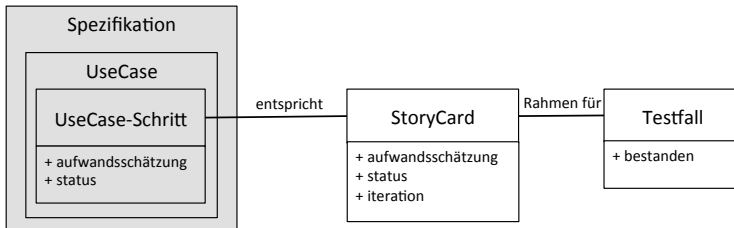


Abbildung 47: Beispielhaftes Anforderungslandschaftsmodell als Referenz für Trace-Operationen

8.4.1 Bestandteile von Trace-Operationen

Im Zentrum jeder Trace-Operation steht die Fähigkeit, anhand von Verknüpfungen automatisiert durch die Artefaktmenge zu navigieren und so Artefakte mit bestimmten Eigenschaften zu finden. Auf diese Art werden Teile von Koexistenz-Aktivitäten teil-automatisiert, sodass ihre Durchführung einfacher wird. Dazu muss allgemein festgelegt werden, wie von einem Artefakttyp zu den gewünschten verknüpften Artefakten bzw. Informationen zu navigieren ist. Dies geschieht durch eine *Trace Query*.

Tabelle 18: Sammlung der Referenz-Trace-Operationen

ID	Kurzbezeichnung	Trace-Operation
CA1	Verknüpfte Stories	Zeige am Use Case-Schritt Anzahl der verknüpften User Stories an
CA2	Verknüpfte Stories	Zeige am Use Case-Schritt per Schnellschaltfläche eine Liste mit Links zu allen verknüpften User Stories an
CA3	Aufwandsumme	Zeige am Use Case die Summe der Aufwandschätzungen aller (disjunkten) verknüpften User Stories an
CA4	Früheste Fertigstellung	Zeige am Use Case die späteste Iteration, der eine verknüpfte User Story zugeordnet ist (inkl. Backlog)
CA5	Ausführbarer Use Case	Markiere Use Case, wenn für alle Schritte mindestens eine User Story fertig implementiert ist, der Ablauf also potenziell ausführbar ist
StK1	Use Case-Schritt ohne Implementierung	Warne im Use Case-Schritt, wenn ein Element keine direkt verknüpften User Stories besitzt
StK2	Use Case-Schritt ohne Testfälle	Weise hin/ biete passiv an, wenn ein Element keine (indirekt) verknüpften Testfälle besitzt
StK3	Unterschätzter Use Case	Warne, wenn Use-Case-Schätzung geringer als die Summe der verknüpften Story Card-Schätzungen ist
StK4	Use-Case-Schritt fortsetzen	Warne, wenn Use Case-Schritt nicht fertig ist, aber die Story Cards schon. (Evtl. muss neue Story (als <i>vertical Slice</i>) angelegt werden.)
StK5	Wertbasierte Story-Verknüpfungen	Warne in der Story, wenn sie keinen verknüpften Use Case-Schritt besitzt, aber nur wenn die Story eine Priorität höher als 2 besitzt
DyK1	Potenzielle Inkonsistenz durch Änderung des Elementes	Wenn User Story geändert wird und verknüpfte Use Case-Schritte besitzt, zeige Hinweis über mögliche Inkonsistenz (und die verknüpften Elemente)
DyK2	Potenzielle Inkonsistenz durch Löschen des Elementes	Wenn Use Case-Schritt gelöscht wird und verknüpfte User Stories besitzt, die nicht bereits erledigt sind, zeige Hinweis über mögliche Inkonsistenz (und die verknüpften Elemente)
DyK3	Potenzielle Inkonsistenz durch Verschieben eines Elementes	Wird eine Aktivität, die verknüpfte Use Case-Schritte besitzt, im Geschäftsprozessmodell verschoben, zeige Warnung, dass das Modell evtl. nicht mehr konsistent ist mit dem Use Case.
DyK4	Potenzielle Inkonsistenz durch geändertes Attribut	Wird die Priorität einer nichtfunktionalen Anforderung geändert, zeige Warnung, dass verknüpfte Use Cases evtl. nicht mehr konsistent mit der Priorität der nichtfunktionalen Anforderung sind.

Bei den statischen und dynamischen Kritiken schließt sich außerdem an die Trace Query eine (heuristische) *Kritik* an. Das Ergebnis der Trace Query wird dazu ausgewertet. Dann wird bei bestimmten Ergebnissen eine Reaktion, beispielsweise eine Warnung, ausgegeben. Auf diese Art können Koexistenz-Aktivitäten angeregt und durch Präsentation der Query-Ergebnisse auch wieder erleichtert werden. Hierzu muss neben der Trace Query eine *Kritik* angegeben werden, die aus einer Auswertung des Trace Query-Ergebnisses und einer Reaktion besteht. Wann eine Trace-Operation durchgeführt wird, wird außerdem über einen *Auslöser* und optionale zugehörige *Bedingungen* gesteuert.

Abbildung 48 fasst die Bestandteile und Zusammenhänge der einzelnen Arten von Trace-Operationen zusammen.

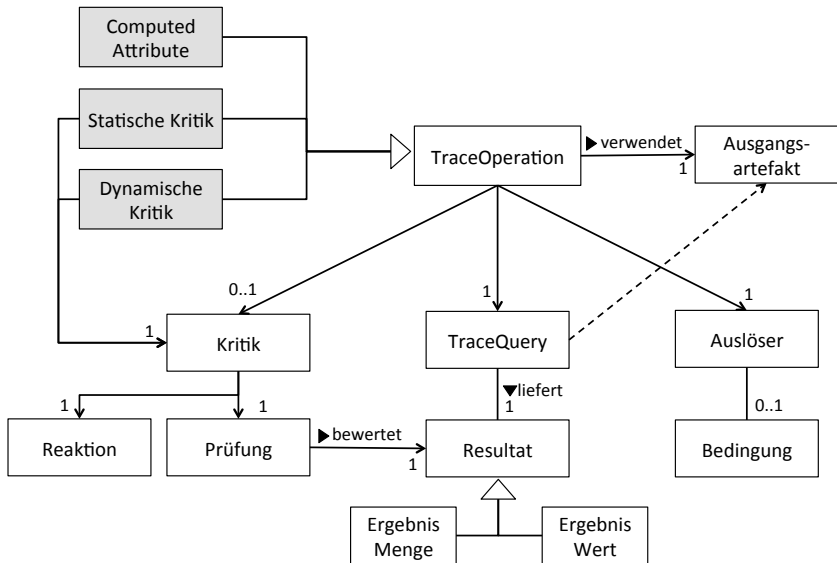


Abbildung 48: Verschiedene Trace-Operationen und ihre Bestandteile

Jede Trace Operation besitzt eine *Trace Query* im Zentrum. Damit wird festgelegt, wie von einem bestimmten *Ausgangsartefakt* aus zu anderen relevanten Artefakten oder Informationen navigiert wird. Basierend auf dem Ausgangsartefakt wird eine Query durchgeführt, die zu einem *Resultat* führt. Das Resultat kann dabei eine Liste von Elementen sein oder einen einzelnen Wert (beispielsweise durch Aggregation) enthalten.

Statische und dynamische *Kritiken* definieren zusätzlich eine *Prüfung* des Trace Query-Resultates. Wenn die Auswertung *wahr* liefert, so wird die *Reaktion* ausgelöst, die beispielsweise eine Warnung anzeigt. Statische und dynamische Kritiken sind größtenteils gleich aufgebaut. Sie unterscheiden sich hauptsächlich durch den Auslöser.

Jede Trace-Operation besitzt außerdem einen *Auslöser*, der festlegt, *wann* die Operation durchgeführt wird. Eine Operation kann bei einem bestimmten Methodenaufruf ausgelöst werden. Alternativ kann es auch gewünscht sein, eine Operation bei jedem Ladevorgang eines Artefaktes durchzuführen und ständig anzuzeigen. Zusätzlich können *Bedingungen* festgelegt werden, die einschränken, wann eine Trace Query und die zugehörige Kritik ausgeführt werden. Beispielsweise kann festgelegt werden, dass Kritiken nur ausgeführt werden, wenn das Ausgangsartefakt eine hohe Priorität besitzt.

Sinnvolle Formen der Reaktion.

Wenn ein potenzielles Problem festgestellt wurde, gibt es verschiedenen Arten, den Nutzer darauf hinzuweisen. Das Problem kann so präsentiert werden, dass es den Nutzer unterbricht und seine Aufmerksamkeit auf das potenzielle Problem lenkt. Dies kann den Nutzer stören und auf Dauer nerven, vor allem wenn dann häufig doch kein relevantes Problem vorliegt. Alternativ kann auf das Problem auf eine subtile, nicht unterbrechende, Art hingewiesen werden, indem beispielsweise nur ein zusätzliches Warnsymbol eingeblendet wird. Dies führt wiederum dazu, dass schneller vergessen wird, das Problem zu überprüfen. Sind außerdem an zu vielen Artefakten solche Warnsymbole vorhanden, fallen einzelne neue Warnungen nicht mehr auf. Je nach dem, wie wichtig die Konsistenzerhaltung an den verschiedenen Stellen ist, sollte die Reaktion daher in den einzelnen Trace-Operationen angepasst werden.

Innerhalb der Forschung zu Kritiksystemen wird unter anderem untersucht, welche Präsentation von Kritiken am sinnvollsten ist [50], [78], [165]. Zettel [166] hat eine Klassifikation aufgestellt, bei der er nach den Eigenschaften *Erzwingung einer Korrektur (Toleranz gegenüber Regelverstößen, automatische Korrektur, Vermeidung von Regelverstößen)*, *Eingreifstrategie (aktiv, passiv)* und *Form des Eingriffs (Unterbrechung und/oder Anzeige)* einer Kritik-Reaktion unterscheidet. Wie eine anschließende Untersuchung von Zettel zeigt, ist es allerdings nicht möglich, klar festzulegen, wann welche Variante optimal ist. Vielmehr hängt es unter anderem von den Nutzern ab, die von den Kritiken betroffen sind. Entsprechend sollte es auch für den Requirements Methodiker möglich sein, für einzelne Trace-Operationen unterschiedliche Reaktionen festzulegen. Die Festlegung basiert beispielsweise auf Wichtigkeit der Konsistenzerhaltung an der entsprechenden Abhängigkeit und auf Wünschen der Projektteilnehmer. In Tabelle 19 sind vier mögliche Reaktionen aufgeführt, die in der hier vorliegenden Situation – der Unterstützung von anforderungsbezogenen Aktivitäten beim Umgang mit einem RE-Tool – sinnvoll sind. Eine automatische Korrektur wäre nur möglich, wenn die Anforderungen entsprechend formal dokumentiert sind. Da der hier vorliegende Ansatz alle Grade von Formalität unterstützt, ist diese Option außerhalb des Scopes dieser Arbeit.

Während hier Zettels Eigenschaften zur Klassifikation, also ob und wie der Nutzer unterbrochen wird, zutreffen und auch angegeben sind, unterscheidet sich die Anwendung allerdings in einem Punkt: In Zettels Ansatz liegen statische Inkonsistenzen vor. Mit dynamischen Kritiken werden in der hier vorliegenden Arbeit aber auch Situationen aufgegriffen, in denen ein potenzielles Problem nur anhand einer Aktion, wie einem Änderungsvorgang, festgestellt werden kann und infolgedessen später nicht mehr angezeigt werden kann. Es ist möglich, solche dynamisch feststellbaren Probleme in statisch anzeigbare Probleme umzuwandeln, indem betroffene Artefakte während der Reaktion einer dynamischen Kritik beispielsweise zusätzlich als *potenziell inkonsistent* markiert werden.

Tabelle 19: Sinnvolle Formen von Reaktionen innerhalb von Kritiken

Bezeichner	Reaktion	ID aus Zettels Klassifikation
<i>Keine Reaktion</i>	-	IGN bzw. CP
<i>Hervorhebung</i>	In dem Moment, in dem mit einem Artefakt umgegangen wird, werden damit verknüpfte Artefakte in ihrer Sicht hervorgehoben. Beispielsweise, indem die Sicht zu diesen Elementen scrollt und ihre Farbe geändert wird.	CT2
<i>Warnsignal</i>	An ein Artefakt, das ein potenzielles Problem enthält, wird ein Warn-Icon geheftet, welches dort so lange angezeigt wird, bis das Problem behoben ist oder durch den Nutzer als überprüft markiert wurde.	CT2 (wie Hervorhebung, aber unterschiedliche Anzeige)
<i>Warndialog</i>	Es wird ein unterbrechender Dialog angezeigt, der das potenzielle Problem erläutert und die betroffenen Artefakte anzeigt. Diese können direkt aus dem Dialog heraus angepasst werden. Wird im Dialog nicht explizit bestätigt, dass Artefakte überprüft wurden, so wird an die entsprechenden Artefakte ein Warnsignal zur späteren Bearbeitung gehängt.	CT4

Zerlegung von Trace-Operationen in ihre Bestandteile.

Zur Veranschaulichung der einzelnen Bestandteile von Trace-Operationen werden in Tabelle 20 die Trace Operationen aus Tabelle 18 in die einzelnen Teile für Auslöser-, Query- und Kritik zerlegt.

Tabelle 20: Aufteilung der Trace-Operation in die Bestandteile Auslöser, Query und Kritik

ID	Operation		
CA1	Zeige am Use Case-Schritt Anzahl der verknüpften User Stories an		
	<i>Auslöser:</i> Statisch/Anfrage	<i>Query:</i> Anzahl der verknüpften User Stories	<i>Kritik:</i> -
CA2	Zeige am Use Case-Schritt per Schnellschaltfläche eine Liste mit Links zu allen verknüpften User Stories an		
	<i>Auslöser:</i> Statisch/Anfrage	<i>Query:</i> Alle verknüpften User Stories	<i>Kritik:</i> -
CA3	Zeige am Use Case die Summe der Aufwandschätzungen aller (disjunkten) verknüpften User Stories an		
	<i>Auslöser:</i> Statisch/Anfrage	<i>Query:</i> Summe der Aufwandsschätzung aller Stories, die mit einem Use Case-Schritt des Use Cases verknüpft sind	<i>Kritik:</i> -
CA4	Zeige am Use Case die späteste Iteration, der eine verknüpfte User Story zugeordnet ist		
	<i>Auslöser:</i> Statisch/Anfrage	<i>Query:</i> Maximum-Wert der Iterationen aller Stories, die mit einem Use Case-Schritt des Use Cases verknüpft sind.	<i>Kritik:</i> -
CA5	Markiere Use Case, wenn für alle Schritte mindestens eine User Story fertig implementiert ist, der Ablauf also potenziell ausführbar ist		
	<i>Auslöser:</i> Statisch/Anfrage	<i>Query:</i> Finde alle Use Case-Schritte, die keine User Story besitzen, die als fertig markiert ist. Gebe true zurück, wenn diese Menge leer ist.	<i>Kritik:</i> -

StK1	Warne im Use Case-Schritt, wenn ein Element keine direkt verknüpften User Stories besitzt		
	<i>Auslöser:</i> Statisch/Anfrage	<i>Query:</i> Anzahl der verknüpften User Stories	<i>Kritik:</i> Wenn Resultat = 0, dann Warnung ausgeben
StK2	Weise hin/ biete passiv an, wenn ein Element keine (indirekt) verknüpften Testfälle besitzt		
	<i>Auslöser:</i> Statisch/Anfrage	<i>Query:</i> Alle verknüpften User Stories, die keine verknüpften Testfälle besitzen	<i>Kritik:</i> Wenn Menge nicht leer, gebe Warnung aus und zeige Stories ohne Tests
StK3	Warne, wenn Use-Case-Schätzung geringer als die Summe der verknüpften Story Card-Schätzungen ist		
	<i>Auslöser:</i> Statisch/Anfrage	<i>Query:</i> Summe der Schätzung aller verknüpften User Stories	<i>Kritik:</i> Wenn Resultat > Use-Case-Schätzung, dann gebe Warnung aus und zeige die Stories
StK4	Warne, wenn Use Case-Schritt nicht fertig ist, aber die Story Cards schon. (Evtl. muss neue Story als <i>vertical Slice</i> angelegt werden.)		
	<i>Auslöser:</i> Statisch/Anfrage [Bedingung: UC-Schritt nicht fertig]	<i>Query:</i> Anzahl der verknüpften User Stories, die nicht fertig sind	<i>Kritik:</i> Wenn Resultat = 0, dann Warnung ausgeben und Stories zeigen.
StK5	Warne in der Story, wenn sie keinen verknüpften Use Case-Schritt besitzt, aber nur wenn die Story eine Priorität höher als 2 besitzt.		
	<i>Auslöser:</i> Statisch/Anfrage [Bedingung: Story Priorität höher 2]	<i>Query:</i> Anzahl der verknüpften Use Case-Schritte	<i>Kritik:</i> Wenn Resultat = 0, dann Warnung ausgeben
DyK1	Wenn User Story geändert wird und verknüpfte Use Case-Schritte besitzt, zeige Hinweis über mögliche Inkonsistenz (und die verknüpften Elemente)		
	<i>Auslöser:</i> edit()	<i>Query:</i> Verknüpfte Use Case-Schritte die nicht abgeschlossen sind	<i>Kritik:</i> Wenn Menge nichtleer, dann zeige Warnung und Query-Resultat
DyK2	Wenn Use Case-Schritt gelöscht wird und verknüpfte Stories besitzt, die nicht bereits erledigt sind, zeige Hinweis über mögliche Inkonsistenz (und die verknüpften Elemente)		
	<i>Auslöser:</i> delete()	<i>Query:</i> Verknüpfte User Stories, die nicht abgeschlossen sind	<i>Kritik:</i> Wenn Menge nichtleer, dann zeige Warnung und Query-Resultat
DyK3	Wird eine Aktivität, die verknüpfte Use Case-Schritte besitzt, im Geschäftsprozessmodell verschoben, zeige Warnung, dass Modell evtl. nicht mehr konsistent mit Use Case ist..		
	<i>Auslöser:</i> move()	<i>Query:</i> Verknüpfte Use Case-Schritte	<i>Kritik:</i> Wenn Menge nichtleer, dann zeige Warnung und Query-Resultat
DyK4	Wird die Priorität einer nichtfunktionalen Anforderung geändert, zeige Warnung, dass verknüpfte Use Cases evtl. nicht mehr konsistent mit der Priorität der nichtfunktionalen Anforderung sind.		
	<i>Auslöser:</i> changePrio()	<i>Query:</i> Verknüpfte Use Cases	<i>Kritik:</i> Wenn Menge nichtleer, zeige Warnung und Query-Resultat

8.4.2 Notwendige Ausdruckskraft innerhalb der Bestandteile von Trace-Operationen

Um eine Trace-Operation später formal zu beschreiben (hierauf wird im folgenden Abschnitt 8.4.3 eingegangen), müssen ein Auslöser, eine Trace Query und eine Kritik formalisiert werden können. Um eine geeignete Formalisierung zu finden, muss daher klar sein, was im Detail damit ausdrückbar sein muss.

Aus den Referenz-Operationen wird abgeleitet, welche Ausdrucksmöglichkeiten Auslöser, Trace Queries und Kritiken mindestens besitzen müssen, um alle nötigen Arten von Trace-Operationen erstellen zu können. Diese Mindest-Erfordernisse werden im Folgenden festgehalten.

Auslöser.

Für die Festlegung eines Auslösers reicht es aus, eine von mehreren vorgegebenen Arten von Auslösern auswählen zu können. Die statischen Arten von Trace-Operationen, also die Computed Attributes und statischen Kritiken, werden entweder durch eine explizite Nutzeranfrage ausgelöst oder permanent durchgeführt und angezeigt. Dann besitzen sie keinen konkreten Auslöser. Innerhalb des Werkzeugs kann dann das Laden des entsprechenden Artefaktes bzw. der Artefakt-Sicht als Auslöser aufgefasst werden.

Dynamische Kritiken werden bei Ausführung einer bestimmten Methode, wie *editiere()* oder *lösche()*, des entsprechenden Artefaktes ausgelöst. Hierfür sollte es möglich sein, den Methodennamen des entsprechenden Artefakttyps als Auslöser anzugeben.

Zum Auslöser kann noch Optional eine **Bedingung** angegeben werden. Eine Bedingung legt fest, wann die Operation ausgeführt wird. Dazu wird ein boolescher Ausdruck¹² festgelegt, dessen Auswertung wahr oder falsch ergibt. Nur wenn die Bedingung erfüllt ist, werden die Query und die Kritik ausgeführt. Dabei bezieht sich die Bedingung auf das Ausgangsartefakt. Beispielsweise wird geprüft, ob das Ausgangsartefakt eine ausreichend hohe Priorität besitzt. Es werden also Attribute des Ausgangsartefaktes abgefragt und mit Werten verglichen. Es ist nicht notwendig, in der Bedingung weitere Queries nach verknüpften Artefakten durchführen zu können.

Trace Query.

Wie die Referenz-Trace-Operationen zeigen, reicht es nicht aus, mit einer Trace Query nur von Artefakttyp zu Artefakttyp zu navigieren, also zu einem Artefakt einfach nur alle verknüpften Artefakte eines bestimmten Typs auszugeben.

¹² Ein boolescher Ausdruck ist ein Element einer booleschen Algebra [154]. Elemente der Algebra sind Ausdrücke, die wahr oder falsch sind. Diese können dann mit den booleschen Operatoren UND, ODER und NICHT zu neuen booleschen Ausdrücken kombiniert werden.

Stattdessen werden auch komplexere Navigationsmechanismen benötigt, die auch aus anderen Query-Sprachen, wie SQL bekannt sind.

Über eine Trace Query sollten mindestens die folgenden Abfragen und Operationen möglich sein:

- Von einem Ausgangselement aus zu verknüpften Artefakten navigieren. (*Verwendet in allen Referenz-Operationen*)
- Aus einer Artefaktmenge Artefakte mit bestimmten Eigenschaften filtern
 - Dabei kann sich die Filter-Bedingung auf Attribute beziehen. (*Verwendet in Referenz-Operationen StK4, DyK2*)
 - Die Filter Bedingung kann jedoch auch weitere Berechnungen bzw. Aggregationen enthalten. In SQL entspricht dies Konstrukten mit *Group By*- und *Having*-Klauseln. (*Verwendet in Referenz-Operationen StK2, StK3, StK4*) („gebe alle Stories aus, die keine Testfälle besitzen“; „gebe späteste Iteration aus, in der ein Use Case-Schritt fertiggestellt wird“)
- Nur bestimmte Attribute der Ergebnis-Artefakte zurückgeben. (*Verwendet in den Referenz-Operationen CA2, CA3,)*
- Eine Aggregationsfunktion (SUM, COUNT, MIN, MAX, AVG) auf die Ergebnis-Elemente anwenden (*Verwendet in den Referenz-Operationen CA1, CA2, CA3, StK1, StK4, StK5*)

Kritik.

Kritiken besitzen zwei Bestandteile, die unabhängig voneinander beschrieben werden können. In Tabelle 20 sind stets eine *Prüfung* und eine *Reaktion* aufgeführt.

In den Referenz-Operationen ist sichtbar, dass die Prüfung auf dem Resultat der Query basiert. Zusätzlich kann es sinnvoll sein, das Query-Resultat auch in der Reaktion aufzugreifen, um es auszugeben. Eine erste Erfordernis ist also, dass man in der Prüfung und in der Reaktion auf das Resultat der Query zugreifen können muss.

Die Kritik selbst besteht dann aus einer **Prüfung**, die das Resultat der vorher durchgeführten Trace Query überprüft. Die Trace Query kann entweder einen einzelnen Wert zurückliefern oder auch eine ganze Liste von Elementen. Entsprechend muss es in der Prüfung möglich sein, einfache boolesche Ausdrücke anzugeben, bei denen Prädikate¹³ in Bezug auf Attribute formuliert werden (wie „die Priorität der Story ist kleiner als 2“). Zusätzlich müssen aber auch Listen ausgewertet werden können, beispielsweise durch Prädikate die auch Aggrega-

¹³ Prädikate und Quantoren sind Konzepte der Prädikatenlogik [154]

tionsfunktionen enthalten („die Anzahl der Elemente der Liste ist größer als null“) oder durch Quantoren („es existiert ein Element mit bestimmten Eigenschaften“).

Schließlich muss eine **Reaktion** festgelegt werden, die ausgeführt wird, falls die Prüfung der Kritik *wahr* ergibt. Wie im vorigen Abschnitt beschrieben, gibt es verschiedene Arten von Reaktionen, die prinzipiell zunächst über ein Schlüsselwort, wie „*Hervorheben*“, „*Warnsymbol*“ oder „*Warndialog*“, festgelegt werden können. Dazu sollte eine Warnmeldung spezifiziert werden können, die genauer darstellt, was genau das Problem ist. In einigen Operationen macht es zusätzlich Sinn, dem Nutzer die problematischen Artefakte oder Verlinkungen darauf direkt anzuzeigen, um ihm zu helfen, das Problem schnell zu überprüfen und gegebenenfalls zu beheben. Hierzu kann automatisch das Ergebnis der Trace Query angezeigt werden, sodass zur Definition nur ein Schlüsselwort *displayQueryResultInReaction* – aber eben keine neue Trace Query – eingegeben werden muss, wenn dies gewünscht ist.

8.4.3 Formalisierung von Trace-Operationen

Es wird eine geeignete Beschreibungssprache für Trace-Operationen gesucht. Die Beschreibung der Trace-Operationen hat dabei mehrere Zwecke, die in Abbildung 49 dargestellt sind. Hauptsächlich wird eine Beschreibungssprache vom Requirements Methodiker verwendet, um ein *CASE-Tool* für die Anwendung projektspezifischer Trace-Operationen zu *konfigurieren*. Jedoch können die Darstellungen von Trace-Operationen auch dazu verwendet werden, *Projektrichtlinien*, die sich auf den Umgang mit Anforderungen und deren Abhängigkeiten beziehen, zu *visualisieren*. Die Festlegung solcher Richtlinien wurde in der Interview-Studie als wichtiger Aspekt zur Unterstützung von Tracing-Aktivitäten erwähnt (siehe Abschnitt 4.4.6). Neben der bloßen Visualisierung der Projektrichtlinien können die Beschreibungen der Trace-Operationen auch zur *gemeinsamen Erarbeitung der Richtlinien* verwendet werden. Indem Projektteilnehmer und der Requirements Methodiker gemeinsam das Anforderungslandschaftsmodell und die Trace-Operationen diskutieren, können sie erarbeiten, welche anforderungsbezogenen Aktivitäten unterstützt werden sollten und wie dabei vorgegangen werden kann. Wenn Projektteilnehmer Koexistenz-Aktivitäten, die sie häufig manuell durchführen, formal beschreiben, können daraus Trace-Operationen erzeugt und damit automatisierte Unterstützung geschaffen werden.

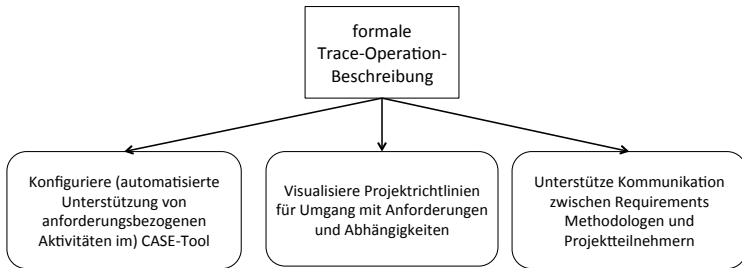


Abbildung 49: Verwendung der Beschreibung von Trace-Operationen

In dieser Arbeit werden zwei Möglichkeiten vorgestellt, die Trace-Operationen zu formalisieren. Die eine basiert auf VTML (*Visual Trace Modeling Language*, siehe [109]), einer Sprache zur grafischen Repräsentation von Trace Queries. Die zweite Möglichkeit basiert auf OCL (*Object Constraint Language*, siehe [22], [122]) und somit einer textuellen Darstellung von Trace Queries und Kritiken.

Diese beiden Möglichkeiten eignen sich in der vorliegenden Situation besonders gut, da sie gut als Erweiterung zum Anforderungslandschaftsmodell, das auch als UML-Modell dargestellt werden kann, dienen. Zudem gibt es für beide Methoden bereits Werkzeug-Unterstützung, was ebenfalls einen Vorteil im Vergleich zu selbstdefinierten Sprachen darstellt.

Abbildung 50 zeigt ein Beispiel, in dem eine Trace-Operation jeweils mit beiden Methoden formalisiert ist. VTML-Queries basieren auf UML-Klassendiagrammen, die einen Teil eines Artefaktmodells aufgreifen und durch zusätzliche Symbole für Einschränkungen und Aggregationen ergänzen. Sie erlauben es, eine Navigation im Landschaftsmodell visuell darzustellen, was für Nutzer einfacher sein kann als textuelle Beschreibungen: Mäder und Cleland-Huang [109] haben in einem Experiment gezeigt, dass Trace Queries mit VTML schneller gelesen und erstellt werden konnten, als textuelle Trace Queries mit SQL. Die Formalisierung von Trace-Operationen mittels VTML hat jedoch den Nachteil, dass der Kritikteil nicht mit VTML darstellbar ist und zum Großteil separat in einer anderen Sprache beschrieben werden muss.

Die Methode, die auf OCL basiert, hat genau hier ihren Vorteil, denn mithilfe von OCL lassen sich sowohl die Trace Query als auch ihre Überprüfung zusammen darstellen. Diese Methode hat den Nachteil, dass die Formalisierung schwieriger zu lesen und zu erstellen ist.

In den folgenden Abschnitten werden die beiden Methoden detaillierter beschrieben. Beide Methoden besitzen die für Trace-Operationen notwendige Ausdruckskraft. Aufgrund der unterschiedlichen Vor- und Nachteile kann jedoch keine der Methoden als überlegen angesehen werden. Aus diesem Grund

werden beide Methoden betrachtet und auch stets die Formalisierungen auf beide Arten angegeben, um die Unterschiede besser einschätzen zu können. Welche Methode in einem konkreten Projekt besser geeignet ist, hängt von den Projektteilnehmern ab und auch davon, inwiefern die Projektteilnehmer in die Festlegung der Anforderungslandschaft einbezogen werden. Soll die Formalisierung nur vom Requirements Methodiker dafür genutzt werden, ein Werkzeug zu konfigurieren, so kann OCL mächtiger und besser geeignet sein. Wird die Formalisierung aber auch zur Diskussion des Umgangs mit Anforderungen herangezogen, so ist die visuelle Variante basierend auf VTML besser geeignet.

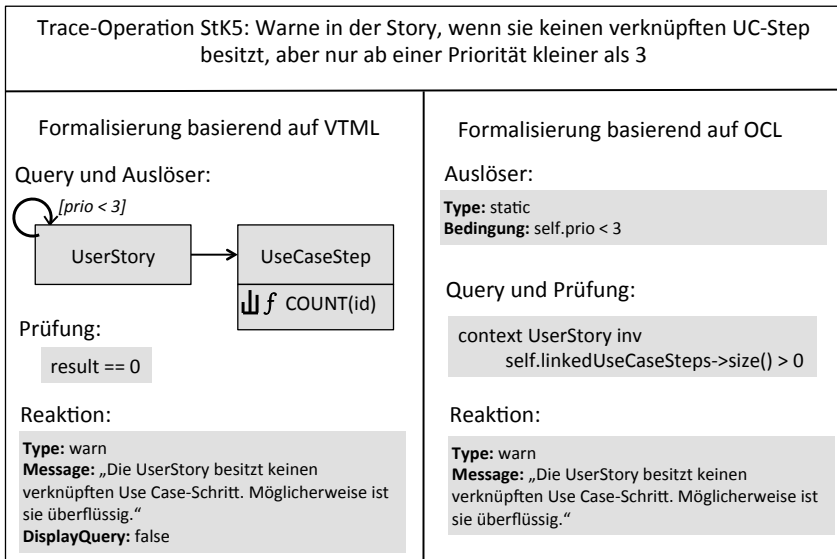


Abbildung 50: Beispielhafte Formalisierung einer Trace-Operation basierend auf VTML und auf OCL

8.4.4 Textuelle Formalisierung von Trace-Operationen basierend auf OCL

Die *Object Constraint Language (OCL)* [122], [159] erweitert UML-Modelle um zusätzliche Einschränkungen sowie abgeleitete Größen und Funktionen. OCL-Konstrukte beziehen sich auf ein Referenz-UML-Klassendiagramm. Es lassen sich generelle Aussagen (ohne Kontext) über Elemente des Klassendiagramms treffen. Alternativ kann eine der Klassen als Kontext-Element angegeben werden. Dann bezieht sich der weitere OCL-Ausdruck auf ein konkretes Objekt dieser Klasse. Dies ist hier wichtig, da sich alle Trace-Operationen auf ein Kontext-Element beziehen, bei dem die Abfrage startet.

Grundaufbau.

Wie im Namen bereits angedeutet, lassen sich mittels der OCL *Constraints*, also Einschränkungen, formulieren. Diese können als Invariante, die immer gelten muss, oder als Vor- bzw. Nachbedingung einer Methode spezifiziert werden. Doch auch *Queries*, also Abfragen, können mittels OCL definiert werden [22]. Dazu werden Variablen definiert. Es ist zu beachten, dass innerhalb eines OCL-Ausdrucks verwendete Funktionen keine Auswirkungen auf den Systemzustand haben können, da OCL seiteneffektfrei ist [139]. Genauso können Variablen definiert werden, diese dann aber nicht mit Daten im System selbst verlinkt. Die folgende Tabelle zeigt die Grundformen für Einschränkungen, und Variablendefinitionen. Als Bezugspunkt dient das Landschaftsmodell der Referenz-Trace-Operationen – bzw. dessen UML-Pendant – wie in Abbildung 51 dargestellt.

Tabelle 21: Grundsätzlicher Aufbau von Invarianten und Variablendefinitionen in der OCL

Konstrukt	OCL-Code
Invariante	context UseCaseStep inv hasLinkedStories: self.storyCard.size() > 0
Query-Definition als Variable	context UseCaseStep def: linkedStoryCount:Integer = self.storyCard.size()

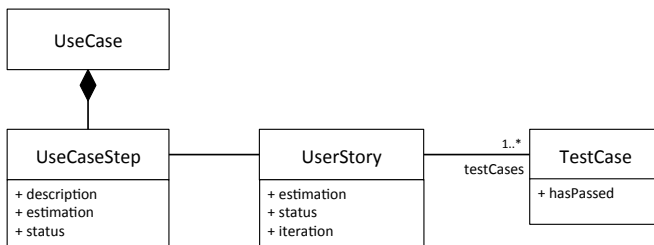


Abbildung 51: Referenz-Klassendiagramm für OCL-Ausdrücke

Navigation.

Navigation erfolgt über Punkt-Notation mit Benennung der Teile zu denen navigiert wird. Es können Attribute der Klasse angesprochen werden. Elemente assoziierter Klassen können über ihren Rollennamen angesprochen werden. Ist kein Rollename hinterlegt, so können die verlinkten Elemente über den kleingeschriebenen Klassennamen der assoziierten Klasse abgerufen werden [136]. Grundsätzlich liefert die Navigation eine Menge von Elementen zurück, wenn das Assoziationsende nicht eine Multiplizität von höchstens 1 besitzt. Entsprechend wird die Punkt-Notation verwendet, um auf Attribute bzw. verlinkte Elemente zuzugreifen. Die Pfeilnotation erlaubt es, vordefinierte OCL-Mengen-Funktionen auf Mengenobjekte anzuwenden. Tabelle 22 zeigt die hierfür relevanten Ausdrücke.

Tabelle 22: Navigation mittels OCL

Konstrukt	OCL-Code
Navigation	<pre>context UseCaseStep def: theDescription : String = self.description def: parentUC : UseCase = self.useCase def: stories : Set<UserStory> = self.userStory</pre>
Navigation durch Listen	<pre>def: storyEsts : Set<Integer> = self.userStory.estimation (entspricht self.userStory.collect (story story.estimate))</pre>
Listenoperationen	<pre>def: estSum : integer = self.userStory.estimate->sum()</pre>

Mengenfunktionen.

Innerhalb von OCL-Ausdrücken kann weiterhin auf *Listen-* bzw. *Mengenfunktionen* zugegriffen werden. Listen und Mengen können darauf geprüft werden, ob sie leer sind, ob sie ein bestimmtes Element enthalten oder welche Größe sie haben. Über Selektionen und Projektionen sind Auswahlen von Listenteilen möglich. Mittels Quantoren können prädikatenlogische Ausdrücke festgelegt werden, die alle Elemente der Menge prüfen oder abfragen, ob ein Element mit bestimmten Eigenschaften in der Menge existiert. Aggregationen lassen sich mittels vordefinierter Funktionen oder sonst mittels des *iterate*-Operators durchführen. Tabelle 23 fasst relevante Mengen- bzw. Listen-Operationen zusammen. Mit diesen Notations-Elementen lassen sich viele simple und komplexe Abfragen und Überprüfungen durchführen. Im Folgenden wird nun dargestellt, wie die Notation genutzt werden kann, um Trace-Operationen zu beschreiben.

Tabelle 23: Listen- bzw. Mengenoperationen in OCL

Konstrukt	OCL-Code
Listenfunktionen	<pre>storyCards->size() storyCards->isEmpty() storyCards->includes(card)</pre>
Auswahl	<pre>storyCards->select(card card.priority > 4) storyCards->collect(card card.estimate) (=: storyCards.estimate)</pre>
Quantoren	<pre>storyCards->forall(story story.priority > 0) storyCards->exists(story story.status <> 'done')</pre>
Aggregation	<pre>storyCards->count(xx) storyCards.estimate->sum() storyCards->iterate{story; int sum = 0; sum = sum + story.estimate }</pre>

Trace Query

Trace Queries werden durch Navigations-Operationen zusammen mit Listen-Operationen durchgeführt. Die vorgestellten Konstrukte erlauben es, alle notwendigen Abfragevarianten durchzuführen.

Wird eine Query für sich ausgeführt und direkt zurückgegeben, wie bei Computed Attributes, so geschieht dies über eine Variablendefinition, wie im folgenden Beispiel angegeben:

```
context <Ausgangs-Artefakt>
def: <Variablenname> : <Variablentyp> = <Trace Query>
bzw.
context UseCaseStep
def: linkedStoryCount : integer = self.storyCard.size()
```

Ist die Trace Query Teil einer Kritik, so kann sie auch direkt als Teil der Invarianten-Beschreibung, wie im folgenden Code-Abschnitt gezeigt, angegeben werden:

```
context <Ausgangs-Artefakt> inv
  <Trace Query> <Überprüfung der Query>
bzw.
context UseCaseStep inv
  self.storyCard.size() > 0
```

So wird eine kompaktere Schreibweise ermöglicht. Dennoch kann es Sinn machen, den Query-Teil als Variable zu definieren und dann in einer Prüfung gesondert abzufragen, da so auch das Query-Resultat in der Reaktion genutzt werden kann. Das obige Beispiel hätte dann die folgende Form:

```
context UseCaseStep
def: linkedStoryCount:integer = self.storyCard.size() --Trace-Query
inv: linkedStoryCount > 0
```

Die folgende Tabelle 24 zeigt, wie alle Ausdrucksmöglichkeiten, die aus den Referenz-Trace-Operationen abgeleitet wurden, durch OCL-Ausdrücke dargestellt werden können.

Tabelle 24: Formalisierung von Navigationselementen mittels OCL

Gewünschte Ausdrucksdruckmöglichkeit	OCL-Notation
Von einem Ausgangselement aus zu verknüpften Artefakten navigieren	(a) context UserStory def: eineQuery:resultType = ... self.useCaseStep ...
Aus einer Artefaktmenge Artefakte mit bestimmten Eigenschaften filtern, basierend auf Attributen	(b) context UseCaseStep def: pendingStories : Set<UserStory> = self.userStory->select(st st.status<>,done')
Aus einer Artefaktmenge Artefakte mit bestimmten Eigenschaften filtern, basierend auf komplexer Filter-Bedingung (hier: Story besitzt fehlgeschlagene Tests)	(c) context UseCaseStep def: failedStories : Set<Story> = self.userStory ->select(story story.testCase->exists(testCase testCase.hasPassed = false))
Nur bestimmte Attribute der Ergebnis-Artefakte zurückgeben	(d) context UseCaseStep def: storyEstimations: Set<Integer> = self.userStory->collect(st st.estimate)
Eine Aggregationsfunktion (SUM, COUNT, MIN, MAX, AVG) auf die Ergebnis-Elemente anwenden	(e) context UseCaseStep def: estimation: integer = iterate{story in self.userStory; int sum = 0; sum = sum + story.estimate }

Bedeutung der Beispiel-Queries: (a) Navigiert von einer bestimmten userStory zur Menge ihrer verknüpften UseCaseSteps. (b) Gibt zu einem UseCaseStep alle Stories aus, die noch nicht erledigt sind. (c) Gibt zu einem UseCaseStep alle verknüpften Stories aus, die mindestens einen fehlgeschlagenen Testfall besitzen. (d) Gibt zu einem UseCaseStep eine Menge mit den Schätzwerten aller verknüpften Stories aus. (e) Gibt zu einem UseCaseStep die Summe der Schätzwerte aller verknüpften UserStories – als Indikator für den eigenen Schätzwert der Umsetzung des gesamten UseCaseSteps – aus.

Auslöser

Der *Auslöser* kann nicht als Teil des OCL-Ausdrucks spezifiziert werden. Er wird separat angegeben, indem ein Schlüsselwort für *statisch*, *Nutzeranfrage*, oder ein Methodenname des Kontextelementes spezifiziert wird.

Eine *Bedingung* wiederum kann als OCL-Ausdruck spezifiziert werden. Die Bedingung wird dann den restlichen Überprüfungen vorangestellt und per *implies* mit diesen verbunden. Ist dann die Bedingung nicht erfüllt, so wird auch der Rest der Überprüfung nicht durchgeführt.

Kritik

Die **Überprüfung**, die zu einer Kritik gehört, kann mittels OCL-Constraints angegeben werden. Wie bereits oben im *Trace Query*-Teil erwähnt, kann in den Constraint selbst direkt eine Trace Query eingebunden werden oder alternativ zuerst als Variable definiert und dann im Constraint aufgegriffen werden. Da so die Variable später weiter verwendet werden kann, wird hier die zweite Variante vorgestellt.

Tabelle 25 zeigt an Beispielen, wie die Ausdruckselemente mittels OCL spezifiziert werden können.

Tabelle 25: Formalisierung von Überprüfungen mittels OCL

Gewünschte Ausdrucksmöglichkeit	OCL-Notation
Überprüfe ein Attribut	(f) context UserStory def: thePriority : integer = self.priority inv thePriority > 0
Logische Verknüpfung von Prüfungen	(g) context UserStory def: theEstimation: integer = self. estimation inv self.iteration=Iteration.current and theEstimation > 0
Überprüfe eine Liste mittels komplexer Ausdrücke im Constraint	(h) context UseCaseStep def: userStories : Set<UserStory> = self.userStory inv self.status <> ‚done‘ and userStories->forall(story story.status = ‚done‘)
Überprüfe das Resultat einer Aggregation einer Liste	(i) context UseCaseStep def linkedStoryCount:integer = self.userStory->size() inv linkedStoryCount > 0

Bedeutung der Beispiel-Constraints: (f) Constraint nicht erfüllt, wenn es eine Story mit einer Priorität, die kleiner oder gleich 0 ist, gibt. (g) Constraint nicht erfüllt, wenn eine UserStory der aktuellen Iteration zugeordnet ist, aber noch keine Aufwandsschätzung besitzt. (h) Constraint nicht erfüllt, wenn eine UseCaseStep als nicht fertig gilt, aber alle seine verknüpften Stories als fertig markiert sind. (i) Constraint nicht erfüllt, wenn ein UseCaseStep keine verknüpften UserStories besitzt.

Die **Reaktion** zu einer Kritik kann nicht mittels OCL angegeben werden. Sie muss separat spezifiziert werden. Hierzu wird ein Schlüsselwort für die Art der

Warnung (*Warnung-Art*) und eine Warnmeldung in Form einer Zeichenkette (*Warnmeldung*) benötigt. Beides kann einfach als textuelles Attribut zur Trace-Operation eingegeben werden. Weiterhin kann es sinnvoll sein, mit der Warnung direkt die betroffenen Artefakte auszugeben. Wenn eine Variable für die Trace Query verwendet wird, so kann über ein zusätzliches Attribut (*Trace Query anzeigen?*) angegeben werden, ob das Resultat der Trace Query angezeigt werden soll. Gibt es diese Möglichkeit nicht, kann natürlich über eine weitere Trace Query ein Resultat angegeben werden, das innerhalb der Meldung der Reaktion verfügbar gemacht werden soll. Tabelle 26 zeigt eine beispielhafte Spezifikation zweier verschiedener Reaktionen.

Tabelle 26: Formalisierung der Reaktion

Gewünschte Ausdrucksmöglichkeit	Formalisierung
Zeige ein Warn-Symbol an ohne den Nutzer zu unterbrechen	Warnung-Art: warn-symbol Warnmeldung: „Mit diesem Use Case-Schritt sind keine User Stories verknüpft. Möglicherweise wird er in der Implementierung vergessen.“ Trace Query anzeigen?: false
Zeige einen Warn-Dialog an, durch den der Nutzer unterbrochen wird. Zeige darin auch das Resultat der Trace Query an.	Warnung-Art: warn-dialog Warnmeldung: „Die User Story, die Sie geändert haben, ist mit Use Case-Schritten verknüpft. Bitte prüfen Sie, ob die neue User Story noch mit diesen Use Case-Schritten zusammenpasst“ Trace Query anzeigen?: true

8.4.5 Grafikbasierte Formalisierung von Trace-Operationen basierend auf VTML

VTML wurde von Mäder und Cleland-Huang vorgestellt [109]. Es wird davon ausgegangen, dass das Artefaktmodell eines Projektes in Form eines Traceability Information Models (siehe [110]) vorliegt. VTML-Queries werden dann als Teilmodelle dieses Traceability Information Models modelliert, in denen genau die für eine Abfrage zu berücksichtigenden Artefakt- und Verknüpfungstypen enthalten sind. Durch zusätzliche Symbole, die alle als Stereotypen der UML-Klassendiagrammnotation angesehen werden können, lassen sich Einschränkungen und Aggregationen vornehmen. Die folgende Abbildung zeigt in Anlehnung an [109] die VMTL-Notationselemente anhand einer Beispiel-Abfrage.

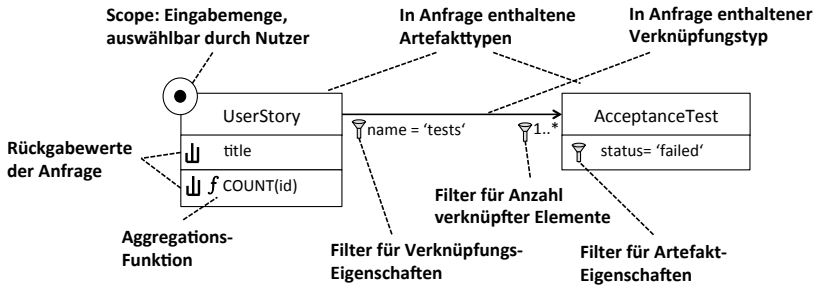


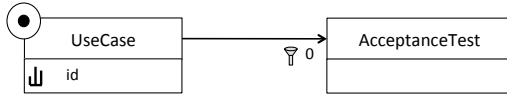
Abbildung 52: VTML-Notationselemente, aus [109]

Um ausführbare Queries zu erzeugen, müssen VTML-Queries innerhalb eines Werkzeugs in ein Format übersetzt werden, mit dem auf die eigentlichen Daten zugegriffen werden kann. Prinzipiell ist eine Übersetzung in verschiedene Formate möglich. Mäder und Cleland-Huang demonstrieren in [109] eine Übersetzung in SQL-Abfragen. Dazu wird davon ausgegangen, dass die Anforderungen in einer Datenbank abgespeichert sind, deren Schema dem Traceability Information Model entspricht. Das bedeutet, dass für jeden Artefakttyp eine Tabelle mit den gespeicherten Artefakten und für jeden Verknüpfungstyp eine Tabelle mit den gespeicherten konkreten Verknüpfungen – repräsentiert durch ein Fremdschlüsselpaar auf die beiden verknüpften Artefakte – existiert.

VTML-Queries eignen sich gut, um Teile von Trace-Operationen darzustellen. Durch ihre grafische Art erlauben sie es, komplexe Navigationspfade leicht visuell darzustellen. Außerdem kann auch hier ein Traceability Information Model zur leichten Ableitung herangezogen werden, indem das Anforderungslandschaftsmodell verwendet wird. So können auch Projektteilnehmer über den Aufbau der Anforderungslandschaft und ihre anforderungsbezogenen Aktivitäten darin diskutieren. Dabei bleiben sie im abstrakten Kontext der Anforderungslandschaft und müssen nicht die Details der eigentlichen Datenhaltung im RE-Tool berücksichtigen.

Die Verwendung der Trace Queries unterscheidet sich im hier vorliegenden Ansatz etwas von der Verwendung, wie bei Mäder und Cleland-Huang vorgestellt. Abbildung 53 zeigt zur Veranschaulichung jeweils eine typische Trace Query für beide Ansätze. Bei Mäder und Huang (a) (Query stammt aus [109]) wird häufig nach mehreren Elementen eines Eingangs-Artefakttyps gefragt. Die Eingangsmenge kann eingeschränkt werden, aber das Resultat ist dennoch häufig eine Teilmenge der Eingangsmenge. Für die Trace-Operationen in dieser Arbeit (b) werden jedoch ausgehend von *einem einzelnen Eingangs-Element* Informationen über die verknüpften Artefakte abgefragt. Für eine Trace-Operation besteht also immer der Sonderfall, dass die Eingangsmenge aus genau einem Element besteht.

(a) Finde alle Use Cases, zu denen es keine Abnahme-Testfälle gibt.



(b) Zu einem konkreten Use Case, gebe die Anzahl an Abnahme-Testfällen an.



Abbildung 53: Verschiedene Arten von Abfragen. (a) sucht alle Elemente, die eine bestimmte Eigenschaft erfüllen; (b) sucht zu einem konkreten Element zugehörige Informationen über verknüpfte Elemente

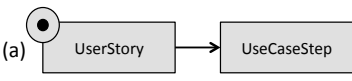
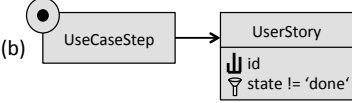
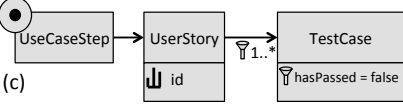
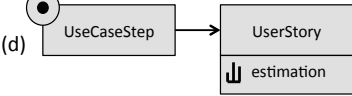
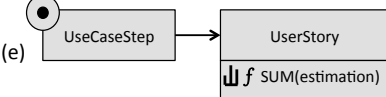
Die Variante (a) wird eher verwendet, um eine Tracing-Aktivität durchzuführen. Eine Abfrage wie in (b) hat einen höheren Bezug zu anforderungsbezogenen Aktivitäten. Außerdem werden im hier vorgestellten Ansatz Trace Queries, zumindest bei Auslösern *statisch* und *Methodenaufwurf*, automatisiert vom System aufgerufen, anstatt durch den Nutzer selbst. Entsprechend wird auch das Eingangsartefakt durch das System und nicht den Nutzer festgelegt, nachdem auf einem Artefakt eine Operation durchgeführt wurde, die eine Trace-Operation auslöst. Diese Arbeit stellt also eine Erweiterung der Verwendungweise von VTML- bzw. Trace-Queries dar, die notwendig ist, um anforderungsbezogene Aktivitäten zu unterstützen.

Neben der eigentlichen Trace Query enthält eine Trace-Operation aber noch weitere Informationen, wie einen Auslöser und eine Kritik. Um die Trace Query-Notation nicht zu überfrachten, müssen Teile dieser Informationen separat angegeben werden. Im Folgenden wird darauf eingegangen, wie die einzelnen Bestandteile formalisiert werden können, um eine Trace-Operation zu beschreiben.

Trace Query

Die Trace Query kann vollständig mit einer VTML-Query abgebildet werden. Tabelle 27 zeigt die benötigten Navigationsmöglichkeiten aus Abschnitt 8.4.2 und die entsprechenden VTML-Query-Elemente.

Tabelle 27: Formalisierung von Navigationselementen mittels VTML

Gewünschte Ausdrucksmöglichkeit	VTML-Notation
Von einem Ausgangselement aus zu verknüpften Artefakten navigieren	(a) 
Aus einer Artefaktmenge Artefakte mit bestimmten Eigenschaften filtern, basierend auf Attributen	(b) 
Aus einer Artefaktmenge Artefakte mit bestimmten Eigenschaften filtern, basierend auf komplexer Filter-Bedingung (hier: Story besitzt fehlgeschlagene Tests)	(c) 
Nur bestimmte Attribute der Ergebnis-Artefakte zurückgeben	(d) 
Eine Aggregationsfunktion (SUM, COUNT, MIN, MAX, AVG) auf die Ergebnis-Elemente anwenden	(e) 

Bedeutung der Beispiel-Queries: (a) Navigiert von einer bestimmten userStory zur Menge ihrer verknüpften UseCaseSteps. (b) Gibt zu einem UseCaseStep alle Stories aus, die noch nicht erledigt sind. (c) Gibt zu einem UseCaseStep alle verknüpften Stories aus, die mindestens einen fehlgeschlagenen Testfall besitzen. (d) Gibt zu einem UseCaseStep eine Menge mit den Schätzwerten aller verknüpften Stories aus. (e) Gibt zu einem UseCaseStep die Summe der Schätzwerte aller verknüpften UserStories – als Indikator für den eigenen Schätzwert der Umsetzung des gesamten UseCaseSteps – aus.

Auslöser

Durch Erweiterung der VTML-Notation lässt sich zusammen mit der Query auch ein Auslöser mit einer optionalen Bedingung festhalten. Der Auslöser wird in die Trace Query-Notation integriert, indem das Symbol, das für die Eingabemenge steht, erweitert wird. Für den Auslöser muss einer der Werte *statisch*, *Nutzeranfrage*, oder *Methodenaufruf* spezifiziert werden. Dies kann entsprechend durch unterschiedliche Symbole passieren, wie in Abbildung 54 angegeben. Wie die Abbildung außerdem zeigt, ist für die dynamischen Kritiken zusätzlich der Name der Methode anzugeben, bei deren Aufruf die Kritik ausgelöst wird. Eine **Bedingung** für den Auslöser kann ebenfalls neben dessen Symbol angegeben werden. Dieser kann wie ein weiterer Filter angegeben werden, der sich auf den Kontext-Artefakttyp bezieht.

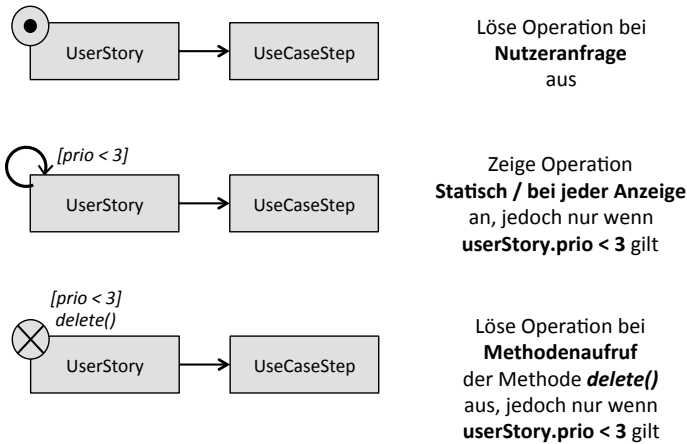


Abbildung 54: Erweiterung der VTML-Notation zur Repräsentation von Auslösern und Bedingungen

Kritik

Die **Überprüfung** des Resultates, nach der entschieden wird, ob eine Kritik angezeigt wird, kann nicht mittels VTML-Notation dargestellt werden. Diese muss separat spezifiziert werden. Mit der Prüfung wird das Resultat der Trace Query überprüft. Das Resultat sollte also als zusätzliche Variable oder als Schlüsselwort angesprochen werden können.

Wie bereits in Abschnitt 8.4.2 erwähnt, sollte es möglich sein, einen einzelnen *Resultatwert* mittels boolescher Ausdrücke zu überprüfen, aber auch das Resultat als eine *Menge* aufzufassen und entsprechende Operationen, wie Aggregationen und Überprüfungen von Quantoren durchzuführen. Prinzipiell können wieder verschiedene Sprachen, wie Java, OCL oder eigene Sprachen, zur Formalisierung der Prüfung verwendet werden. In dieser Arbeit wird die Überprüfung mittels OCL-Ausdrücken beschrieben, da diese bereits aus der OCL-basierten Methode (Abschnitt 8.4.4) bekannt sind. Der Zugriff auf das Resultat erfolgt hier über einen speziellen Variablennamen ‚result‘.

Die **Reaktion** kann, wie auch bereits bei der OCL-Variante im vorigen Abschnitt, wieder über Festlegung einer Kritik-Art und einer Nachricht geschehen. Zusätzlich kann hier über ein Schlüsselwort angegeben werden, dass das Resultat der Trace Query ebenfalls ausgegeben werden soll.

8.5 Ausführung der Trace-Operationen in einem Werkzeug

Nachdem Trace-Operationen nun formal beschrieben werden können, stellt sich die Frage, wie diese nun *angewendet* werden, um Projektteilnehmer in anforderungsbezogenen Aktivitäten zu unterstützen. Wie bereits erwähnt, wird davon ausgegangen, dass die Anforderungen mithilfe eines CASE- bzw. RE-Tools verwaltet werden. Dieses RE-Tool sollte nun während die Projektteilnehmer darin mit Anforderungen arbeiten, Trace-Operationen ausführen, um die Projektteilnehmer zu unterstützen. Hierzu wird in diesen Abschnitt vorgestellt, wie sich das Konzept von Trace-Operationen in ein RE-Tool einbinden lässt, sodass das Werkzeug basierend auf VTML- bzw. OCL-Modellen Trace-Operationen auf seiner Anforderungsmenge ausführt.

Die Trace-Operationen gehen immer von einem bestimmten zugrundeliegenden Artefaktmodell aus. Im vorigen Abschnitt wurde dazu beispielsweise stets ein Referenzmodell, wie in Abbildung 47 und Abbildung 51, angegeben.

Im Extremfall könnte der Requirements Methodiker dieses Artefaktmodell projektspezifisch festlegen und als Konfiguration für das RE-Tool verwenden. Entsprechende Ansätze wurden im Zuge von Meta-CASE-Tools erforscht [4]. Damit würde aber ein hoher Konfigurationsaufwand einhergehen, um beispielsweise für alle dort definierten Artefakttypen geeignete Anzeigen und Methoden so zu beschreiben, dass ein RE-Tool dies automatisiert umsetzen kann.

In dieser Arbeit wird daher davon ausgegangen, dass ein RE-Tool mit einem festen Artefaktmodell vorliegt, welches durch seine Modell-Klassen und ihre Verknüpfungen festgelegt ist. Der Requirements Methodiker legt dann mit einem Anforderungslandschaftsmodell für die Projektteilnehmer fest, welche der angebotenen Artefakt- und Verknüpfungstypen sie verwenden sollen.

Die Trace-Operationen nun wiederum sollen individuell für jedes Projekt eingegeben werden können. Zwar gibt es auch hier die Möglichkeit, einige besonders häufige Operationen fest vorzugeben und den Requirements Methodiker dann nur noch entscheiden zu lassen, ob eine Operation verwendet werden soll oder nicht. Doch treten hier eher projektspezifische Erfordernisse auf, die nicht alle im Vorhinein festgehalten werden können.

Es gibt drei Möglichkeiten, Trace-Operationen in ein RE-Tool einzugeben, sodass dieses die Operationen dann auf seine Daten anwendet. Diese sind in Abbildung 55 zusammengefasst.

Die Regeln lassen sich zunächst manuell *fest codieren*, was bedeutet, dass der bestehende Code einfach um die gewünschte Funktionalität ergänzt wird. Beispielsweise wird dann die *löschen()*-Methode einer User Story um zusätzlichen Code ergänzt, der die verlinkten Use Case-Schritte überprüft. Dazu müssen die Trace-Operationen bei der Erstellung des CASE-Tools bekannt sein oder nachträglich implementiert werden können. Diesen Ansatz verfolgt beispielsweise

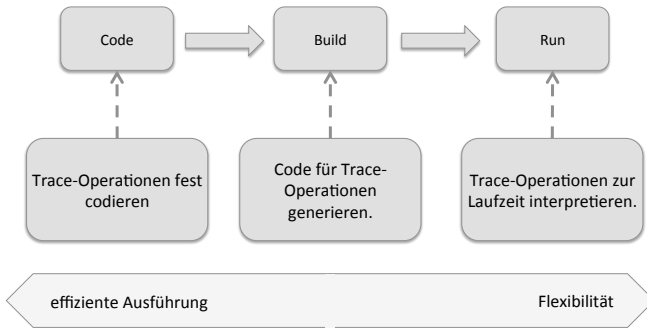


Abbildung 55: Verschiedene Ansätze zur Verwendung von Trace-Operationen im RE-Tool

Zettel [165], indem er zeigt, wie ein CASE-Tool basierend auf Konsistenzregeln in OCL implementiert wird.

Prinzipiell ist mit dieser Methode die effizienteste Ausführung erreichbar, da einfach statischer Code erzeugt und direkt in die bestehenden Strukturen eingefügt wird. Der Code operiert direkt auf den Laufzeit-Objekten, wie zum Beispiel einem Objekt `userStory12:UserStory`, ohne dass zusätzliche Übersetzungen innerhalb des Programms notwendig sind. Es ist aber auch die unflexibelste Methode, weil die Trace-Operationen außerhalb der Tool-Erstellung – bis auf kleine Anpassungen, wie dem Warn-Typ oder der generellen Aktivierung/Deaktivierung einer Operation – gar nicht änderbar sind. Insbesondere lassen sich so keine neuen Regeln erstellen, ohne dass wieder der Code geändert wird.

Diese Variante ist denkbar, weil die Regeln nicht sehr häufig geändert werden müssen. Wenn Zugriff auf den Code besteht und der Requirements Methodiker programmieren kann oder wenn er Zugriff zu Personen hat, die das Tool für ihn anpassen können, so kann diese Methode gewählt werden, bringt aber immer eine gewisse Inflexibilität in Bezug auf die schnelle Anpassung von Trace-Operationen mit sich.

Auf diese Variante wird im Folgenden nicht weiter eingegangen. Der zu erstellende Code lässt sich relativ simpel manuell aus den Formalisierungen ableiten. Meist werden nur verlinkte Elemente überprüft oder Listenelemente aggregiert, was zumeist bereits mit vordefinierten Funktionen möglich ist.

Will man nun auch während der Verwendung des CASE-Tools projektspezifische Trace-Operationen festlegen können, so muss eine Möglichkeit geschaffen werden, Trace-Operationen auch nach der Codierung in das System einzubringen.

Die erste dieser Möglichkeiten besteht in der **Code-Generierung**. Während bereits die grobe Struktur des RE-Tools existiert, können Trace-Operationen in Konfigurationsdateien festgelegt werden. Beim Bauen des Tools (dem sogenannten *Build*) werden dann durch Generierung weitere Klassen-Dateien erzeugt, die den Code in der Ziel-Programmiersprache enthalten. Im Gegensatz zur vorhergehenden Methode, muss hierfür ein generischeres System-Design geschaffen werden, das explizit beliebige Trace-Operation-Klassen ausführen und deren Ergebnisse anzeigen kann.

Die Ausführung dieser Variante ist nach wie vor effizient, da auch hier letztlich statischer Code erzeugt wird. Sie ist jedoch ineffizienter als die vorhergehende Variante mit der festen Codierung, da die Möglichkeit, generisch auf beliebige Trace-Operationen eingehen zu können, in der Regel zusätzlichen Aufwand erfordert. Beispielsweise müssen die verwendeten Trace-Operationen nun in einer zusätzlichen Liste gehalten werden, die unter Umständen häufig durchsucht werden muss.

Gleichzeitig ist diese Variante flexibler, da nun für Änderungen nicht mehr der Code angepasst werden muss, sondern nur noch Konfigurationsdateien verändert und das Werkzeug neu gebaut werden muss. Auch diese Variante ist plausibel, da davon auszugehen ist, dass der Requirements Methodiker einen vorgefertigten Build-Prozess anstoßen kann und die Regeln nicht sehr häufig verändert werden müssen. Die Variante der Code-Generierung wird im folgenden Abschnitt 8.5.1 erläutert.

Die dritte Möglichkeit zur Eingabe von Trace-Operationen ist, die Trace-Operationen über eine Konfigurationsdatei oder eine Maske im Werkzeug selbst zur Laufzeit einzugeben, sodass diese vom CASE-Tool ebenfalls **zur Laufzeit interpretiert** werden. Eine Trace-Operation stellt damit dynamischen Code dar, der die Ausführung von Funktionen im Werkzeug zur Laufzeit steuert.

Dabei kann die Trace-Operation beim Einlesen auch umgewandelt – beispielsweise in eine SQL-Anfrage oder gar Java-Code – und in einer Datenbank abgelegt werden. Für die Ausführung wird sie dann aber immer zur Laufzeit interpretiert.

Diese Variante ist sehr flexibel, da damit jederzeit neue Regeln eingegeben oder bestehende verändert werden können. In der Ausführung kann sie jedoch ineffizient werden. Neben den generischen Mechanismen zur Ausführung einer beliebigen Trace-Operation, wie bei der Generierung, wird zusätzlich eine Logik benötigt, welche die Formalisierungen nun auf die Ausführung der bestehenden Methoden im statischen Code übersetzt. Bei jeder Ausführung einer Trace-Operation wird ein zusätzlicher Interpretations-Schritt notwendig. Beispielsweise muss beim Einlesen des Ausdrucks „*COUNT(id)*“ die Methode *.size()* auf

einem Listenobjekt ausgelöst werden. Hierzu sind zusätzliche Pasing- oder Zuordnungsmechanismen, wie beispielsweise Java Reflections, notwendig, welche die Performanz verschlechtern.

Diese Variante wird im Abschnitt 8.5.2 vorgestellt.

Um eine geeignete Mischung aus Flexibilität und Performanz zu erreichen, sind auch Kombinationen der angegebenen Methoden möglich. Trace-Operationen, die in vielen Projekten verwendet werden, werden im Tool fest codiert. Diese können dann bei Bedarf durch eigene Trace-Operationen, die zur Laufzeit interpretiert werden müssen, ergänzt werden.

Die Variante mit dem geringsten Aufwand für die Umsetzung wäre die Verwendung eines bereits existierenden Interpreters für OCL-Ausdrücke. Damit können Formalisierungen, wie in Abschnitt 8.4.4 angegeben, direkt eingelesen werden, um das Modell damit zu untersuchen. Die praktische Umsetzung dieser Variante gestaltet sich jedoch schwierig, da die meisten OCL-Interpreter auf dem *Eclipse Modeling Framework (EMF)* aufsetzen und verlangen, dass die Daten, in einem *.ecore*-Modell vorliegen. Dies passt jedoch nicht zu der Voraussetzung, dass mit der vorgestellten Methode bestehende CASE-Tools erweitert werden können, da diese ihre Daten meist in Datenbanken halten und auf einem Instanzmodell beispielsweise aus Java-Objekten operieren. Daher werden in dieser Arbeit auch alternative Methoden detaillierter untersucht.

In den folgenden zwei Abschnitten wird dazu jeweils ein Softwareentwurf vorgestellt, der zeigt, wie eine Software ergänzt wird, um die Verwendung von Trace-Operationen durch Generierung bzw. Interpretation zu ermöglichen. Anschließend werden in Abschnitt 8.5.3 konzeptuelle Übersetzungstabellen genannt, die als Grundlage für die eigentliche Generierung bzw. Interpretation dienen.

8.5.1 Softwareentwurf für die Verwendung und Auswertung generierter Trace-Operationen

In diesem Abschnitt wird der Aufbau einer Softwarekomponente gezeigt, welche die Verwaltung und Auswertung von Trace Operationen, die durch Generierung erstellt werden, durchführt. Die Komponente lässt sich in bestehende Software integrieren, indem Klassen, die Anforderungsartefakte darstellen, eine abstrakte Klasse *Artifact* erweitern. Abbildung 56 zeigt den Aufbau anhand eines Klassendiagramms. Zur besseren Übersichtlichkeit sind zu den Klassenattributen keine Getter- und Setter-Methoden angegeben. Die Notizen enthalten Pseudo-Code, der zeigt, wie bestimmte Methoden-Teile implementiert werden können.

8.5 Ausführung der Trace-Operationen in einem Werkzeug

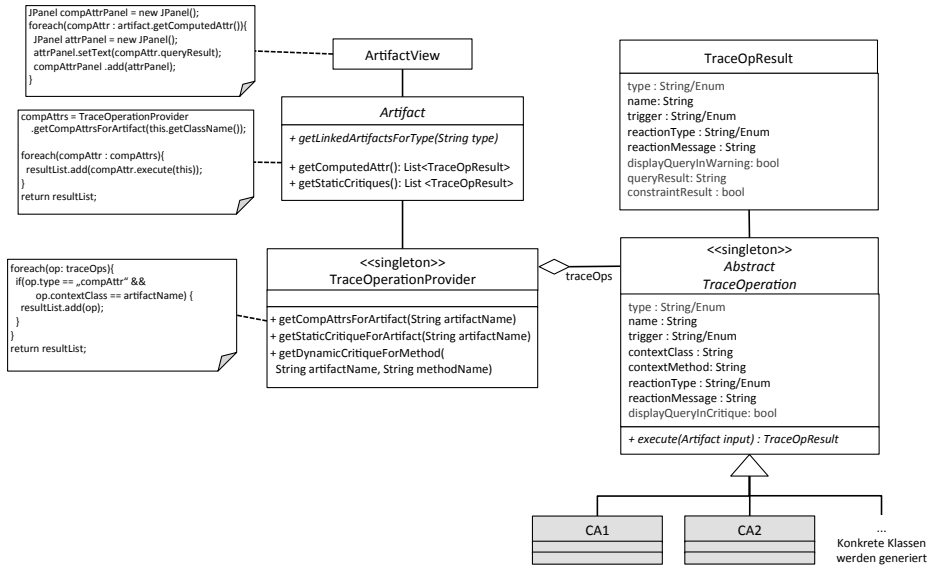


Abbildung 56: Softwareentwurf für System, das generierte TraceOperationen verarbeitet

Im Zentrum des Systems steht die Klasse *TraceOperation*, die alle Informationen zu einer Trace Operation kapselt. Mittels der Methode *execute()* wird diese in Bezug auf ein konkretes Artefakt ausgeführt. Das Resultat wird dann in einem *TraceOpResult*-Objekt gekapselt. Dabei werden auch weitere Informationen zur Trace Operation, wie der Warn-Typ (*reactionType*) oder die Warn-Meldung (*reactionMessage*) gespeichert, damit diese später bei der Anzeige durch die *ArtifactView* berücksichtigt werden können. Die konkreten Trace Operationen werden im Generierungs-Schritt aus den VTML- bzw. OCL-Dateien erzeugt. Dabei wird insbesondere die Methode *execute()* generiert, welche die Trace Query und Überprüfung der Trace Operation innerhalb des Programms durchführt. Auf die Generierung der TraceOperation-Klassen aus den VTML- bzw. OCL-Formalisierungen wird in Abschnitt 8.5.3 eingegangen.

Die Klasse *TraceOperationProvider* enthält alle Trace Operationen des Systems und ordnet diese den Artefakttypen und Methoden zu, auf welche die Trace-Operationen sich beziehen. Ein Artefakt gelangt so an genau die Trace-Operationen, die es auswerten muss.

Alle Artefakte des Systems müssen die abstrakte Klasse *Artifact* erweitern. Die *TraceOperation*-Klassen arbeiten auf *Artifact*-Objekten. Damit es in ihrer *execute*-Methoden möglich ist, zu verknüpften Artefakten zu navigieren, implementieren *Artifact*-Objekte die abstrakte Methode *getLinkedArtifactsForType()*.

Über die Methoden *getComputedAttrs()* und *getStaticCritiques()* stellt ein *Artifact*-Objekt alle auf es bezogenen *TraceOpResults* zur Verfügung. Beim Aufruf

der Methoden stößt das *Artifact*-Objekt die Auswertung an und gibt die *TraceOpResult*-Objekte zurück.

Die Anzeige der Ergebnisse einer Trace Operation geschieht in der *Artifact-View*. Für ein konkretes Artefaktobjekt geht die Anzeige alle *TraceOpResult*-Objekte durch, erstellt Unter-Sichten und zeigt darin die Auswertung an.

Eine beispielhafte generierte Klasse, die eine konkrete *TraceOperation* darstellt, ist im Folgenden dargestellt. Es ist zu sehen, dass die eigentliche Ausführung der Trace Query und der Überprüfung mittels Java-Code in der Methode *execute()* bzw. deren Untermethoden geschieht. Dies ist in der alternativen Variante, der Interpretation, auf die im folgenden Abschnitt 8.5.2 eingegangen wird, anders. Wie die Methode *execute()* aus den verschiedenen VTML- bzw. OCL-Formalisierungen erstellt wird, wird in Abschnitt 8.5.3 erläutert.

```
public class StK5 extends TraceOperation{
    private static TraceOperation instance = null;

    final TraceOperationType type = TraceOperationType.StaticCritique;
    final String name = „NoLinkedUseCaseSteps“;
    final TriggerType trigger = TriggerType.StaticTrigger;           // Auslöser
    final String contextClass = „UserStory“;
    final String contextMethod = null;
    final ReactionType reactionType = ReactionType.Warn;           // Reaktion
    final String ReactionMessage reactionMessage = „Die User Story besitzt keinen
        verknüpften Use Case-Schritt. Möglicherweise ist sie überflüssig.“
    final boolean displayQueryInCritique = false;

    private StK5(){
    }

    public static TraceOperation getInstance(){
        if(instance == null) instance = new StK5();
        return instance;
    }

    public TraceOpResult execute(Artifact input){
        TraceOpResult result = new TraceOpResult(this.type, this.name, this.trigger,
            this.reactionType, this.reactionMessage, this.displayQueryInCritique);

        UserStory userStory = (UserStory) input;
        String queryResult = this.executeQuery(userStory);
        result.queryResult = queryResult;
        result.constraintResult = this.executeQuery(queryResult);
        return result;
    }

    private String executeQuery(UserStory userStory){
        if(userStory.prio < 3) {                                     // Bedingung
            List<Artifact> useCaseSteps =                             // Trace Query
                userStory.getLinkedArtifactsForType(„UseCaseStep“);
            return String.valueOf(useCaseSteps.size());
        }
    }

    private boolean executeConstraint(String result){
        int resultAsInt = Integer.parseInt(result);
        return resultAsInt == 0;                                     // Überprüfung
    }
}
```

8.5.2 Softwareentwurf für die Verwendung und Auswertung von Trace-Operationen, die zur Laufzeit interpretiert werden

Der Grundaufbau einer Softwarekomponente, die Trace-Operationen verwendet, welche zur Laufzeit interpretiert werden, sieht sehr ähnlich zu dem Aufbau im letzten Abschnitt aus, bei dem die Trace-Operationen als eigenständige Klassen ins Programm generiert wurden. Hier werden nun Trace-Operationen nicht durch neue Klassen einbezogen, sondern durch Objekte, die zur Laufzeit erstellt werden. Abbildung 57 zeigt den entsprechenden Softwareentwurf.

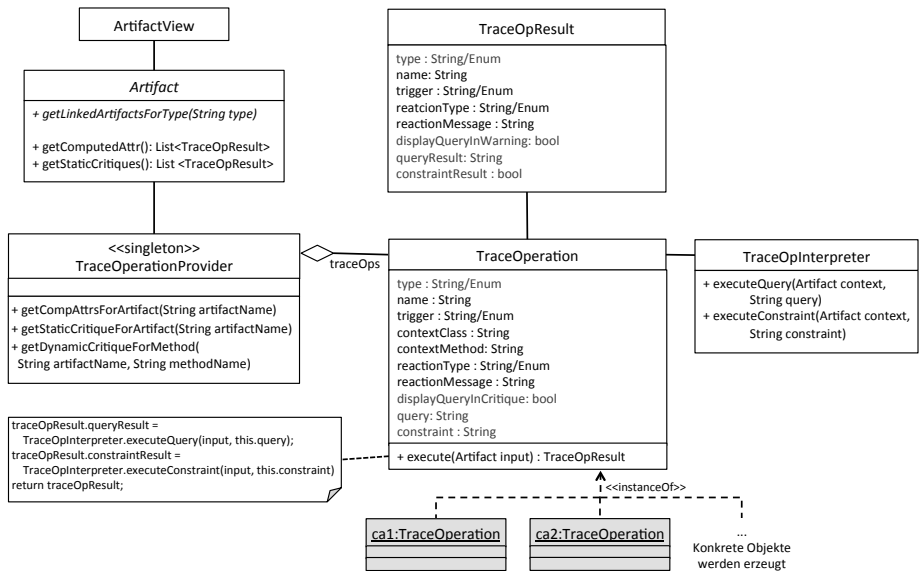


Abbildung 57: Softwareentwurf für System, das Trace-Operationen zur Laufzeit verarbeitet

Die Klasse *TraceOperation* ist nun nicht mehr abstrakt, sondern eine normale Klasse. Für jede Trace Operation wird nun ein neues Objekt erzeugt. Entsprechend sind Trace Query und Constraint nun nicht mehr in der Methode *execute()* enthalten, sondern werden zu jedem Objekt als Attribut gespeichert. Jede TraceOperation führt nun die Methode *execute()* so aus, dass sie die Attribute für *query* und *constraint* dem *TraceInterpreter* übergibt, welcher nun die Query ausführt.

Es gibt zwei denkbare Möglichkeiten, die VTML- bzw. OCL-Formalisierungen einzulesen und zu verarbeiten. Abbildung 58 zeigt hierzu einen Überblick. Zum einen kann eine VTML- bzw. OCL-Formalisierung direkt interpretiert werden. Hierzu ist ein Interpreter notwendig, der die VTML bzw. OCL-Formalisierung

versteht und die Trace Operationen in Aktionen auf dem Objektmodell oder direkt auf der Datenbank übersetzt. Ein solcher Interpreter muss unter Umständen explizit erstellt werden.

Die alternative Variante besteht darin, die VTML- bzw. OCL-Formalisierungen beim Einlesen in eine weitere Sprache zu übersetzen. Dann wird ein Interpreter in dieser entsprechenden anderen Sprache benötigt. Hier bieten sich Sprachen, wie SQL, Java oder JavaScript an, da hierzu bereits Interpreter existieren. Die zu implementierenden Anweisungen zur Übersetzung in Code – beispielsweise Java-Code – der dann dynamisch ausgeführt wird, sind dann sehr ähnlich zu denen, die für die Generierung, wie im vorigen Abschnitt vorgestellt, verwendet werden. Hierzu lässt sich entsprechend der folgende Abschnitt 8.5.3 heranziehen.

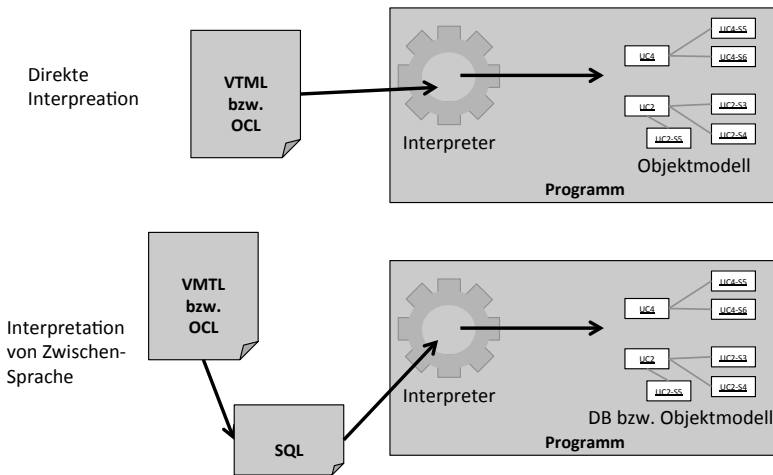


Abbildung 58: Alternativen zur Interpretation von Trace-Operations-Formalisierungen

Eine Möglichkeit, die besonders bei der Verwendung von OCL naheliegend ist, ist die Verwendung eines bestehenden OCL-Interpreters. Für die in dieser Arbeit vorliegende Situation gibt es dafür zwei Voraussetzungen:

- Der Interpreter muss auf Daten, die in einer Datenbank liegen, oder auf die Modellelemente, wie Java-Objekte, zugreifen können.
- Der Interpreter muss sich als Bibliothek in ein bestehendes CASE-Tool einbinden lassen.

Im Java-Bereich gibt es zur Zeit keine existierende Software, die beide Voraussetzungen erfüllt. Die bekannten Interpreter basieren auf dem *Eclipse Modeling Framework (EMF)*. Da mit dem EMF direkt Referenz-Modelle erzeugt werden können, auch welche die OCL-Ausdrücke sich beziehen, bietet sich diese Herangehensweise an. Allerdings verwenden diese Interpreter dann zur Eingabe der

Modell-Instanzen *.ecore*-Dateien, anstelle von Datenbanken oder den Modellobjekten in der Programmiersprache.

Mit Dresden-OCL [37], [161] existiert zwar ein Interpreter, der auf dem Java-Objektmodell operiert oder OCL-Ausdrücke in SQL umwandeln kann. Jedoch handelt es sich dabei um ein Eclipse Plugin und nicht um eine Bibliothek, die in ein bestehendes Programm eingebunden werden kann. Genauso gibt es für VTML-Queries nach aktuellem Wissensstand nur die Möglichkeit, sie mittels eines XSLT-Stylesheets in SQL-Anfragen umzuwandeln [109]. Ein passender Interpreter ist beispielsweise unter Verwendung der Java Reflection API [118], [123] denkbar, müsste aber in jedem Fall zuerst erstellt werden.

Eine bessere existierende Interpreter-Unterstützung gibt es hingegen für SQL-Anfragen oder die dynamische Ausführung von Programmiersprachen, wie Java-Code. Daher wird hier diese Variante weiter verfolgt. Mittels einer Umwandlung der VTML- bzw. OCL-Ausdrücke in SQL- oder Java-Code kann zur Ausführung der Trace-Operationen ein bestehender Interpreter verwendet werden. Die Umwandlung in Java-Code ist sehr ähnlich zur Generierung, wie im vorigen Abschnitt vorgeschlagen. Sie wird konzeptuell vom folgenden Abschnitt 8.5.3 abgedeckt.

Analog zum Beispiel einer generierten Klasse aus Abschnitt 8.5.1, wird hier ein entsprechendes Beispiel-Objekt gezeigt, das eine Trace Operation für die Auswertung zur Laufzeit kapselt.

<u>stK5:TraceOperation</u>	
type	= TraceOperationType.StaticCritique;
name	= „NoLinkedUseCaseSteps“;
trigger	= TriggerType.StaticTrigger;
contextClass	= „UserStory“;
contextMethod	= null;
reactionType	= ReactionType.Warn
reactionMessage	= „Die User Story besitzt keinen verknüpften Use Case-Schritt. Möglicherweise ist sie überflüssig.“
displayQueryInCritique	= false;
query	= „if(self.prio < 3) { List<Artifact> useCaseSteps = self.getLinkedArtifactsForType(„UseCaseStep“); return String.valueOf(useCaseSteps.size());“
constraint	= „int resultAsInt = Integer.parseInt(result); return resultAsInt == 0;“

8.5.3 Konzeptuelle Umwandlung

Wie in den beiden vorausgegangenen Abschnitten gezeigt, müssen für die in dieser Arbeit vorgestellte Vorgehensweise die Trace-Operationen in Code umgewandelt werden. Dies geschieht entweder innerhalb eines Generierungsschrittes zur Build-Zeit oder innerhalb einer Umwandlung zur Laufzeit. In diesem Abschnitt wird an Beispielen gezeigt, in welchen Code die einzelnen Konstrukte umgewandelt werden müssten. Auf die Ausführung, wie genau die Umwandlung durchgeführt wird wird hier verzichtet.

Wie die einzelnen Elemente in eine Klasse bzw. ein Objekt gespeichert werden, wurde am Beispiel bereits in den beiden vorausgegangenen Abschnitten gezeigt. Hier wird noch einmal die Beispiel-Klasse zusammen mit der Trace-Operations-Formalisierung gezeigt. Die wichtigsten und variabelsten Teile sind die Trace Query und die Überprüfung für eine Kritik. Für diese wird im Anschluss näher beschrieben, wie die verschiedenen Ausdrücke übersetzt werden.

Als Beispiel wird die Trace-Operation StK5 verwendet:

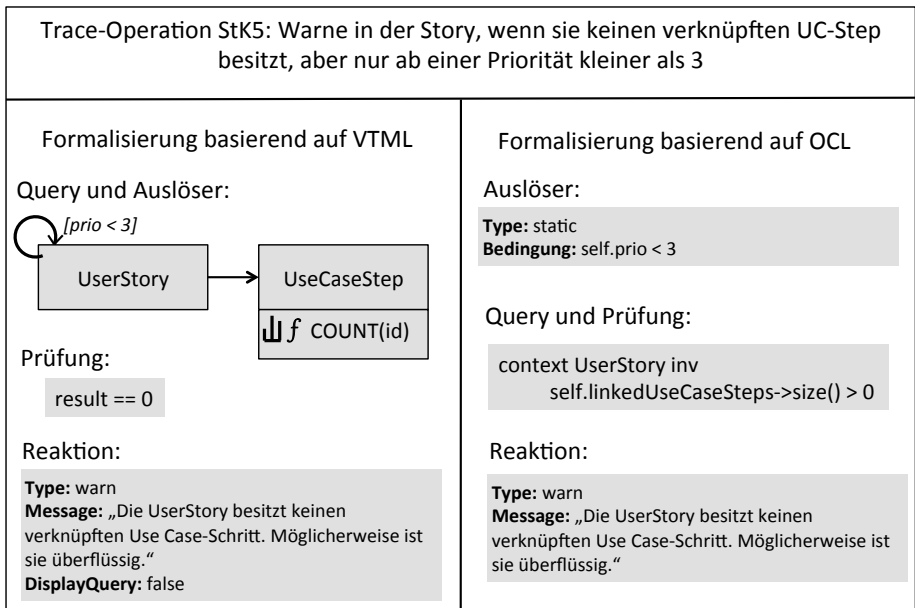


Abbildung 59: Trace-Operations-Beispiel zur Demonstration der Umwandlung in Code

Die daraus zu erstellende Klasse ist im Folgenden aufgeführt. Aus Gründen der Übersichtlichkeit werden die Attribute der Klasse als *protected* dargestellt, statt sie als *private* zu deklarieren und die entsprechenden Getter und Setter anzubieten. Außerdem sind sämtliche Überprüfungen von Ausnahme- bzw. Fehlersituationen ausgelassen.

```

public class StK5 extends TraceOperation{
    private static TraceOperation instance = null;

    final TraceOperationType type = TraceOperationType.StaticCritique;
    final String name = „NoLinkedUseCaseSteps“;
    final TriggerType trigger = TriggerType.StaticTrigger;           // Auslöser
    final String contextClass = „UserStory“;
    final String contextMethod = null;
    final ReactionType reactionType = ReactionType.Warn;           // Reaktion
    final String ReactionMessage reactionMessage = „Die User Story besitzt keinen
        verknüpften Use Case-Schritt. Möglicherweise ist sie überflüssig.“
    final boolean displayQueryInCritique = false;

    private StK5(){
    }

    public static TraceOperation getInstance(){
        if(instance == null) instance = new StK5();
        return instance;
    }

    public TraceOpResult execute(Artifact input){
        TraceOpResult result = new TraceOpResult(this.type, this.name,
            this.trigger, this.reactionType, this.reactionMessage,
            this.displayQueryInCritique);

        UserStory userStory = (UserStory) input;

        String queryResult = this.executeQuery(userStory);
        result.queryResult = queryResult;
        result.constraintResult = this.executeQuery(queryResult);

        return result;
    }

    private String executeQuery(UserStory userStory){
        if(userStory.prio < 3) {                                     // Bedingung
            List<Artifact> useCaseSteps =                             // Trace Query
                userStory.getLinkedArtifactsForType(„UseCaseStep“);
            return String.valueOf(useCaseSteps.size());
        }
    }

    private boolean executeConstraint(String result){
        int resultAsInt = Integer.parseInt(result);
        return resultAsInt == 0;                                     // Überprüfung
    }
}

```

Um zu zeigen, wie die verschiedenen VTML- bzw. OCL-Ausdrücke umgewandelt werden, werden in Tabelle 28 die Beispiele für Trace Queries und Überprüfungen übersetzt, die in Tabelle 24, Tabelle 25 und Tabelle 27 (Abschnitt 8.4) als Repräsentanten der gewünschten Ausdrucksmöglichkeiten vorgestellt wurden.

Tabelle 28: Übersetzung von OCL-Ausdrücken in Java-Code

OCL-Notation	Java-Code
<pre>(a) context UserStory def: eineQuery:resultType = ... self.useCaseStep ...</pre>	<pre>UserStory userStory = (UserStory)input; List<Artifact> allSteps = userStory .getLinkedArtifactsForType(„UseCaseStep“);</pre>
<pre>(b) context UseCaseStep def: pendingStories : Set<UserStory> = self.userStory->select(st st.status<>,done')</pre>	<pre>List<Artifact> allStories = step .getLinkedArtifactsForType(„UserStory“); List<Artifact> result = new ArrayList<Artifact>(); for(UserStory st : allStories){ if(((Story)st).status != „done“){ result.add(st); } } return result;</pre>
<pre>(c) context UseCaseStep def: failedStories :Set<Story> = self.userStory->select(story story.testCase->exists(testCase testCase.hasPassed = false))</pre>	<pre>List<Artifact> allStories = step .getLinkedArtifactsForType(„UserStory“); List<Artifact> result = new ArrayList<Artifact>(); for(UserStory story : allStories) { List<Artifact> allTestCases = story. getLinkedArtifactsFor- Type(„TestCase“); List<Artifact> result2 = new Array- List<Artifact>(); for(Artifact testCase : allTestCases){ if(((TestCase)testCase).hasPassed == false){ result2.add(testCase); } } if(result2.size()>0){ result.add(story); } } return result;</pre>
<pre>(d) context UseCaseStep def:storyEstimations:Set<Integer> = self.userStory->collect(st st.estimate)</pre>	<pre>List<Artifact> allStories = step.getLinkedArtifactsOf(„UserStory“); List<Integer> result = new ArrayList<Integer>(); for(Artifact story: allStories){ result.add(((Story)story).estimate); } return result;</pre>

8.5 Ausführung der Trace-Operationen in einem Werkzeug

<pre>(e) context UseCaseStep def: estimation: integer = iterate{story in self.userStory; int sum = 0; sum = sum + story.estimate }</pre>	<pre>List<Artifact> allStories = step.getLinkedArtifactOfType(„UserStory“); List<Integer> resultList = new ArrayList<Integer>; for(Artifact story: allStories){ resultList.add((Story)story).estimate); } Integer result = sum(resultList); //Hilfs-Methode</pre>
<pre>(f) context UserStory def: thePriority : integer = self.priority inv thePriority > 0</pre>	<pre>UserStory userStory = (UserStory)input; int thePriority = userStory.priority; return (thePriority > 0)</pre>
<pre>(g) context UserStory def: theEstimation: integer = self.estimate inv self.iteration=Iteration.current and theEstimation > 0</pre>	<pre>int theEstimation = userStory.estimate; return (userStory.iteration == Iteration.current) && (theEstimation > 0)</pre>
<pre>(h) context UseCaseStep def: userStories : Set<UserStory> = self.userStory inv self.status <> ‚done‘ and userStories->forall(story story.status = ‚done‘)</pre>	<pre>List<UserStory> userStories = step.getLinkedArtifactsOfType(„UserStory“); boolean exp1 = step.status != „done“; boolean exp2 = true; for(UserStory story : userStories){ if(! story.status == „done“) exp2 = false; } return exp1 && exp2;</pre>
<pre>(i) context UseCaseStep def linkedStoryCount:integer = self.userStory->size() inv linkedStoryCount > 0</pre>	<pre>int linkedStoryCount = step .getLinkedArtifactsForType(„UserStory“).size(); return linkedStoryCount > 0;</pre>

8.5.4 Verarbeitung der Trace-Operations-Ergebnisse in Artefakt-Sichten

Dieser Abschnitt zeigt am Beispiel einer Java Swing-Oberfläche, wie Artefakt-Sichten nun beliebige Trace-Operationen berücksichtigen können. Es wird ein spezieller Bereich definiert, der zu jeder Trace-Operation eine Unter-Ansicht enthält. Mittels der Attribute *name* und *queryResult* können ein Bezeichner und der anzuzeigende Ergebniswert extrahiert werden.

```
public class StoryCardView extends JPanel{

    private JPanel storyPanel;
    private JPanel storyDetailPanel;
    private JPanel traceOpsPanel;

    public StoryCardView(UserStory story){
        storyPanel = new JPanel(new GridLayout(2,1));

        storyDetailPanel = new JPanel();
        storyDetailPanel.setText(story.id + story.description);
        storyPanel.add(storyDetailPanel);

        traceOpsPanel = new JPanel(new FlowLayout());
        for(TraceOpResult compAttr : story.getComputedAttr()){
            JPanel compAttrPanel = new JPanel();
            compAttrPanel.setToolTipText(compAttr.name);
            compAttrPanel.setText(compAttr.queryResult);
            traceOpsPanel.add(compAttrPanel);
        }

        for(TraceOpResult statCrit : story.getStaticCritiques()){
            JPanel statCritPanel = new JPanel();
            statCritPanel.setToolTipText(statCrit.name);
            statCritPanel.setText(statCrit.reactionMessage);
            traceOpsPanel.add(statCritPanel);
        }

        storyPanel.add(traceOpsPanel);
    }
}
```

9 Validierung des IRE-Ansatzes

Zur Validierung des IRE-Ansatzes stellen sich mehrere Evaluationsfragen, die im Folgenden aufgelistet sind. Da der Ansatz sowohl die Methodik- als auch die Projektebene berührt, müsste auch auf beiden Ebenen untersucht werden, wie gut die Ziele erreicht werden. Mögliche Evaluationsfragen sind:

1. Können Requirements Methodiker Trace-Operationen auf der methodischen Ebene gut spezifizieren? Fällt ihnen dieser Vorgang leicht?
2. Lassen sich alle benötigten Operationen mit den gegebenen Ausdrucksmitteln spezifizieren?
3. Wie gut lässt sich der Ansatz technisch in aktuell existierende CASE-Tools integrieren?
4. Hat der IRE-Ansatz die gewünschte Wirkung, Projektteilnehmer im Umgang mit Anforderungen zu unterstützen?
5. Steht der Aufwand zur Erstellung einer verknüpften Anforderungsmenge, die für einige Trace-Operationen notwendig ist, in einer ausreichend guten Relation zum Nutzen des IRE-Ansatzes?

Als besonders wichtig sind dabei die Fragen nach dem Nutzen auf der Projektebene (Fragen 4 und 5) herauszustellen. Während sich auf der Projektebene zeigt, wie gut die *Wirkung des Ansatzes* ist, wird auf der Methodik-Ebene festgelegt, wie gut Bestandteile des IRE-Ansatzes, wie Trace-Operationen, konzeptuell und technisch (d.h. für ein Computersystem verständlich) *definiert werden* können.

Die Frage nach der Wirkung ist zunächst deswegen wichtiger, weil diese über die Sinnhaftigkeit des gesamten Ansatzes entscheidet. Bei einer guten Unterstützung der *Definition*, aber einer schlechten *Wirkung* des Ansatzes hat der gesamte Ansatz keinen Nutzen. Ist hingegen eine gute *Wirkung* des Ansatzes belegt, so kann man im Falle einer schlechten Unterstützung der *Definition* noch versuchen, die Aspekte zur Definition zu verändern. Ist beispielsweise die technische Integration schwieriger als erwartet (Frage 3), so lassen sich die Softwareentwürfe aus Abschnitt 8.5 möglicherweise anpassen, um besser zur Architektur bestehender CASE-Tools zu passen. Wenn sich nicht alle Operationen formalisieren lassen (Frage 2), so kann der Ansatz möglicherweise um zusätzliche Befehle und Symbole zur Formalisierung neuer Elemente erweitert werden. Wenn Requirements Methodiker Trace-Operationen nicht gut spezifizieren können (Frage 1), so lässt sich möglicherweise eine neue Sprache finden oder weitere Werkzeugunterstützung zur Spezifikation erstellen. Dass die projektspezifische *Definition* grundsätzlich durchführbar ist, ist zudem im Rahmen von Kapitel 8 bereits gezeigt worden.

Für die Wirkung auf Projektebene sind nun zwei Aspekte relevant. Zum einen darf es nicht zu aufwändig sein, Verknüpfungen zwischen Anforderungen herzustellen (Frage 5). Viele der Trace-Operationen sind auf bestehende Verknüpfungen auf Projektebene angewiesen, um hilfreiche Hinweise zu erstellen. Andererseits ist es aber auch entscheidend zu evaluieren, ob die Trace-Operationen in einer vollständig verknüpften Anforderungsmenge überhaupt ihre Wirkung erzielen (Frage 4), oder ob es bereits dabei Probleme gibt. Beispielsweise ist denkbar, dass Projektteilnehmer während ihrer anforderungsbezogenen Aktivitäten überhaupt nicht auf Meldungen von Trace-Operationen reagieren oder dass die integrierte Sicht sie stört.

Entsprechend wird in dieser Evaluation zunächst geprüft, ob sich unter der Annahme einer vollständig verknüpften Anforderungsmenge grundsätzlich ein Nutzen erzielen lässt. Damit dient Frage 4 als Forschungsfrage für diese Evaluation.

Dennoch ist es auch wichtig, den Aufwand für die Erstellung der Verknüpfungen zu berücksichtigen und in einer zukünftigen Evaluation auch zu überprüfen. Sind die Kosten für die Erstellung von Traceability-Links zu hoch, so kann dies den Einsatz und damit den Erfolg des ganzen Ansatzes mindern. Jedoch gibt es in Zusammenhang mit dem IRE-Ansatz auch Möglichkeiten, die Kosten für die Erstellung von Verknüpfungen zu senken:

Meist ist es einfacher, Artefakte direkt bei ihrer Erstellung zu verknüpfen, als später die gesamte Anforderungsmenge zu durchsuchen und die Artefakte einander zuzuordnen. Während der Erstellung ist der Projektteilnehmer gedanklich gerade im Kontext der entsprechenden Anforderung, was es einfacher macht, an abhängige Anforderungen zu denken. Trace-Operationen können an genau dieser Stelle helfen. Beispielsweise zeigt die statische Kritik StK1 (siehe Tabelle 18, Seite 177) eine Warnung, wenn es zu einem Artefakt kein verknüpftes anderes Artefakt gibt. Wird diese Kritik direkt bei der Erstellung eines Artefaktes angezeigt, so erinnert sie die Projektteilnehmer auch daran, das Artefakt direkt zu verknüpfen.

Außerdem wird die Schwelle, während einer anforderungsbezogenen Aktivität Artefakte eines anderen Typs zu betrachten, niedriger, wenn man dazu nicht den aktuellen Anforderungskontext zu verlassen braucht. Genau dies wird durch integrierte Sichten ermöglicht.

Wenn bereits einige Verknüpfungen bestehen, so wird die Navigation zu relevanten Elementen erleichtert, weil dafür Verknüpfungen von benachbarten Elementen genutzt werden können. Leitet beispielsweise der Projektteilnehmer eine User Story aus einer anderen User Story ab, so kann er die Verknüpfungen der bestehenden User Story nutzen, um schnell zu einem relevanten Use Case zu gelangen und dort nach potenziellen Abhängigkeiten zu suchen.

Zudem lässt sich der Ansatz um automatisierte Vorschläge zu Verknüpfungen erweitern, was ebenfalls hilft, die Kosten für die Erstellung von Verknüpfungen zu senken. So kann das System beispielsweise bei der Erstellung einer neuen User Story mithilfe von Textanalyse, wie in [65], [116], oder der Analyse bereits bestehender Verknüpfungen direkt potentiell abhängige Artefakte vorschlagen. In [162] wird beispielsweise das Vorschlagen textuell verwandter Artefakte als zusätzliche Trace-Operation in den Prototyp des IRE-Ansatzes integriert.

9.1 Evaluationsziele

Die Evaluation soll zeigen, ob der entwickelte Ansatz Projektteilnehmer bei der Durchführung von anforderungsbezogenen Aktivitäten unterstützt (Frage 4 im vorigen Abschnitt).

Bei der Durchführung anforderungsbezogener Aktivitäten gibt es verschiedene Herausforderungen. Einige davon entstehen durch Koexistenz-Aktivitäten, wie in der Interview-Studie (Abschnitt 4) festgestellt wurde. Die unterschiedlichen Aspekte von Schwierigkeiten, die dabei eine Rolle spielen, sind in Abbildung 60 dargestellt.

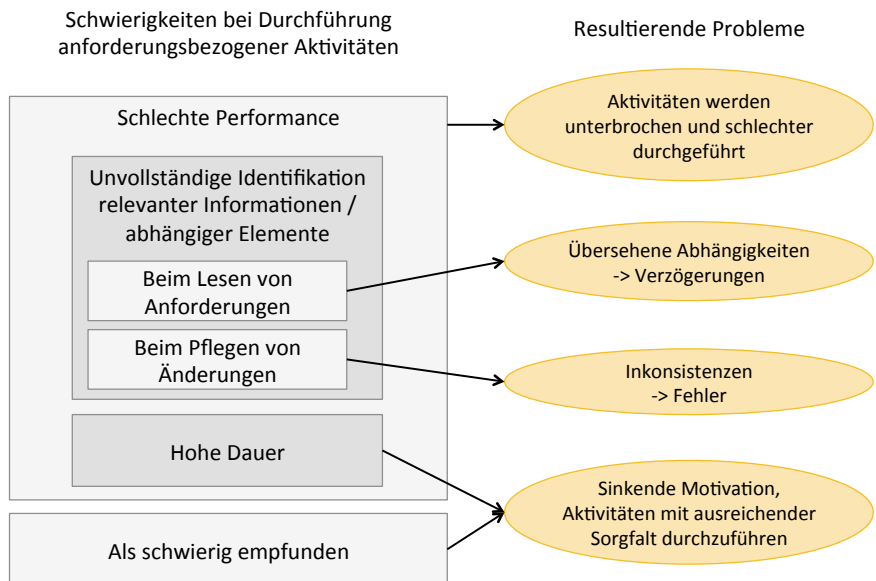


Abbildung 60: Herausforderungen bei der Durchführung anforderungsbezogener Aktivitäten

Die Probleme im Umgang mit Anforderungen sind im Folgenden erläutert. Die konkreten Herausforderungen helfen dabei, die Evaluation auf konkrete Aspekte zu fokussieren. Herausforderungen sind:

- Grundsätzlich führt *schlechte Performance* in der Durchführung von Koexistenz-Aktivitäten dazu, dass die eigentlichen anforderungsbezogenen Aktivitäten gestört werden. Dies kann zu Verzögerungen führen, aber auch generell zu schlechteren Ergebnissen, wenn nicht alle Abhängigkeiten ausreichend berücksichtigt werden.
- Werden relevante Informationen in den Anforderungen nur *unvollständig gefunden*, so führt dies zu übersehenen Abhängigkeiten, was wiederum die Planung und Implementierung verzögern kann.
- Wenn Anforderungsänderungen *nicht ausreichend vollständig* an allen relevanten Stellen *nachgepflegt* werden, führt dies zu Inkonsistenzen und später möglicherweise zu Fehlern.
- Wenn Anforderungsaktivitäten zu lange dauern oder als zu schwierig empfunden werden, verlieren Projektteilnehmer die Motivation, diese Aktivitäten, bzw. die damit zusammenhängenden Koexistenz-Aktivitäten, mit ausreichender Sorgfalt auszuführen. Im Extremfall vernachlässigen sie dann ganze Anforderungsartefakte.

Die Evaluation verfolgt zwei Ziele: (i) verstehen, wie explizite Verknüpfungen und integrierte Sichten verwendet werden und (ii) bewerten, ob die oben gelisteten Herausforderungen durch die Verwendung des IRE-Ansatzes – genauer von expliziten Verknüpfungen und Trace-Operationen – abgeschwächt werden können. Aus diesen Zielen werden die folgenden Forschungsfragen für eine Studie abgeleitet:

Forschungsfrage 1: Hilft der IRE-Ansatz dabei, die Performance von anforderungsbezogenen Aktivitäten zu verbessern?

Forschungsfrage 2: Hilft der IRE-Ansatz dabei, die empfundene Schwierigkeit von anforderungsbezogenen Aktivitäten zu verringern?

Forschungsfrage 3: Hilft der IRE-Ansatz dabei, relevante Informationen bei der Ausführung anforderungsbezogener Aktivitäten vollständiger zu identifizieren und anzupassen?

Forschungsfrage 4: Wie empfinden Projektteilnehmer die integrierten Sichten und die automatischen Reaktionen von Trace-Operationen?

Eine weitere Fokussierung im Rahmen der Studie ist die Konzentration auf Trace-Operationen, welche neben integrierten Sichten nur einer der Bestandteile des IRE-Ansatzes sind. Integrierte Sichten stärken Trace-Operationen indem sie eine direktere Navigation zu abhängigen Artefakten erlauben und Kontextwechsel vermeiden. Doch hat prinzipiell auch bereits die bloße Anwendung von integrierten Sichten im Vergleich zu isolierten Sichten eine eigene Wirkung. Diese wird im Rahmen der vorliegenden Studie jedoch nicht untersucht. Die stärkere Fokussierung auf Trace-Operationen hilft, gemessene Effekte besser zu konkreten Konzepten zuordnen zu können.

9.2 Evaluierungsmethode

Die Studie besteht aus einem *randomisierten Experiment*, in dem Experimentteilnehmer zufällig zu einer von zwei Gruppen zugeordnet werden. Teilnehmer beider Gruppen erhalten dieselbe *Anforderungsmenge* in derselben *integrierten Sicht* und führen dieselben *anforderungsbezogenen Aktivitäten* durch. Die *experimentelle Behandlung*, die eine der beiden Gruppen (*LinkedArtifacts*) erhält, ist, dass ihre Anforderungsmenge mit expliziten Verknüpfungen versehen ist und sie durch Trace-Operationen unterstützt wird. Die Kontrollgruppe (*NoLinks*) kann nicht auf Verknüpfungen zurückgreifen und wird nicht durch Trace-Operationen unterstützt. Dabei werden die Performance und das Ergebnis der Aktivitäten beider Gruppen evaluiert und verglichen.

Das *two-group design* [138], bei dem nur eine von zwei Personengruppen die Behandlung erhält, wird dem *within design* [138] vorgezogen, bei dem jeder Teilnehmer das Experiment einmal mit Behandlung und einmal ohne durchführt. Der Aufbau nach *two-group design* wird gewählt, um die gesetzten Zeitgrenzen für die Experimentdauer einhalten zu können und um Lerneffekte zu vermeiden.

Randbedingungen: Eine der größten Herausforderungen beim Finden einer passenden Anforderungsmenge und der passenden Aufgabenstellungen ist es, die richtige Balance zwischen (i) **angemessener Experimentdauer**, (ii) **realistischem Projektumfang** und (iii) **hoher Vielfalt an durchgeführten Aktivitäten** zu finden.

Hier werden die Bedingungen und deren Resultate kurz erläutert:

- (i) Die Probanden nehmen freiwillig und unabhängig von Lehrveranstaltungen an der Studie teil. Es wird angenommen, dass eine längere Experimentdauer zu geringeren Interessentenzahlen für das Experiment führt. Der Zeitumfang soll daher so gering wie möglich gehalten werden und bei höchstens 60 Minuten liegen.
- (ii) Es ist wichtig, dass die Anforderungsbasis möglichst den Anforderungen in einem realen Projekt entspricht und nicht beispielsweise nur aus

einem Use Case und einer Handvoll passender Story Cards besteht. Insbesondere soll das Beispielprojekt auch Teile beinhalten, an denen Use Cases und Story Cards in komplexeren Beziehungen zueinander stehen, um auch komplexere Aufgaben zu ermöglichen. Insgesamt soll das Beispielprojekt mindestens von der Größe eines studentischen Softwareprojektes innerhalb der Lehrveranstaltung *Softwareprojekt* sein [105], und über mindestens eine Instanz von Anforderungen verfügen, in denen dieselbe Funktionalität für mehrere Abläufe verwendet wird (d.h. an mehreren Stellen der Spezifikation auftaucht).

- (iii) Es soll möglich sein, auf Basis mehrerer Aufgaben innerhalb des Experimentes, verschiedene Aktivitäten (wie Finden und Ändern von Anforderungen oder Inkonsistenzen) und Komplexitätsgrade zu testen.

Um einen Aufbau zu finden, der diesen Bedingungen entspricht, wurde eine fiktive Software vom ungefähren Umfang eines Softwareprojektes spezifiziert. Dazu wurden Aufgaben erstellt, welche die verschiedenen Aktivitäten und Schwierigkeitsgrade abdecken. Schließlich wurde die Anforderungs- und Aufgabenmenge mithilfe von Pilotstudien so angepasst (d.h. verkleinert), dass sie die vorgegebene Zeitdauer von 60 Minuten nicht überschreitet. Das Beispielprojekt entspricht einer Anforderungsmenge von 6 Use Cases und 49 Story Cards und wird in 18 Aufgaben bearbeitet.

Die meisten gewünschten Funktionalitäten werden von Story Cards *und* Use Case-Schritten beschrieben, welche in einer *entspricht*-Beziehung stehen. Für die NoLinks-Gruppe sind diese Zusammenhänge, nicht explizit sichtbar, während die *LinkedArtifacts*-Gruppe zu jedem Artefakt verlinkte Elemente einsehen sowie in der Anforderungsübersicht direkt zu ihnen navigieren kann.

Aufgabenstellung: Die Probanden erhielten insgesamt 18 verschiedene Aufgaben, bei denen sie Artefakte (Use Case-Schritte, Use Case-Erweiterungen und Story Cards) in der vorgegebenen Anforderungsmenge *finden* (insg. 16 Aufgaben), *ändern* (insg. 8 Aufgaben) oder *neu einpflegen* (insg. 4 Aufgaben) mussten. Auch Kombinationen dieser Aktivitäten waren möglich. Dabei ist zwischen *simplem* und *komplexen* Aufgaben zu unterscheiden. Bei *simplem* Aufgaben (insg. 11 Aufgaben) reicht es, Informationen aus Artefakten mit nur einem *Artefakttyp* (also Use Cases *oder* Story Cards) zu beziehen. Bei *komplexen* Aufgaben (insg. 7 Aufgaben) ist es notwendig, Informationen aus Artefakten beider Artefakttypen zu kombinieren, um zum richtigen Ergebnis zu gelangen.

Tabelle 29 zeigt die Arten von Aufgaben, die im Experiment verwendet wurden. Um unterscheiden zu können, ob Effekte durch die unterschiedlichen Komplexitäten der Aufgaben entstanden oder nur, weil die Teilnehmer zum Ende des Experimentes hin weniger aufmerksam wurden, wurde ein Kreuzdesign verwen-

det. Dazu wurde die Reihenfolge der Blöcke B und C für die Hälfte der beiden Gruppen jeweils vertauscht.

Tabelle 29: Unterteilung der Aufgabenstellungen im Experiment

Aufgaben-Block	Beispiel-Aufgabe
A (6 einleitende Aufgaben)	„Verändere die Priorität der Story [... <i>Story Titel</i>] zu Priorität 3.“
B (5 simple Aufgaben, 1 komplexe Aufgabe)	„Der Kunde möchte die Funktionalität, die in Use Case 4 – Schritt 3 beschrieben ist, ändern. Er möchte, dass [... <i>neue Anforderung</i>]. Dokumentiere diese Anforderung.“
C (4 komplexe Aufgaben)	„Welcher der Schritte in Use Case 3 kann bereits durchgeführt werden?“
D (2 Aufgaben zum Finden von Inkonsistenzen)	„Nicht alle Schritte in Use Case 1 besitzen bereits verknüpfte Story Cards. Finde drei solche Schritte und erstelle die entsprechenden Stories.“

Im Anschluss an die Aufgaben erhielten die Teilnehmer einen Fragebogen mit Fragen zu ihrem Eindruck der Arbeit mit den Anforderungen und dem Werkzeug.

Die Teilnehmer erhielten eine schriftliche Anleitung innerhalb des Werkzeugs, in der ihnen die Funktionsweise des Tools, die Anforderungslandschaft sowie der Experimentablauf erklärt wurden. Alle Teilnehmer wurden gebeten, die Aufgaben so schnell und gründlich wie möglich zu erledigen und die Anforderungsmenge so vollständig und konsistent wie möglich zu machen.

Variablen: Tabelle 30 zeigt die Variablen, die in diesem Experiment verwendet werden. Die abhängigen Variablen werden für jede Experimentaufgabe einzeln gemessen. Dann werden die Werte über alle Aufgaben derselben Komplexitätsstufe aggregiert, um die Forschungsfragen zu beantworten. Die statistische Signifikanz der Ergebnisse wird mittels des Mann-Whitney-U-Tests überprüft. Neben der direkten Messung von abhängigen Variablen, wie der Dauer oder Vollständigkeit bei der Lösung der Aufgaben, wird auch eine abgeleitete Größe gebildet: die *Performance der Fertigstellung einer Aufgabe* berechnet sich als Quotient der Vollständigkeit der Lösung geteilt durch die Dauer der Lösung einer Aufgabe. Sie dient als kombiniertes Maß zum besseren Vergleich von Situationen, in denen Teilnehmer eine Aufgabe vollständiger lösen, aber gleichzeitig dafür mehr Zeit benötigen.

Tabelle 30: Abhängige und unabhängige Variablen, die im Experiment betrachtet werden

Unabhängige Variablen	Abhängige Variablen
Faktoren	Forschungsfragen 1-3
Verknüpfungsmodus <ul style="list-style-type: none"> • Explizite Verknüpfungen und Trace-Operationen • Nicht verknüpft 	Dauer der Lösung einer Aufgabe
Komplexität der Aufgaben <ul style="list-style-type: none"> • Simple • Komplex 	Vollständigkeit der Lösung einer Aufgabe (Quotient: $\frac{\#(\text{identifizierte} + \text{angepasste} + \text{erstellte Artefakte})}{\#(\text{Artefakte, die identifiziert} + \text{angepasst} + \text{erstellt werden müssen, basierend auf Musterlösung})}$)
Kontrollierte Variablen	Performance der Lösung einer Aufgabe (Quotient: Vollständigkeit / Dauer der Lösung einer Aufgabe)
Vertrautheit mit den vorliegenden Anforderungen	Empfundene Schwierigkeit der Aufgabenlösung (Likert-Skala mit 1 = sehr einfach bis 5 = sehr schwierig)
Inhalt und Darstellung der Artefakte	Forschungsfrage 4
Komplexität der Anforderungsmengen	Empfundene Häufigkeit der Störung durch zweite Sicht
	Empfundene Häufigkeit der Betrachtung verknüpfter Artefakte, die hervorgehoben wurden

9.3 Experimentdurchführung und Datensammlung

Die Experimentteilnehmer (15 Personen in der *LinkedArtifacts*-Gruppe und 14 Personen in der *NoLinks*-Gruppe¹⁴) waren größtenteils Studierende der Leibniz Universität Hannover, die mindestens an der Lehrveranstaltung *Softwareprojekt* teilgenommen hatten [105]. Somit haben alle Teilnehmer bereits in mindestens einem Projekt den praktischen Umgang mit Anforderungen erfahren.

Das IRE-Tool als Prototyp für ein RE-Tool mit Unterstützung für Trace-Operationen und integrierte Sichten

Während des Experimentes arbeiteten die Teilnehmer innerhalb eines Prototyp-RE-Tools (IRE – Integrated Requirements Environment), das auf die Integration von Use Cases und User Stories ausgelegt ist und entsprechende integrierte Sichten und Trace-Operationen bereitstellt. Das Werkzeug wurde im Zuge dieser Arbeit für die Demonstration der Konzepte aus Kapitel 8 entwickelt. Abbildung 61 zeigt die zentrale Ansicht, in der die Interaktion mit Use Cases und User Stories sowie deren Verknüpfungen stattfindet. Das Werkzeug ist unter <http://www.se.uni-hannover.de/ire-tool> einsehbar.

¹⁴ Ein Ergebnis aus der NoLinks-Gruppe musste entfernt werden, da der Experimentteilnehmer – wie die Person auch selbst in den Kommentaren angegeben hat – die Fragen und das Beispiel-Projekt allgemein nicht verstanden hat und daher praktisch keine Lösungen angeben konnte.

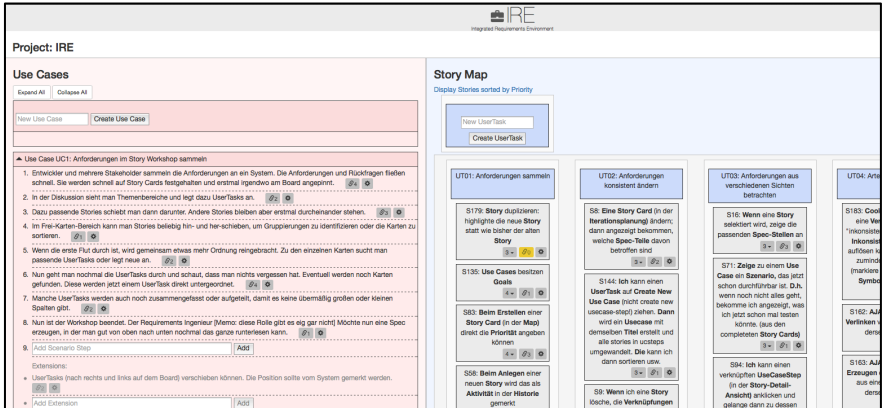


Abbildung 61: Zentrale Ansicht des IRE-Tools mit integrierter Sicht für Use Cases und User Stories

Die implementierten Trace-Operationen sind in der folgenden Tabelle aufgeführt. Diese Trace-Operationen wurden im Prototyp fest codiert.

Tabelle 31: Für Evaluation verwendete Trace-Operationen

Auslöser	Query	Kritik
Kontext: Use Case-Schritt		
<i>statisch</i>	Für einen Use Case-Schritt zeige eine Liste der verknüpften User Stories.	-
<i>statisch</i>	Für einen Use Case-Schritt zeige die Anzahl der verknüpften User Stories.	Wenn die Anzahl gleich null ist, zeige Warnsymbol
<i>select()</i>	Für einen Use Case-Schritt finde die Menge der verknüpften User Stories	Wenn Menge nichtleer, dann Hervorheben
<i>edit()</i>	Für einen Use Case-Schritt finde die Menge der verknüpften User Stories	Wenn Menge nichtleer, dann Hervorheben
<i>delete()</i>	Für einen Use Case-Schritt finde die Menge der verknüpften User Stories	Wenn Menge nichtleer, dann Hervorheben
Kontext: User Story		
<i>statisch</i>	Für eine User Story zeige eine Liste der verknüpften Use Case-Schritte.	-
<i>statisch</i>	Für eine User Story zeige die Anzahl der verknüpften Use Case-Schritte.	Wenn die Anzahl gleich null ist, zeige ein Warnsymbol
<i>select()</i>	Für eine User Story finde die Menge der verknüpften Use Case-Schritte	Wenn Menge nichtleer, dann Hervorheben
<i>edit()</i>	Für eine User Story finde die Menge der verknüpften Use Case-Schritte	Wenn Menge nichtleer, dann Hervorheben
<i>delete()</i>	Für eine User Story finde die Menge der verknüpften Use Case-Schritte	Wenn Menge nichtleer, dann Hervorheben

Für das Experiment wurde das Werkzeug um zusätzliche Funktionen des *System Monitoring* [149] ergänzt, um die Durchführung des Experimentes und die Datensammlung zu unterstützen. Dazu wurden die folgenden Funktionen ergänzt:

1. **Experiment-Anleitungs-Komponente:** In einem zusätzlichen Bereich oberhalb der eigentlichen Benutzeroberfläche werden die Aufgabenstellungen angezeigt und können textuelle Lösungen eingegeben werden.
2. **Logging-Komponente:** Alle relevanten Aktivitäten, die innerhalb der Benutzeroberfläche durchgeführt werden, werden zusammen mit einem Zeitstempel protokolliert. Zu den Aktivitäten zählen die Erstellung, Anpassung, Löschung von Artefakten, die Erzeugung von Verknüpfungen und der Aufruf der nächsten Experimentaufgabe, somit also der Abschluss einer Aufgabe.
3. **Projektspezifische Einstellmöglichkeit zur Unterdrückung aller Tool-Funktionen, die in Bezug zu Verknüpfungen stehen:** Für ein gegebenes Projekt kann vom Administrator die Einstellung *NoLinks* gewählt werden. Projekte mit dieser Einstellung zeigen keine verknüpften Artefakte an und erlauben es auch nicht, zu verknüpften Artefakten zu navigieren. Experimentteilnehmer der *NoLinks*-Gruppe erhielten jeweils ein so eingestelltes Projekt für ihren Experimentdurchgang.

9.4 Ergebnisse

Die Ergebnisse werden anhand der Forschungsfragen erläutert. Abbildung 62 fasst hierzu die verschiedenen gemessenen Variablen zusammen. Die Box-Plots zeigen Unterschiede zwischen den Ergebnissen der beiden Gruppen, also basierend auf den unterschiedlichen Verknüpfungs-Modi. Zu jeder Variable wird zudem jeweils ein Ergebnis für komplexe und eines für simple Aufgaben angezeigt.

Forschungsfrage 1: Hilft das IRE-Konzept dabei, die Performance von anforderungsbezogenen Aktivitäten zu verbessern?

Wie Abbildung 62 (a) zeigt, erreichen Teilnehmer der *LinkedArtifacts*-Gruppe eine bessere Performance für komplexe Aufgaben als die Teilnehmer der *NoLinks*-Gruppe. Diese Unterschiede sind statistisch signifikant mit $p < 0,01$, basierend auf dem Mann-Whitney-U-Test. Der Median-Wert für die Performance mit Treatment, also innerhalb der Gruppe *LinkedArtifacts*, ist (mit 4,44) um 60% höher als der Median-Wert ohne Treatment (2,78).

Bei simplen Experimentaufgaben sind jedoch die Median-Werte nahezu gleich (2,90 zu 2,72). Wie sich in Forschungsfrage 3 weiter zeigen wird, ist auch die Vollständigkeit der Lösung bei simplen Aufgaben für beide Gruppen ähnlich, sodass sich schließen lässt, dass sowohl Vollständigkeit als auch Bearbeitungsdauer bei simplen Aufgaben ähnlich sind, unabhängig davon in welcher Gruppe ein Teilnehmer war.

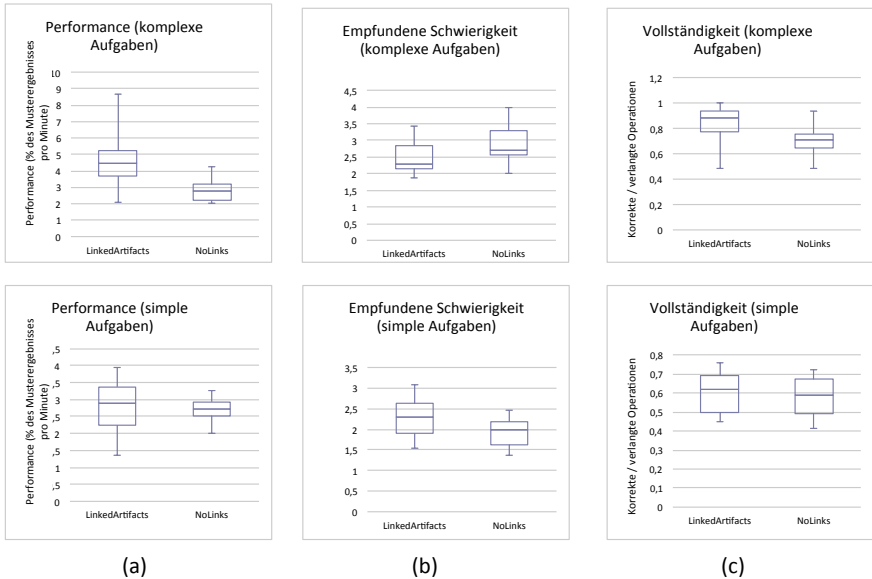


Abbildung 62: Box-Plots der Verteilung von gemessenen Variablen

Forschungsfrage 2: Hilft das IRE-Konzept dabei, die empfundene Schwierigkeit von anforderungsbezogenen Aktivitäten zu verringern?

Wie in Abbildung 62 (b) zu sehen ist, wurde die empfundene Schwierigkeit bei komplexen Aufgaben von Teilnehmern der *LinkedArtifacts*-Gruppe als weniger schwierig bewertet als von Teilnehmern der *NoLinks*-Gruppe (2,29 zu 2,71 im Median; Unterschiede nicht statistisch signifikant). Während bei komplexen Aufgaben die Teilnehmer mit Treatment hauptsächlich die verknüpften Artefakte durchgehen mussten, mussten Teilnehmer der Kontrollgruppe manuell nach verwandten Artefakten suchen. Diese zusätzliche Schwierigkeit resultierte in einer als höher empfundenen Gesamtschwierigkeit.

Simple Aufgaben wurden von Teilnehmern mit Treatment (i) als schwieriger eingestuft als die von Teilnehmern ohne Treatment (2,3 zu 2,0 im Median).

Forschungsfrage 3: Hilft das IRE-Konzept dabei, relevante Informationen bei der Ausführung anforderungsbezogener Aktivitäten vollständig zu identifizieren und anzupassen?

Wie in Abbildung 62 (c) abgebildet, haben die Probanden mit Treatment komplexe Aufgaben mit einer mittleren Vollständigkeit von 88% gelöst, während Teilnehmer ohne Treatment nur 71% erreicht haben. Dieser Unterschied ist statistisch signifikant mit $p < 0,05$, basierend auf dem Mann-Whitney-U-Test. Jedoch sind die Unterschiede der Vollständigkeit bei simplen Aufgaben nur

gering (62% zu 59%). Die Vollständigkeit der Lösung von simplen Aufgaben ist bei Teilnehmern der *LinkedArtifacts*-Gruppe um 26 Prozentpunkte niedriger als bei komplexen Aufgaben. Genauso, wie Teilnehmer aus der *NoLinks*-Gruppe haben *LinkedArtifacts*-Mitglieder es ebenfalls versäumt, abhängige Artefakte anzupassen, wenn sie Änderungen an Anforderungen vorgenommen haben. Nur wenn eine Experimentaufgabe explizit darauf ausgelegt war, beide Seiten zu betrachten (also wenn es sich um eine komplexe Aufgabe handelte), haben Teilnehmer mit Treatment die Aufgaben auch vollständig gelöst.

Forschungsfrage 4: Wie empfinden Projektteilnehmer die integrierten Sichten und die automatischen Reaktionen von Trace-Operationen?

In Abbildung 63 (a) ist aufgezeigt, dass die meisten Projektteilnehmer nie oder selten eine der beiden parallelen Sichten schließen wollten. Die meisten Teilnehmer wollten die Use Case-Sicht häufiger schließen als die User Story-Sicht. Eine Vermutung ist, dass dies daran lag, dass die Story Map horizontal nicht vollständig auf den Bildschirm gepasst hat und die Teilnehmer sich so einen besseren Überblick verschaffen wollten.

Abbildung 63 (b) zeigt, dass die meisten Teilnehmer, die mit verknüpften Artefakten und Trace Operationen gearbeitet haben (das sind nur die Teilnehmer der *LinkedArtifacts*-Gruppe), die hervorgehobenen verknüpften Artefakte auch zumindest grob betrachtet haben. Der Einsatz der IRE-Methode kann also die Aufmerksamkeit auf verknüpfte Artefakte lenken.

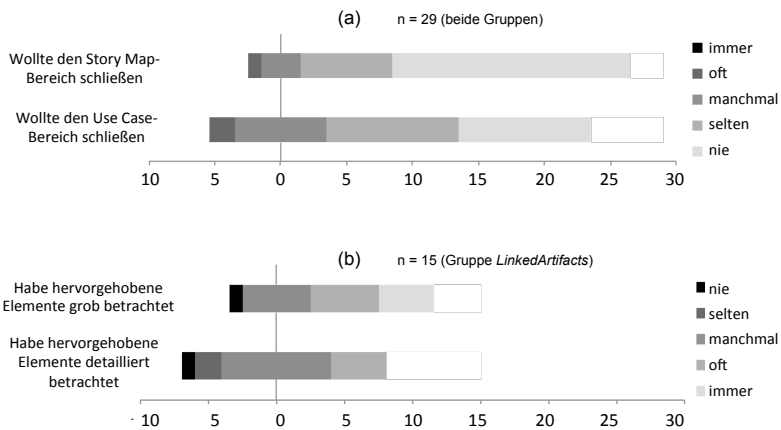


Abbildung 63: Zusammenfassung der Antworten des Fragebogens

9.5 Diskussion

Entgegen der Erwartungen war die Vollständigkeit, mit der die Aufgaben gelöst wurden für beide Gruppen ähnlich. Das typische Problem, dass ein Projektteilnehmer schnell ein primäres Artefakt ändert, jedoch vergisst, die Änderung auch auf andere verwandte Artefakte zu übertragen, ist vor allem bei simplen Aufgaben aufgetreten.

Als Trace-Operationen wurden bewusst nur relativ leichtgewichtige Mechanismen, wie die Hervorhebung verknüpfter Artefakte, eingesetzt, anstatt auffälliger Reaktionen, wie die Einblendung eines Warndialogs auszuführen. Dies sollte dazu dienen, herauszufinden, ob es bereits hilfreich ist, nur die Aufmerksamkeit stärker auf verknüpfte Elemente zu lenken. Es hat sich jedoch gezeigt, dass stärkere Mechanismen notwendig sind, um eine ausreichende Berücksichtigung von Abhängigkeiten zu bewirken. Beispielsweise können zusätzliche Richtlinien in die Entwicklungsmethode integriert werden, wie dass die Konsistenz einer User Story der kommenden Iteration noch einmal explizit überprüft wird. Außerdem können auffälliger Reaktionen der Trace-Operationen helfen, nachrücklicher auf die Überprüfung verknüpfter Elemente hinzuweisen. Solche Mechanismen tragen allerdings auch neue Risiken, wie die Unterbrechung der Projektteilnehmer bei ihren Aufgaben. Diese Risiken müssten weiter untersucht werden.

9.6 Validität der Ergebnisse

In diesem Abschnitt wird die Validität der Experiment-Ergebnisse diskutiert. Hierzu werden die verschiedenen Aspekte der Validität nach Wohlin et al. [164] erläutert.

Interne Validität (Internal Validity):

Um Seiteneffekte zu vermeiden, wurden die Unterschiede zwischen den Bedingungen beider Gruppen minimiert. Beide Gruppen arbeiteten im selben Werkzeug und erhielten dieselbe Anforderungsmenge. Somit können Effekte, die durch verfügbare oder fehlen Werkzeugfunktionen, die Usability des Werkzeugs oder durch Inhalt sowie Erscheinung der Anforderungen hervorgerufen werden, ausgeschlossen werden.

Die Anleitung zum Experiment war schriftlich auf der Startseite des Werkzeugs verfasst, sodass auch hier vermieden wurde, die Experimentteilnehmer durch unterschiedliche mündliche Erklärungen zu beeinflussen. Dies führt allerdings zu einer neuen Abhängigkeit davon, wie aufmerksam die Teilnehmer die Einleitungstexte lesen. Um diese Abhängigkeit kontrollieren zu können, wurden die ersten sechs Aufgaben (Block A in Tabelle 29 mit leichten Einleitenden Aufgaben) als Pre-Test verwendet. Wird eine dieser Fragen falsch beantwortet, so wird dieser Teilnehmer nicht berücksichtigt.

Validität des Aufbaus (Construct Validity):

Es werden Dauer, Vollständigkeit und Performance der Lösung der Experiment-Aufgaben gemessen. Die *Dauer* wurde aus der Differenz der Zeitstempel von Beginn der entsprechenden Aufgabe und dem Beginn der folgenden Aufgabe berechnet. Dies ist nur eine Annäherung an die eigentliche Dauer der Lösung der Aufgabe, da hier beispielsweise auch die Dauer zum Lesen der Aufgabe und zur Orientierung in der Anforderungsmenge enthalten ist.

Um die *Vollständigkeit* zu berechnen, wird das Verhältnis zwischen durchgeführten zu notwendigen Operationen (nennen, erstellen, ändern eines Elementes) aus der Musterlösung berechnet. Dabei wird jede Operation gleich gewichtet. Dies kann wieder nur als Annäherung dienen, da in der Realität unterschiedliche Operationen unterschiedliche Anteile an der Gesamtlösung darstellen können.

Die *Performance* ist eine abgeleitete Größe und könnte auf unterschiedliche Art konstruiert werden. Um Unklarheiten zu vermeiden, wird in Tabelle 30 explizit definiert, was Performance im Kontext dieser Studie bedeutet.

Um einen allgemeinen Wert für simple bzw. komplexe anforderungsbezogene Aktivitäten zu erhalten, wurden die Ergebnisse zu den einzelnen Aufgaben aggregiert. Um sicherzustellen, dass eine ausreichend umfassende Bandbreite an möglichen Aktivitäten abgedeckt wird, wurden 18 Experimentaufgaben zusammengestellt, um alle Basisoperationen (erstellen, lesen, anpassen, löschen) für verschieden komplexe anforderungsbezogene Aktivitäten abzudecken.

Statistische Validität (Conclusion Validity):

Die Resultate zeigen, dass die Einführung der Behandlung auch zu Unterschieden in den abhängigen Variablen führen. Jedoch sind nicht alle Ergebnisse statistisch signifikant. Statistische Signifikanz wurde auf Basis des Mann-Whitney-U-Tests berechnet und ist bei statistisch signifikanten Ergebnissen im Abschnitt *Ergebnisse* aufgeführt.

Um Unklarheiten bei der Bewertung der Vollständigkeit einer Aufgabenlösung zu vermeiden, wurde eine Musterlösung angefertigt, die explizit aufzeigt, welche der Anforderungen für eine Aufgabe identifiziert, geändert oder erstellt werden müssen.

Externe Validität (External Validity):

Es kann nicht nachgewiesen werden, dass die Ergebnisse auf einen industriellen Kontext übertragbar sind. Insbesondere der Umstand, dass die Teilnehmer mit den Anforderungen, mit denen sie im Experiment gearbeitet haben, nicht vertraut waren, kommt in Softwareprojekten selten vor. Die Teilnehmer bestanden größtenteils aus Studierenden. Jedoch haben alle Teilnehmer an mindestens einem Projekt teilgenommen, in dem sie bereits umfangreich mit Anforderungen gearbeitet haben.

Um die Experimentdauer und –komplexität kontrollieren zu können, musste anstatt einer realen eine konstruierte Anforderungsmenge verwendet werden.

Um jedoch zu vermeiden, dass eine unrealistisch simple Anforderungsmenge gewählt wird, wurden Umfang und Komplexität der Anforderungsmenge an typische Werte aus Projekten der Lehrveranstaltung *Softwareprojekt* an der Leibniz Universität Hannover [105] angepasst.

9.7 Zusammenfassung

Der Aufbau des Experimentes stellt eine erfolgreiche Fallstudie dar, bei der gezeigt wurde, dass eine hybride Anforderungslandschaft zu einem Projekt aufgebaut und Trace-Operationen darauf angewendet werden können. In der Durchführung sind die Experimentteilnehmer mit der hybriden Anforderungslandschaft, den Verknüpfungen und dem Werkzeug insgesamt zurecht gekommen. Sie haben, wie beabsichtigt, mehr auf Artefakte eines abhängigen Artefakttyps geachtet und sich nicht durch die parallele Sicht gestört gefühlt.

Die Evaluation zeigt, dass das IRE-Konzept dabei helfen kann, die Performance von komplexen anforderungsbezogenen Aktivitäten zu verbessern. Bei simplen anforderungsbezogenen Aktivitäten reicht es allerdings nicht aus, verknüpfte Artefakte nur hervorzuheben, um zu erreichen, dass Projektteilnehmer ausreichend auf Konsistenz zu abhängigen Artefakten achten. Hier sind nachdrücklichere Mechanismen notwendig, die in der Diskussion in Abschnitt 9.5 vorgestellt werden.

Anknüpfungspunkte für zukünftige Forschung bestehen darin, näher zu untersuchen, ob nachdrücklichere Trace-Operationen die Handhabung abhängiger Artefakte besser unterstützen können. Zudem wäre es zu untersuchen, welchen konkreten Aufwand die Erstellung einer verknüpften Anforderungsmenge darstellt und wie dieser zum Nutzen durch die expliziten Verknüpfungen steht. Außerdem wäre zu prüfen, wie gut Personen in der Rolle eines Requirements Methodikers Trace-Operationen projektspezifisch definieren und in ein System einpeisen können, das diese zur Laufzeit ausführt.

10 Zusammenfassung und Ausblick

10.1 Zusammenfassung

In dieser Arbeit wurde ein Ansatz für die Unterstützung der Koexistenz von Anforderungsartefakten erarbeitet.

Dazu wurde zunächst in einer Studie herausgearbeitet, welche Aktivitäten von Anforderungsartefakten beeinflusst werden und wie umgekehrt Entwicklungs- und Kommunikationsaktivitäten die Wahl von Anforderungsartefakten beeinflussen. Es wurde erarbeitet, welche Herausforderungen das Zusammenspiel verschiedener Anforderungsartefakte beeinflussen und mit welchen Mitteln Abhängigkeiten bisher repräsentiert werden.

Ein auf den identifizierten Herausforderungen aufbauendes Konzept zur Modellierung von Artefakten, Abhängigkeiten und Verknüpfungen ermöglicht es, in einem Projekt eine Anforderungslandschaft aufzubauen, welche die Blickwinkel und Erfordernisse der Projektteilnehmer abdeckt und die entsprechenden Artefakte integriert. Auf diese Art wird es auch möglich, mit hybriden Landschaften agile und traditionelle Blickwinkel und damit auch die Methoden zu vereinen. Das zusätzliche Konzept der Trace-Operationen erweitert Anforderungslandschaften um zusätzliche CASE-Tool-Unterstützung für den täglichen Umgang mit Anforderungen und deren Abhängigkeiten.

In der Evaluation wurde gezeigt, dass Trace-Operationen dazu beitragen können, abhängige Anforderungsartefakte schneller und vollständiger zu finden. Allerdings haben die Experimentteilnehmer nicht in allen Aktivitäten mit Anforderungen auch von sich aus abhängige Elemente betrachtet, wodurch potenziell Inkonsistenzen entstehen können. Es werden entsprechend stärkere Mechanismen benötigt, die Nutzer an abhängige Anforderungen erinnern.

10.2 Kritischer Rückblick

Die Modellierung und die risikobasierte Optimierung von Anforderungslandschaften fußen auf Ergebnissen der Interview-Studie und damit auf Erfahrungen aus der Praxis. Die Konzepte wurden systematisch abgeleitet und haben einen direkten Bezug zu den Problemen, die in der Interview-Studie festgestellt wurden. Dennoch ist eine empirische Überprüfung der Ergebnisse in Zukunft wichtig, um zu validieren, dass sie auch tatsächlich die Probleme in der Praxis verbessern können. Hierunter fällt auch die Handhabbarkeit der Ansätze – also die Frage, wie leicht es einem Requirements Methodiker fällt, eine Anforderungslandschaft zu modellieren oder zu optimieren. Der Requirements Methodiker ist ein Spezialist, von dem auch erwartet werden kann, dass er sich in die Konzepte einarbeitet und sie entsprechend auch beherrschen wird. Dennoch können die

Ansätze mithilfe von Erkenntnissen über ihre Handhabbarkeit potenziell verbessert werden.

Weitere Limitierungen des Anforderungslandschafts-Konzeptes liegen in ihren Möglichkeiten, komplexe Beziehungen zwischen Anforderungsartefakten darzustellen. Aus dem Ziel heraus, eine möglichst simple Darstellung für Anforderungslandschaften zu finden – und auch in Anlehnung an die bestehenden Literatur [85], [109] – werden in Anforderungslandschaftsmodellen nur wenige Assoziationsformen definiert. Die Unterscheidung zwischen Verknüpfungen und Abhängigkeitsrelationen sowie die spezielle Darstellung zusammengesetzter Artefakte erweitert die bestehende Literatur bereits um zwei zusätzliche Beziehungsformen. Diese gehen genau auf die in der Praxis festgestellten Herausforderungen der Koexistenz von Artefakten ein und sind deswegen auch sinnvoll. Dennoch wären auch weitere Beziehungsformen denkbar, die derzeit nicht dargestellt werden können:

- Es kann Sinn machen, auch Vererbungsbeziehungen zwischen Artefakten zu kennzeichnen, wie in [113]. Ohne Vererbungsbeziehung lässt sich die folgende Situation beispielsweise nur schwer darstellen: Eine User Story muss immer mit mindestens einem Element im Use Case verknüpft werden, wobei ein Element ein Use Case-Schritt oder eine Erweiterung sein kann.
- Zusammengesetzte Artefakte stellen eine Möglichkeit dar, eine spezielle hierarchische Beziehung zwischen Artefakttypen darzustellen. User Story Maps [125] sind nun zweidimensionale Gebilde, bei denen User Stories in zwei Dimensionen – zu User Tasks (übergeordnete grobgranulare Nutzeraktivität) und Releases – zugeordnet werden. Story Maps erlauben eine effiziente Navigation von User Stories zu beiden Artefakttypen, den Activities und den Releases. Jedoch lässt sich das in der Anforderungslandschaft nicht darstellen, wie in Abbildung 21 (Seite 100) sichtbar.

Die vorgestellte Methode zur risikobasierten Optimierung bietet eine gute systematische Möglichkeit, um auf bestehende Erfahrungen zur Handhabung von Anforderungsartefakten zuzugreifen. Auch die Widersprüchlichkeit, die oft in Abwägungen von Vor- und Nachteilen zu einer Alternative herrscht, wird mit dem Ansatz beherrscht. Dennoch ist damit die globale Optimierung – also das Finden der optimalen Landschaft unter allen möglichen Kombinationen von Anforderungsartefakten – verhältnismäßig aufwändig, da potenziell viele Aspekte zu vielen Dimensionen bewertet werden müssen. Die Eignung liegt daher eher in der lokalen Optimierung von Landschaftsaspekten, welche die Projektteilnehmer beispielsweise in Reflexions-Workshops als problematisch identifizieren.

Ähnlich dazu ist auch mit einem hohen Aufwand zu rechnen, wenn der Requirements Methodiker alle Trace-Operationen zu einer Anforderungslandschaft einzeln festlegen muss. Die meisten Operationen werden symmetrisch angelegt. Wenn verlangt wird, dass jede User Story mit mindestens einem Use Case-Schritt verknüpft wird, so wird man häufig auch die Regel festlegen, dass jeder Use Case-Schritt mit einer User Story verknüpft wird. Genauso ist es möglich, dass in einem Projekt für verschiedene Verknüpfungen dieselben Operationen festgelegt werden. Wird in einem Projekt eine dynamische Kritik zwischen User Story und Geschäftsprozess-Aktivität festgelegt, so ist auch wahrscheinlich, dass dieselbe dynamische Kritik zwischen User Story und Use Case gewünscht sein wird. Solche Situationen müssen durch konzeptuelle oder werkzeuggestützte Bündelung von Trace-Operationen unterstützt werden, um den Aufwand im Rahmen zu halten. Beispielsweise ist in Zukunft ein Ansatz denkbar, bei dem der Requirements Methodiker zu einer Verknüpfung eine gewünschte Verknüpfungsfestlegung festlegt und das System daraufhin eine Grundkonfiguration an passenden Trace-Operationen vornimmt. Der Requirements Methodiker muss dann nur noch zur Feinjustierung einzelne Trace-Operationen abwandeln. Die Idee, die Trace-Operationen mit Eigenschaften der Anforderungslandschaft zu verbinden, wurde in Abschnitt 8.2.2 erläutert.

Eine weitere Herausforderung für die Trace-Operationen ist, dass sie bestehende Verknüpfungen benötigen. Wie auch in der Interview-Studie festgestellt, sehen viele Projektteilnehmer in der Erstellung von Verknüpfungen einen zu hohen Aufwand und nutzen daher bestehende Tracing-Ansätze nicht. Der IRE-Ansatz setzt zunächst am Nutzen-Aspekt des Kosten-Nutzen-Verhältnisses von Traceability an. Das bedeutet, dass mithilfe von Trace-Operationen der Nutzen von Verknüpfungen erhöht wird, sodass Projektteilnehmer Verknüpfungen als lohnenswerter erachten. Entsprechend stellt auch die Validierung des IRE-Ansatzes – zumindest in bestimmten Situationen – einen Nutzen von Verknüpfungen und Trace-Operationen fest. Dennoch muss für eine korrekte Einschätzung des IRE-Ansatzes auch der Kosten-Aspekt betrachtet werden. In Zukunft sollte man dazu den Aufwand der Verknüpfungserzeugung mithilfe des IRE-Ansatzes evaluieren und ihn mit geeigneten Methoden reduzieren. Dabei können auch bereits die Mechanismen des Ansatzes selbst, also Trace-Operationen und integrierte Sichten, zur Aufwandsreduktion genutzt werden. Beispiele hierzu sind Trace-Operationen, die basierend auf textuellen Analysen der Anforderungen direkt Vorschläge für Verknüpfungen anbieten. Integrierte Sichten können beispielsweise durch Drag&Drop-Mechanismen helfen, Artefakte aus den unterschiedlichen Bereichen leichter zu verknüpfen. Weitere Beispiele wurden zu Beginn von Kapitel 9 erläutert.

10.3 Ausblick

Die Arbeit reißt viele Punkte an, die an vielen Stellen weiter entwickelt oder weiter evaluiert werden können.

Evaluation der Verwendung des Anforderungslandschafts-Konzeptes. Das Konzept der Anforderungslandschaften und ihrer relevanten Eigenschaften ist aus den Ergebnissen der Interview-Studie abgeleitet. Somit besitzen die Elemente eine Berechtigung. Es wäre dennoch zu prüfen, ob Anforderungslandschaften tatsächlich in der Praxis anwendbar und auch nützlich sind. Insbesondere da Anforderungslandschaften auch in Workshops mit Entwicklern oder Stakeholdern eingesetzt werden sollen, ist zu prüfen, inwiefern die Notation für diese Personengruppen verständlich ist. Die Nützlichkeit des Anforderungslandschaftskonzeptes ist noch schwieriger festzustellen. Nützlich ist das Konzept erst, wenn es tatsächlich dazu führt, dass in einem Projekt bessere Anforderungslandschaften eingesetzt werden, als ohne die explizite Modellierung der Anforderungslandschaften. Für die Feststellung der Nützlichkeit wären demnach Langzeitstudien in realen Softwareprojekten notwendig.

Unterstützung der inhaltlichen Erstellung von Anforderungslandschaften durch konkrete Integrationsprozesse. Der beschriebene Ansatz ist generisch auf alle möglichen Anforderungsartefakte anwendbar. Requirements Methodiker müssen selbst entscheiden, welche konkreten Artefaktarten sie in einem Projekt verwenden wollen. Es wurden beispielhafte Kombinationen in Kapitel 6.7 genannt. Dennoch wäre es auch nützlich, konkrete typische Konstellationen systematischer und genauer zu untersuchen, um so auch zu einer inhaltlichen Hilfestellung für Requirements Methodiker zu kommen. Beispielsweise ist dazu zu klären, welche Artefakte gut auseinander abgeleitet werden können und welche Überlappungen und disjunkten Elemente Artefaktarten wie Geschäftsprozessmodelle, User Stories und Spezifikationen typischerweise besitzen. Eine User Story kann beispielsweise einem Use Case oder alternativ einem Use Case-Schritt zugeordnet werden. Ähnlich ist es möglich, eine User Story mit einem Prozess oder nur einer einzelnen Aktivität im Prozessmodell zu verknüpfen. Zu einigen Elementen gibt es möglicherweise auch gar kein Pendant, mit dem das Element verknüpft werden könnte. Durch Abwandlungen lassen sich aber auch untypische Informationen in bestimmten Artefaktarten abbilden. Beispielsweise stellt Leffingwell [94] eine Variante vor, mit der ein Teil der nichtfunktionalen Anforderungen als User Stories repräsentiert werden können.

Mit dem Mapping von Use Cases auf Geschäftsprozesse in Form von ereignisgesteuerten Prozessketten hat sich Lübke [104] bereits befasst. Die Umwandlung von Use Cases in BPMN-Geschäftsprozesse wird in [106] und [98] aufgegriffen. Erste Ideen zum parallelen Einsatz von Geschäftsprozessmodellen und User Story Maps wurden in [64] entworfen. Der parallele Einsatz von GUI-Mockups und User Story Maps wurde in [99], [124], [10] und [40] angerissen. Entsprechend können diese Arbeiten fortgesetzt, ihre Anwendung in der Praxis

weiter evaluiert und allgemein weitere Konstellationen von Anforderungsartefakten untersucht werden.

Integration mit FLOW-Modellen. FLOW-Modelle, wie von Stapel und Schneider [155] beschrieben, zeigen Informationsflüsse zwischen Personen und Dokumenten in einem Softwareprojekt. Mit ihrer Hilfe lassen sich Probleme in der Kommunikation zwischen Projektteilnehmern aufdecken und Informationsflüsse optimieren [142]. Wenn beispielsweise zwei eng verzahnte Rollen nur indirekt über Dokumente kommunizieren, aber besser direkt miteinander sprechen sollten, so lässt sich dies im FLOW-Modell festlegen. Genauso kann bei ausschließlich direkter Kommunikation festgelegt werden, dass wichtige Information dokumentiert werden soll, da sie sonst mit der Zeit verloren gehen kann. Die Sichtweisen aus FLOW-Modellen können eine gute Ergänzung zur Sicht auf die Anforderungslandschaft eines Projektes sein. Damit können Informationen über die Rollen, die bestimmte Artefakte nutzen, dargestellt werden. Diese Information kann dann herangezogen werden, um zu hinterfragen, ob ein Artefakt einen passenden Blickwinkel für eine bestimmte Rolle darstellt und um zu bestimmen, welche Artefakte in der Landschaft verknüpft werden sollten. Grundsätzlich ist in Softwareprojekten die direkte Kommunikation eine wichtige Ergänzung zur Dokumentation von Anforderungen (siehe bspw. [58], [143]). Da Anforderungslandschaften jedoch bisher nur darstellen, welche Information auch tatsächlich auf Artefakten dokumentiert wird, kann nicht berücksichtigt werden, dass weitere Information informell ausgetauscht wird und möglicherweise dafür keine Anforderungsartefakte benötigt werden. Mit zusätzlichen Notationselementen aus FLOW-Modellen könnte dem ergänzenden Informationsaustausch durch Kommunikation auch in Anforderungslandschaften Rechnung getragen werden. Bei einer Integration wäre zu beachten, dass die Assoziationen – und insbesondere die Unterscheidung zwischen durchgezogenen und gestrichelten Linien – in beiden Modellen unterschiedliche Bedeutungen haben.

Evaluation der risikobasierten Optimierung von Anforderungslandschaften. Ähnlich wie das Konzept der Anforderungslandschaften muss auch der risikobasierte Ansatz zur Optimierung der Landschaften evaluiert werden, da auch hier die Inhalte aus Erfahrungen aus der Interview-Studie abgeleitet, aber nicht unabhängig davon validiert wurden. Auch hier sind die Dimensionen der Anwendbarkeit und der Nützlichkeit zu betrachten. Es wäre zu prüfen, wie leicht oder schwer es Projektteilnehmern fällt, die Wahrscheinlichkeit und den Schaden der konkreten Risikofaktoren abzuschätzen. Zur Nützlichkeit ist festzustellen, ob der Optimierungsansatz tatsächlich zu besseren Anforderungslandschaften führt. Hierzu wäre ein Maß der Güte von Anforderungslandschaften ebenfalls eine hilfreiche weitere Entwicklung.

Weitere Trace-Operationen. Neben den hier vorgestellten Trace-Operationen, die *statische und dynamische Kritiken* sowie *Computed Attributes* darstellen,

lassen sich noch weitere Operationen beschreiben. Werden beispielsweise als Reaktion nicht nur das Anzeigen der Query-Ergebnisse, sondern *beliebige Funktionsaufrufe* zugelassen, so lassen sich Trace-Operationen stärker für die Konsistenzsicherung oder die leichtere Erstellung von Verknüpfungen nutzen. Beispielsweise können dann an Funktionen, wie das *Duplizieren* eines Anforderungsartefaktes zusätzliche Aktionen, wie die automatisierte Kopie der Verknüpfungen, angehängt werden. Bei der Erstellung eines neuen Artefaktes könnte automatisch ein zugehöriges Artefakt von einem abhängigen Artefakttyp erzeugt werden.

Ein weiteres unterstützendes Konzept könnte die Erfassung von *Transaktionen mit zusammengehörigen Erstellungs- oder Änderungsvorgängen* sein. Werden beispielsweise bei der Änderung einer Anforderung mehrere Artefakte angepasst, die Informationen zu dieser Anforderung enthalten, so können diese Anpassungen gruppiert werden. So kann später besser nachvollzogen werden, warum ein einzelnes Anforderungsartefakt geändert wurde, indem zur Änderung die zugehörige allgemeine Transaktion betrachtet wird. Ähnlich könnte eine ganze Transaktion auf einmal rückgängig gemacht werden, wenn eine Anpassung nicht zulässig war.

Stärkere Verbindung der Konzepte Anforderungslandschaft und Trace-Operation. Wie in Abschnitt 8.2.2 beschrieben, können durch die stärkere Verbindung von Verknüpfungstypen und Trace-Operationen in Zukunft neue Möglichkeiten geschaffen werden, die beiden Konzepte noch umfangreicher zu verwenden.

Beispielsweise kann das Konzept so erweitert werden, dass eine *Verknüpfungstärke* als Attribut von Verknüpfungstypen direkt festlegt, welche Richtlinien und Trace-Operationen mit der Verknüpfung verbunden werden. Andersherum kann der Umgang mit Anforderungen Hinweise auf Abhängigkeiten oder Relevanz der Artefakttypen in der Anforderungslandschaft geben. Werden Artefakte besonders häufig benutzt, können diese als besonders relevant in der Landschaft dargestellt werden. Werden Artefaktpaare besonders häufig gemeinsam verwendet, so kann deren Abhängigkeitsrelation in der Landschaft als besonders relevant dargestellt werden.

Bessere Unterscheidung von relevanten und irrelevanten Anpassungen der Anforderungen. Bisher werden dynamische Kritiken ausgelöst, wenn eine bestimmte Methode, wie die Methode *edit()*, bei einem Artefakt ausgeführt wird. So wird die dynamische Kritik jedoch bei jedem Aufruf ausgeführt, unabhängig davon, ob es sich bei der Änderung um eine inhaltliche Änderung oder zum Beispiel nur um die Korrektur eines Rechtschreibfehlers handelt. Dies kann dazu führen, dass dynamische Kritiken schnell stören, wenn sie in irrelevanten Situationen ausgelöst werden. Besser wäre es, zum Auslöser genauere – beispielsweise heuristische – Regeln angeben zu können, die den Auslöser auf bestimmte Situationen einschränken. Heuristische Regeln für statische Kritiken

zu Anforderungsartefakten wurden bereits von Knauss [83] vorgestellt. Das Konzept lässt sich auf dynamische Kritiken ausweiten, indem die möglichen Auslöser einer Trace-Operation angepasst werden. Außerdem lassen sich die Möglichkeiten, relevante Auslöser zu spezifizieren durch Anwendung von linguistischen Analysen aus dem Natural Language Processing-Bereich ausweiten. So kann beispielsweise die Änderung eines Objektes in einem Satz vom Singular in den Plural als relevant gelten, während die Änderung eines Objektes zu einem Synonym nicht in jedem Fall relevant ist.

Abwägung von Erstellung einer verlinkten Anforderungsmenge zu Nutzen durch darauf aufbauende Trace-Operationen. Es muss untersucht werden, ob sich der Aufwand für die Erstellung der Traceability im Vergleich zu dessen Nutzen lohnt. Bisher wird Traceability nur in sehr komplexen Projekten oder in regulierten Umgebungen eingesetzt. Das liegt auch daran, dass der Erstellungs- und Wartungsaufwand für gültige und nützliche Links als sehr hoch gilt. In dieser Arbeit ist es das Ziel, durch Trace-Operationen auch leichtgewichtiger Projekte zu unterstützen. Daher ist es wichtig, zu prüfen, ob Verknüpfungen ohne hohen Zusatzaufwand erstellt werden können. Dazu können auch helfende Prozesse definiert werden, welche die Effizienz bei der Erstellung von Verknüpfungen steigern. Werden Elemente beispielsweise direkt bei ihrer Erstellung verknüpft oder sogar direkt auseinander *abgeleitet* und automatisiert verknüpft, so ist der Aufwand geringer, als wenn die Elemente nachträglich betrachtet werden und von neuem nach abhängigen Elementen gesucht wird.

Visualisierung von Anforderungen und Verknüpfungen. Mit den *integrierten Sichten* ist ein erster Schritt in die Richtung einer neuen Visualisierung von abhängigen Anforderungen und Verknüpfungen erfolgt. Auch diese Richtung lässt sich weiter verfolgen, indem genauer untersucht wird, welche weiteren Visualisierungsmöglichkeiten bestehen und welche davon besonders hilfreich im täglichen Umgang mit Anforderungen sind. Hier ist beispielsweise eine Integration mit dem Ansatz von Ghazi [55] denkbar, bei dem alle Artefakte stets sichtbar sind und Zoom-Mechanismen relevante Artefakte in den Fokus und irrelevante Artefakte in die Peripherie rücken.

Auch für die Visualisierung von Verknüpfungen gibt es unterschiedliche Ansätze. Abhängige Artefakte zu einer Anforderung können farblich hervorgehoben, in einer Liste dargestellt oder wie in einem Graph durch Linien zwischen den Artefakten dargestellt werden. Es stellt sich weiterhin die Frage, ob die Anforderungen stets in ihrem natürlichen Kontext, also zusammen mit allen benachbarten Anforderungen, angezeigt werden sollten oder ob sie aus diesem Kontext herausgenommen werden sollten, um die Sicht nur auf abhängige Anforderungen zu konzentrieren. Studien zum Verhalten bezüglich unterschiedlich visueller Abhängigkeiten könnten beispielsweise durch Eye-Tracking unterstützt werden.

Auch die Frage, wie Kritiken angezeigt werden müssen, um die Aufmerksamkeit der Nutzer noch stärker auf abhängige Anforderungen zu lenken, ist interessant. Da aufdringlichere Visualisierungen die Nutzer auch stören, muss hier ein Mittelweg gefunden werden, der Nutzer dazu bewegt, abhängige Artefakte zu überprüfen, sie aber nicht zwingt, ihre eigentliche Aktivität zu unterbrechen. Muss beispielsweise der Nutzer explizit bestätigen, dass er abhängige Artefakte überprüft und angepasst hat, bevor er mit einer Aktivität fortfahren kann, kann dies zu höherer Ignoranz oder gar zum Verzicht auf die Verwendung des Werkzeugs führen.

Unterstützung einer dynamischen Zuordnung zu Artefakttypen. Im hier vorgestellten Ansatz muss zu einem Anforderungsartefakt bei dessen Erstellung direkt festgelegt werden, von welchem Artefakttyp dieses sein soll. Es kann jedoch von Vorteil sein, Information zunächst einfach festzuhalten, ohne direkt einen endgültigen Typ festzulegen [32]. Beispielsweise werden so in einem Workshop Informationen zunächst schnell in Form von Notizen festgehalten und dann im Anschluss vom Requirements Engineer in vollwertige Use Cases oder Geschäftsprozessmodelle umgewandelt. Correia et al. [31] haben mit *Weak-i* ein schwach typisiertes Wiki vorgestellt, das so einen Entstehungsprozess für Anforderungsartefakte unterstützt. Dort kann der Typ eines Artefaktes – also eine Wiki-Seite – nachträglich festgelegt und später auch geändert werden.

Dieses Prinzip könnte auch auf Anforderungsmengen in dieser Arbeit angewendet werden. Dadurch könnte eine bessere Unterstützung von anforderungsbezogenen Aktivitäten und ein direkteres Zusammenwirken abhängiger Artefakte erreicht werden. Neben beliebigen Änderungen des Typs zu einem Artefakt könnten auch spezielle *erlaubte Ableitungen* festgelegt werden, die den typischen Entstehens- und Konkretisierungsprozess von Anforderungen unterstützen. Beispielsweise hat Ambler in [5] zu allen 37 Artefakttypen auch angegeben, welche davon in welche anderen Typen überführt werden können.

Ernicke [45] hat am Beispiel der dokumenten- bzw. graphenorientierten Datenbank OrientDB untersucht, wie Anforderungslandschaften mithilfe einer generischen Graphenstruktur mit beliebigen Verknüpfungsmöglichkeiten instanziiert werden können. Über eine eigene Präsentationsschicht hat er eine Typisierung der Datenbankobjekte zu speziellen Artefakttypen vorgenommen und dann über Regeln innerhalb dieser Präsentationsschicht erlaubte Überführungsmechanismen festgelegt. Auch dieser Ansatz lässt sich als Basis für Anforderungen mit dynamischen Artefakttypen verwenden.

Um die Änderung von Artefakttypen zu ermöglichen, muss jedoch ein Ansatz gefunden werden, der die Verknüpfungen zwischen den konkreten Artefakten konsistent mit den erlaubten Verknüpfungen hält. Wird der Typ eines Anforderungsartefaktes geändert, so sind manche Verknüpfungen möglicherweise nicht mehr zulässig.

Anhang A – Interview-Aussagen mit Ableitung von Risiken

Die in Abschnitt 7.6 genannten Risikofaktoren für die Optimierung von Anforderungslandschaften sind durch Interpretation und Weiterentwicklung von Aussagen aus der Interviewstudie entstanden. Zur Nachvollziehbarkeit werden hier die konkreten Aussagen dargestellt.

Dabei ist zu beachten, dass die Aussagen als Anregung für die Risikofaktoren zu verstehen sind, jedoch nicht unbedingt direkt den damit verknüpften Risiken entsprechen. Durch Interpretation und Weiterentwicklung hat die Autorin aus den Aussagen auf mögliche Eintrittsbedingungen oder Folgen von Risiken geschlossen. Entsprechend beziehen sich einige Aussagen nur auf Teile eines Risikofaktors oder stellen einen indirekten Bezug dar, in dem Erfahrungen aus gegenteiligen Situationen berichtet werden.

Risikofaktoren für Artefakte	
<i>Risikofaktoren bei Aufnahme eines Artefaktes</i>	
R-1.1	Verlängerung der Analysephase
<p>„[Interviewer: <i>Auch in den großen Projekten, in denen zuerst die Analyse stattfindet, ist es nicht einfach möglich, die Anforderungen detailliert zu beschreiben?</i>] Genau. [Warum?] [...] Meist ist es ja so, dass ein sehr hoher Druck ist. Man möchte einfach <i>beginnen</i>. Man möchte sehr schnell zu Ergebnissen kommen. [...] Es ist natürlich schon so, dass der Auftraggeber wissen möchte, was das ungefähr kostet, wenn ich ein neues System baue. [...] Deswegen wird das schon aufgeschrieben, aber eben nur so grob, wie man es relativ schnell hinbekommt.“ (114 – 31:47)</p> <p>„[Use Cases ausformulieren.] Das war natürlich langwieriger, als mit diesen Powerpoints. [...] Es ist schon ein bisschen aufwendiger.“ (113 – 8:02)</p> <p>„Es herrscht permanenter Zeitdruck. Am Anfang war eine sehr lange Phase der Entwicklung der Requirements [mit detaillierter Spezifikation der Aktivitätsdiagramme]. Und dann wurde losgelegt. Jetzt ist es eher sehr zeitnah. Die Kunden schicken uns eine Powerpoint, das wird im wöchentlichen Meeting besprochen. Eine Woche drauf sagen wir denen, wie wir es machen würden.“ (113 – 25:34)</p> <p>„[Interviewer: <i>Die Spezifikateure arbeiten jetzt mit Stories, nachdem sie vorher mit Dokumenten gearbeitet haben. Was hat sich seitdem verbessert?</i>] Mein Eindruck war, dass die Aufgaben völlig unterschätzt wurden. So eine Spezifikation [Teil-Fachkonzept] zu schreiben, ist immer zu klein angesetzt worden. Am Ende des Sprints hat man geguckt und dann war die Hälfte nicht fertig spezifiziert. Da wurden immer wieder Termine gerissen.“ (115 – 75:50)</p>	

R-1.2	Zu detaillierte Spezifikation legt Projekt auf ungeeignete / ungünstige Lösungen fest
<p>„Man hat Sachen, die eigentlich gar nicht so genau spezifiziert werden müssen, so tief spezifiziert, dass es mit den Systemen, in die es letztlich integriert werden soll, nicht umsetzbar ist. Weil es von der Struktur her nicht passt.“ (I3 – 25:51)</p> <p>„Wasserfall-mäßig durchspezifizieren – da haben wir gesehen, dass wir das nicht schaffen. Es sind komplexe Systeme, und es ist schwierig. Fachlich kann man auch nicht immer alles durchdringen und man vergisst sehr viel. Agile Aspekte, wie <i>Responding to Change</i>, das brauchen wir hier.“ (I15 – 18:05)</p> <p>„Selbst mit der Erfahrung, die ich jetzt nach 3 Jahren habe: mal eben den einen oder anderen Teilaspekt <i>vollständig</i> zu spezifizieren, da weiß ich, dass ich etwas vergesse. Dem Fachbereich und den Business Analysten geht es genauso. Das heißt, wir wussten von Anfang an, dass wir uns nicht hinstellen konnten ‚wir machen eine Spezifikation bis zum Ende‘. Dann hätten wir einen tollen Papiertiger gebaut, und in der Umsetzung merken wir nach 3 Monaten, dass man die Hälfte wieder wegwerfen kann.“ (I15 – 18:26)</p> <p>„In so einem Word-Dokument hat man mehr Platz und kann alles mehr ausformulieren, detaillierter machen und Bilder einfügen. Man denkt, es wird sich wahrscheinlich sowieso nichts mehr ändern, denn man hat es jetzt verstanden. Was leider nicht der Fall ist. Es ändert sich immer etwas.“ (I6 – 19:53)</p>	
R-1.3	Viele Relationen zu anderen Artefakten; häufige Anpassungen der Anforderungen -> Hoher Maintenance-Aufwand
<p>„[Hinzufügen von Aktivitätsdiagrammen mit angehängten textuellen Erläuterungen] Dann hat man ja das nächste Modell, wo wieder Text drin steht, der sich auch wieder ändern kann. Und die Modelle lassen sich auf keinen Fall [nach bestimmten Informationen] durchsuchen. Der Text hängt da rum und den muss im Zweifel wieder jemand anpassen. Dann habe ich das Word-Dokument, die Übersicht und noch ein weiteres Dokument mehr, das gepflegt werden muss.“ (I16 – 60:00)</p> <p>„[Interviewer: <i>Warum gibt es keine Verknüpfungen zwischen den Artefakten?</i>] Es ist aufwendig, so etwas zu pflegen. Aber ich denke, das Problem ist auch: wir haben E-Mails als Meeting-Minutes, Powerpoints als Requirements, dann die einzelnen IT-Tasks [im RE-Tool], dann das Design-Dokument. Das muss alles gepflegt werden - und sollte auch gepflegt werden. Aber das bleibt oftmals durch die Zeit auf der Strecke. Und natürlich auch durch diese verschiedenen Systeme. Hätte man ein Requirements Management-System, vielleicht würde das helfen. Es ist immer zeitaufwendig und es ist auch nervig. Das kenne ich aus Projekten, wo das ganz detailliert betrieben wird mit den ganzen Verbindungen. Da muss nachher eine Person für abgestellt werden, die das Pflegt. Das ist nachher wirklich nicht mehr ohne. Gerade bei komplexen Projekten.“ (I13 – 35:26)</p>	

R-1.4	Zähheit -> lähmt Projekt
<p>„Sachen, die einen halben oder einen Tag dauern, das wird bei uns als Requirement Defect behandelt, läuft im normalen Fehler-Fixing-Prozess mit. Das ist dann ok, bei so kleinen Sachen. Wenn es größere sind, dann wird das ein Change. Aber wir brauchen eben auch etwas, um ein bisschen [leichtgewichtiger zu sein]. Z.B. wenn jemand einen Text geändert haben will. Wir können uns da hinstellen und sagen "Nee, wir haben es so gebaut, wie ihr es spezifiziert habt.", und können einen Change Request stellen. Und dann diskutieren wir 2-3 Tage über eine Label-Änderung. Das kann ja nicht sein. Also haben wir da etwas Leichtgewichtigeres.“ (I15 – 72:45)</p> <p>„Es ist natürlich Aufwand, erstmal etwas [Change Request] erstellen zu müssen, das dann vielleicht in der Spezifikation anzupassen. Aber man will ja die Prozesse haben. Warum es jetzt manchmal so schnell gemacht wird, ist, dass früher alles schneller ging. Da gab es so einen Change Prozess gar nicht. Die Entwickler waren alle allen namentlich bekannt. Und dann wurde gesagt, hier das funktioniert nicht, ändere das doch nochmal. Dann wurde es geändert und dann war es direkt im Produktivsystem. Jetzt fängt man auf einer Testumgebung an, dann geht es in die nächste Testumgebung und irgendwann geht es in die Produktion. Und das muss man auch noch beantragen mit 5 Tagen Vorlauf. Und dann hat man einen Riesen-Rattenschwanz, den man hinter sich herzieht. Aber der Fachbereich ist es von früher gewohnt, und sagt, „Das muss doch funktionieren. Es kann ja nicht sein, dass es schlechter wird.“ (I16 – 17:05)</p>	
<i>Risikofaktoren bei Verzicht auf ein Artefakt</i>	
R-1.5	Projekt läuft aus dem Rahmen
<p>„Wir wollten agil bleiben, aber wir hatten auch gemerkt, dass wir in der ersten Projektphase nicht genau wussten, wo es langgehen soll. Eine Roadmap, die hat gefehlt. Damit man weiß, wo man hin will. [...] Man muss auch das Ziel kennen, um zu sagen, wie weit man ist. Hier muss man diesen Rahmen schon mal abgesteckt haben.“ (I15 – 19:04)</p> <p>„Das große Problem bei User Stories in dem Gesamtkontext ist natürlich, dass man nur einzelne Teilbereiche beleuchtet, aber nirgends das Gesamtkonzept erfasst. Das ist vielleicht der einzige große Nachteil von User Stories selbst.“ (I3 – 14:54)</p> <p>„Was in jedem Fall selbst bei einem agilen Softwareprojekt bestehen muss, ist in gewisser Form ein Pflichten- bzw. Lastenheft. [...] Das ist extrem sinnvoll, damit man eine Ebene hat, auf die man sich in jedem Fall referenzieren kann und auch darauf einigen kann, wo jetzt der gemeinsame Stand ist. [...] Und vor allem auch weiß, wie die Verbindlichkeit damit gekoppelt wird, speziell wenn es, wie bei uns, ein agiler Prozess ist.“ (I3 – 33:36)</p> <p>„Ich kann bei den Stories schlecht entscheiden, ist die Story jetzt das, was ich wirklich will? Ist der Umfang der Story deckungsgleich mit meinen Requirements? Ich möchte mich auf meine Requirements referenzieren und nicht auf Stories, die jemand anderes abgeleitet hat.“ (I4 – 74:00)</p>	

R-1.6	Häufige Anforderungswechsel oder viele Stakeholder -> Überanpassung wegen fehlendem Bezugspunkt
<p>„Es war nicht ganz nach dem agilen Muster. Sondern schon so, dass auch eine Spezifikation erstellt wird und man dort schaut, was man im nächsten Sprint machen kann. Mit dem agilen Ansatz hat man keine guten Erfahrungen gesammelt. Das Problem war wohl einfach die Stabilität der Anforderungen. Das ist auch immer noch das Problem. Es passt nicht zu dem Konzept, wenn du erstmal vier Wochen lang etwas baust, das du die nächsten vier Wochen wieder umbaut. Und dann wieder umbaut. Dann kommst du nicht voran.“ (I16 – 3:06)</p> <p>„Man hat dann irgendwann gesehen: "Anforderungen? Wo hatten wir unsere Anforderungen denn aufgeschrieben?" Wir hatten rudimentäre Beschreibungen, aber an sich wussten wir nicht, wie wir es aufschreiben sollten. Das Ergebnis war: Keine Eingabemaske [als Teil der Software] hat länger als drei Monate überlebt. Sie wurde immer wieder neu geändert.“ (I15 – 15:39)</p> <p>„Man muss auch das Ziel kennen, um zu sagen, wie weit man ist. Hier muss man diesen Rahmen schon mal abgesteckt haben. [...] Wenn nicht jeder das Ziel vor Augen hat, wenn sich nicht jeder darauf committed hat – dann möchte es der eine so haben, der andere anders.“ (I15 – 19:58)</p> <p>„Es sind Menschen darunter - teilweise aus unterschiedlichen Abteilungen, die traditionsgemäß nicht miteinander können - und natürlich gibt es da Konflikte. Konflikte werden dann so ausgetragen, dass man eben blockiert. Daher braucht man eine Roadmap, damit alle committed haben ‚wir wollen in diese Richtung‘.“ (I15 – 21:23)</p>	
R-1.7	Fehlendes kundenverständliches Artefakt -> falsche Spezifikationen oder falsche Priorisierung
<p>„Was immer hilfreich war, ist dass man ein gemeinsames Bild mit den Beteiligten bekommt. Das ist nicht immer einfach auf abstraktem Level. Wenn man jetzt so Konstrukte hat, wie Use Cases, da kann nicht jeder Anwender etwas mit anfangen. Auch auf Datenmodell-Ebene kann man in der Regel nicht diskutieren. Und so ein Workflow ist auch schon grenzwertig.“ (I7 – 5:40)</p> <p>„Was auch sehr geholfen hat, ist, Screenshots zu malen und auf dieser Basis mit Usern zu sprechen. Das sehe ich jetzt auch in anderen Projekten. Alles, was davor kommt, ist für sie manchmal sehr abstrakt. Sie können sich das besser vorstellen, wenn sie Screenshots sehen. Auf der Basis kann man besser diskutieren.“ (I7 – 18:20)</p> <p>„Als Techniker kann ich die Funktionsbeschreibungen [ähnlich zu komplexen Use Cases] lesen. Aber auch ich muss da stark hin und her [springen], wenn man z.B. 10 Querverweise hat. [...] Probleme sind das vor allem für den Fachbereich. Der kann die Dinger nicht lesen. Und was wir auch gemerkt haben, ist, dass die Tester [haben Fachwissen, aber keinen IT-Hintergrund] das auch nicht können.“ (I15 – 37:23)</p> <p>„Ich hatte diese 30-Seiten-Funktionsbeschreibung [verfasst vom Business Analysten] und ich habe den Business Analysten dann täglich einbestellt. Täglich</p>	

tauchten Dinge auf, die so nicht passten. Allein so eine Funktionsbeschreibung zu bauen, das kriegt man nicht hin. Das kann man nicht so beschreiben, dass es vollständig erklärt ist. Irgendein Detail vergisst man immer. (I15 – 38:56)

„[Spezieller Dokumenttyp: Berechnungsdiagramm]. Da hat mir jemand damals eine Formel vorgesetzt mit 40 Eingangsvariablen und allen möglichen Berechnungen. Ich sollte das programmieren. Das habe ich 10 Minuten probiert und mir dann gesagt, "nee, das schreibst du erstmal in einer anderen Form auf [...]". Dann habe ich [mit dieser anderen Notation] angefangen. [...] Das hat etwas Struktur reingebracht. Das war dann so erfolgreich, dass der Fachbereich, der diese Funktionsbeschreibungen auch nicht lesen kann, das total toll fand und ausgedrückt haben wollte. Dann haben wir nur noch über dieses Diagramm geredet. Nicht mehr darüber, was im textuellen Dokument drin steht. Denn der Fachbereich kann so etwas verstehen – das ist einfacher als so ein Word-Dokument.“ (I15 – 54:07)

Risikofaktoren für Zähheit

Risikofaktoren eines zähen Artefaktes

R-2.1	Abhängigkeiten zu flexiblen Artefakt -> ungewünschte Änderungen treten an anderer Stelle auf
-------	--

„Bei besonders relevanten Eingabemasken sind die Funktionsbeschreibungen auseinander gelaufen, inkonsistent geworden. [...] Wir haben uns dann entschieden, ein Activity-Diagramm zu nehmen. Die Funktionsbeschreibungen für besonders komplizierte Masken wurden darin auch schon überführt. Interessanterweise war bei manchen Diagrammen dann die Vorlage der Code und nicht mehr das Spezifikationsdokument.“ (I15 – 50:24)

„[Umgehen des Change Prozesses] Es gibt die unterschiedlichsten Kanäle. Es kann sein, dass jemand anruft und sagt "ich bräuchte da mal". Das ist natürlich nicht der gewünschte Weg. Zum Teil wird auch versucht, Anforderungen in Incidents unterzubringen, indem gesagt wird "das ist ein Fehler.“ (I17 – 5:57)

„Problematisch wird es, wenn die Änderungen übergreifende Teilaspekte betreffen, wo die Eingabemasken Abhängigkeiten zueinander haben. Wenn dort dann mehrere Änderungen in die Masken kommen - und das ist die Realität [dass es viele Änderungen gibt] - dann werden die umgebaut und dann werden die inkonsistent. Es ist zwar immer der Auftrag an den Spezifikateur, auch die Spezifikation gerade zu ziehen, aber wenn der ursprüngliche z.B. das Projekt verlassen hat [ist es schwierig, die Änderungen einzuarbeiten].“ (I15 – 40:25)

R-2.2	Unnötige Verzögerung durch Abstimmungsprozess
-------	---

(Indirekt): „In diesem Dokument ist eine Liste von allen Use Cases und da wird dahinter geschrieben "dieser Use Case ist jetzt vom Business freigegeben für die Implementierung". Sodass wir nicht auf das ganze Dokument warten müssen, sondern schon häppchenweise damit anfangen können.“ (I13 – 28:49)

(Indirekt): „Das Hauptproblem, das ich sehe, ist, dass oft - gerade auf der Business-Seite - einige Verantwortliche bestimmt werden und Ressourcen festgelegt werden, die dann aber nicht da sind. Es werden Leute bestimmt, um Input zu geben für die Requirements. Und wenn man die dann fragt, haben sie keine Zeit.“ (I11 – 3:29)

(Indirekt): „In ganz seltenen Fällen, wenn wir größere [Erweiterungsaufträge] haben oder mehrere Systeme involviert sind, dann gucke ich auf [die Spezifikation] nochmal drauf [ob alles richtig ist]. Ansonsten fehlt die Kapazität. [...] Dann sprechen [die Stakeholder] direkt mit den Entwicklern. Ich habe aber den Eindruck, das geht nicht komplett in die Hose.“ (I4 – 60:00)

R-2.3 | Artefakt wird nicht gepflegt

„Das würde ich nicht machen. Ich müsste es zuerst auschecken, dann müsste ich es bearbeiten, wieder hochladen. Dann stehe ich als letzter Bearbeiter da drin. Ist vielleicht auch nicht so toll.“ (I16 – 54:55)

„Theoretisch ist [bei Änderungen] vorgegeben [für Fachkonzept, das von allen abgenommen ist]: mit allen absprechen, das Dokument ändern, erst dann die Implementierung machen. In der Praxis, klar, wurde das nicht immer so komplett durchgeführt.“ (I15 – 08:05)

„[Interviewer: Warum werden Änderungen manchmal nicht dokumentiert?] Das ist aufgekommen, nachdem die Anwendung in Produktion gegangen ist. Bestes Beispiel: man hat festgestellt, es sind tausend Rechnungen [an Kunden] falsch. Wenn man jetzt losgeht, erstmal einen Change Request erstellt, alles schön aufschreibt, dann die Spezifikation anpasst und es dann irgendwann umsetzt... Dann ist vielleicht schon eine Woche vergangen, aber die Rechnungen sind alle falsch und es sind falsche Daten rausgegangen. So eine Änderung soll sofort rein. Und dann wird dem Entwickler gesagt „so ist es richtig, und die Spezifikation ändere ich dann später“. Und wenn es dann einmal umgesetzt ist, kümmert es auch keinen mehr. So rutschen Sachen rein, weil es schnell gehen muss.“ (I16 – 16:16)

„[Interviewer: Es wäre möglich, weitere Informationen oder Verweise, die zu einem Thema relevant sind, in dem Moment wenn man sie selbst gefunden hat, im Dokument nachzupflegen. Würden Sie das tun?] Nein. Ich habe ja ein Problem, das ich lösen will. Da sitzen mir vielleicht 10 Leute im Nacken. Da fange ich nicht an, Dokumente zu pflegen.“ (I16 – 52:23)

Risikofaktoren eines flexiblen Artefaktes

R-2.4 | Überanpassung

„Man hat dann irgendwann gesehen: "Anforderungen? Wo hatten wir unsere Anforderungen denn aufgeschrieben?" Wir hatten rudimentäre Beschreibungen, aber an sich wussten wir nicht, wie wir es aufschreiben sollten. Das Ergebnis war: Keine Eingabemaske [als Teil der Software] hat länger als drei Monate überlebt. Sie wurde immer wieder neu geändert.“ (I15 – 15:39)

„Jetzt wo wir in der Testphase sind, haben wir gemerkt, so ein Change

Prozess ist zwar toll, aber am Entwicklungsende gibt es jetzt erstmal keine Changes mehr! Wir müssen jetzt erstmal den Deckel zukriegen. Sonst kommt immer nochmal jemand auf die Idee, dass er etwas anders haben möchte. Sehr häufig passiert das.“ (I15 – 71:00)

„Ein weiteres Problem ist natürlich wieder das Übliche am agilen Prozess. Wenn man jederzeit Einfluss auf die Entwicklung nehmen kann, kann es natürlich auch zu so einem Prozess des dauerhaften Vergoldens eines Produktes kommen.“ (I3 – 17:12)

„Die Business-Seite testet gerade. Es kommen Dinge, wie "Das Datum fehlt mir gerade", "Ich habe einen Artikel gefunden, da sind die Stammdaten nicht richtig". Aufnehmen kann man es ja, aber wenn man das Problem jetzt direkt löst, das kostet Zeit und Energie. Ich muss neu deployen, nachtesten. Wir kommen mit den Dingen, die jetzt wichtig sind, nicht wirklich vorwärts. [...] Wenn wir alle zwei Wochen mit dem Business sprechen, kriegen wir dadurch Moving Targets rein.“ (I4 – 23:59)

R-2.5	Projektteilnehmer oder angrenzende Projekte bekommen Änderungen nicht rechtzeitig mit
-------	---

„Das Problem ist, dass (1) ein Bereich die Verwaltung der Spezifikation übernimmt, (2) der Fachbereich sich um die fachlichen Geschichten kümmert und (3) wir uns um die Umsetzung. Es ist ein Dreieck. Und unweigerlich kommt es immer wieder zu einem gewissen Kommunikations-Ausfall. Weil immer irgendwo Personen nicht mit informiert werden, wenn sich irgendwo etwas ändert.“ (I3 – 45:15)

„Meistens fängt das Problem schon vorher an. Man hat einmal eine Spezifikation aufgeschrieben und dann wurde es irgendwie umgesetzt. Dann gibt es eine Testphase und die Tester sind nicht die Leute, die es spezifiziert haben. Teilweise gab es dann auch bidirektionale Abstimmungen zwischen Spezifikateur und dem Umsetzenden. Und dann steht das nicht direkt im Fachkonzept. Oder es wurde an einer anderen Stelle ergänzt, wo es nicht direkt gesehen wird. D.h. die Tester stellen erstmal Fehler ein und wenn man dann nicht guckt... der Entwickler sieht dann, das ist als Fehler eingestuft und ändert das wieder. Und es geht nochmal auseinander. Wenn es dann wirklich in der Produktion ankommt, dann ist es vielleicht schon etwas ganz anderes als damals mal aufgeschrieben wurde. Man kann das halt nicht nachschlagen, das ist das größte Problem.“ (I16 – 6:41)

R-2.6	Änderungen an Anforderungen sind nicht mehr nachvollziehbar
-------	---

„Es wäre wichtig, sagen zu können, dass eine Anforderung bei der Abnahme *so-und-so* war und dass sie jetzt anders ist. Man könnte feststellen, ob das jetzt eigentlich ein Change ist, ob man da jetzt mehr Budget bekommt. Wenn man das sehr sauber beschreibt, dann wäre die IT in einer viel besseren Lage. Weil sie dann sagen kann, dass sich von da bis da etwas geändert hat. Man könnte es wirklich zeigen und einen Vergleich [in den Versionen der Requirements] machen. Wenn ich Anforderungsdetails sowohl vorn nicht beschreibe, als auch es hinten einfach nur implementiere ohne es zu beschreiben, dann habe ich gar keine solchen Möglichkeiten. Dann ist plötzlich 20-30% Budget-Überschreitung und keiner weiß genau, wa-

rum. Alle ahnen es vielleicht, aber keiner kann es genau belegen.“ (I14 – 30:40)

„Vor zwei Jahren war das Projekt beendet. Da wussten alle relativ genau, was das Tool machen sollte. Jetzt sind wir zwei Jahre weiter und haben 500 oder mehr Changes gemacht. Und diese Changes finden sich nur in den Issue-Objekten des RE-Tools. Das Operating von dem Tool wird von einer anderen Gruppe gemacht. Wenn eine Business-Person anruft und sagt, das Tool funktioniert nicht richtig, weiß der vom Operating nicht, ob das stimmt. Jetzt muss er nachgucken, ob es dazu einen Change gab, findet den aber gar nicht mehr. Er weiß gar nicht, wie das Tool zum jetzigen Zeitpunkt funktionieren müsste. Der muss jetzt in diesen hunderten von Issue-Objekten suchen.“ (I4 – 65:00)

Risikofaktoren für Umgang mit Abhängigkeiten

Risikofaktoren bei Halten zweier sich entsprechender Artefakttypen

R-3.1 | Bei Anpassungen eines Artefaktes wird das entsprechende andere Artefakt nicht angepasst

„Man selbst ist schon so weit im Thema drin, dass man nicht mehr ins Fachkonzept reinschaut. Sodass es irgendwann vom Fachkonzept in den E-Mail- oder Telefonverkehr übergeht.“ (I18 : 12:58)

„Es war schwer, [Spezifikation und User Stories] synchron zu halten. Es gab viele [externe] Änderungen, [von denen manche nur] in die Spezifikation eingepflegt wurden. Wir wussten aber nicht genau, wo. Es gibt Unterschiede zwischen der Spezifikation und dem Product Backlog und ich bin jetzt hinterher, das Product Backlog so weit es geht zu aktualisieren. Dieses Änderungs-Tracking macht das ganze schwierig. In der Excel-Datei, wo der [fachliche] Workflow beschrieben wurde, standen auch einige Anforderungen drin. Auch das wurde nicht aktuell gehalten. In beiden Dokumenten waren ein Paar Anforderungen versteckt. Das macht das ganze auch schwieriger.“ (I6 – 13:32)

„Changes wurden im Testmanagement-Tool als Defects dokumentiert. Das Problem ist, die ganzen Spezifikationen sind Word-Dokumente und dann hat man schon wieder das erste Gap. Man hat den Change Request irgendwo im Tool liegen, der wird dann auch umgesetzt, aber die Spezifikation wird nicht angefasst. Wenn dann jemand später [auf die Spezifikation] schaut, der sucht ja nicht im Tool, sondern er schaut in die Spezifikation und da steht der alte Stand.“ (I16 – 13:03)

R-3.2 | Informationen müssen aus mehreren Artefakten zusammengetragen werden

„Es war schwer, [Spezifikation und User Stories] synchron zu halten. Es gab viele [externe] Änderungen, [von denen manche nur] in die Spezifikation eingepflegt wurden. Wir wussten aber nicht genau, wo. Es gibt Unterschiede zwischen der Spezifikation und dem Product Backlog und ich bin jetzt hinterher, das Product Backlog so weit es geht zu aktualisieren. Dieses Änderungs-Tracking macht das ganze schwierig. In der Excel-Datei, wo der [fachliche] Workflow beschrieben wurde, standen auch

einige Anforderungen drin. Auch das wurde nicht aktuell gehalten. In beiden Dokumenten waren ein Paar Anforderungen versteckt. Das macht das ganze auch schwieriger.“ (I6 – 13:32)

„Changes wurden im Testmanagement-Tool als Defects dokumentiert. Das Problem ist, die ganzen Spezifikationen sind Word-Dokumente und dann hat man schon wieder das erste Gap. Man hat den Change Request irgendwo im Tool liegen, der wird dann auch umgesetzt, aber die Spezifikation wird nicht angefasst. Wenn dann jemand später [auf die Spezifikation] schaut, der sucht ja nicht im Tool, sondern er schaut in die Spezifikation und da steht der alte Stand.“ (I16 – 13:03)

„Zum Beispiel musste ich etwas an Feature X verändern. Jetzt wusste ich gar nicht, in welchen [Teil-]Spezifikationen hängt Feature X überall mit drin? Ich weiß, es ist auf der Eingabemaske Y, da kann ich es ändern. Dort wurde das Feature geändert. Und hinterher sind [andere] Systemteile kaputt gegangen, weil da überall auch etwas mit Feature X gemacht wird. Hätte man [die Anforderungen in einer Reihe von Teil-Spezifikationen] jetzt vielleicht einfach durchsuchen können, hätte man vielleicht die Stellen gefunden.“ (I16 – 23:34)

„Mich hat gestört, dass Änderungen diskutiert wurden, auch irgendwo in Meeting Minutes aufgeschrieben wurden, aber nicht wirklich ins Dokument eingeflossen sind. Sodass man am Ende das Dokument und 10 Meeting Minutes im Kopf haben musste, um zu wissen, was da rauskommen soll.“ (I13 – 48:12)

„[Interviewer: In welchen Situationen ist es wichtig, Artefakte nachzupflegen?] Weil Fehler auftreten, z.B. ist der Rechnungsbetrag am Ende falsch. [...] Man fragt sich, wie ist der Rechenweg dahin? Den Rechenweg kann ich im Code nachvollziehen, aber das hilft mir ja nicht, denn ich muss wissen, wie es richtig ist. Und das sollte in der Spezifikation stehen. Wenn ich im Testmanagement-Tool suche, dann gibt es vielleicht noch andere Dokumente, wo irgendwelche Informationen drinstehen. Dann kriege ich das Hinterher nicht wieder zusammen. Wichtig ist, dass man eine Stelle hat, wo die Wahrheit steht.“ (I16 – 14:24)

R-3.3 | Viele Änderungen -> hoher Maintenance-Aufwand

„Dann hat man ja das nächste Modell, wo wieder Text drin steht, der sich auch wieder ändern kann. Und die Modelle lassen sich auf keinen Fall durchsuchen. Der Text hängt da rum und den muss wieder jemand im Zweifel anpassen. D.h. dann habe ich den Text, das Word-Dokument, die Übersicht, also noch ein Dokument mehr, das gepflegt werden muss.“ (I16 – 60:00)

Das Problem ist, dass sich die Blickwinkel [fachliche Probleme und technische Umsetzung] verwischen. Wenn theoretisch nur eine Seite geändert werden soll, müssen oft doch beide Seiten geändert werden. Sowohl das, was umzusetzen ist, als auch wie es umzusetzen ist und auch was eigentlich die Gesamtziele des Projektes sind usw. Das lässt sich dann deutlich schlechter separieren. (I3 – 40:46)

„Ich mache ein Paar Stichpunkte in die Tasks und wenn das nicht reicht, beschreibe ich das im Design-Dokument. Und dann verweise ich vom Task in dieses Design-Dokument, sodass ich das nicht doppelt pflegen muss. Damit man es nur an einer Stelle hat, wo es detailliert beschrieben steht.“ (I13 – 17:41)

„[Interviewer: Warum gibt es keine Verknüpfungen zwischen den Artefakten?] [...] Ich denke, das Problem ist auch: wir haben E-Mails als Meeting-Minutes, Powerpoints als Requirements, dann die einzelnen IT-Tasks [im RE-Tool], dann das Design-Dokument. Das muss alles gepflegt werden. Aber das bleibt oftmals durch die Zeit auf der Strecke. Und natürlich auch durch diese verschiedenen Systeme. Hätte man ein Requirements Management-System, vielleicht würde das helfen. Es ist halt immer zeitaufwendig und es ist auch nervig. Das kenne ich aus Projekten, wo das ganz detailliert betrieben wird mit den ganzen Verbindungen. Da muss nachher eine Person für abgestellt werden, die das Pflegt. Das ist nachher wirklich nicht mehr ohne. Gerade bei komplexen Projekten.“ (I13 – 35:26)

„Prozessdiagramme kann man ja auch als Artefakte zählen. Der Übergang von Diagramm zu Text und zurück und Konsistenzprüfung dazwischen, das ist auch [...] ein großer Aufwand.“ (I2 – 23:42)

Risikofaktoren bei Zusammenfassen der Informationen zu einem Artefakttyp

R-3.4 | Artefakt wird komplexer und weniger lesbar

„[Zusammenfassen von Aktivitätsdiagrammen und Vorgangsbeschreibungen, die vorher wie Use Case-Schritte waren.] Aktivitätsdiagramme in UML sind toll, um den groben Ablauf darzustellen, aber nicht um zu spezifizieren, wie eine Eingabemaske funktioniert. [...] Meist ist die Logik ja ein bisschen mehr. D.h. die Logik versteckt sich wieder im Text. Ich muss drauf klicken, dann sehe ich unten Text. Jetzt klicke ich aufs nächste und sehe den nächsten Text. Die Texte hängen wieder so lose rum, ich muss hin und her klicken. Dann ist es wieder schwierig, den Zusammenhang herzustellen, finde ich.“ (I16 – 59:12)

„Je größer die Organisation, umso größer ist der Overhead mit schwerewichtigen Dokumenten, wie Spezifikationen, die nicht unbedingt gut aufteilbar sind. Speziell wenn wir Änderungen im Projektverlauf unterliegen, wo man dann nicht mehr verfolgen kann, wo genau [im Dokument] die Änderungen passiert sind. Wenn man ein Paar Hundert Seiten Spezifikation hat und irgendwo ändern sich ein, zwei Sätze - das macht erstmal Spaß, das zu suchen.“ (I3 – 12:03)

„Man kann auch nicht schnell etwas nachschlagen, weil jedes Requirement zwei Seiten ausformulierter Text sind. [...] Da sind Bilder drin, da kann man schnell draufgucken, wie nochmal die Idee war, ohne den Text zu lesen. Dafür ist es gut – auch weil es die einzige Stelle ist, wo solche Bilder auftauchen. Aber sonst...“ (I6 – 25:17)

R-3.5 | Komplexe Beziehungen zwischen Artefakten -> Doppelung von Informationen

(Indirekt): Wir sind jetzt nicht in der Spezifikation auf die Datenmodelle eingegangen, die waren ja in dieser separaten Datei. Wir sind nicht auf die Oberflächen eingegangen, die waren in der Oberflächen-Datei. Beziehungsweise, sie wurden notwendigerweise kopiert und dann in die Spezifikation eingefügt. [Das erzeugte die zusätzliche] Aufgabe, darauf aufzupassen, dass da keine Widersprüche sind.“ (I8 – 11:05)

„[Interviewer: Gibt es auch Situationen, wo keine baumartige Struktur zwischen den Artefakten herrscht?] Klar, baumartige Strukturen treten mit der Dekomposition der Anforderungen auf, aber natürlich kann nachher auch Wiederverwendung einsetzen. Wenn man z.B. für ein bestimmtes Feature einen bestimmten Dialog entwickelt hat. Den realisieren Sie den in einem Feature und verwenden den, oder Teile davon, in 2-3 weiteren Features wieder mit. Daher gibt es da entsprechende Quer-Abhängigkeiten. Oder man möchte auch entsprechend ‚abilities‘ [nichtfunktionale Anforderungen] darstellen.“ (I2 – 34:32)

Risikofaktoren für Verknüpfungen

Risikofaktoren bei Verknüpfung zweier abhängiger Artefakttypen

R-4.1	Viele <i>False Positives</i> -> Verknüpfte Elemente müssen zusätzlich gefiltert werden
-------	--

„Vor allem: wie lange sind die Verlinkungen aktuell, das ist die nächste Frage. Vielleicht hat sich im Text wieder etwas geändert, aber keiner fasst die Verlinkungen an. Dann haben wir wieder das nächste Problem.“ (I16 – 55:47)

„Wir haben versucht, im Product Backlog die Kapitel, in denen die entsprechenden Anforderungen im Word-Dokument beschrieben waren, zu verlinken, damit man das wiederfindet. Aber da sich die Kapitel geändert haben, war das nicht mehr nachzuvollziehen.“ (I13 – 18:05)

„Ich habe keine Motivation, im Word-Dokument einen Link auf ein anderes Word-Dokument einzufügen, das vielleicht im nächsten Monat wieder verschoben wird, weil es eine neue Version gibt.“ (I16 – 54:23)

„Toll wäre es, wenn man eine Volltextsuche über alle [Teil-]Spezifikationen hätte. Wir haben Sharepoint, da kann man auch über alles suchen. Aber da kriegt man zu viele Ergebnisse von allen möglichen Ecken und Enden, das sind keine sinnvollen Ergebnisse.“ (I16 – 23:14)

R-4.2	Viele <i>False Negatives</i> -> Relevante Informationen werden missachtet oder nicht angepasst
-------	--

„[In der Sprintplanung wurde informell erwähnt, welche Teil-Spezifikation (ca. 12 Seiten) für eine User Story relevant ist. Interviewer: Wären Verknüpfungen zwischen Story und Spezifikation hilfreich?] Das würde ich eher nicht so sehen. Irgendjemand muss diese Verknüpfung ja auch erstellen. Und das ist auch wieder Verwaltungsaufwand. Und dann heißt es "du hast gesagt, es steht auf Seite 3-6, aber auf Seite 7 stand der eigentlich interessante Teil, den habe ich jetzt nicht gemacht." Dann sind wir wieder in dem Bereich, wo die Präzision vielleicht zu groß ist, um wirklich das Gesamtverständnis beizubehalten. Ich glaube, das ist zu genau. Die Dokumente sind meist nicht so lang.“ (I16 – 32:45)

R-4.3	<p>Hohe Kosten für Erstellung von Verknüpfungen</p> <p>„Es würde mir viel zu lange dauern, in den Dokumenten herum zu ändern. Aber wenn man sagen kann, ich klicke drauf und gebe ein Stichwort ein, dann funktioniert das. Also, wenn es weniger als 20 Sekunden dauert, dann könnte man es sich vorstellen. Aber eine Minute fände ich schon zu viel. [...] Ich müsste es zuerst auschecken, dann müsste ich es bearbeiten, wieder hochladen. Dann stehe ich als letzter Bearbeiter da drin. Ist vielleicht auch nicht so toll. Die Alternative ist, dass man ein Wort anklickt und einen Kommentar hinterlässt. [...] Aber es müsste eben schnell gehen.“ (I16 – 54:13)</p> <p>„[In der Sprintplanung wurde informell erwähnt, welche Teil-Spezifikation (ca. 12 Seiten) für eine User Story relevant ist. <i>Interviewer: Wären Verknüpfungen zwischen Story und Spezifikation hilfreich?</i>] Das würde ich eher nicht so sehen. Irgendjemand muss diese Verknüpfung ja auch erstellen. Und das ist auch wieder Verwaltungsaufwand.“ (I16 – 32:45)</p> <p>„[<i>Interviewer: Warum gibt es keine Verknüpfungen zwischen den Artefakten?</i>] Es ist aufwendig, so etwas zu pflegen. Ich denke, das Problem ist auch: wir haben E-Mails als Meeting-Minutes, Powerpoints als Requirements, dann die einzelnen IT-Tasks [im RE-Tool], dann das Design-Dokument. Das muss alles gepflegt werden. Aber das bleibt oftmals durch die Zeit auf der Strecke. Und natürlich auch durch diese verschiedenen Systeme. Hätte man ein Requirements Management-System, vielleicht würde das helfen. Es ist halt immer zeitaufwendig und es ist auch nervig. Das kenne ich aus Projekten, wo das ganz detailliert betrieben wird mit den ganzen Verbindungen. Da muss nachher eine Person für abgestellt werden, die das pflegt. Das ist nachher wirklich nicht mehr ohne. Gerade bei komplexen Projekten.“ (I13 – 35:26)</p> <p>„[Verknüpfungen erstellen:] Es ist relativ aufwändig, wenn man das nicht von vornherein macht. Und es ist wahrscheinlich auch sehr Änderungsanfällig. Darum ist unsere Empfehlung, es nicht zu übertreiben. Eher nicht von Einzelanforderung zu Einzelanforderung, sondern von Gruppe zu Gruppe zu verlinken. Z.B. sind in einem Folder 10 Anforderungen und dieser Folder deckt dann eine andere Anforderung ab. Ich glaube nicht, dass es praktikabel ist, 100% Abdeckung hinzubekommen. Aber es gibt ganz viele Gründe, warum man das trotzdem machen sollte. Man muss da einen Zwischenweg finden.“ (I14 – 80:05)</p> <p>„Inkonsistenzen sind dabei auch immer ein großes Problem. Das muss man über unterschiedliche Artefakte gewährleisten. Das ist schon eine Sache, die trotz entsprechender Requirements Management-Tools schwer zu handhaben ist. Wo man schwer auf eine Automatisierung setzen kann. Das ist schon auch letztendlich Kompetenz der Menschen, die in solchen Projekten arbeiten, das auch zu gewährleisten.“ (I2 – 22:05)</p>
<p>Risikofaktoren bei Verzicht auf explizite Verknüpfung zweier Artefakttypen</p>	
R-4.4	<p>Relevante abhängige Informationen werden missachtet oder nicht angepasst</p> <p>„Changes wurden im Testmanagement-Tool als Defects dokumentiert. Das Problem ist, die ganzen Spezifikationen sind Word-Dokumente und dann hat man schon wieder das erste Gap. Man hat den Change Request irgendwo im Tool liegen,</p>

der wird dann auch umgesetzt, aber die Spezifikation wird nicht angefasst. Wenn dann jemand später [auf die Spezifikation] schaut, der sucht ja nicht im Tool, sondern er schaut in die Spezifikation und da steht der alte Stand.“ (I16 – 13:03)

„Zum Beispiel musste ich etwas an Feature X verändern. Jetzt wusste ich gar nicht, in welchen [Teil-]Spezifikationen hängt Feature X überall mit drin? Ich weiß, es ist auf der Eingabemaske Y, da kann ich es ändern. Dort wurde das Feature geändert. Und hinterher sind [andere] Systemteile kaputt gegangen, weil da überall auch etwas mit Feature X gemacht wird. Hätte man [die Anforderungen in einer Reihe von Teil-Spezifikationen] jetzt vielleicht einfach durchsuchen können, hätte man vielleicht die Stellen gefunden.“ (I16 – 23:34)

„[Verlinkung zwischen Use Cases und abgeleiteten IT-Tasks] Die Entwickler sollten eigentlich nur noch die IT-Tasks brauchen. Ich mache aber noch die Use Case-Nummer an die IT-Tasks dran, damit man in einem halben Jahr weiß, wofür wir das eigentlich nochmal machen wollten. Das geht sonst verloren, aus meiner Erfahrung.“ (I13 – 15:27)

Mit vielen Artefakten zu arbeiten ist vielleicht kompliziert in der Relationierung der Artefakte. Wenn ich mir mal erschließen will, was wo zu gehört, sollte man sich schon Gedanken über eine entsprechende Werkzeugunterstützung machen. Dass man entsprechend ermöglicht, User Stories zu finden, die zu einem bestimmten Epic gehören.“ (I2 – 20:34)

R-4.5	Suche nach abhängigen Anforderungen nimmt viel Zeit in Anspruch und unterbricht Aktivitäten
-------	---

„Es gab viele [externe] Änderungen, [von denen manche nur] in die Spezifikation eingepflegt wurden. Wir wussten aber nicht genau, wo. Es gibt Unterschiede zwischen der Spezifikation und dem Product Backlog [...]. In der Excel-Datei, wo der [fachliche] Workflow beschrieben wurde, standen auch einige Anforderungen drin. Auch das wurde nicht aktuell gehalten. In beiden Dokumenten waren ein Paar Anforderungen versteckt. Das macht das ganze auch schwieriger.“ (I6 – 13:32)

„Derjenige, der die Spezifikation geschrieben hat, sollte die Punkte, die im Product Backlog nicht auftauchen, nochmal ansprechen. [...] Jedes Mal, wenn sich die Spezifikation ändert, kann man ja nicht wieder 40 Seiten lesen und die Änderungen suchen.“ (I6 – 26:26)

„[Interviewer: In welchen Situationen ist es wichtig, Artefakte nachzupflegen?] Weil Fehler aufgetreten sind, z.B. ist der Rechnungsbetrag am Ende falsch. [...] Man fragt sich, wie ist der Rechenweg dahin? Den Rechenweg kann ich im Code nachvollziehen, aber das hilft mir ja nicht, denn ich muss wissen, wie es richtig ist. Und das sollte in der Spezifikation stehen. Wenn ich im Testmanagement-Tool suche, dann gibt es vielleicht noch andere Dokumente, wo irgendwelche Informationen drin stehen. Dann kriege ich das Hinterher nicht wieder zusammen. Wichtig ist, dass man eine Stelle hat, wo die Wahrheit steht. Und die sollte nicht der Code sein.“ (I16 – 14:24)

Risikofaktoren für Aktualität von Anforderungen	
<i>Risikofaktoren wenn Artefakttyp stets aktuell gehalten wird</i>	
R-5.1	Hohe Anpassungskosten -> Entwickler vermeiden Dokumentation von Anforderungen
<p>„Es wurden sehr viele Konzepte umgeworfen. Da hat man sich dann nicht mehr die Mühe gemacht, die Aktivitätsdiagramme weiter zu pflegen. Weil sich das zu der Zeit entfernt hatte. Man hat gesagt, wir machen jetzt hier einen cut und beschreiben neu, wie es eigentlich gedacht war.“ (I13 – 10:24)</p> <p>„[Interviewer: Es wäre möglich, weitere Informationen oder Verweise, die zu einem Thema relevant sind, in dem Moment wenn man sie selbst gefunden hat, im Dokument nachzupflegen. Würden Sie das tun?] Nein. Ich habe ja ein Problem, das ich lösen will. Da sitzen mir vielleicht 10 Leute im Nacken. Da fange ich nicht an, Dokumente zu pflegen.“ (I16 – 52:23)</p> <p>„Das Problem ist auch: wir haben E-Mails als Meeting-Minutes, Powerpoints als Requirements, dann die einzelnen IT-Tasks [im RE-Tool], dann das Design-Dokument. Das muss alles gepflegt werden. Aber das bleibt oftmals durch die Zeit auf der Strecke.“ (I13 – 35:26)</p>	
R-5.2	Unterbrechung von Aktivitäten, die Erstellung und Änderung von Artefakten beinhalten
<p>„[Interviewer: Warum werden Änderungen manchmal nicht dokumentiert?] Das ist aufgekommen, nachdem die Anwendung in Produktion gegangen ist. Bestes Beispiel: man hat festgestellt, es sind tausend Rechnungen [an Kunden] falsch. Wenn man jetzt losgeht, erstmal einen Change Request erstellt, alles schön aufschreibt, dann die Spezifikation anpasst und es dann irgendwann umsetzt... Dann ist vielleicht schon eine Woche vergangen, aber die Rechnungen sind alle falsch und es sind falsche Daten rausgegangen. So eine Änderung soll sofort rein. Und dann wird dem Entwickler gesagt „so ist es richtig, und die Spezifikation ändere ich dann später“. Und wenn es dann einmal umgesetzt ist, kümmert es auch keinen mehr. So rutschen die Sachen rein, weil es schnell gehen muss.“ (I16 – 16:16)</p> <p>„[Interviewer: Es wäre möglich, weitere Informationen oder Verweise, die zu einem Thema relevant sind, in dem Moment wenn man sie selbst gefunden hat, im Dokument nachzupflegen. Würden Sie das tun?] Nein. Ich habe ja ein Problem, das ich lösen will. Da sitzen mir vielleicht 10 Leute im Nacken. Da fange ich nicht an, Dokumente zu pflegen.“ (I16 – 52:23)</p>	
<i>Risikofaktoren wenn Artefakttyp vernachlässigt wird (Lazy Updates)</i>	
R-5.3	Hohe Kosten für Identifikation und Anpassung veralteter Artefakte -> Projektteilnehmer verzichten ganz auf die Verwendung des Artefakttyps
<p>„Haben zuerst ein Aktivitätsdiagramm verwendet. Davon wurde dann irgendwann abgegangen. Weiß nicht genau, woran das gelegen hat. [...] Ich denke, da sind dann Realität und Anforderungen auseinander gelaufen. Und das Dokument wurde dann nicht mehr so richtig verfolgt. [...] Es wurden sehr viele Konzepte um-</p>	

geworfen. Da hat man sich dann nicht mehr die Mühe gemacht, die Aktivitätsdiagramme weiter zu pflegen. Weil sich das zu der Zeit entfernt hatte. Man hat gesagt, wir machen jetzt hier einen cut und beschreiben neu, wie es eigentlich gedacht war.“ (I13 – 3:40 & 10:24)

„Ich habe mir [die Spezifikation] einmal durchgelesen. Dann gab es größere Änderungen, dann habe ich nicht mehr reingeguckt. Es ist sehr viel Aufwand und sehr viel Text und wenn man nicht weiß, wo sich etwas geändert, hat... Wenn man weiß, da sind ein Paar Sachen drin, die nicht aktuell sind oder mit dem Kunden nicht nochmal abgestimmt sind, das ist problematisch.“ (I6 – 24:44)

R-5.4	Unterbrechung von Aktivitäten, die Identifikation von Informationen beinhalten
-------	--

„[In welchen Situationen ist es wichtig, Artefakte nachzupflegen?] Weil Fehler aufgetreten sind, z.B. ist der Rechnungsbetrag am Ende falsch. [...] Man fragt sich, wie ist der Rechenweg dahin? Den Rechenweg kann ich im Code nachvollziehen, aber das hilft mir ja nicht, denn ich muss wissen, wie es richtig ist. Und das sollte in der Spezifikation stehen. Wenn ich im Testmanagement-Tool suche, dann gibt es vielleicht noch andere Dokumente, wo irgendwelche Informationen drinstehen. Dann kriege ich das Hinterher nicht wieder zusammen. Wichtig ist, dass man eine Stelle hat, wo die Wahrheit steht. Und die sollte nicht der Code sein.“ (I16 – 14:24)

„Es gab viele [externe] Änderungen, [von denen manche nur] in die Spezifikation eingepflegt wurden. Wir wussten aber nicht genau, wo. Es gibt Unterschiede zwischen der Spezifikation und dem Product Backlog [...]. In der Excel-Datei, wo der [fachliche] Workflow beschrieben wurde, standen auch einige Anforderungen drin. Auch das wurde nicht aktuell gehalten. In beiden Dokumenten waren ein Paar Anforderungen versteckt. Das macht das ganze auch schwieriger.“ (I6 – 13:32)

Anhang B – Unterlagen zur Experiment-Studie

Im Folgenden sind die Fragen angegeben, welche die Experimentteilnehmer bearbeitet haben sowie die Story Map mit den dazu vorgegebenen Anforderungen.

Block	ID	Aufgabe
A	1	Welche ID hat die Story Folienmaster erstellen ?
	2	Wie wird dem Presenter signalisiert, dass er die letzte Folie erreicht hat?
	3	Welche Möglichkeiten habe ich, eine neue Ansicht zu erstellen ? Bitte schreibe nur die IDs der entsprechenden Anforderungselemente auf. (z.B. S12 für eine Story, UC3-S2 für einen UseCase-Schritt)
	4	Markiere die Story Präsentation ab beliebiger Folie beginnen als <i>erledigt</i> .
	5	Verändere die Priorisierung der Story Eine Folie in der Folienübersicht selektieren, sie dann im Bearbeitungsbereich angezeigt bekommen zu Prio 3.
	6	Verändere die Story Die Präsentation wird über den Button gespeichert zu Die Präsentation wird nach jeder Aktion automatisch gespeichert .
B	7	Finde alle 5 Artefakte (Story Cards oder Use Case-Schritte), die Informationen zur 1-Folien-Sicht und zur 3-Folien-Sicht enthalten.
	8	Der Kunde möchte gern die Funktionalität, die in Step UC4-S3 beschrieben ist, ändern: Statt copy&pasten der Inhalte soll der Presenter die Folie in der Folienübersicht per drag&drop von der Mitte nach links verschieben können. Dokumentiere diese Anforderung.
	9	Der Kunde merkt außerdem an, dass beim Verschieben einer Folie auf eine andere Präsentationsfläche (aus voriger Aufgabe) die Animationen der Art „bewegen“ gelöscht werden sollen. Dokumentiere diese Anforderung.
	10	Du unterhältst dich mit dem Kunden über Story 7 Neue Ansicht aus nur einer Folie erzeugen . Der Kunde möchte gern, dass man die Funktion „nächste Folie“ umbenennt in „Folie ableiten“.
	11	Du reviewst mit dem Kunden Use Case UC6 Präsentation durchführen . Er möchte gern, dass man als letzten Schritt die Präsentation mit der Geste „beide Hände von oben nach unten führen“ komplett beenden kann. Dann sollen alle Präsentationsflächen schwarz werden. Alternativ wählt man dafür

		Strg+ESC. Dokumentiere diese Anforderung.
	12	Lösche die Story Elemente highlighten (das war eine geplante Animation).
C	13	Du möchtest gern den Use Case UC 3 Animationen einfügen mit dem Kunden an der bisherigen Software durchgehen. Welche Stories sind für den Use Case relevant? (Beachte, dass das fertige sowie unfertige Stories sein können.)
	14	Welche der Schritte in UC 3 Animationen einfügen sind schon durchführbar?
	15	Der Kunde will UC 5 Hineinzoomen und Herausziehen verwenden möglichst bald durchführen können. Priorisiere die dafür notwendigen Stories mit Prio 1.
	16	In welchen Situationen kann es sinnvoll sein, eine Präsentation ab einer beliebigen Folie beginnen zu können? (Nenne den/die Use Case-Schritt(e) aus dem/denen du das herausgelesen hast.)
D	17	Nicht alle Schritte aus Use Case Präsentation erstellen sind schon in den Stories enthalten. Finde drei fehlende Teile und schreibe die passenden Stories dazu.
	18	Es ist ein Widerspruch zwischen Use Case UC6 Eine Präsentation durchführen und seinen Stories enthalten. Nenne den UC-Schritt und die Story ID, die inkonsistent sind.

Anhang B – Unterlagen zur Experiment-Studie

<p>UT5: Eine 3-Lenwand-Präsentation durchführen</p>	<p>S2: Beim Betätigen der „Weiter“-Taste wird die nächste Animation durchgeführt bzw. (wenn es keine mehr gibt) die nächste Ansicht gestartet.</p> <p>10 - ⚙️</p>	<p>S6: Der Presenter verwendet zur Präsentation den Rechner des Infolabs. Er kann sich über den Rechner in der Webapp einloggen und dort seine Präsentation öffnen.</p> <p>10 - ⚙️</p>	<p>S29: Der Presenter sieht während der Präsentation einen Vorschau-Modus auf dem Bildschirm</p> <p>10 - ⚙️</p>	<p>S51: Zum Weiterschalten (Folien bzw. Animationen) verwendet man Pfeiltasten. Zum Überspringen der Animationen verwendet man Strg+Pfeiltasten.</p> <p>10 - ⚙️</p>	<p>S37: Wenn die Präsentation fertig ist, kann ich nach der letzten Folie nicht weiter schalten (erst beim Beenden soll die Präsentationsfläche schwarz werden)</p> <p>10 - ⚙️</p>
<p>UT6: Die Präsentation per Gessen steuern</p>	<p>S16: Mit der normalen Wischgeste kann ich einen Schritt nach vorn (bzw. hinten) gehen</p> <p>10 - ⚙️</p>	<p>S17: Mit der „großen Wischgeste“ kann ich direkt zum nächsten „Abschnittswechsel“ springen.</p> <p>10 - ⚙️</p>	<p>S18: Wenn eine Ansicht enthält, springt die Präsentation bei der großen Wischgeste zur letzten Folie.</p> <p>10 - ⚙️</p>	<p>S60: Die Präsentation wird mit der Geste „beide Hände von oben nach unten führen“ komplett beendet.</p> <p>10 - ⚙️</p>	<p>Add User Story Add</p>
<p>UT1: Präsentation erstellen</p>	<p>S1: Ich kann in der Bearbeitungsansicht zwischen der "1-Folien-Ansicht" und der "3-Folien-Ansicht" wechseln.</p> <p>10 - ⚙️</p>	<p>S6: In der Folienübersicht sehe ich statt einer Folie drei nebeneinander</p> <p>10 - ⚙️</p>	<p>S10: Der Bearbeitungsbereich zeigt die selektierte Folie an und am Rand keine Teile der benachbarten Folien (so ähnlich wie die verschiedenen "Desktops" auf Handys und Tablets)</p> <p>10 - ⚙️</p>	<p>S39: Beim Anlegen einer neuen Ansicht wird im 1-Folien-Bearbeitungsmodus automatisch die mittlere Folie gezeigt</p> <p>10 - ⚙️</p>	<p>S33: Ich kann zwischen den Folienansichten "Titelfolie" und "Abschnittswechsel" unterscheiden. Jede Folie besitzt ein spezielles Textfeld "Titel".</p> <p>10 - ⚙️</p>
<p>UT2: Folien erzeugen</p>	<p>S49: Der Presenter dupliziert die Ansicht Unter der aktuellen Ansicht erscheint dieselbe Ansicht nochmal.</p> <p>10 - ⚙️</p>	<p>S38: Aus einer Folie herauszoomen, dabei werden die 3 Folien auf die mittlere Folie ausgetauscht</p> <p>1 - ⚙️</p>	<p>S7: Ich kann unter einer Folie einen von drei Buttons (Pfeil runter, "Neue Folie", "Folie ableiten") anklicken. Dann wird die Ansicht (mit den drei Präsentationsflächen) übernommen. Die Folie unter die ich geklickt habe ist, jetzt aber eine neue Folie</p> <p>10 - ⚙️</p>	<p>Add User Story Add</p>	
<p>UT3: Eine Folie bearbeiten</p>	<p>S21: Formen Gruppieren</p> <p>10 - ⚙️</p>	<p>S22: Gruppen bearbeiten</p> <p>10 - ⚙️</p>	<p>S25: Eine Menge von Formen ausrichten</p> <p>10 - ⚙️</p>	<p>S27: Pfeil an "Klebpunkte" einer Form snappen</p> <p>10 - ⚙️</p>	<p>S31: Ich kann eine Form-Aktion rückgängig machen</p> <p>10 - ⚙️</p>
<p>UT4: Animationen einfügen</p>	<p>S13: Die Animationen beziehen sich immer auf die gesamte aktuelle Foliengruppe (bestehend aus den drei Folien)</p> <p>10 - ⚙️</p>	<p>S14: Ich kann eine Animation auf einer und danach auf einer anderen Präsentationsfläche durchführen</p> <p>10 - ⚙️</p>	<p>S59: Aktion zu einer Animation ändern können</p> <p>10 - ⚙️</p>	<p>S3: Ich kann Formen von einer Stelle zu einer anderen "bewegen". Dazu erhält die Form einen speziellen Pfeil, den ich an die Z-position ziehe.</p> <p>10 - ⚙️</p>	<p>S15: In eine Folie "hineinzoomen". Dabei wird eine Folie auf die drei ausgetauscht.</p> <p>1 - ⚙️</p>
<p>S55: Eine Form verschieben</p> <p>10 - ⚙️</p>	<p>S53: Größe einer Form ändern</p> <p>10 - ⚙️</p>	<p>S43: Der Presenter kann Bilder zur Folie hinzufügen</p> <p>10 - ⚙️</p>	<p>S53: Größe einer Form ändern</p> <p>10 - ⚙️</p>	<p>S55: Eine Form verschieben</p> <p>10 - ⚙️</p>	<p>S57: Text in Form einfügen</p> <p>10 - ⚙️</p>
<p>S58: Eine Form anpassen</p>	<p>Add User Story Add</p>	<p>Add User Story Add</p>	<p>Add User Story Add</p>	<p>Add User Story Add</p>	<p>Add User Story Add</p>

1. kann man nicht ändern

kann ich nach der letzten Folie nicht weiter schalten (erst beim Beenden soll die Präsentationsfläche schwarz werden)

10 - ⌂2 ⚙

Add User Story

Add

2. kann man nicht ändern

Textfeld "Titel".

10 - ⌂1 ⚙

S34: Ich kann die Folienart ändern. Dabei wird der Titel einer Folie in das jeweilige Feld geschrieben (oben, in der Mitte, etc.)

10 - ⌂1 ⚙

S32: Das System umwandelt in der Folienübersicht die aktuell bearbeitbare Folie (bzw. Foliengruppe wenn der 3-Folien-Modus gewählt ist)

10 - ⌂1 ⚙

S35: Ich kann eine Folie in der Folienübersicht selektieren. Diese wird mir dann im Bearbeitungsbereich angezeigt

3 - ⌂1 ⚙

S30: Das System speichert die Präsentation automatisch nach jeder Aktion

10 - ⌂1 ⚙

S31: Ich kann eine neue Präsentation erstellen

10 - ⌂1 ⚙

Add User Story

Add

Add

10 - ⌂1 ⚙

S57: Text in Form einfügen

10 - ⌂1 ⚙

S58: Eine Form anpassen

1 - ⌂6 ⚙

S48: Eine Form löschen

1 - ⌂5 ⚙

S63: Das System zeigt kopierte Folien in der Folienübersicht: graulich an, solange sie exakt mit der vorhergehenden Folie übereinstimmen.

10 - ⌂1 ⚙

S64: Form einfügen

10 - ⌂1 ⚙

Add User Story

Add

Anhang B – Unterlagen zur Experiment-Studie

Alle Aufgaben

Teilnehmer	<= mt Treatment (Linkadressen)														ohne Treatment (NoLinks) ->															
	BPf-y24	BPf-y33	BPf-y23	BPf-y21	BPf-y26	BPf-y29	BPf-y30	BPf-y22	BPf-y44	BPf-y49	BPf-y41	BPf-y50	BPf-y03	BPf-y46	BPf-y47	BPf-y53	BPf-y55	BPf-y61	BPf-y54	BPf-y59	BPf-y65	BPf-y58	BPf-y61	BPf-y58	BPf-y61	BPf-y58	BPf-y61	BPf-y61		
Gesamtpunktzahl	62	59	52	42	35	53	45	53	49	49	50	51	45	45	39	49	35	35	43	34	46	50	34	37	35	42	49	46	59	
erreichbare Punkte	0,75	0,61	0,84	0,68	0,58	0,85	0,73	0,84	0,65	0,73	0,77	0,81	0,84	0,73	0,47	0,65	0,58	0,68	0,58	0,77	0,81	0,35	0,6	0,56	0,88	0,65	0,94	0,62		
Anzahl	113	147	154	92	164	198	91	152	116	101	146	102	202	142	188	143	159	144	145	142	212	230	118	128	162	146	153	169	211	
Avg Dauer / AuÖg (in sek)	34	44	46	28	49	60	27	46	35	30	44	30	61	43	57	43	48	43	43	43	64	69	36	39	49	44	46	51	63	
Summe in min	2,15	1,39	1,83	2,43	1,18	1,42	2,7	1,83	1,86	2,63	1,75	2,7	1,36	1,7	0,82	1,51	1,17	1,35	1,38	1,31	1,2	1,17	1,33	1,54	1,14	1,55	1,41	1,45	1,08	
Performance																														
Empfohlene Schwierigkeit																														
	1,83	2,28	2,56	2,06	3,17	2,89	2	2,33	2,59	1,67	2,83	2,11	2,33	1,83	2,83	2,5	3,06	2,56	2,17	1,94	2	1,89	2	2,72	2	2,72	2	2,72	2	

Simple Aufgaben

Teilnehmer	<= mt Treatment (Linkadressen)														ohne Treatment (NoLinks) ->															
	BPf-y24	BPf-y23	BPf-y21	BPf-y26	BPf-y29	BPf-y30	BPf-y44	BPf-y49	BPf-y41	BPf-y50	BPf-y03	BPf-y46	BPf-y47	BPf-y53	BPf-y55	BPf-y61	BPf-y54	BPf-y59	BPf-y65	BPf-y58	BPf-y61	BPf-y58	BPf-y61	BPf-y58	BPf-y61	BPf-y61	BPf-y61	BPf-y61		
Gesamtpunktzahl	29	16	13	21	13	19	20	14	19	16	18	22	20	15	13	17	17	12	15	14	17	21	18	32	14	20	15	21	20	
erreichbare Punkte	0,35	0,45	0,72	0,45	0,66	0,69	0,48	0,66	0,59	0,62	0,78	0,76	0,69	0,52	0,46	0,59	0,59	0,41	0,52	0,48	0,59	0,73	0,62	0,41	0,48	0,69	0,52	0,72	0,69	
Anzahl	86	127	136	63	161	181	90	118	110	92	128	105	157	123	182	115	122	91	97	90	100	177	111	114	96	115	104	151	167	
Avg Dauer / AuÖg (in sek)	16	23	25	12	30	33	17	22	20	17	23	19	29	23	31	21	22	17	18	16	18	33	20	21	18	21	19	28	31	
Summe in sek	3,41	1,96	2,88	3,75	2,2	2,09	2,82	3	2,75	3,65	3,3	4	2,38	2,26	1,36	2,81	2,68	2,41	2,89	3	3,28	2,25	3,1	1,95	2,67	3,29	2,74	2,57	2,23	
Performance																														
Empfohlene Schwierigkeit																														
	1,64	2,27	2,73	2	3,09	2,55	1,82	2,45	2,3	1,55	2,91	2,09	2,09	1,36	2,18	1,73	2,45	2,86	2	1,36	1,64	1,36	2	2,86	2,57	2,71	2	2,86	2	

Komplexe Aufgaben

Teilnehmer	<= mt Treatment (Linkadressen)														ohne Treatment (NoLinks) ->															
	BPf-y24	BPf-y33	BPf-y23	BPf-y21	BPf-y26	BPf-y29	BPf-y30	BPf-y44	BPf-y49	BPf-y41	BPf-y50	BPf-y03	BPf-y46	BPf-y47	BPf-y53	BPf-y55	BPf-y61	BPf-y54	BPf-y59	BPf-y65	BPf-y58	BPf-y61	BPf-y58	BPf-y61	BPf-y58	BPf-y61	BPf-y61	BPf-y61		
Gesamtpunktzahl	33	29	25	31	29	17	33	31	33	24	31	26	28	33	30	16	23	18	24	27	20	31	29	16	25	21	22	25	25	22
erreichbare Punkte	0,68	0,76	0,94	0,88	0,52	1	0,94	1	0,73	0,94	0,79	0,65	0,97	0,91	0,48	0,71	0,55	0,73	0,82	0,61	0,94	0,68	0,48	0,76	0,64	0,67	0,76	0,76	0,67	
Anzahl	155	179	181	136	169	255	93	205	125	117	174	96	274	173	199	187	218	227	220	223	388	312	129	152	266	194	280	198	281	
Avg Dauer / AuÖg (in sek)	18	21	21	16	20	26	11	24	15	14	20	23	20	23	20	22	25	27	26	26	45	36	45	36	45	36	23	27	23	33
Summe in sek	4,69	3,62	4,48	5,5	2,6	3,85	8,55	4,17	4,87	6,71	3,95	7,73	3,03	4,55	2,69	3,18	2,2	2,7	3,15	2,95	2,09	2,44	3,2	4,22	2,06	2,91	2,81	3,3	2,05	
Performance																														
Empfohlene Schwierigkeit																														
	2,14	2,29	2,29	2,14	3,29	3,43	2,29	2,14	3	1,86	2,71	2,14	2,71	2,57	3,86	3,71	4	2,86	2,43	2,86	2,57	2,71	4	2,86	2,43	2,86	2,57	2,71	2	3,29

Anhang C – Mathematische Erläuterung zur Addition von Risk Exposure-Werten

In Abschnitt 7.5.2 wird angegeben, wie sich die Risk Exposure verhält, wenn mehrere Risiken bestehen. Dabei wird die Risk Exposure mit dem Erwartungswert des eintretenden Schadens verglichen. Jedoch sind Erwartungswerte nur auf Zufallsvariablen, aber nicht auf Ereignissen definiert. Zur Ergänzung der dortigen anschaulichen Ausführungen wird daher hier noch einmal mathematisch korrekt mit Zufallsvariablen argumentiert.

Zufallsvariable eines Risikos

Eine Zufallsvariable X ist ein mathematisches Konstrukt, welches mit einer gewissen Wahrscheinlichkeit einen gewissen Wert annimmt. Dieses Verhalten wird durch die Verteilung repräsentiert. Die Verteilung ist eine Funktion P , die zu Zahlen x aus dem Ergebnisbereich von X eine Wahrscheinlichkeit $P(X=x)$ zuordnet.

Beispiel. Sei X die Zufallsvariable, die das Ergebnis eines Münzwurfes repräsentiert. Der Ergebnisbereich von X ist dann $\{0,1\}$ (z.B. steht 1 für Kopf und 0 für Zahl).

Die Verteilung von X ist $P(X=1) = 1/2$, $P(X=0) = 1/2$.

Zu einem Risiko mit Ereignis r_1 kann man nun zwei Zufallsvariablen M_1 und S_1 definieren.

Die Zufallsvariable M_1 soll das *Eintreten des Ereignisses* r_1 repräsentieren. Beim Eintreten des Ereignisses soll $M_1=1$ sein, sonst $M_1=0$.

Beispiel. Betrachten wir das Risiko mit dem Ereignis r_1 ="Die Analysephase verlängert sich". Sei die Wahrscheinlichkeit des Ereignisses $P(r_1) = 1/2$ und sein Schaden $S(r_1) = 40$.

Die Zufallsvariable M_1 , die das Eintreten von r_1 darstellt, nimmt den Wert 1 mit einer Wahrscheinlichkeit von $1/2$ an. Sie nimmt den Wert 0 ebenfalls mit einer Wahrscheinlichkeit von $1/2$ an.

Wir können M_1 mit einem Münzwurf (mit fairer Münze) vergleichen.

Nun hilft uns diese Zufallsvariable noch nicht, um zu bestimmen, wie sich der Schaden eines Risikos verhalten wird. Dazu konstruieren wir eine neue Zufallsvariable $S_1 := 40 \cdot M_1$ (siehe [81] S. 8).

40 ist dabei genau der Schaden $S(r_1)$ aus obigem Beispiel.

Die Konstruktion S_1 ist nun eine *neue* Zufallsvariable, bei der man M_1 als *zufällige Eingabe* auffasst. Man kann sich das so vorstellen, dass sich zuerst M_1 mit den bereits gegebenen Wahrscheinlichkeiten realisiert. S_1 nimmt dann den entsprechenden errechneten Wert an.

Beispiel. In unserem Beispiel wird mit einer Wahrscheinlichkeit von $1/2$ $M_1=1$ und damit $S_1=40$ werden. Mit einer Wahrscheinlichkeit von $1/2$ wird $M_1=0$ und $S_1=0$ werden.

S_1 ist vergleichbar mit folgendem Spiel: Werfe eine Münze. Wenn diese 1 zeigt, so zahle 40 Euro Strafe. Wenn diese 0 zeigt, so zahle 0 Euro Strafe.

In Bezug auf das Risiko ist S_1 also die Zufallsvariable, die den eintretenden Schaden darstellt. Mithilfe dieser Zufallsvariable können wir nun einen Erwartungswert bestimmen, der zeigt, welchen erwarteten Schaden das Risiko besitzt.

Der Erwartungswert einer Zufallsvariable X ist definiert durch

$$E(X) = \sum_{x \text{ aus Ergebnisbereich von } X} P(X = x) \cdot x$$

Beispiel. Für den erwarteten Schaden des Risikos ist

$$E(S_1) = P(S_1 = 40) \cdot 40 + P(S_1 = 0) \cdot 0 = \frac{1}{2} \cdot 40 = 20.$$

Dies entspricht genau der Risk Exposure $RE(r_1)$, bei der $P(r_1) = P(S_1=40)$ und $S(r_1) = 40$ war.

Zusammensetzen mehrerer Risiken zu einer Alternative

Nehmen wir nun an, dass es zwei Risiken gibt.

Beispiel. Das Risiko zu r_1 wurde oben bereits definiert. Wir haben es durch einen Münzwurf dargestellt, deren Münze wir nun als rote Münze bezeichnen.

Es kommt ein zweites Risiko hinzu, für das gilt:
 r_2 tritt mit einer Wahrscheinlichkeit von $1/6$ ein und erzeugt einen Schaden von 60.

Dieses Risiko repräsentieren wir durch den Wurf mit einer schwarzen gezinkten Münze (dargestellt durch die Zufallsvariable M_2).

Dann ist $M_2=1$ mit einer Wahrscheinlichkeit von $1/6$ und $M_2=0$ mit einer Wahrscheinlichkeit von $5/6$.

Die Zufallsvariable für den Schaden berechnen wir durch $S_2 = 60 \cdot M_2$.

Wenn wir nun eine Alternative A betrachten, bei der beide Risiken auftreten können, so benötigen wir wieder eine neue Zufallsvariable S_A , die den Schaden der gesamten Alternative repräsentiert. Unter der Annahme, dass sich die Schäden addieren, können wir diese Zufallsvariable aus den bisherigen Variablen konstruieren (siehe [81] S. 8):

$$S_A = S_1 + S_2 \text{ bzw. anschaulicher } S_A = 40 \cdot M_1 + 60 \cdot M_2.$$

Wieder ist die Konstruktion so zu verstehen, dass zuerst M_1 und M_2 zufällig realisiert werden und sich dann S_A je nach dem Ausgang von M_1 und M_2 berechnet. Das entspricht folgendem Spiel:

Werfen der roten und der schwarzen Münze. Dann Zahlen der jeweiligen Strafe für die rote und die schwarze Münze. Zeigen die rote und die schwarze Münze jeweils eine 1, so muss man 100 Euro Strafe zahlen. Zeigt nur die rote Münze eine 1 und die schwarze Münze eine 0, so muss man 40 Euro Strafe zahlen usw.

Beispiel. Zur genaueren Anschauung wird hier noch einmal das gesamte Verhalten der Zufallsvariable S_A dargestellt, die sich als Konstruktion aus der zusammengesetzten Variable (M_1, M_2) ergibt. Mit Kleinbuchstaben werden hier die Realisierungen der Zufallsvariablen bezeichnet. Die beiden rechten Spalten zeigen die Verteilung von S_A .

Mögliche Werte (m_1, m_2) der zusammengesetzten Zufallsvariable (M_1, M_2)	Wahrscheinlichkeit, dass (M_1, M_2) den Wert (m_1, m_2) annimmt. Das ist $P(M_1=m_1, M_2=m_2)$. Bei Unabhängigkeit von M_1 und M_2 , dann auch $P(M_1=m_1) \cdot P(M_2=m_2)$	Entsprechende Werte s der konstruierten Zufallsvariable S_A . Es gilt $s = 40 \cdot m_1 + 60 \cdot m_2$
(1, 1)	$1/2 * 1/6 = 1/12$	100
(1, 0)	$1/2 * 5/6 = 5/12$	40
(0, 1)	$1/2 * 1/6 = 1/12$	60
(0, 0)	$1/2 * 5/6 = 5/12$	0

Risk Exposure als Erwarteter Schaden einer Alternative

Wir nehmen an, dass die beiden Risiko-Ereignisse r_1 und r_2 – und damit auch die Zufallsvariablen M_1 und M_2 – unabhängig sind. Das entspricht dem unabhängigen Münzwurf mit der roten und der schwarzen Münze.

Der Erwartungswert des Schadens aus der Alternative A ist

$$E(S_A) = E(40 \cdot M_1 + 60 \cdot M_2)$$

Den Erwartungswert könnte man nun wieder mittels der Definition bestimmen

$$E(S_A) = \sum_{s \in \{100, 40, 60, 0\}} P(S_A = s) \cdot s$$

Wie in der obigen Tabelle zu sehen, können wir dabei $P(S_A=s)$ mithilfe von Satz (5.3) aus [81] (S. 22) durch

$$P(S_A = s) = \sum_{m_1+m_2=s} P(M_1 = m_1, M_2 = m_2)$$

berechnen.

Beispiel.

$$\begin{aligned} E(S_A) &= P(M_1 = 1, M_2 = 1) \cdot 100 + P(M_1 = 1, M_2 = 0) \cdot 40 \\ &\quad + P(M_1 = 0, M_2 = 1) \cdot 60 + P(M_1 = 0, M_2 = 0) \cdot 0 \\ &= 1/12 \cdot 100 + 5/12 \cdot 40 + 1/12 \cdot 60 + 0 \\ &= 30 \end{aligned}$$

Dies ist analog zur Berechnung aus Abschnitt 7.5.2.

Unter der Annahme der Unabhängigkeit der beiden Zufallsvariablen M_1 und M_2 können wir aber auch die Linearität des Erwartungswertes [81] direkt ausnutzen, um $E(S_A)$ zu bestimmen. Dazu stellen wir $40 \cdot M_1$ und $60 \cdot M_2$ wieder als S_1 und S_2 dar.

$$\begin{aligned} E(S_A) &= E(40 \cdot M_1 + 60 \cdot M_2) = E(40 \cdot M_1) + E(60 \cdot M_2) \\ &= E(S_1) + E(S_2) \end{aligned}$$

Nun soll die Risk Exposure der Alternative A genau den erwarteten Schaden der Alternative darstellen. Damit ergibt sich:

$$RE(A) := E(S_A) = E(S_1) + E(S_2) = RE(r_1) + RE(r_2).$$

Literaturverzeichnis

- [1] U. Abelein and B. Paech, "A proposal for enhancing user-developer communication in large IT projects," in *Proceedings of the 5th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2012, pp. 1–3.
- [2] J. Adersberger, *Modellbasierte Extraktion, Repräsentation und Analyse von Traceability-Informationen. Dissertation*. Herbert Utz Verlag, 2013.
- [3] R. S. Aguilar-Savén, "Business process modelling: Review and framework," *Int. J. Prod. Econ.*, vol. 90, no. 2, pp. 129–149, 2004.
- [4] A. Alderson, "Meta-CASE Technology," in *Proceedings of Software Development Environments and CASE Technology: European Symposium Königswinter, June 17--19*, A. Endres and H. Weber, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 81–91.
- [5] S. Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. New York, NY, USA: John Wiley & Sons, Inc., 2002.
- [6] P. O. Antonino, T. Keuler, N. Germann, and B. Cronauer, "A Non-invasive Approach to Trace Architecture Design, Requirements Specification and Agile Artifacts," in *Proceedings of the 23rd Australian Software Engineering Conference (ASWEC)*, 2014, pp. 220–229.
- [7] Atlassian, "JIRA," 2016. [Online]. Available: <https://www.atlassian.com/software/jira>. [Accessed: 25-May-2016].
- [8] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, 2nd Editio. Addison Wesley, 2004.
- [9] J. Bergsmann, *Requirements Engineering für die Agile Softwareentwicklung*. Heidelberg: dpunkt.Verlag, 2014.
- [10] J. Bock, "Konzept zur Unterstützung der parallelen Erstellung und Verwendung von GUI-Mockups und Story Cards (Bachelorarbeit)," Leibniz Universität Hannover, Fachgebiet Software Engineering, 2015.
- [11] B. Boehm, "Get ready for agile methods, with care," *IEEE Comput.*, vol. 35, no. 1, pp. 64–69, Jan. 2002.

- [12] B. Boehm, "Value-based Software Engineering," *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 2, pp. 1–12, 2003.
- [13] B. Boehm, "Requirements that handle IKIWISI, COTS, and rapid change," *IEEE Comput.*, vol. 33, no. 7, pp. 99–102, Jul. 2000.
- [14] B. Boehm, "The Spiral Model as a Tool for Evolutionary Acquisition," *CrossTalk J. Def. Softw. Eng.*, vol. 14, no. 5, p. 4, 2001.
- [15] B. Boehm, "Software risk management: principles and practices," *IEEE Softw.*, vol. 8, no. 1, pp. 32–41, Jan. 1991.
- [16] B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*. Pearson Education, 2003.
- [17] B. Boehm and R. Turner, "Using Risk to Balance Agile and Plan-Driven Methods," *IEEE Comput.*, vol. 36, no. 6, pp. 57–66, 2003.
- [18] B. Boehm and R. Turner, "Management challenges to implementing agile processes in traditional development organizations," *IEEE Softw.*, vol. 22, no. 5, pp. 30–39, Sep. 2005.
- [19] J. Brandstätter, "Risikomanagement in der Softwareentwicklung," in *Agile IT-Projekte erfolgreich gestalten: Risikomanagement als Ergänzung zu Scrum*, Wiesbaden: Springer Fachmedien Wiesbaden, 2013, pp. 21–32.
- [20] C. Braun, F. Wortmann, M. Hafner, and R. Winter, "Method Construction - a Core Approach to Organizational Engineering," in *Proceedings of the 2005 ACM Symposium on Applied Computing*, 2005, pp. 1295–1299.
- [21] S. Buehne, G. Halmans, K. Pohl, M. Weber, H. Kleinwechter, and T. Wierczoch, "Defining requirements at different levels of abstraction," in *Proceedings on 12th IEEE International Requirements Engineering Conference*, 2004, pp. 346–347.
- [22] J. Cabot and M. Gogolla, "Object Constraint Language (OCL): a Definitive Guide," in *Advanced Lectures on Formal Methods for Model-Driven Engineering: 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM 2012)*, M. Bernardo, V. Cortellessa, and A. Pierantonio, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 58–90.
- [23] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. Natt och Dag, "An industrial survey of requirements interdependencies in software product release planning," in *Proceedings of the Fifth IEEE*

- International Symposium on Requirements Engineering*, 2001, pp. 84–91.
- [24] E. Ben Charrada, *Supporting Requirements Update During Software Evolution. Dissertation*. University of Zurich, Faculty of Economics, 2013.
- [25] E. Ben Charrada, A. Koziolok, and M. Glinz, “Supporting requirements update during software evolution,” *J. Softw. Evol. Process*, vol. 27, no. 3, pp. 166–194, 2015.
- [26] J. Cleland-Huang, G. Zemont, and W. Lukasik, “A heterogeneous solution for improving the return on investment of requirements traceability,” in *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE)*, 2004, pp. 230–239.
- [27] A. Cockburn, *Agile Software Development*. Addison Wesley, 2002.
- [28] A. Cockburn, “Selecting a Project’s Methodology,” *IEEE Softw.*, vol. 17, no. 4, pp. 64–71, 2000.
- [29] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley, 1999.
- [30] M. Cohn, *User Stories Applied: For Agile Software Development*. Prentice Hall, 2004.
- [31] F. F. Correia, H. S. Ferreira, N. Flores, and A. Aguiar, “Incremental Knowledge Acquisition in Software Development Using a Weakly-typed Wiki,” in *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, 2009, pp. 31:1–31:2.
- [32] F. F. Correia, “Supporting the Evolution of Software Knowledge with Adaptive Software Artifacts,” in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, 2010, pp. 231–232.
- [33] O. Creighton, M. Ott, and B. Bruegge, “Software Cinema-Video-based Requirements Engineering,” in *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE)*, 2006, pp. 109–118.
- [34] H. Dahlberg, F. S. Ruiz, and C. M. Olsson, “The Role of Extreme Programming in a Plan-Driven Organization,” in *The Transfer and Diffusion of Information Technology for Organizational Resilience: IFIP TC8 WG 8.6 International Working Conference, June 7--10, 2006, Galway, Ireland*, B. Donnellan, T. J. Larsen, L. Levine, and J. I. DeGross, Eds. Boston, MA: Springer US, 2006, pp. 291–312.

- [35] Å. G. Dahlstedt and A. Persson, "Engineering and Managing Software Requirements," in *Requirements Interdependencies: State of the Art and Future Challenges*, A. Aurum and C. Wohlin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 95–116.
- [36] A. Delater, "Tracing requirements and source code during software development. Dissertation.," Ruprecht-Karls-Universität Heidelberg, 2013.
- [37] B. Demuth and C. Wilke, "Model and object verification by using Dresden OCL," in *Proceedings of the Russian-German Workshop Innovation Information Technologies: Theory and Practice, Ufa, Russia*, 2009, pp. 687–690.
- [38] E. Derby, D. Larsen, and K. Schwaber, *Agile retrospectives: Making good teams great*. Pragmatic Bookshelf Raleigh, NC, 2006.
- [39] E. W. Dijkstra, *A Discipline of Programming*, vol. 1. Prentice-Hall Englewood Cliffs, 1976.
- [40] C. Dirkes, "Konzept zur teilautomatisierten Erzeugung von GUI-Mockups aus User Stories (Bachelorarbeit)," Leibniz Universität Hannover, Fachgebiet Software Engineering, 2015.
- [41] R. Dömges and K. Pohl, "Adapting Traceability Environments to Project-specific Needs," *Commun. ACM*, vol. 41, no. 12, pp. 54–62, 1998.
- [42] N. Drivalos, D. S. Kolovos, R. F. Paige, and K. J. Fernandes, "Engineering a DSL for Software Traceability," in *Proceedings of the First International Conference on Software Language Engineering (SLE). Revised Selected Papers*, D. Gašević, R. Lämmel, and E. Van Wyk, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 151–167.
- [43] A. Egyed, "A scenario-driven approach to trace dependency analysis," *IEEE Trans. Softw. Eng.*, vol. 29, no. 2, pp. 116–132, Feb. 2003.
- [44] A. Egyed, S. Biffel, M. Heindl, and P. Grünbacher, "A Value-based Approach for Understanding Cost-benefit Trade-offs During Automated Software Traceability," in *Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering*, 2005, pp. 2–7.
- [45] J. H. Ernicke, "Evaluation und Implementierung einer graphenorientierten Datenbank zur Umsetzung einer Work Product

- Factory (Masterarbeit),” Fachgebiet Software Engineering, Leibniz Universität Hannover, 2016.
- [46] A. Espinoza, P. P. Alarcon, and J. Garbajosa, “Analyzing and Systematizing Current Traceability Schemas,” in *Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop*, 2006, pp. 21–32.
- [47] H. Femmer, J. Mund, and D. Méndez Fernández, “It’s the Activities, Stupid!: A New Perspective on RE Quality,” in *Proceedings of the Second International Workshop on Requirements Engineering and Testing*, 2015, pp. 13–19.
- [48] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke, “Viewpoints: A Framework for Integrating Multiple Perspectives in System Development,” *Int. J. Softw. Eng. Knowl. Eng.*, vol. 02, no. 01, pp. 31–57, 1992.
- [49] A. Finkelstein and I. Sommerville, “The viewpoints FAQ,” *BCS/IEE Softw. Eng. J.*, vol. 11, no. 1, pp. 2–4, 1996.
- [50] G. Fischer, “Domain-Oriented Design Environments,” *Autom. Softw. Eng.*, vol. 1, no. 2, pp. 177–203, 1994.
- [51] M. Fowler and J. Highsmith, “The agile manifesto,” *Softw. Dev.*, vol. 9, no. 8, pp. 28–35, 2001.
- [52] J. Friedrich, U. Hammerschall, M. Kuhrmann, and M. Sihling, “Das V-Modell XT: Für Projektleiter und QS-Verantwortliche,” in *Informatik im Fokus*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–32.
- [53] R. E. Gallardo-Valencia, V. Olivera, and S. E. Sim, “Are Use Cases Beneficial for Developers Using Agile Requirements?,” in *Proceedings of the Fifth International Workshop on Comparative Evaluation in Requirements Engineering (CERE)*, 2007, pp. 11–22.
- [54] P. Ghazi, “A magnet-and-spring based visualization technique for enhancing the manipulation of requirements artifacts,” in *Proceedings of the 23rd IEEE International Requirements Engineering Conference (RE)*, 2015, pp. 400–405.
- [55] P. Ghazi, N. Seyff, and M. Glinz, “FlexiView: A Magnet-Based Approach for Visualizing Requirements Artifacts,” in *Requirements Engineering: Foundation for Software Quality (REFSQ)*, A. S. Fricker and K. Schneider, Eds. Springer International Publishing, 2015, pp. 262–269.

- [56] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine de Gruyter, 1967.
- [57] M. Glinz, "On Non-Functional Requirements," in *Proceedings of the 15th IEEE International Requirements Engineering Conference (RE)*, 2007, pp. 21–26.
- [58] M. Glinz and S. A. Fricker, "On shared understanding in software engineering: an essay," *Comput. Sci. - Res. Dev.*, vol. 30, no. 3, pp. 363–376, 2015.
- [59] O. Gotel and A. Finkelstein, "Contribution Structures," in *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, 1995, pp. 100–107.
- [60] O. Gotel, F. T. Marchese, and S. J. Morris, "On Requirements Visualization," in *Second International Workshop on Requirements Engineering Visualization*, 2007, p. 11.
- [61] O. Gotel and A. Finkelstein, "An analysis of the requirements traceability problem," in *Proceedings of the First International Conference on Requirements Engineering*, 1994, pp. 94–101.
- [62] A. Gross and J. Doerr, "What you need is what you get!: The vision of view-based requirements specifications," in *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE)*, 2012, pp. 171–180.
- [63] P. Grünbacher, S. Köszegi, and S. Biffel, "Stakeholder Value Proposition Elicitation and Reconciliation," in *Value-Based Software Engineering*, S. Biffel, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 133–154.
- [64] L. Hahn, "Verbesserung der agilen Entwicklung von prozessunterstützenden Systemen durch Mapping von Prozessmodellen und User Stories (Bachelorarbeit)," Leibniz Universität Hannover, Fachgebiet Software Engineering, 2015.
- [65] J. H. Hayes, A. Dekhtyar, and J. Osborne, "Improving requirements tracing via information retrieval," in *Proceedings of the 11th IEEE International Requirements Engineering Conference*, 2003, pp. 138–147.
- [66] L. T. Heeager, "Meshing Agile and Documentation-Driven Methods in Practice," Aalborg University, Faculty of Engineering and Science, Information Systems, 2012.

- [67] L. T. Heeager, "Introducing Agile Practices in a Documentation-Driven Software Development Practice: A Case Study," *J. Inf. Technol. Case Appl. Res.*, vol. 14, no. 1, pp. 3–24, 2012.
- [68] M. Heindl and S. Biffli, "A Case Study on Value-based Requirements Tracing," in *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2005, pp. 60–69.
- [69] R. Hoda, J. Noble, and S. Marshall, "Developing a grounded theory to explain the practices of self-organizing Agile teams," *Empir. Softw. Eng.*, vol. 17, no. 6, pp. 609–639, 2012.
- [70] HP, "HP ALM Software," 2016. [Online]. Available: <http://www8.hp.com/us/en/software-solutions/application-lifecycle-management.html>. [Accessed: 25-May-2016].
- [71] IBM Rational, "DOORS," 2016. [Online]. Available: <http://www.ibm.com/software/products/de/ratidoor>. [Accessed: 25-May-2016].
- [72] IEEE, "Systems and Software Engineering -- Vocabulary," *ISO/IEC/IEEE 24765:2010(E)*, pp. 1–418, 2010.
- [73] IEEE, "Systems and Software Engineering -- Life Cycle Processes -- Requirements engineering," *ISO/IEC/IEEE 29148:2011(E)*, pp. 1–94, 2011.
- [74] IEEE, "IEEE Standard for Software Life Cycle Processes - Risk Management," *IEEE Std 1540-2001*, pp. 1–24, 2001.
- [75] M. Imaz and D. Benyon, "How Stories Capture Interaction," in *Proceedings of INTERACT'99*, 1999, p. 321 pp.–328.
- [76] ISO, "Software Engineering -- Metamodel for Development Methodologies," *ISO/IEC 24744:2014*, 2014.
- [77] H. Jaakkola and B. Thalheim, "Visual SQL -- High-Quality ER-Based Query Treatment," in *Workshop Proceedings of Conceptual Modeling for Novel Application Domains*, M. A. Jeusfeld and Ó. Pastor, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 129–139.
- [78] R. Jeffery, F. Utbult, K. Chung, and S. Bruyninx, "The Effect of Constraint Notification within a Case Tool Environment on Design Productivity and Quality," in *Proceedings of the Second International Conference on Product Focused Software Process Improvement (PROFES)*, F. Bomarius and M. Oivo, Eds. Berlin, Heidelberg:

Springer Berlin Heidelberg, 2000, pp. 115–125.

- [79] R. Jeffries, “Essential XP: Card, Conversation, Confirmation,” *30. August 2001*, 2001. [Online]. Available: <http://ronjeffries.com/xprog/articles/expcardconversationconfirmation/>. [Accessed: 25-May-2016].
- [80] Jonathan I. Maletic and M. L. Collard, “TQL: A query language to support traceability,” in *Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, 2009, pp. 16–20.
- [81] G. Kersting and A. Wakolbinger, *Elementare Stochastik*. Basel: Springer Basel, 2010.
- [82] E. Knauss, D. Lübke, and T. Flohr, “Learning to tailor documentation of software requirements,” *J. Univers. Knowl. Manag.*, vol. 1, no. 2, pp. 103–111, 2006.
- [83] E. W. Knauss, *Verbesserung der Dokumentation von Anforderungen auf Basis von Erfahrungen und Heuristiken. Dissertation*. Hannover, Germany: Cuvillier Verlag, 2010.
- [84] A. von Knethen and M. Grund, “QuaTrace: a tool environment for (semi-) automatic impact analysis based on traces,” in *International Conference on Software Maintenances (ICSM)*, 2003, pp. 246–255.
- [85] S. Konrad and H. Degen, “Lessons Learned from the Use of Artifact Models in Industrial Projects,” in *Proceedings of the 17th IEEE International Requirements Engineering Conference (RE)*, 2009, pp. 349–354.
- [86] G. Kotonya and I. Sommerville, “Viewpoints for requirements definition,” *Softw. Eng. J.*, vol. 7, no. 6, pp. 375–387, 1992.
- [87] P. Kruchten, *The Rational Unified Process: an Introduction*. Addison-Wesley Professional, 2004.
- [88] T. Kühne, “Matters of (Meta-) Modeling,” *Softw. Syst. Model.*, vol. 5, no. 4, pp. 369–385, 2006.
- [89] M. Kuhrmann, “Konstruktion modularer Vorgehensmodelle. Dissertation,” Technische Universität München, 2008.
- [90] M. Kuhrmann and S. Beecham, “Artifact-based Software Process Improvement and Management: A Method Proposal,” in *Proceedings of the 2014 International Conference on Software and System Process*, 2014, pp. 119–123.

- [91] M. Kuhrmann and I. Richardson, "How Do Artifact Models Help Direct SPI Projects?," in *Proceedings of the 2015 International Conference on Software and System Process*, 2015, pp. 122–126.
- [92] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, 2001, pp. 249–262.
- [93] C. Larman and B. Vodde, *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*. Pearson Education, 2010.
- [94] D. Leffingwell, *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley Professional, 2010.
- [95] P. Letelier, "A Framework for Requirements Traceability in UML-based Projects," in *In Proc. of 1st Intl. Workshop on Traceability in Emerging Forms of Softw. Eng*, 2002, pp. 32–41.
- [96] A. E. Limón, "An Advanced Traceability schema as a Baseline to Improve Supporting Lifecycle Processes. Dissertation," Universidad Politécnica de Madrid, 2009.
- [97] O. Liskin, "How Artifacts Support and Impede Requirements Communication," in *Requirements Engineering: Foundation for Software Quality (REFSQ)*, A. S. Fricker and K. Schneider, Eds. Springer International Publishing, 2015, pp. 132–147.
- [98] O. Liskin, "Automatisierte Visualisierung von Use Cases durch BPMN-Modelle (Studienarbeit)," Leibniz Universität Hannover, Fachgebiet Software Engineering, 2008.
- [99] O. Liskin and T. Baum, "Integrating GUI Mockups and User Stories (Poster Abstract)," in *Joint Proceedings of REFSQ-2015 Workshops, Research Method Track, and Poster Track*, 2015, pp. 226–227.
- [100] O. Liskin, C. Herrmann, E. Knauss, T. Kurpick, B. Rumpe, and K. Schneider, "Supporting Acceptance Testing in Distributed Software Projects with Integrated Feedback Systems: Experiences and Requirements," in *Proceedings of 7th IEEE International Conference on Global Software Engineering (ICGSE '12)*, 2012, pp. 84–93.
- [101] O. Liskin, R. Pham, S. Kiesling, and K. Schneider, "Why We Need a Granularity Concept for User Stories," in *Agile Processes in Software Engineering and Extreme Programming (XP)*, G. Cantone and M. Marchesi, Eds. Springer International Publishing, 2014, pp. 110–

- [102] O. Liskin and K. Schneider, "Improving Project Communication with Virtual Team Boards," in *Workshop on Global Software Engineering for Agile Teams (Globagile '12) at ICGSE '12*, 2012, pp. 35–36.
- [103] O. Liskin, K. Schneider, F. Fagerholm, and J. Münch, "Understanding the Role of Requirements Artifacts in Kanban," in *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2014, pp. 56–63.
- [104] D. Lübke, *An Integrated Approach for Generation in Service-Oriented Architecture Projects*. Hamburg: Verlag Dr. Kovac, 2008.
- [105] D. Lübke, T. Flohr, and K. Schneider, "Serious Insights Through Fun Software-Projects," in *Proceedings of the 11th European Conference on Software Process Improvement (EuroSPI)*, T. Dingsøyr, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 57–68.
- [106] D. Lübke, M. Weidlich, and K. Schneider, "Visualizing Use Case Sets as BPMN Processes," in *Proceedings of the Requirements Engineering Visualization Workshop (REV)*, 2008.
- [107] P. Mäder, *Rule-based maintenance of post-requirements traceability. Dissertation*. MV-Verlag, 2010.
- [108] P. Mäder and J. Cleland-Huang, "A Visual Traceability Modeling Language," in *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, D. C. Petriu, N. Rouquette, and Ø. Haugen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 226–240.
- [109] P. Mäder and J. Cleland-Huang, "A visual language for modeling and executing traceability queries," *Softw. Syst. Model.*, vol. 12, no. 3, pp. 537–553, 2013.
- [110] P. Mäder, O. Gotel, and I. Philippow, "Getting back to basics: Promoting the use of a traceability information model in practice," in *Proceedings of the ICSE Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, 2009, pp. 21–25.
- [111] D. Méndez Fernández, "Requirements Engineering: Artefact-Based Customisation. Dissertation," Technische Universität München, 2011.
- [112] D. Méndez Fernández, K. Lochmann, B. Penzenstadler, and S. Wagner, "A case study on the application of an artefact-based requirements engineering approach," in *Proceedings of the 15th*

Annual Conference on Evaluation Assessment in Software Engineering (EASE), 2011, pp. 104–113.

- [113] D. Méndez Fernández and B. Penzenstadler, “Artefact-based requirements engineering: the AMDiRE approach,” *Requir. Eng.*, vol. 20, no. 4, pp. 405–434, 2015.
- [114] D. Méndez Fernández and S. Wagner, “Naming the pain in requirements engineering: A design for a global family of surveys and first results from Germany,” *Inf. Softw. Technol.*, vol. 57, pp. 616–643, 2015.
- [115] D. Méndez Fernández and R. Wieringa, “Improving Requirements Engineering by Artefact Orientation,” in *Proceedings of the 14th International Conference on Product-Focused Software Process Improvement (PROFES)*, J. Heidrich, M. Oivo, A. Jedlitschka, and M. T. Baldassarre, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 108–122.
- [116] J. Natt och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell, “Speeding up requirements management in a product software company: linking customer wishes to product requirements through linguistic engineering,” in *Proceedings of the 12th IEEE International Requirements Engineering Conference*, 2004, pp. 283–294.
- [117] L. Nguyen and A. P. Swatman, “Managing the requirements engineering process,” *Requir. Eng.*, vol. 8, no. 1, pp. 55–68, 2003.
- [118] P. Niemeyer and D. Leuck, *Learning Java*. O’Reilly Media, 2013.
- [119] K. Niklas, S. Gärtner, and K. Schneider, “Consistency Checks of Design Specifications against Requirements using Graph-Based Linguistic Analysis,” in *Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC)*, 2016.
- [120] B. Nuseibeh, J. Kramer, and A. Finkelstein, “A framework for expressing the relationships between multiple views in requirements specification,” *IEEE Trans. Softw. Eng.*, vol. 20, no. 10, pp. 760–773, 1994.
- [121] Object Management Group, “Unified Modeling Language (UML) - Version 2.5,” *Unified Modeling Language (UML) - Version 2.5*, 2015. [Online]. Available: <http://www.omg.org/spec/UML/2.5/>. [Accessed: 11-Mar-2016].
- [122] OMG, “Object Constraint Language (OCL) Version 2.4,” 2014. [Online]. Available: <http://www.omg.org/spec/OCL/2.4/>.

[Accessed: 25-May-2016].

- [123] Oracle, “The Reflection API.” [Online]. Available: <https://docs.oracle.com/javase/tutorial/reflect/>. [Accessed: 25-May-2016].
- [124] W. Ortlieb, “Unterstützung der Anforderungskommunikation durch Umwandlung von Webanwendungs-Oberflächen in editierbare Mockups (Masterarbeit),” Leibniz Universität Hannover, Fachgebiet Software Engineering, 2015.
- [125] J. Patton, *User Story Mapping*. O’Reilly Media, 2014.
- [126] B. Penzenstadler, J. Eckhardt, and D. Méndez Fernández, “Two Replication Studies for Evaluating Artefact Models in RE: Results and Lessons Learnt,” in *Proceedings of the 3rd International Workshop on Replication in Empirical Software Engineering Research (RESER)*, 2013, pp. 66–75.
- [127] F. A. C. Pinheiro, “Requirements Traceability,” in *Perspectives on Software Requirements*, J. C. S. do Prado Leite and J. H. Doorn, Eds. Boston, MA: Springer US, 2004, pp. 91–113.
- [128] K. Pohl, *Process-Centered Requirements Engineering*. John Wiley & Sons, Inc., 1996.
- [129] K. Pohl, *Requirements Engineering: Grundlagen, Prinzipien, Techniken*. dpunkt.Verlag, 2008.
- [130] K. Pohl and E. Sikora, “COSMOD-RE: Supporting the Co-Design of Requirements and Architectural Artifacts,” in *Proceedings of the 15th IEEE International Requirements Engineering Conference (RE)*, 2007, pp. 258–261.
- [131] P. Pruski, S. Lohar, R. Aquanette, G. Ott, S. Amornborvornwong, A. Rasin, and J. Cleland-Huang, “TiQi: Towards natural language trace queries,” in *Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE)*, 2014, pp. 123–132.
- [132] B. Ramesh and M. Jarke, “Toward reference models for requirements traceability,” *IEEE Trans. Softw. Eng.*, vol. 27, no. 1, pp. 58–93, Jan. 2001.
- [133] A. Rashid, P. Sawyer, A. Moreira, and J. Araujo, “Early aspects: a model for aspect-oriented requirements engineering,” in *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, 2002, pp. 199–202.

- [134] A. Rausch and M. Broy, "Die V-Modell XT Grundlagen," in *Das V-Modell XT: Grundlagen, Methodik und Anwendungen*, R. Höhn and S. Höppner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–27.
- [135] L. Richardson and S. Ruby, *RESTful Web Services*. " O'Reilly Media, Inc.," 2008.
- [136] M. Richters and M. Gogolla, "OCL: Syntax, Semantics, and Tools," in *Object Modeling with the OCL: The Rationale behind the Object Constraint Language*, T. Clark and J. Warmer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 42–68.
- [137] J. Robertson and S. Robertson, "Volere Requirements Specification Template - Edition 18," 2016. [Online]. Available: <http://volere.co.uk/template.htm>. [Accessed: 25-May-2016].
- [138] C. Robson, *Real World Research*. Wiley, 2011.
- [139] B. Rumpe, *Agile Modellierung mit UML: Codegenerierung, Testfälle, Refactoring*. Springer-Verlag, 2012.
- [140] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empir. Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2009.
- [141] C. Rupp and D. SOPHISTEN, *Requirements-Engineering und -Management: aus der Praxis von klassisch bis agil*, 6. Auflage. München: Hanser, 2014.
- [142] K. Schneider and O. Liskin, "Exploring FLOW Distance in Project Communication," in *Proceedings of 8th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2015.
- [143] K. Schneider and O. Liskin, "Exploring FLOW Distance in Project Communication," in *Proceedings of the 8th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2015, pp. 117–118.
- [144] K. Schneider, O. Liskin, H. Paulsen, and S. Kauffeld, "Requirements compliance as a measure of project success," in *IEEE Global Engineering Education Conference (EDUCON)*, 2013, pp. 1276–1283.
- [145] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Prentice Hall, 2002.
- [146] H. Schwarz, J. Ebert, V. Riediger, and A. Winter, "Towards Querying

- of Traceability Information in the Context of Software Evolution.," in *10. Workshop Software-Reengineering (WSR 2008) der GI-Fachgruppe Software Reengineering (SRE). GI Lecture Notes in Informatics 126*, R. Gimnich, U. Kaiser, J. Quante, and A. Winter, Eds. GI Bonn, 2008.
- [147] S. A. Sherba, K. M. Anderson, and M. Faisal, "A framework for mapping traceability relationships," in *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering*, 2003.
- [148] M. Silva and T. Oliveira, "Towards Detailed Software Artifact Specification with SPEMArti," in *Proceedings of the 2011 International Conference on Software and Systems Process*, 2011, pp. 213–217.
- [149] J. Singer, S. E. Sim, and T. C. Lethbridge, "Software Engineering Data Collection for Field Studies," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. London: Springer London, 2008, pp. 9–34.
- [150] R. van Solingen, "Measuring the ROI of software process improvement," *IEEE Softw.*, vol. 21, no. 3, pp. 32–38, 2004.
- [151] I. Sommerville, *Software Engineering*, 8. Auflage. Pearson Studium, 2007.
- [152] Sparx Systems, "Enterprise Architect," 2016. [Online]. Available: <http://www.sparxsystems.com/products/ea/>. [Accessed: 25-May-2016].
- [153] I. Spence and L. Probasco, "Traceability strategies for managing requirements with use cases," *White Pap. Ration. Softw. Corp.*, 1998.
- [154] F. Staab, *Logik und Algebra: Eine praxisbezogene Einf{ü}hrung f{ü}r Informatiker und Wirtschaftsinformatiker*. De Gruyter, 2012.
- [155] K. Stapel and K. Schneider, "Managing Knowledge on Communication and Information Flow in Global Software Projects," *Expert Syst. - Spec. Issue Knowl. Eng. Glob. Softw. Dev.*, 2012.
- [156] H. Störrle, "VMQL: A visual language for ad-hoc model querying," *J. Vis. Lang. Comput.*, vol. 22, no. 1, pp. 3–29, 2011.
- [157] G. Theocharis, M. Kuhrmann, J. Münch, and P. Diebold, "Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices," in *Proceedings of the 16th International Conference on Product-Focused Software Process Improvement*

- (PROFES), P. Abrahamsson, L. Corral, M. Oivo, and B. Russo, Eds. Cham: Springer International Publishing, 2015, pp. 149–166.
- [158] S. Walderhaug, U. Johansen, E. Stav, and J. Aagedal, “Towards a Generic Solution for Traceability in MDD,” in *Proceedings of the ECDMA Traceability Workshop*, 2006, pp. 41–50.
- [159] R. S. Wazlawick, “Functional Modeling with OCL Contracts (Chapter 8),” in *Object-Oriented Analysis and Design for Information Systems*, R. S. Wazlawick, Ed. Boston: Morgan Kaufmann, 2014, pp. 193–225.
- [160] R. Wieringa, “An Introduction to Requirements Traceability,” no. IR-389. Free University, Faculty of Mathematics and Computer Science, Amsterdam, the Netherlands, Sep-1995.
- [161] C. Wilke, M. Thiele, B. Freitag, and L. Schütze, “Dresden OCL - Manual for Installation, Use and Development,” 2016. [Online]. Available: http://moz-code.org/uqam/cours/INF3143/ressources/dresden_manual.pdf. [Accessed: 25-May-2016].
- [162] T. Winter, “Unterstützung der Konsistenzsicherung von Anforderungsartefakten durch Natural Language Processing (Bachelorarbeit),” Leibniz Universität Hannover, 2015.
- [163] C. Wohlin and A. Aurum, “Criteria for Selecting Software Requirements to Create Product Value: An Industrial Empirical Study,” in *Value-Based Software Engineering*, S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 179–200.
- [164] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer Science & Business Media, 2012.
- [165] J. Zettel, *Anpassbare Methodenassistenz in CASE-Werkzeugen. Dissertation*. Fraunhofer-IRB-Verlag, 2003.
- [166] J. Zettel, “Methodology Support in CASE Tools and Its Impact on Individual Acceptance and Use: A Controlled Experiment,” *Empir. Softw. Eng.*, vol. 10, no. 3, pp. 367–394, 2005.
- [167] D. Zowghi and N. Nurmuliani, “A study of the impact of requirements volatility on software project performance,” in *Proceedings of the Ninth Asia-Pacific Software Engineering Conference*, 2002, pp. 3–11.

Lebenslauf

Persönliche Daten

Name: Olga Boruszewski (geb. Liskin)

Email: olga.boruszewski@gmail.com

Geburtsdatum: 27.06.1986

Geburtsort: Kiew, Ukraine

Ausbildung

Sept 1998 – Mai 2005: Leibnizgymnasium, Hannover
Abschluss: Abitur

Okt 2005 – Nov 2010: Studium der Mathematik mit Studienrichtung Informatik an der Leibniz Universität Hannover
Abschluss: Diplom

Thema der Diplomarbeit: *Anreicherung von Web-Service-Antworten mit möglichen Zustandsübergängen*

Jan 2009 – Mai 2009: Auslandsstudium an der University of Colorado at Boulder, USA

Jan 2011 – Dez 2016: Promotionsstudium am Fachgebiet Software Engineering der Leibniz Universität Hannover.

Promotionsthema: *Unterstützung der Koexistenz von agilen und traditionellen Anforderungsartefakten*

Beruflicher Werdegang

Okt 2007 – Dez 2008 und
Jan 2010 – Sept 2010: Wissenschaftliche Hilfskraft am Fachgebiet Software Engineering, Leibniz Universität Hannover

Jun 2009 – Sept 2009: Praktikum als Software-Entwicklerin bei Rally Software Development, Boulder, USA

Jan 2011 – Feb 2016: Wissenschaftliche Mitarbeiterin am Fachgebiet Software Engineering, Leibniz Universität Hannover