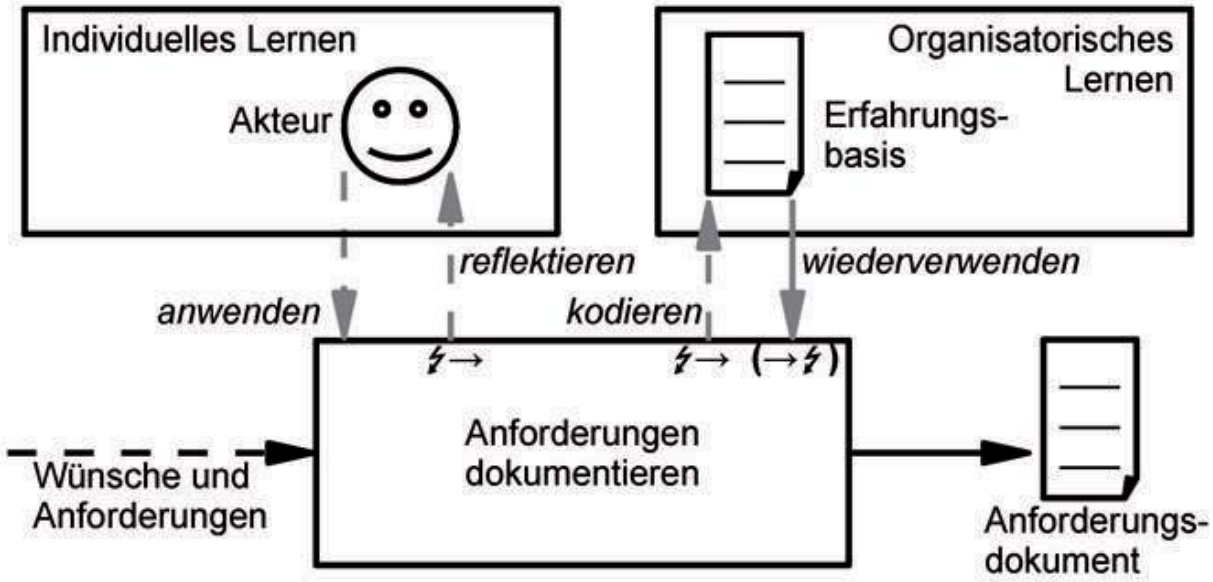
Eric Werner Knauss

Verbesserung der Dokumentation von Anforderungen auf Basis von Erfahrungen und Heuristiken



Cuvillier Verlag Göttingen
Internationaler wissenschaftlicher Fachverlag

Verbesserung der Dokumentation von Anforderungen auf Basis von Erfahrungen und Heuristiken

Von der Fakultät für Elektrotechnik und Informatik der
Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des akademischen Grades eines

Doktor-Ingenieur
(abgekürzt: Dr.-Ing.)

genehmigte Dissertation

von M. Sc. Eric Werner Knauss
geboren am 17. September 1977 in Hattingen

2010

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.
1. Aufl. - Göttingen: Cuvillier, 2010
Zugl.: Hannover, Univ., Diss., 2009

978-3-86955-593-5

1. Referent: Prof. Dr. Kurt Schneider
2. Referent: Prof. Dr. Martin Glinz
Tag der Promotion: 22. Oktober 2010

Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de

1. Auflage, 2010
Gedruckt auf säurefreiem Papier

978-3-86955-593-5

Zusammenfassung

Auch 40 Jahre nach der Software-Krise haben Organisationen Schwierigkeiten, funktionierende Software zu liefern, die die Kundenwünsche in ausreichender Qualität erfüllt. Dies ist zu einem beträchtlichen Teil auf mangelhaftes Requirements Engineering zurückzuführen: Wenn wichtige Aspekte der Anforderungen nie geklärt oder sogar missverstanden werden, führen offene Fragen dazu, dass Entwickler durch deren Klärung aufgehalten werden oder die Lücken durch eigene Annahmen schließen. Beides kann zu späten, grundlegenden Änderungen, einem hohen Mehraufwand und schlimmstenfalls zu einem für den Kunden unbrauchbaren Produkt führen. Auf diese Weise scheitern Projekte, verspäten sich oder überschreiten ihr Budget deutlich. Systematische und präzise Dokumentation von Anforderungen hilft, weil sie eine intensive Analyse der Anforderungen erfordert. Häufig können so versteckte Widersprüche entdeckt werden, die sonst erst bei der Implementierung oder Inbetriebnahme aufgefallen wären. Gute Anforderungsdokumentation erleichtert es Qualitätsmanagern zu prüfen, ob alle Anforderungen erfüllt sind. Projektmanager schätzen den Aufwand und kontrollieren den Fortschritt auf Basis dokumentierter Anforderungen. Vor allem erlaubt gute Anforderungsdokumentation den Entwicklern, sich auf die Umsetzung zu konzentrieren, statt Lücken mit dem Kunden zu klären – eine Aufgabe, für die sie meistens nicht ausgebildet sind. Trotz der Schlüsselfunktion von Anforderungen wird das Requirements Engineering oft vernachlässigt. Möglichst schnell wollen Kunden Ergebnisse sehen und Manager mit der Erstellung der Lösung beginnen. Die Aufgabenklärung wird zu wenig als Teil der Lösungserbringung gesehen. Kürzere Releasezyklen und Time-to-Market bauen zusätzlichen Zeitdruck auf. Organisationen müssen unter hohem Zeitdruck und mit beschränkten Ressourcen die Anforderungen immer komplexerer Systeme in hoher Qualität dokumentieren. Erfahrung ist bei diesem Ziel einer der wichtigsten Erfolgsfaktoren. Viele Organisationen greifen daher auf erfahrene externe Berater zurück, oder lassen ihre Erfahrung und Reife formal bestimmen und verbessern.

Diese Arbeit zeigt einen systematischen Ansatz, wie Organisationen vorhandene Erfahrungen mit dem etablierten Dokument- und Vorgehensmodell besser nutzen und ausbauen können. Dazu wird untersucht, wie Computer die Erstellung natürlichsprachlicher Anforderungsdokumente unterstützen können, die sich etablierten Verfahren wie Model-Checking oder Simulation verschließen. Dabei wird kein spezielles Dokument- oder Vorgehensmodell vorausgesetzt.

Die Arbeit belegt eine Korrelation zwischen Qualität von Anforderungsdokumenten und Projekterfolg: Projekte mit schlechten Anforderungen scheitern öfter. Meistens gibt es für inhaltliche Probleme der Anforderungen syntaktische Indikatoren. Computer sind in der Lage, diese Indikatoren automatisch zu nutzen und dem Analysten mit entsprechenden Hinweisen zu helfen, möglicherweise vorliegende inhaltliche Probleme zu erkennen und zu beheben. Das Konzept, syntaktische Indikatoren automatisch auszunutzen um

einen hilfreichen Hinweis zu geben, wird in dieser Arbeit als *heuristische Kritik* bezeichnet. Heuristische Kritiken erlauben es, die Erfahrungen einer Organisation abzubilden und sind ein Weg, um Erfahrungen computerbasiert bei der Dokumentation von Anforderungen anzuwenden. So können relevante Erfahrungen effizient gefunden, aktiviert und um neue Erfahrungen ergänzt werden. Die Arbeit zeigt die wichtigsten Eigenschaften solcher erfahrungsbasierter Werkzeuge des Requirements Engineering. Jede dieser Eigenschaften wird empirisch evaluiert.

Die Beiträge dieser Dissertation sind i) ein konzeptionelles Modell über den Zusammenhang von computerbasierter Analyse natürlichsprachlicher Anforderungen und Erfahrungsmanagement, ii) eine Strategie zur empirischen Untersuchung dieses Zusammenhangs und iii) wichtige Datenpunkte zum Potential dieses Ansatzes.

Schlagwörter: Requirements Engineering, Erfahrungsmanagement, heuristische Kritiken

Abstract

40 years after the software crisis, software developing organisations have still problems to deliver working software that fulfils the customer's wishes in sufficient quality. The largest part of these problems is caused by insufficient requirements engineering: Important aspects are never clarified or are misunderstood. Open questions related to requirements can either slow down developers who need to clarify them, or force them to fill the gaps with their own assumptions. Both can lead to fundamental changes later on in the project and thus cause high additional effort. Furthermore, misunderstood requirements can result in a product that is useless for the customer. This way, projects fail, are delayed, or seriously exceed their budget. Systematic and precise documentation of requirements helps, because it demands an intensive analysis of requirements. Such analysis can often identify hidden contradictions and misunderstandings that otherwise would not be discovered before implementation or operation. Good requirements documents can be used by quality managers to check whether all requirements have been met. Project managers estimate effort and control the progress based on the documented requirements. Most important, good requirements documents allow developers to focus on the implementation, instead of discussing gaps with the customer – a job they are most often not trained for. Despite this key role of requirements documents, requirements engineering is often neglected. Customers want to see results and managers intend to start creating solutions early. Understanding the problem is not sufficiently regarded as part of its solution. Short *release cycles* and *time to market* introduce additional time pressure. Software developing organisations have to document high quality requirements of more and more complex systems under high time pressure with limited resources. Experience is one of the main success factors to reach this goal. Therefore, many organisations hire experienced external consultants. Others formally assess and improve the experience and maturity of their organisation.

This dissertation shows a systematic approach allowing organisations to leverage and extend their existing experiences with established documents and processes of development. It investigates how computers can support creating natural language requirements documents that cannot be accessed by model checking or simulation approaches. It does not require a special document model or development process.

The dissertation shows a correlation between requirements quality and project success: Projects with bad requirements fail more often. Most often, syntactical indicators exist which point to existing problems in natural language requirements documents. Computers can be used to automatically use such indicators. Once detected, hints and warnings help analysts to identify and solve the potential problems. In this work, the concept of automatically exploiting such indicators in order to give useful hints is called *heuristic critique*. Heuristic critiques allow an organisation to explicitly represent their experiences. Computers can support the application of such experiences with the help of indicators. This allows to identify relevant experiences, to activate them, and

to complement them with new experience very efficiently. This dissertation shows the most important properties of such experience based tools for requirements engineering. Each of these properties is empirically evaluated.

The contributions of this dissertation are i) a conceptual model of the relationship between the computer based analysis of natural language requirements and experience management, ii) a strategy to empirically analyse this relationship, and iii) important data points showing the potential of this approach.

Keywords: Requirements Engineering, Experience Management, Heuristic Critiques

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

1.1.1 Problemumfeld

Requirements Engineering umfasst die Analyse und Verwaltung von Anforderungen. Es bildet die Grundlage für planbare Erfolge bei der Erstellung von Software. Entsprechend wird Requirements Engineering seit langer Zeit in der Literatur, auf Konferenzen und in lokalen Arbeitskreisen diskutiert. Trotzdem bleiben in der Praxis viele Fragen offen: Auf welche Aspekte ist, vor allem unter Zeitdruck, besonders zu achten? Wann sind Anforderungen gut genug spezifiziert, um in den Entwurf und die Implementierung überzugehen? Enthält eine Spezifikation noch Missverständnisse?

Die Grundannahme dieser Arbeit ist: Die praktischen Probleme bei der Anwendung von Requirements Engineering liegen nicht darin, dass diese Fragen in der Literatur nicht beantwortet werden. Es liegt auch nicht an schlechter Ausbildung, Faulheit oder Böswilligkeit von Anforderungsingenieuren. Vor dem Hintergrund enger Zeitpläne und hoher Komplexität des zu erstellenden Software-Systems liegen die Ursachen dieser Probleme in der Schwierigkeit der Aufgabe:

- Softwareprojekte werden immer komplexer. Dies führt auch bei der Klärung der Rahmenbedingungen und Anforderungen eines Projekts zu einer kaum überschaubaren Informationsmenge. Die Konsistenz aller anforderungsbezogenen Dokumente und Modelle eines Softwareprojekts ist nur schwer zu wahren.
- Requirements Engineering hängt sehr stark von der persönlichen Erfahrung und den Fähigkeiten einzelner Personen ab. Selbst diesen Experten fällt es aber auf Grund der Komplexität und des Zeitdrucks schwer, ihre Erfahrungen im entscheidenden Augenblick zu aktivieren und anzuwenden.

Um Anforderungsingenieure bei ihrer Aufgabe zu unterstützen, müssen diese zunächst einmal entlastet werden. So bieten sich für die Verarbeitung und Konsistenzprüfung großer Mengen anforderungsbezogener Informationen in unterschiedlichen Dokumenten computerbasierte Lösungen an. Im Rahmen dieser Arbeit wird gezeigt, dass dadurch enorme Zeit (bis zu einem Personenmonat in größeren Projekten) für die formale Konsistenzprüfung eingespart werden kann, die dann für

wichtigere, inhaltliche Aufgaben zur Verfügung steht (siehe Abschnitt 8.3). Zudem wird gezeigt, wie Computer für eine gegebene Situation im Requirements Engineering relevante Erfahrungen ermitteln und dem Nutzer zugänglich machen können. Da dabei hauptsächlich natürlichsprachliche Anforderungsbeschreibungen verarbeitet werden müssen, ist es schwer, Algorithmen zu finden, die eindeutige Ergebnisse liefern. Durch die Mehrdeutigkeit und Kontextabhängigkeit natürlicher Sprache führen die meisten dieser Algorithmen bei ihrer Analyse mit einer gewissen Wahrscheinlichkeit zu Fehlern. Im Rahmen dieser Arbeit werden daher heuristische Regeln verwendet, um Indikationen aufzuspüren, die darauf hinweisen, dass eine Erfahrung anwendbar oder eine Inkonsistenz enthalten ist. Heuristische Regeln liefern häufig das gewünschte Ergebnis, aber nicht immer (vergleiche Abschnitt 5.2), sollen jedoch dennoch nützlich sein. Durch diese im Vergleich zu anderen Algorithmen geringeren Anforderungen lassen sich heuristische Regeln einfacher implementieren und verbessern, in vielen Fällen sogar im Betrieb durch den Nutzer. Abbildung 1.1 zeigt in einem Beispiel, wie heuristische Regeln bei der Erstellung eines Use-Cases eingesetzt werden können. Der Nutzer erhält als computerbasiertes Feedback die Kritik, dass sich passive Formulierungen nicht für Anforderungen eignen.

Abbildung 1.1: Eine heuristische Kritik weist auf ein mögliches Problem hin (hier in der Beispielimplementierung HeRA).

Kritiken auf Basis heuristischer, automatisch prüfbarer Regeln (sogenannte *heuristische Kritiken*) können Inkonsistenzen und Verstösse gegen Best Practices und andere Erfahrungen also direkt aufzeigen (ähnlich der Rechtschreibprüfung bei der Textverarbeitung). Sie können aber auch helfen, Konsistenz zu sichern und Erfahrungen zwischen Personen und Projekten zu transferieren.

1.1.2 Hintergrund

Um die Dokumentation von Anforderungen zu verbessern, sind in Literatur und Praxis verschiedene Ansätze gebräuchlich. Die folgenden beiden Ansätze erfordern zum Teil recht umfangreiche Änderungen im Vorgehen einer Organisation:

- *Prozess:* Durch Verbesserungen am Erstellungsprozess der Dokumente kann eine hohe Steigerung der Qualität von Anforderungsdokumenten erreicht

werden (z.B. [139]). Verbesserungen am Prozess führen dazu, dass Anforderungen verschiedene Bearbeitungs-, Verifikations- und Validierungsschritte durchlaufen müssen, bis sie als ausreichend dokumentiert gelten. Die Einbeziehung unterschiedlicher Rollen erleichtert es, Fehler zu finden und zu beheben.

- *Modelle:* Bessere Modelle (zum Beispiel [175, 82, 21]) führen zu einer formalen Repräsentation von Anforderungen, die mit Hilfe von Model-Checkern oder Simulation überprüft werden kann [22].

Beide Ansätze haben gemeinsam, dass sie die Qualitätskontrolle erleichtern. Sowohl Änderungen am Prozess, als auch Änderungen an der Modellierungsart stellen jedoch große Einschnitte für Software erstellende Organisationen dar. Erfahrungen mit dem alten Prozess oder den bisher verwendeten Modelltypen werden damit größtenteils wertlos.

Etwas weniger einschneidende Verbesserungstypen beziehen sich auf die Verwendung und Verbesserung von Vorlagen (englisch: Templates) oder Qualitätssicherungsmaßnahmen:

- *Vorlagen:* Vorlagen sind ein gutes Mittel, Erfahrungen in Softwareprojekte einzubringen. Sie können als eine Art Checkliste gesehen werden, die verhindert, dass wichtige Aspekte vergessen werden. Zudem geben Vorlagen eine Strukturierung vor, die sich bereits in Projekten bewährt hat. Prominente Vorlagen für die Anforderungsdokumentation sind die Anforderungsschablone von Rupp [142] und die Use-Case-Vorlagen von Cockburn [42].
- *Qualitätssicherungsmaßnahmen:* Für die Qualitätssicherung haben sich verschiedene, überwiegend analytische Verfahren bewährt. Beispiele sind Quality Gates [66] sowie Checklisten und Bewertungskriterien für Anforderungsreviews [48].

Diese beiden Ansätze haben gemeinsam, dass sie eine kontinuierliche Verbesserung erlauben. Nach jeder Anwendung können die Vorlagen und Checklisten angepasst werden, um im gegebenen Kontext einer Organisation noch bessere Resultate zu liefern (siehe [102]).

In der Praxis steht zudem noch die Möglichkeit offen, besonders erfahrene Mitarbeiter mit einzubinden, zum Beispiel als externe Berater. Diese können häufig wertvolles Feedback liefern und helfen, die häufigsten Fallstricke zu vermeiden. Experten sind in der Regel aber knapp und gute Berater teuer. Daher wird diese Option auch dann oft nicht wahrgenommen, wenn der Bedarf an Beratung erkannt wurde.

Der Versuch, fehlende Experten zu substituieren, liegt daher nahe. Die computerbasierte Analyse von Anforderungsdokumentation erlaubt es, mehr oder weniger

komplexe Indikatoren für Fehler zu nutzen [59, 18, 90], um dann geeignete Verbesserungsvoschläge wie durch einen Experten zu unterbreiten. Die Ansätze reichen von der einfachen Suche nach Schlüsselwörtern, über die Analyse von Aspekten der Anforderungsmodelle bis zur Verarbeitung natürlicher Sprache. Je nach Einsatz dieser Ansätze kann so ein Experte zwar nicht vollständig ersetzt werden. Es kann jedoch wertvolles Feedback geben werden, durch das ein vorhandener Experte entlastet wird oder das Fehlen eines Experten abgefedert wird. Die Analyse von Modellen hängt von der spezifischen Repräsentation der Modelle ab und ist damit oft abhängig von bestimmten Modellierungswerkzeugen. Die Analyse natürlicher Sprache gilt als sehr komplex. Dies mögen Gründe dafür sein, dass computerbasierte Ansätze zur Analyse von Anforderungsdokumentation relativ schwach verbreitet sind.

1.2 Zielsetzung und Methodik

Heuristische Kritiken scheinen also eine gute Möglichkeit zu sein, um die Nutzung von Erfahrungen computerbasiert zu unterstützen. Diese Arbeit untersucht, ob sich dieses Konzept eignet, um Erfahrungen im Requirements Engineering zur kontinuierlichen Verbesserung der (natürlichsprachlichen) Dokumentation von Anforderungen zu nutzen. Die Arbeit bedient sich damit bei Ansätzen aus dem Erfahrungsmanagement[1] und wendet sie auf das Requirements Engineering an.

1.2.1 Zielsetzung und Forschungsfragen

Das Ziel der Arbeit ist es zu zeigen, wie Dokumentation von Anforderungen durch erfahrungsbasierte und heuristische Konzepte verbessert werden kann. Aus der Perspektive dieser Arbeit ergeben sich daraus die folgenden Forschungsfragen:

1. Lässt sich durch heuristische Kritiken das systematische Lernen einer Software erstellenden Organisation unterstützen? Wie groß sind mögliche Effekte dieser Unterstützung? Dabei wird a) das Lernen der Organisation und b) das Lernen der Individuen der Organisation unterschieden.
2. Lässt sich durch heuristische Kritiken die Effektivität (d.h. die Qualität) oder die Effizienz (d.h. die Kosten) von Anforderungsdokumenten verbessern? Wie groß sind mögliche Effekte dieser Unterstützung?

Die folgenden Abschnitte zeigen die wissenschaftliche Methodik und das Vorgehen in dieser Arbeit, um dieses Ziel zu erreichen und Antworten auf die Forschungsfragen zu finden.

[1] Erfahrungsmanagement ist ein relativ neuer Begriff und wird in der Regel als Teilgebiet des Wissensmanagement definiert. Vergleiche [23, Seite 14], [151] sowie Def. 10 dieser Arbeit.

Tabelle 1.1: Vergleich des Vorgehens bei konstruktiven und empirischen Arbeiten.

Ansatz	**Konstruktiv**	**Empirisch**
Fokus	Entwickeln und Evaluieren	Hypothesen aufstellen und belegen
Vorgehen	1. Problem der Praxis identifizieren 2. Problem auf Theorie zurückführen 3. Lösung konstruieren 4. Lösung an Problem messen	1. Fragestellung durch anekdotische Beobachtung 2. Hypothesen aufstellen 3. Empirische Untersuchung 4. Hypothesen bestätigen oder verwerfen

1.2.2 Wissenschaftliche Methodik

Wissenschaftliche Arbeiten im Software Engineering lassen sich grob in zwei Kategorien einordnen: *konstruktive Arbeiten*, die neue technische Lösungen für ein Problem vorschlagen, und *empirische Arbeiten*, die existierende Fragestellungen beantworten. Tabelle 1.1 gibt einen Vergleich des typischen Vorgehens in den beiden Kategorien. Diese Arbeit lässt sich nicht eindeutig in eine der beiden Kategorien einordnen. Es werden zwar Forschungsfragen empirisch beantwortet, die Evaluation ist jedoch ohne neue technische Lösungen nicht möglich. Die technischen Lösungen hingegen beruhen auf existierenden Techniken und Konzepten aus den Bereichen Requirements Engineerung und Erfahrungsmanagement. Die Leistung dieser Arbeit ist weniger in den technischen Lösungen zu suchen, sondern in den Konzepten, Requirements Engineering und Erfahrungsmanagement zusammenzubringen.

Das Problem, dass eine neue Idee nicht evaluiert werden kann, ohne den Beobachtungsgegenstand zu verändern, ist für wissenschaftliche Arbeiten nicht neu. So kann der Einfluss des Beobachters auf das Beobachtungsergebnis in naturwissenschaftlichen Experimenten häufig auf das Einbringen von (systematischen) Messungenauigkeiten reduziert werden (zum Beispiel bei der *persönlichen Gleichung* in der Astronomie). Bei sozialen Wissenschaften kommt die soziale Interaktion zwischen Beobachter und Beobachteten noch hinzu. Je nach Kontext stellen sich hier noch zwei weitere Probleme: Durch den Aufbau eines klinischen Experiments, das versucht, alle unerwünschten Einflüsse zu eliminieren, entfernt man sich zu weit von dem zu beobachteten sozialen Gefüge. Zudem ist es teilweise moralisch verwerflich oder auf andere Weise unerwünscht, einer Kontrollgruppe den zu erwartenden Vorteil einer zu evaluierenden Idee vorzuenthalten, wenn dieser dadurch ein Schaden entsteht. Dies hat zur Entwicklung des Action Research geführt [112]. Da diese Schwierigkeiten der sozialen Wissenschaften auch für Un-

tersuchungen von Softwareprojekten gelten, hat das Action Research mittlerweile auch in die Informatik Einzug erhalten [55] und auch diese Arbeit beeinflusst:

Diese Arbeit beruht auf den schwer zu evaluierenden Grundannahmen, dass a) Erfahrung ein entscheidender Erfolgsfaktor für das Requirements Engineering ist und b) Requirements Engineering ein entscheidender Erfolgsfaktor für Softwareprojekte ist. Daraus leitet sich die Hoffnung ab, dass erfahrungsbasierte Konzepte im Requirements Engineering einen wichtigen Beitrag zum Gelingen von Softwareprojekten leisten. Für die Evaluation ist daher ein konstruktiver Anteil zur Erstellung dieser Konzepte unumgänglich. Die dabei erstellten neuen technischen Lösungen sind jedoch exemplarisch gemeint - Ziel der Evaluation ist eigentlich die dahinter stehende Idee. Aus Sicht dieser Arbeit ist eine Idee gut, wenn sie einen Vorteil beim Requirements Engineering einer Organisation bringt.

Da der Nutzen der Ergebnisse im Vordergrund steht, lässt sich der Forschungsansatz nach Easterbrook et al. als Pragmatismus bezeichnen [55, S. 292]. Durch den konstruktiven Anteil besitzt die Arbeit im Großen Anteile des Action Research. Methodisch wird hautpsächlich auf Fallstudien zurückgegriffen, ergänzt um Experimente an zentralen Stellen. Nach Easterbrook et al. ist dies charakteristisch für den Pragmatismus.

1.2.3 Vorgehen

Für diese Arbeit leitet sich nach dem vorherigen Abschnitt und Tabelle 1.1 folgendes Vorgehen ab:

1. *(Empirisch) Fragestellung durch anekdotische Beobachtung:* Läßt sich Anforderungsdokumentation mit Erfahrungen und Heuristiken verbessern? Die Forschungsfragen dieser Arbeit werden im folgendem Abschnitt kurz angerissen, aber auf Basis der Theorie erst in Abschnitt 6.2 detailliert.

2. *(Konstruktiv) Problem auf Theorie zurückführen:* Wie hängen Requirements Engineering, Erfahrungsmanagement und Heuristiken zusammen? Der Rahmen der Arbeit wird gesteckt, indem das Ziel (Verbesserungen im Requirements Engineering, Abschnitt 4.1), das Problemfeld (Anforderungsdokumentation, Abschnitt 4) und der Ansatz (erfahrungsbasiertes Requirements Engineering, Abschnitt 5) theoretisch erarbeitet werden. Die relevanten Aspekte werden definiert und sind Grundlage für den ersten wichtigen Beitrag dieser Arbeit: Der konzeptionellen Definition des Zusammenhangs computerbasierter Analyse von Anforderungsdokumentation und erfahrungsbasierten Ansätzen im Software Engineering.

3. *(Empirisch) Hypothesen aufstellen:* Wie sind die Wirkzusammenhänge von Heuristiken, Dokumentation von Anforderungen und Erfahrungen? Auf wel-

che Faktoren kommt es dabei an? Auf Basis des Konzepts wird die Problemstellung für die Arbeit in Abschnitt 6.2 konkretisiert und mögliche Lösungen charakterisiert. Dies ist ein wichtiger Schritt, denn er leitet über von der theoretischen Beschreibung von Ziel, Problemfeld und Ansatz hin zu konkret mess- und evaluierbaren Lösungen.

4. *(Konstruktiv) Lösung konstruieren:* Mit Hilfe des Konzepts und der Charakterisierung möglicher Lösungen werden systematisch Beispielimplementierungen erarbeitet. Die Auswahl der Beispielimplementierungen erlaubt es, verschiedene Aspekte heuristischer und erfahrungsbasierter Werkzeuge zu evaluieren.

5. *(Empirisch) Hypothesen bestätigen oder verwerfen:* Um tatsächliche von vermeintlichen Wirkzusammenhängen zu trennen, werden die Beispielimplementierungen in unterschiedlichen Evaluationsszenarien angewendet. Die Auswahl der Evaluationsszenarien erlaubt es, empirisch zu untersuchen, welche Arten von Anforderungsdokumentation und welcher Kontext durch erfahrungsbasierte und heuristische Ansätze wirkungsvoll unterstützt werden können.

Abbildung 1.2 zeigt ein Beispiel aus der Evaluation: Die Beispielimplementierung wendet heuristische Kritiken bei der Erstellung von Use-Cases an. Im Evaluationsszenario *studentisches Softwareprojekt* zeigt sich, dass die beiden Gruppen, die auf diese Weise mit Erfahrungen unterstützt werden, bessere Use-Cases schreiben, als die drei Vergleichsgruppen (vergleiche Abschnitt 8.2.3).

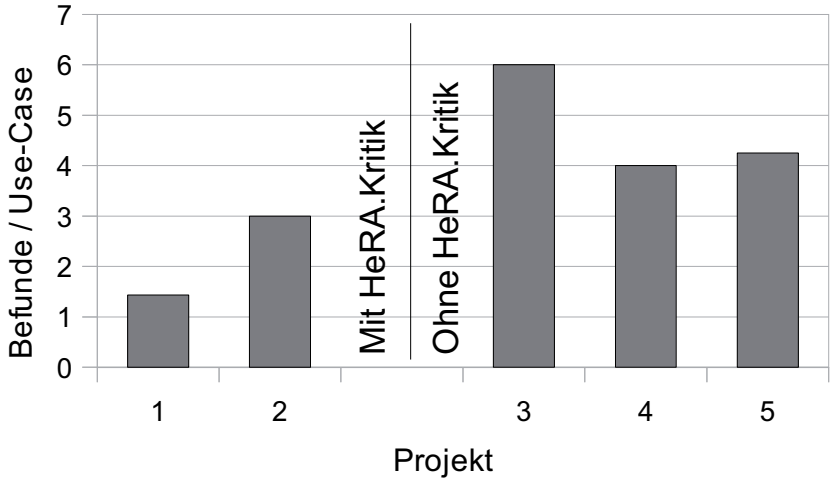


Abbildung 1.2: Evaluatation der Anforderunsqualität: Use-Cases in Projekten mit heuristischer Unterstützung (links) beinhalten weniger Befunde nach Abschluss der Anforderungsanalyse.

1.3 Ergebnisse

Dieser Abschnitt zeigt zunächst Resultate, die im Rahmen dieser Arbeit erreicht wurden und diskutiert dann den wissenschaftlichen Beitrag und die Grenzen dieser Arbeit.

1.3.1 Resultate der Arbeit

Von verwandten Ansätzen hebt sich diese Arbeit vor allem dadurch ab, dass hier der Zusammenhang zwischen dem Einsatz computerbasierter Analyse von Anforderungsdokumentation und der systematischen Verwaltung von Erfahrungen hergestellt wird. Die Arbeit liefert dazu die folgenden Resultate:

1. Es wird nachgewiesen, dass es einen statistisch signifikanten Zusammenhang zwischen der Qualität der Anforderungsdokumente und dem Erfolg eines Projekts gibt. Dieses Ergebnis ist für Andere relevant, weil es die Wichtigkeit von gutem Requirements Engineering zeigt. Für diese Arbeit ist wichtig, dass die Definition von Qualität überwiegend auf Metriken beruht, die sich automatisch durch Computerprogramme messen lassen.
2. Ein abstraktes Modell zur Reife und Formalität von Anforderungsdokumenten erlaubt es, Einsatzbereiche für computerbasierte Analyse von Anforderungen festzulegen. Es zeigt sich, dass heuristische Kritiken im Projektverlauf schon deutlich vor anderen Methoden der Qualitätssicherung eingesetzt werden können, selbst wenn die Anforderungen erst formlos skizziert werden. Dieses Ergebnis ist hauptsächlich für die Abgrenzung der Konzepte dieser Arbeit von anderen Arbeiten wichtig.
3. Die Arbeit gibt als Zwischenergebnis eine Übersicht von Stakeholdern der Dokumentation von Anforderungen. Heuristische Kritiken können auf die speziellen Interessen und Rollen dieser Stakeholder abgestimmt werden, um so die Eintrittswahrscheinlichkeit typischer Risiken zu reduzieren. Zudem ist die persönliche Erfahrung und der Typ eines Projektbeteiligten entscheidend dafür, wie Befunde heuristischer Kritiken präsentiert werden müssen, um eine Akzeptanz zu erreichen.
4. Das Lernmodell dieser Arbeit erlaubt es zu beschreiben, wie Organisationen und Individuen durch heuristische Kritiken profitieren und neue Erfahrungen sammeln. Andere können sich an diesem Lernmodell orientieren, wenn sie erfahrungsbasierte Konzepte in das Requirements Engineering einer Organisation einführen.
5. Ein Steckbrief für erfahrungsbasierte Werkzeuge für das Requirements Engineering wird eingeführt und charakterisiert diese nach addressiertem Doku-

mentyp (Reife, Formalität), Nutzertyp (Einstellung gegenüber dem Werkzeug und persönliche Erfahrung) und Eigenschaften des erfahrungsbasierten Werkzeugs nach dem Lernmodell. Der Steckbrief erlaubt es, erfahrungsbasierte und heuristische Werkzeuge voneinander abzugrenzen. Dies wird im Rahmen dieser Arbeit für die Abgrenzung von anderen Arbeiten genutzt, der Steckbrief kann aber auch zur systematischen Planung geeigneter erfahrungsbasierter Werkzeuge verwendet werden.

6. Die Evaluation mit verschiedenen Beispielimplementierungen und Evaluationsszenarien zeigt das Potential der Ansätze. Unabhängig von der Erfahrung der Beteiligten (Student bis erfahrener Analyst) konnte selbst mit einfachen heuristischen Regeln eine deutliche Verbesserung der Anforderungsdokumentation erreicht werden. Der Nachweis der Nützlichkeit ist eine Argumentationshilfe für die Einführung ähnlicher Konzepte in softwareerstellenden Organisationen. Zudem sind dadurch Datenpunkte gegeben, auf die sich Forscher mit ähnlichen Vorhaben beziehen können.

Die systematische Verwaltung der Erfahrungen einer Organisation durch heuristische Kritiken kann die positiven Effekte langfristig sogar noch steigern. Diese Arbeit stellt die Infrastruktur zur Verfügung, um auch diese langfristigen Effekte zu beschreiben und zu evaluieren. Die Messung des langfristigen Erfahrungsgewinns einer Organisation bleibt jedoch zukünftigen Arbeiten überlassen.

1.3.2 Beitrag der Arbeit

Aus diesen Ergebnissen ergeben sich die folgenden wichtigen Beiträge dieser Arbeit:

1. Konzeptionelles Modell des Zusammenhangs der computerbasierten Analyse von Anforderungsdokumentation und dem Erfahrungsmanagement einer Software erstellenden Organisation.
2. Evaluationsstrategie zur qualitativen Analyse des oben genannten Zusammenhangs.
3. Wichtige Datenpunkte zu den relevanten Aspekten des konzeptionellen Modells.

Diese Beiträge fördern vor allem Praxis und Forschung im Requirements Engineering. Andere können von diesen Beiträgen profitieren, indem sie ihre eigenen Ansätze um die hier beschriebenen erfahrungsbasierten Konzepte erweitern, oder sich bei der Planung der Evaluation einschlägiger Forschungsfragen an der hier vorgestellten Evaluationsstrategie orientieren. Bei beiden Vorhaben sind die vorliegenden Datenpunkte eine wichtige Entscheidungsgrundlage.

1.3.3 Grenzen der Arbeit

Der Fokus dieser Arbeit liegt auf der kontinuierlichen Verbesserung der Anforderungsdokumentation einer Software erstellenden Organisation. Der Vorteil des Fokus auf der kontinuierlichen Verbesserung ist, dass sich die vorliegenden Konzepte und Techniken recht einfach in bestehende Strukturen integrieren lassen. Nachteilig ist, dass die einzelnen Verbesserungsschritte recht klein und das Verbesserungspotential früher oder später erschöpft ist. Will sich eine Organisation weiter verbessern, so kommt sie an strukturellen Änderungen des Prozesses und der Modellbildung für das Requirements Engineering nicht vorbei.

1.4 Eigene Vorarbeiten

In diesem Kapitel wurde die Motivation und Zielsetzung der Arbeit vorgestellt. Zudem wurden die Ergebnisse der Arbeit sowie die Methodik zu ihrer Erreichung skizziert. Dieser Abschnitt stellt Vorarbeiten vor und schließt die Einleitung mit dem logischen Aufbau der Arbeit. Die eigenen Vorarbeiten zu dieser Dissertation lassen sich wie folgt ordnen:

Relevanz von Erfahrungen im Requirements Engineering. Die Wichtigkeit von Erfahrung für das Requirements Engineering ist eine wichtige Grundannahme für diese Arbeit. Basis für diese Grundannahme sind Beobachtungen in studentischen Projekten [102] und bei Zusammenarbeiten mit Industriepartnern [99].

Heuristische Unterstützung bei der Dokumentation von Anforderungen. Heuristisches Feedback kann Analysten beim Dokumentieren von Anforderungen unterstützen [91, 104, 103, 106] und ist eine gute Ergänzung zu analytischen Qualitätssicherungsmaßnahmen [98].

Abgeleitete Modelle zur Unterstützung des Requirements Engineering. Abgeleitete Modelle können im Requirements Engineering Überblick schaffen [128], die Zusammenarbeit verschiedener Rollen unterstützen [110, 169] und durch die Analyse vermeintlicher Inkonsistenzen zu vollständigeren Anforderungsdokumenten führen [101].

Struktur- und Qualitätseigenschaften von Anforderungsdokumenten. Um die Verbesserung von Anforderungsdokumentation beschreiben zu können, ist eine intensive Beschäftigung mit der Struktur [93], zentralen Qualitätseigenschaften [97, 94] und dem Nutzen von Anforderungen in Softwareprojekten erforderlich [92, 105].

Weitere relevante Arbeiten. Einige Vorarbeiten haben die vorliegende Dissertation beeinflusst, konnten aber hinsichtlich der Forschungsziele keinen neuen Beitrag zu den hier dargestellten Ergebnissen liefern. Dazu zählen insbesondere Beispielimplementierungen, durch die die hier vorgestellten Konzepte in kollaborativen

Szenarien [161, 95, 96, 100] oder bei der Entwicklung sicherer Systeme eingesetzt werden können [79].

Diese Vorarbeiten sind in die vorliegende Arbeit eingeflossen. Über diese Vorarbeiten hinaus stellt diese Arbeit den konzeptionellen Zusammenhang her und rundet die Vorarbeiten durch neue Ergebnisse ab.

1.5 Aufbau der Arbeit

Die Arbeit gliedert sich wie folgt: In Kapitel 2 geht es zunächst um die Grundlagen, auf denen diese Arbeit aufbaut. Dabei wird der Stand der Technik der beiden Säulen dieser Arbeit wiedergegeben: Requirements Engineering und Erfahrungsmanagement.

Kapitel 3 gibt ein erstes Beispiel für den Einsatz heuristischer Kritiken bei der Dokumentation von Anforderungen. Der Heuristic Requirements Assistant (HeRA) ist eine Plattform, die zur Evaluation der Konzepte dieser Arbeit erstellt worden ist.

Die Kapitel 4 und 5 bilden das theoretische Fundament dieser Arbeit. Dabei werden die Beispiele aus Kapitel 3 zum besseren Verständnis aufgegriffen:

- *Ziel* und *Problemfeld*: Kapitel 4 beschreibt zunächst Rollen und Nutzerprofile, die von einer Verbesserung von Anforderungsdokumentation profitieren. Zudem werden die relevanten Aspekte von Anforderungsdokumentation modelliert und gezeigt, dass gute Anforderungsdokumentation einen Einfluss auf den Projekterfolg hat.
- *Ansatz:* In Kapitel 5 wird der Ansatz beschrieben, Erfahrungen systematisch zur Verbesserung von Anforderungsdokumentation zu nutzen. Es wird gezeigt, wie heuristische Kritiken genutzt werden können, um Erfahrungen in Erfahrungs-Pakete zu kodieren. Zudem wird gezeigt, wie Organisationen und Individuen von dieser Technik profitieren können.

Kapitel 6 fasst das konzeptionelle Modell der Arbeit zusammen. Dies dient zur Übersicht und Klassifikation von heuristischen Anforderungswerkzeugen. Mit Hilfe des konzeptionellen Modells wird die Problemstellung der Arbeit konkretisiert.

Kapitel 7 führt einen Steckbrief für erfahrungsbasierte Werkzeuge des Requirements Engineering ein. Zudem werden die für die Evaluation benötigten Beispielimplementierungen vorgestellt. Mit Hilfe des Steckbriefs wird gezeigt, dass die Beispielimplementierungen alle vorgestellten Konzepte abdecken.

In Kapitel 8 werden die Aspekte aus dem konzeptionellen Modell systematisch evaluiert. Zentral für dieses Kapitel ist eine Matrix, die verschiedene Beispielim-

plementierungen den Evaluationsszenarien zuordnet. Dies erlaubt es nachzuvollziehen, dass alle Aspekte des konzeptionellen Modells und alle Charakteristika heuristischer Anforderungswerkzeuge abgedeckt sind.

Die vorliegende Arbeit hat Anknüpfpunkte an viele andere Arbeiten, die zum Teil ähnliche Mechanismen nutzen, zum Teil ähnliche Ziele verfolgen. In Kapitel 9 werden verwandten Arbeiten vorgestellt und gezeigt, wie sich diese Arbeit davon abgrenzt.

Kapitel 10 bildet schließlich den Abschluss dieser Arbeit. Es greift diese Einleitung auf und zeigt, wie die hier beschriebenen Ergebnisse tatsächlich umgesetzt worden sind. Dabei wird auch zusammengefasst, wie die einzelnen Teile dieser Arbeit den Stand der Forschung erweitern und so einen Beitrag zum Wissen über Requirements Engineering und der Anwendung von Erfahrungsmanagement im Requirements Engineering leistet. Die Arbeit schließt mit einer Übersicht über interessante Forschungsfragen, die sich aus diesem Beitrag ableiten.

2 Grundlagen: Dokumentation und Erfahrung

Weder im Requirements Engineering noch im Erfahrungsmanagement gibt es einen einheitlichen Sprachgebrauch. Vielmehr existieren verschiedene „Schulen“, von denen selbst zentrale Begriffe widersprüchlich verwendet werden. Als Grundlage für diese Arbeit liefert dieses Kapitel zwei Dinge:

- *Begriffsklärung.* Die in der Arbeit verwendeten Begriffe werden erläutert. Wenn unterschiedliche Interpretationen eines Begriffs bekannt sind, wird darauf hingewiesen.
- *Standortbestimmung.* Dieses Kapitel zeigt den Stand der Forschung und der Praxis, auf den diese Arbeit aufbaut. Neuere Ansätze, die in Methodik oder Zielsetzung vergleichbar sind, werden dagegen in Kapitel 9 mit den Ergebnissen dieser Arbeit verglichen.

Um als Grundlage für die weitere Arbeit zu dienen, unterteilt sich dieses Kapitel in drei Bereiche: In Abschnitt 2.1 wird das Spannungsfeld zwischen formaler Modellierung und natürlichsprachlicher Dokumentation von Anforderungen betrachtet. Dies dient nicht nur zur Motivation von heuristischem Feedback, sondern zeigt auch die verschiedenen Ziele, die das Requirements Engineering verfolgt. Wichtige Grundlagen für diese Arbeit sind darüberhinaus Regeln und Richtlinien zur Dokumentation von Anforderungen sowie Konzepte und Metriken, um die Qualität von Anforderungen zu bestimmen.

Software Engineering beschäftigt sich mit der systematischen Erstellung von Software nach Ingenieursprinzipien. Eine Grundlage dafür ist, dass die wesentlichen Aspekte der Software Erstellung messbar sind. Dies gilt insbesondere, wenn Verbesserungen am Erstellungsprozess erreicht werden sollen. Erst, wenn man den Effekt einer Verbesserung objektiv messen kann, ist die Grundlage für eine systematisch lernende Software erstellende Organisation wirklich gegeben. In Abschnitt 2.2 werden die Grundlagen lernender Organisationen dargestellt. Diese Grundlagen liegen bereits auf der Schnittstelle zwischen Software Engineering und Erfahrungsmanagement. Der Abschnitt erläutert Grundlagen des Lernens einer Organisation während der Durchführung eines Projekts. Es zeigt sich, dass viele Erfahrungen nicht einfach weitergegeben werden können. Dazu ist es notwendig den Software Entwickler kurz aus seiner täglichen Routine zu reißen und ihm die Möglichkeit zur Reflexion zu geben.

Das Kapitel schließt mit einer kurzen Zusammenfassung in Abschnitt 2.3. Hier werden die wichtigsten Erkenntnisse aus den Grundlagen dieser Arbeit mit ihrem Bezug zu den folgenden Kapiteln dargestellt.

2.1 Dokumentation von Anforderungen

Dieser Abschnitt klärt im Rahmen dieser Arbeit die grundlegende Begriffswelt zu Dokumentation von Anforderungen. In Abschnitt 2.1.1 wird zunächst gezeigt, wie sich die Dokumentation von Anforderungen im Rahmen des Requirements Engineering einordnet. Nachdem geklärt ist, woher die zu dokumentierenden Anforderungen kommen und wozu die Dokumentation der Anforderungen dient, wird auf verschiedene Konzepte zur Unterstützung von Dokumentation eingegangen. Abschnitt 2.1.2 gibt einen Überblick über Vorschläge zur Strukturierung von Anforderungsdokumenten. Abschnitt 2.1.3 stellt Dokumentations- und Formulierungsregeln vor.

Grundsätzlich hält sich diese Arbeit an die folgende Definition einer Anforderung aus dem IEEE Standard Glossary of Software Engineering Terminology:

Definition 1: ***Anforderung (engl: Requirement)***

1. Eine Bedingung oder Fähigkeit die von einem Nutzer benötigt wird, um ein Problem zu lösen oder ein Ziel zu erreichen.
2. Eine Bedingung oder Fähigkeit die ein System oder eine Komponente erfüllen oder besitzen muss, um einen Vertrag, einen Standard, eine Spezifikation oder ein anderes formales Dokument zu erfüllen.
3. Eine dokumentierte Repräsentation, Bedingung oder Fähigkeit nach (1) oder (2).

(aus IEEE Standard Glossary of SE Terminology [1, Seite 62])

Die zweite grundsätzliche Definition aus der selben Quelle betrifft das zentrale Anforderungsdokument, die Spezifikation:

Definition 2: ***Anforderungsspezifikation (engl: Requirements Specification)***

Ein Dokument, das die Anforderungen für ein System oder eine Komponente spezifiziert. Ein solches Dokument beinhaltet typischerweise funktionale Anforderungen, Leistungsanforderungen, Schnittstellenanforderungen, Entwurfsanforderungen und Entwicklungsstandards.

(aus IEEE Standard Glossary of SE Terminology [1, Seite 69])

2.1.1 Einordnung: Dokumentation von Anforderungen

Die Dokumentation von Anforderungen ist eine der zentralen Aktivitäten des Requirements Engineering. Dieser Abschnitt zeigt, wie die Dokumentation mit anderen Aktivitäten des Requirements Engineering zusammenhängt.

Einen guten Überblick über den grundlegenden Kontext von Anforderungsdokumentation gibt Pohl in seinem Buch [131]. Pohl beschreibt Requirements Engineering durch die drei Dimensionen der *Dokumentation*, der *Übereinstimmung* und des *Inhalts* (vergleiche auch [130]). Ziel des Requirements Engineering ist es, in diesen drei Dimensionen die notwendige Qualität zu erreichen:

- *Inhaltsdimension.* Ziel der Inhaltsdimension ist es, alle relevanten Anforderungen in dem erforderlichen Detaillierungsgrad zu kennen und zu verstehen.
- *Übereinstimmungsdimension.* Ziel der Übereinstimmungsdimension ist die Etablierung einer ausreichenden Übereinstimmung über die bekannten Anforderungen.
- *Dokumentationsdimension.* Ziel dieser Dimension ist die Dokumentation und Spezifikation aller Anforderungen konform zu den vorgegebenen Dokumentations- und Spezifikationsvorschriften.

Anforderungsdokumente müssen die Kenntnis und das Verständnis aller relevanten Anforderungen wiederspiegeln, also die Inhaltsdimension. Zudem müssen die dokumentierten Anforderungen auch Konsens der Projektbeteiligten sein. Laut Pohl ist dies Aufgabe der Dokumentationsvorschriften:

- *Allgemeine Dokumentationsvorschriften* geben Struktur und Inhalt von Dokumenten vor, die im Requirements Engineering relevant sind.
- *Dokumentations- und Spezifikationsvorschriften für Anforderungen* legen fest, wie Anforderungen abhängig vom jeweiligen Verwendungszweck dokumentiert werden sollen.

Diese Vorgabe hilft dabei, inhaltlich keine wesentlichen Teile zu vernachlässigen. Auch die Übereinstimmungsdimension kann unterstützt werden, indem beispielsweise vorgeschrieben wird, wie Konflikte zu dokumentieren sind.

Verschiedene Rahmenwerke und Referenzmodelle geben einen Überblick über die Aktivitäten, die zur Erreichung einer hohen Qualität in allen drei Dimensionen erforderlich sind (z.B. [46, 53, 142, 131]). Die folgenden Definitionen (basierend auf dem Dagstuhl Seminar 'Requirements Engineering', 1998 [46]) gelten für diese Arbeit:

Definition 3: *Requirements Engineering*

Requirements Engineering ist der ingenieursmäßige Ansatz, Anforderungen zu analysieren (*Requirements Analysis*) und zu verwalten (*Requirements Management*).

(aus Dagstuhl Seminar 'Requirements Engineering', 1998 [46])

Für diese prinzipielle Unterscheidung in Analyse und Verwaltung von Anforderungen gibt es verschiedene Gründe. Unter anderem sind Kontext und notwendige Fähigkeiten recht unterschiedlich: ein guter Analyst muss nicht unbedingt ein guter Requirements Manager sein.

Definition 4: *Requirements Management*

Das Requirements Management (deutsch: Anforderungsmanagement) beinhaltet die Verwaltung von Anforderungen und zugehöriger Informationen, mit dem Ziel, Änderungen an den Anforderungen zu unterstützen und Verfolgbarkeit zu gewährleisten.

Change Management (deutsch: Änderungsmanagement) ist der systematische Umgang mit Änderungenswünschen, die systematische Verwaltung von Versionen und die organisierte Weitergabe von Änderungen.

Tracing (deutsch: Nach- oder Rückverfolgung) ist die Zuordnung von Annahmen, Entscheidungen und Zusammenhängen zu Anforderungen.

(aus Dagstuhl Seminar 'Requirements Engineering', 1998 [46])

Das Requirements Management steht nicht im Fokus dieser Arbeit, da in der Regel bereits dokumentierte Anforderungen verwaltet werden. Gut dokumentierte Anforderungen erleichtern die Aufgaben des Requirements Management. Daher können und sollen die Konzepte dieser Arbeit genutzt werden, um Erfahrungen und Feedback aus dem Requirements Management in die Requirements Analysis einfließen zu lassen.

Definition 5: *Anforderungsanalyse (engl: Requirements Analysis)*

Die Anforderungsanalyse dient dazu, Anforderungen in einem Projekt

1. zu ermitteln und abzustimmen,
2. zu formulieren und dokumentieren,
3. zu validieren.

(aus Dagstuhl Seminar 'Requirements Engineering', 1998 [46])

Bis Anforderungen am Ende der Anforderungsanalyse dokumentiert sind, sind eine Reihe von Aktivitäten nötig. Die Aktivitäten der Requirements Analysis lassen sich nach ihrer Zielsetzung gruppieren.

Elicitation. Stakeholder sowie weitere Quellen für Anforderungen müssen identifiziert werden. Zudem müssen die Rohanforderungen dieser Quellen erhoben werden.

Interpretation. Auf Basis der Rohanforderungen müssen echte Anforderungen identifiziert werden. Rohanforderungen müssen dazu strukturiert, klassifiziert, konkretisiert und zueinander in Verbindung gesetzt werden.

Negotiation. In den interpretierten Anforderungen müssen Abhängigkeiten und Inkonsistenzen gelöst sowie Prioritäten gesetzt werden, in der Regel durch Verhandlung mit den Stakeholdern der Anforderungen.

Documentation. Anforderungen müssen fixiert werden, auch um Zwischenstände und Annahmen festzuhalten.

Verification und Validation. Die dokumentierten Anforderungen müssen formal und inhaltlich geprüft werden, ob sie die Vorgaben und Kundenwünsche aus vorherigen Dokumenten korrekt beschreiben.

Aktivitäten dieser Gruppen werden in der Regel nicht in einer festgeschriebenen Reihenfolge durchgeführt. Dokumentation steht mit allen anderen Aktivitäten in einem direkten Zusammenhang. Angaben über die Quellen und die Rohanforderungen aus der Elicitation werden dokumentiert. Die Resultate der Interpretation spiegeln sich in der Dokumentation wieder. Häufig werden Widersprüche erst erkennbar, wenn die Anforderungen in ausreichender Qualität dokumentiert werden. Die Verifikation und Validierung schließlich ist ebenfalls erst auf Basis einer belastbaren Dokumentation möglich.

2.1.2 Modelle und natürlichsprachliche Anforderungsdokumente

Typen von Anforderungsdokumenten

Es gibt zwei gängige Arten, Anforderungen zu dokumentieren: (grafische) Modelle oder natürlichsprachliche Dokumente. Unter den grafische Modellen zur Dokumentation von Anforderungen sind neben vielen anderen die UML [124, 125] (vor allem Use-Case-Diagramme) und i* [175] bekannt.

Nach Glinz sind UML Use-Case-Diagramme für die Dokumentation von Anforderungen alleine nicht ausreichend [75]. Es fehlen vor allem Möglichkeiten, die Interaktion zwischen verschiedenen Use-Cases darzustellen, den Kontext des Systems ausreichend zu beschreiben und Use-Cases unterschiedlicher Abstraktionsebenen hierarchisch zu strukturieren.

Im Gegensatz zu den UML Use-Case-Diagrammen erlaubt es die i* Methode, unterschiedliche (auch widersprüchliche) Arten von Zielen zu dokumentieren. Im Gegensatz zu UML Use-Case-Diagrammen versucht man bei i* nicht, einen leichten Übergang zu Architektur und Entwurf zu schaffen: Die Methode hat besonders die frühen Phasen des Requirements Engineering im Fokus. Daher empfehlen Santander und Castro Anforderungen zwar mit Hilfe von i* Modellen zu erheben, aber zum Zweck der Dokumentation Use-Cases abzuleiten [144].

Von der Verwendung grafischer Modelle im Requirements Engineering verspricht man sich einen leichten Übergang in die Architektur und das Design. Dies ist vor allem durch UML Diagramme zu erreichen. Durchgängige Methoden dazu werden immer wieder vorgeschlagen (zum Beispiel von Maciaszek [116]). Die von Glinz beschriebenen Probleme sind aber auch mit UML 2.0 und bei Verwendung verschiedener Modellarten nicht von der Hand zu weisen [75]. Zudem muss man zunächst genügend Informationen über die Anforderungen zusammentragen, um diese in ausreichender Qualität modellieren zu können. Je nach Domäne sind die Stakeholder zudem nicht in der Lage, die dabei entstehenden komplexen Modelle zu verstehen. Dies ist jedoch auch deshalb wichtig, da Stakeholder nur so auf Missverständnisse hinweisen können. Ambriola und Gervasi leiten daraus den Bedarf ab, UML Modelle automatisch aus natürlichsprachlichen Anforderungen zu extrahieren [7]. In diesem Kontext zielt diese Arbeit darauf ab, die Anforderungen bis zu diesem Punkt in einen guten Zustand zu bringen. Glinz empfiehlt Szenarien als primäres Anforderungsartefakt [74], da diese von den Stakeholdern des Systems leicht verstanden werden können und bereits ein sinnvolles Subsystem beschreiben, bevor alle Anforderungen vollständig erhoben wurden.

Aus diesem und anderen Gründen spielt die Dokumentation von Anforderungen mit natürlicher Sprache in der Industrie eine große Rolle [117, 77]. Ein wichtiges Problem, das beim Umgang mit natürlichsprachlichen Anforderungen (insbesondere auch bei Szenarien, [74]) gelöst werden muss, ist die Konsistenz verschiedener Teilbeschreibungen. Die Konsistenzprüfung wird auch dadurch erschwert, dass die Anforderungsbeschreibungen in den frühen Phasen eines Projekts noch stark im Fluss sind. Zudem sind natürlichsprachliche Anforderungen nicht so leicht automatisch auswertbar wie formale Modelle [73].

Strukturierung durch Vorlagen

Eines der wichtigsten Hilfsmittel bei der Dokumentation von Anforderungen sind Vorlagen (englisch: Templates). Vorlagen helfen bei der Strukturierung von Anforderungen auf verschiedenen Ebenen:

(i) Vorlagen für Anforderungsspezifikationen strukturieren ein Dokument nach verschiedenen Typen von Anforderungen (siehe Definition 2). Beispiele hierfür

sind die Vorlagen für Lasten- und Pflichtenhefte des V-Modell XTs [81], die Volere-Vorlage [139] und der IEEE Standard 1998 zur Spezifikation von Anforderungen [3].

(ii) Vorlagen für spezielle Teile einer Anforderungsspezifikation wie Szenarien, Use-Cases und Qualitätsmodelle helfen dabei, bestimmte Teile der Anforderungsspezifikation zu strukturieren. Besondere Aufmerksamkeit kommt im Rahmen dieser Arbeit den Use-Case-Vorlagen in Tabelle 2.1 nach Cockburn zu (siehe Kapitel 3) [42].

(iii) Vorlagen für einzelne Anforderungen helfen dabei, diese klar und eindeutig zu formulieren. Bekannt ist die Volere-Vorlage für Anforderungsspezifikationen [139] und die Anforderungsschablone von Rupp [142]. Letztere kann auch als Formulierungsregel für die Dokumentation von Anforderungen aufgefasst werden, da Rupps Schablone stark in die Formulierung und den Satzbau eingreift und so bestimmte Arten von Mehrdeutigkeiten oder Auslassungen unterbindet.

Vorlagen sind auch deshalb eine große Hilfe, weil sie als eine Art Checkliste dienen. Sie spiegeln damit die Erfahrung ihrer Autoren wieder. Wurden alle vorgegebenen Bereiche oder Felder einer Vorlage ausgefüllt, ist sichergestellt, dass nichts vergessen wurde, was dem Autor der Vorlage wichtig erschien. Durch die systematische Anpassung (englisch: Tailoring) solcher Vorlagen kann auch projekt- und organisationsspezifische Erfahrung mit einfließen. Dies ist ein sehr praktikabler Weg, die Dokumentation von Anforderungen auf Basis von Erfahrungen zu verbessern, der in Vorarbeiten zu dieser Arbeit auch untersucht worden ist [102].

2.1.3 Regeln zur Dokumentation von Anforderungen

Die Strukturierung von Anforderungen mit Vorlagen gibt dem Analysten mehr Sicherheit. Im Rahmen von Untersuchungen der Anforderungsspezifikationen studentischer Softwareprojekte als Grundlage für diese Arbeit wurde festgestellt, dass gut strukturierte Vorlagen die Qualität und Vollständigkeit von Anforderungen stark erhöhen [31, 94] (siehe auch Abschnitt 4.3). So lassen sich in Use-Case-Beschreibungen viel weniger sprachliche Fehler finden, da die vorgegebene Vorlage viele ungenaue grammatikalische Konstrukte nicht zulässt (zum Beispiel ist für Schritte des Hauptszenarios ein Feld für den Akteur und ein Schritt für die Aktion vorgesehen. Dadurch sind sprachliche Konstrukte, die das Subjekt auslassen, deutlich erschwert).

Häufig wird eine Vorlage erst dadurch richtig wertvoll, dass es pragmatische Hinweise gibt, wie Anforderungen damit zu strukturieren sind. So schlägt Cockburn in seinem Buch nicht nur eine ganze Serie von Use-Case-Vorlagen für verschiedene Situationen vor, sondern gibt auch eine Vielzahl von Tipps und Tricks, wie man

Tabelle 2.1: Use-Case-Vorlage nach Cockburn [42].

Feld	*Beschreibung*
Use-Case ID	Eindeutiger Bezeichner für den Use-Case, erleichtert die Verwaltung von Use-Cases.
Titel	Als Titel sollte ein kurzer aktiver Satz gewählt werden, der das Ziel des Use-Cases ankündigt.
Erläuterung	Die Erläuterung beschreibt den Use-Case ausführlicher als der Titel.
Umfang / Systemgrenzen	Der Umfang grenzt den Use-Case ab. Welcher Teil des zu erstellenden Systems gehört noch zu dem Use-Case, welcher Teil liegt außerhalb?
Ebene	Die Ebene ordnet den Use-Case nach seiner Abstraktionsebene in Überblick, Hauptziel und Unterfunktion ein.
Vorbedingung	Die Vorbedingung beschreibt, in welchem Zustand das zu entwickelnde System sein muss, bevor dieser Use-Case ausgeführt werden kann.
Mindestgarantie	Mit der Mindestgarantie können Bedingungen definiert werden, die auch beim Fehlschlag des Use-Cases eingehalten werden müssen.
Erfolgsgarantie	Die Erfolgsgarantie beschreibt den Zustand des Systems, nachdem der Use-Case erfolgreich durchgeführt worden ist.
Stakeholder und Interessen	Bei den Stakeholdern werden alle Rollen und Personen aufgeführt, die ein Interesse an dem Use-Case haben.
Hauptakteur	Hauptakteur ist die Rolle oder Person, deren Ziel durch den Use-Case unterstützt werden soll.
Auslöser	Auslöser ist eine Bedingung, die zum Start des Use-Cases führt.
Hauptszenario	Eine Liste der Schritte, die zur Durchführung des Use-Cases notwendig sind.
Erweiterungen	Erweiterungen sind alternative Szenarien. Zu jeder Erweiterung sollte neben der Liste der Schritte angegeben werden, welcher Schritt des Hauptszenarios unter welcher Bedingung erweitert wird.
Technische Variationen	Je nach technischer Realisierung kann ein Use-Case sehr unterschiedliche Szenarien haben. Solche alternativen Verläufe können als technische Variation dargestellt werden.
Weitere	Die Vorlage kann um weitere Felder ergänzt werden (zum Beispiel offene Punkte, Priorität, Verwendungshäufigkeit, Autor, Implementierungsideen usw.)

mit diesen Vorlagen *gute* Use-Cases schreibt [42]. Diese Tipps und Tricks werden von Adolph, Bramble, Cockburn und Pols noch mit weiteren Patterns und Best Practices abgerundet [5].

Darüber hinaus gibt es aber auch für unstrukturierte Anforderungen Anleitungen und Regeln, mit deren Hilfe bekannte Fehler vermieden werden können. Die bereits erwähnte Anforderungsschablone von Rupp [142] erlaubt es beispielsweise nicht, bei funktionalen Anforderungen wichtige Details auszulassen. Da es durch die Schablone aber außerdem so gut wie unmöglich ist, grammatikalisch korrekte Sätze zu schreiben, wird sie oft nicht angewendet.

Weniger in die sprachliche Gestaltung eingreifend sind Vorschläge zur Formulierung von Anforderungen. So hat sich herausgestellt, dass Plural in Anforderungen fast immer problematisch ist, da so unbeabsichtigt verallgemeinert wird (vergleiche Berry und Kamsties [25]). Dies kann bereits von Grund auf unzulässig sein oder aber beim Leser falsch interpretiert werden. Wenn möglich sollte also bei der Formulierung auf Plural verzichtet werden. Mit ihrem Handbuch zur Mehrdeutigkeit [27] geben Berry, Kamsties und Krieger eine hervorragende Sammlung dieser und ähnlicher vermeidbarer Mehrdeutigkeiten.

Viele dort erwähnten Ursachen von Mehrdeutigkeiten sind in der Forschung weiter untersucht worden. Zum Beispiel ist die Verknüpfung zweier Teilsätze durch *und* / *oder* ebenfalls problematisch, da sie laut Chantree et al. in den meisten Fällen vom Leser auf verschiedene Weise dekodiert werden können [39].

Es existiert also eine Vielzahl von Regeln für die Dokumentation von Anforderungen. Es scheint praktisch unmöglich zu sein, all diese Regeln immer zu berücksichtigen. Daher beschäftigen sich immer mehr Ansätze damit, die Einhaltung von Regeln automatisch zu prüfen (z.B. [59, 118, 39, 90, 18]). Diese Ansätze stellen verwandte Arbeiten dar und werden in Kapitel 9 als verwandte Arbeiten genauer betrachtet.

2.1.4 Qualität von Anforderungsdokumenten

Definition 6: ***Metrik für Anforderungsdokumentation***

Eine Metrik für Anforderungsdokumentation, oder kurz Metrik, ist eine (meist mathematische) Funktion, die eine Eigenschaft von Software in einen Zahlenwert, auch Maßzahl genannt, abbildet. Hierdurch werden formale Vergleichs- und Bewertungsmöglichkeiten geschaffen.

Qualitätsaspekte und Metriken für Anforderungen werden in der Literatur umfassend diskutiert. Über die Schwierigkeiten, wie sie beispielsweise von Costello

beschrieben werden [44], herrscht dabei weitgehend Einigkeit. Demnach ist es im Kontext von Anforderungen besonders schwierig, objektive Messdaten zu erlangen. So ist es beispielsweise schwer festzustellen, ob eine Anforderungsspezifikation vollständig ist oder Lücken enthält. Ohne die ursprünglichen Stakeholder, von denen die Anforderungen stammen, kann nicht nachvollzogen werden, ob alle Anforderungen enthalten sind.

Eine ganze Reihe von Lehrbüchern beschreibt, wie qualitativ hochwertige Anforderungsdokumentation erstellt werden kann [48, 70, 139, 142, 27, 26]. In ihrem „Handbuch der Mehrdeutigkeiten“ geben Berry und Kamsties einen Überblick über Typen von Mehrdeutigkeiten [27, 26]. Für verschiedene Typen von Mehrdeutigkeiten geben sie mehr oder weniger effektive Methoden an, um diese zu vermeiden oder aus einer Anforderungsspezifikation zu entfernen. So können durch Homonyme[1] und Synonyme[2] Missverständnisse verursacht werden. Eine mögliche Gegenmaßnahme sind Glossare, deren Effektivität vor allem darunter leidet, dass missverständliche Homonyme und Synonyme zunächst entdeckt werden müssen.

Das Quality Gateway der Robertsons [139] ist ein prozessorientiertes Beispiel um hohe Qualität von Anforderungen zu erstellen. Nur *gute* Anforderungen können es passieren und damit Gültigkeit für die folgenden Phasen der Lösungserstellung erlangen. *Gute* Anforderungen erfüllen eine Reihe von Qualitätskriterien. So dürfen zum Beispiel nur Anforderungen das Quality Gateway passieren, für die ein Erfüllbarkeitskriterium gegeben ist. Derartige konstruktive und prozessgetriebene Ansätze sind wichtig, um qualitativ hochwertige Anforderungsspezifikationen zu erstellen. Im Rahmen des Nachweises der Relevanz guter Anforderungsdokumentation ist dieses Vorgehen jedoch ungeeignet, da es keinen direkten Vergleich der Qualität von Anforderungen verschiedener Projekte erlaubt.

Im Gegensatz zu diesen konstruktiven Ansätzen vertritt Rupp [142] einen eher analytischen Ansatz, um gute Qualität von Anforderungen zu erreichen. Dabei werden bekannte Qualitätsaspekte untersucht, wie zum Beispiel die Vollständigkeit. Die Robertsons [139] definieren Vollständigkeit über den Grad zu dem Anforderungsformulare (wie z.B. Use-Cases) ausgefüllt wurden. Im Gegensatz dazu zeigt Rupp, wie unvollständige Anforderungen auf Basis der Analyse natürlicher Sprache gefunden werden können. Sie bedient sich dabei der neurolinguistischen Programmierung (NLP). Beide Ansätze helfen, die Qualität von Anforderungsspezifikationen zu verbessern, können aber nicht direkt verwendet werden, um diese quantitativ zu vergleichen.

Glinz gewichtet die klassischen Qualitätsaspekte für Anforderungen neu [74]. Demnach seien die Qualitätsaspekte Eindeutigkeit und Vollständigkeit kaum erreichbar und ihr Erreichen kaum überprüfbar. Wichtiger sei es, angemessene, konsistente, leicht änderbare und überprüfbare Anforderungsdokumente zu erstellen.

[1] gleiche lexikographische Begriffe mit unterschiedlicher Semantik
[2] unterschiedliche lexikographische Begriffe mit gleicher Semantik

Davis [48] macht konkrete Vorschläge zur Quantifizierung der Qualität eines Anforderungsdokuments. Befunde werden gezählt und auf Basis ihrer Kritikalität gewichtet. Die Qualität einer Anforderungsspezifikation ist damit gegeben durch die Anzahl der Fehler multipliziert mit dem Gewicht der Fehlerklasse. Je niedriger dieser Wert ist, um so besser ist die Spezifikation.

Neben den sehr grundlegenden Qualitätsmetriken, die bisher diskutiert worden sind, gibt es auch eine Reihe von fortgeschrittenen Maßnahmen. So können beispielsweise die in Abschnitt 2.1.3 beschriebenen Verfahren, die die Einhaltung von Formulierungsregeln automatisch prüfen, ebenfalls verwendet werden, um die Qualität von Dokumenten zu bestimmen.

2.2 Erfahrungen im Software Engineering nutzen

Nach Schneider sind im Software Engineering Erfahrungen und Wissen ein kritischer Erfolgsfaktor. Sie stellen eine wichtige Grundlage dar, um Kompromisse und Entscheidungen unter Ungewissheit zu treffen [151]. Demnach gibt es im Software Engineering häufig keine Möglichkeit, vorab zu wissen, welches die optimale Entscheidung in einer Situation ist. Beispielsweise bieten moderne Softwareentwicklungsprozesse in der Regel einen Mechanismus zur Anpassung der verwendeten Dokumente auf den Projektkontext [81] (so genanntes *Tailoring*). In Vorarbeiten zu dieser Dissertation hat sich jedoch gezeigt, dass diese Aufgabe oft vermieden wird: Projektmitglieder sind beim Anpassen oft überfordert, weil sie die Auswirkungen von Tailoring-Entscheidungen nicht einschätzen können [102]. Um eine Entscheidungsgrundlage zu schaffen, könnten die Auswirkungen von Tailoring-Entscheidungen durch sehr aufwendige Experimente nachgewiesen werden. Die Ergebnisse sind dann aber oft nicht verallgemeinerbar. Alternativ kann Wissen und Erfahrung einer Organisation in einer solchen spezifischen Situation helfen, schneller und besser zu (re)agieren. So kann beispielsweise eine Expertenmeinung herangezogen werden. Experten, die bereits Erfahrung mit dem Anpassen von Softwareentwicklungsprozessen haben sind jedoch selten. Eine Software erstellende Organisation kann alternativ auch versuchen, die notwendige Erfahrung selbst zusammenzutragen. Hier kann nur der systematische Umgang mit Erfahrungen helfen.

Der bekannteste Ansatz zur Unterstützung dieses schwierigen Lernprozesses ist die *Experience Factory* von Basili [13, 11]. Dazu wird eine einheitliche Wissensmanagement-Infrastruktur in die Entwicklungsprojekte einer Software erstellenden Organisation eingeführt. Erfahrungen aus dem einen Projekt können direkt in anderen Projekten angewandt werden, die vor ähnlichen Problemen stehen. Wesentlich sind dafür 3 Schritte:

1. Feststellen, dass ein Problem vorliegt.
2. Erkennen, dass dieses Problem bekannt ist und Erfahrungen dazu existieren.
3. Anwenden einer vorhandenen Erfahrung. Dies kann wiederum zu neuen Erfahrungen mit dem Problem führen, die möglicherweise in die Erfahrungsbasis zurückgeführt werden sollten.

An dieser Stelle setzt diese Arbeit auf: Heuristische Kritiken können helfen, mögliche Probleme zu identifizieren (1) und mit einer vorhandenen Erfahrung zu verknüpfen (2). Je nach Einbettung der Heuristiken können neue Erfahrungen (3) implizit durch Beobachtung des Nutzers oder explizit durch Feedback des Nutzers gewonnen werden. Kapitel 5 zeigt, wie dies konzeptionell erreicht werden kann.

2.2.1 Grundlagen des organisatorischen Lernens

Nach Schneider ist es bei organisatorischem Lernen wichtig, zwischen Wissen und Erfahrung zu unterscheiden [151]. Die von Schneider verwendete Definition von Erfahrung erfüllt wichtige pragmatische und für diese Arbeit nützliche Anforderungen:

Definition 7: *Erfahrung*

Erfahrung ist als 3-Tupel definiert und besteht aus:

- einer Beobachtung
- einer Emotion (bezüglich der Beobachtung)
- einer Schlussfolgerung oder Hypothese (abgeleitet aus der Beobachtung und der zugehörigen Emotion).

(aus Schneider, Experience and Knowledge Management [151])

Erfahrung ist also eine spezielle Form von Wissen, die aus der täglichen Arbeit abgeleitet wird. Die Trennung von Beobachtung und Schlussfolgerung erleichtert die Verwaltung und Überarbeitung. Die explizite Nennung der Emotion hilft, den Kontext einer Erfahrung besser beurteilen zu können. Im Gegensatz zu Wissen wird von Erfahrungen nicht gefordert, dass sie allgemein gültig und richtig sind. Daher ist im Umgang mit Erfahrung Vorsicht angebracht.

Eine Software erstellende Organisation, die systematisch mit Erfahrungen umgeht, nennt man *lernende Software erstellende Organisation* (LSO) [168]. Sie definiert sich nach Schneider über die Mechanismen, mit denen eine Organisation Wissen und Erfahrungen *lernen* kann [147, 151]:

Definition 8: *Lernende Software erstellende Organisation (LSO)*

Eine lernende Software erstellende Organisation (LSO) pflegt die folgenden Aspekte organisatorischen Lernens, um die Erstellung von Software zu optimieren:

1. Lernen der Individuen der Organisation
2. Organisationsweite Sammlung von Wissen und Erfahrung
3. Kultivieren einer organisationsweiten Infrastruktur für den Austausch von Wissen und Erfahrungen.

(nach Schneider et al. [147, 151])

LSO haben also einen Erfahrungskreislauf etabliert: Vorhandene Erfahrungen werden genutzt und neue Erfahrungen in das vorhandene Wissen integriert. Im Rahmen dieser Arbeit liegt der Fokus auf den Aspekten, die eine LSO im Rahmen des Requirements Engineering dabei betrachtet. Eine LSO ist in der Lage, die Qualität der Anforderungsspezifikation von Projekt zu Projekt zu verbessern. Dabei lohnt es sich, in diese kontinuierliche Verbesserung zu investieren, weil sich die Qualität der in den Projekten erstellten Softwareprodukte gemessen an der Kundenzufriedenheit erhöht (vergleiche Abschnitt 4.3).

Einer LSO ist also der systematische Umgang mit Erfahrungen und Wissen wichtig. Die damit verbundene Tätigkeit nennt man nach Schneider [151] Wissensmanagement.

Definition 9: *Wissensmanagement*

Wenn Erfahrung und Wissen systematisch erzeugt, evaluiert, verwaltet, überarbeitet, verteilt und angewendet wird, um ein Problem zu lösen, wird dies als Wissensmanagement bezeichnet.

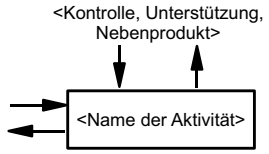
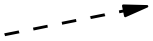
(aus Schneider, Experience and Knowledge Management [151])

Im Rahmen dieser Arbeit liegt der Fokus auf dem Umgang mit Erfahrung:

Definition 10: *Erfahrungsmanagement*

Teil des Wissensmanagement, in dem Erfahrung systematisch erzeugt, evaluiert, verwaltet, überarbeitet, verteilt und angewendet wird, um ein Problem zu lösen.

Tabelle 2.2: FLOW-Syntax nach Schneider et al. [152].

Aggregat-zustand	*Informations-speicher*	*Informations-fluß*	*Erfahrungs-fluß*	*Aktivität*
fest	 <Dokumentname>	 <Informationstyp> (optional)	 <Erfahrung> (optional)	 <Kontrolle, Unterstützung, Nebenprodukt> <Name der Aktivität> <Ein- / Ausgehende Information>
flüssig	 <Person>	 <Informationstyp> (optional)	 <Erfahrung> (optional)	

2.2.2 Erfahrungsflüsse

Diese Arbeit ist stark von den Schwerpunkten der Forschungsgruppe beeinflusst, in der sie durchgeführt wurde. Einige dieser Einflüsse sind so fundamental für die Arbeit, dass sie ebenfalls in den Grundlagen erläutert werden müssen. Das gilt besonders für die FLOW-Methode [152]. Die Informationsflussmodelle aus FLOW sind wesentlich, um wichtige Aspekte des Einsatzes von Heuristiken im Requirements Engineering zu modellieren. Andersherum haben sich Heuristiken im Kontext dieser Arbeit als hilfreich erwiesen, um kritische Informationsflüsse zu unterstützen (vergleiche [161, 79]).

Die FLOW-Notation zeichnet aus, dass sie die Unterscheidung verschiedener Arten von Informationsflüssen erlaubt. Zum Einen kann zwischen informellen und dokumentbasierten Informationsflüssen unterschieden werden. Diese werden nach der FLOW-Metapher der Aggregatzustände von Informationsflüssen als *flüssig* (informell) oder *fest* (dokumentbasiert) bezeichnet. Zum Anderen erlaubt es die Notation, zwischen Flüssen von für das Projekt relevanten Informationen und Flüssen von projektunabhängigen Erfahrungen zu unterscheiden. Tabelle 2.2 zeigt die Syntax dieser Notation.

Der Notation liegt die folgende Semantik zugrunde:

- Bezeichner beziehen sich auf Individuen oder Rollen, je nach Ziel des Modells.
- Flüsse werden durch Pfeile repräsentiert. Alle Flüsse, die aus einem festen Speicher (Dokument) kommen, sind selbst fest, alle Flüsse, die aus einem flüssigen Speicher (Smiley) kommen, sind flüssig.
- Beschriftungen der Pfeile können verwendet werden, um bestimmte Typen von Informationsflüssen auszuzeichnen. Anforderungen und daraus abgeleitete Informationen sind der Standardtyp und benötigen keine weitere Beschriftung.

- Aktivitäten werden als Rechteck mit einer Bezeichnung in der Mitte dargestellt.
  - Eine Aktivität verarbeitet eingehende Informationen zu ausgehenden Informationen. Diese Flüsse werden mit der linken oder rechten Seite des Rechtecks verbunden.
  - Flüsse, die aus einer Aktivität kommen, sind entweder fest oder flüssig: So kann in einem Modell eine Intention oder Annahme zu der Aktivität ausgedrückt werden.
  - Zudem kann die Aktivität durch zusätzliche Informationsflüsse gesteuert oder unterstützt werden (häufig Erfahrungen). Diese Flüsse werden mit der oberen oder unteren Seite verbunden.
  - Ausgehende Informationsflüsse, die mit der oberen oder unteren Seite einer Aktivität verbunden sind, bezeichnen Nebenprodukte der Aktivität.

Die Aktivität dient vor allem als Blackbox, durch die unnötige Details von Informationsflüssen versteckt werden können. Eine Aktivität kann durch verschiedene konkrete Informationsflüsse ersetzt werden, so genannte Implementierungen einer Aktivität. Diese Implementierungen müssen dann allerdings die FLOW-Schnittstelle der Aktivität einhalten, also genauso viele eingehende und ausgehende Informationsflüsse des gleichen Typs haben. Abbildung 2.1 illustriert diesen Mechanismus.

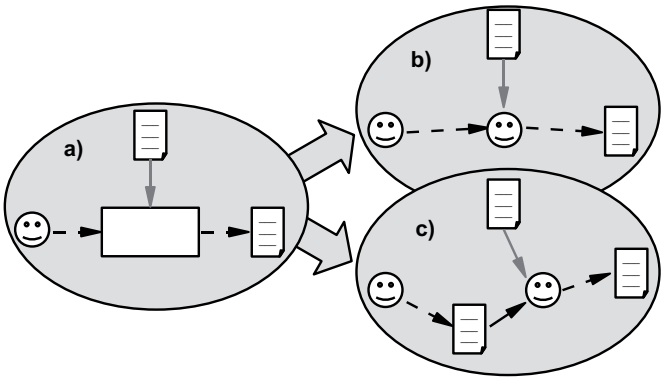


Abbildung 2.1: FLOW-Aktivität: a) als Blackbox und zwei gültige Implementierungen b) und c) der Aktivität.

2.2.3 Experience Factory

Mit der *Experience Factory* haben Basili, Caldiera und Rombach eine wichtige Grundlage für die systematische Nutzung von Erfahrungen in der Software-Entwicklung gelegt [13]. Die Experience Factory ist eine organisatorische Einheit, die ein Projekt beim Erfahrungsmanagement unterstützt:

1. Die Experience Factory analysiert ein Projekt und leitet Erfahrungen ab.
2. Die Experience Factory enthält einen Speicher für Erfahrungen (die Erfahrungsbasis).
3. Die Experience Factory stellt Erfahrungen verschiedenen Projekten zur Verfügung.

Ein wichtiger Beitrag der Arbeiten um die Experience Factory ist die Unterscheidung in Projekt-Aktivitäten und Aktivitäten des Erfahrungsmanagements. Dadurch ergeben sich konzeptionelle Schnittstellen zwischen dem Projektgeschäft und dem Erfahrungsmanagement.

Abbildung 2.2 gibt eine Übersicht über die Experience Factory. Zur Projektorganisation gehört die *Planung* und die *Durchführung* des Projekts. In der *Planung* wird das Projekt charakterisiert, es werden Ziele für das Projekt festgelegt und es wird geplant, wie diese Ziele erreicht werden sollen. Während der Planung übermittelt die Projektorganisation Charakteristika und Plandaten des Projekts und seiner Umgebung an die Erfahrungsbasis. Mit Hilfe dieser Daten kann die Experience Factory konkrete Erfahrungen zur Verfügung stellen, die in dem Projekt nützlich sein können. Als Beispiele nennen Basili et al. vor allem Prozessgrößen, die bei der Definition von Zielen hilfreich sind und die Planung erleichtern [13].

Basierend auf der Planung wird das Projekt dann *durchgeführt*. Während der *Durchführung* misst die Projektorganisation, in wie weit die gesetzten Ziele erreicht werden und greift gegebenenfalls steuernd ein. Die dabei erhobenen Daten werden der Experience Factory übermittelt und dienen dort der Analyse zur Ableitung neuer Erfahrungen. Nach Basili [12] ähnelt die Experience Factory Demings Plan-Do-Check-Act (PDCA) Paradigma [52]. Beim PDCA wird ein Produktionsverfahren schrittweise verbessert. Dazu wird zunächst ein *Plan* entwickelt, wie ein besseres Produkt gefertigt werden kann. Dieser Plan wird dann ausgeführt (*Do*). In der *Check*-Phase werden die Effekte des Plans erfasst, indem das Produkt mit den individuellen Qualitätsmetriken aus der Plan-Phase geprüft wird. Das Ergebnis dieser Prüfung wird in der *Act*-Phase analysiert, um Lehren aus dem Zyklus zu ziehen und den Plan für den nächsten Durchlauf zu fassen. Laut Basili geht die Experience Factory über diese Produktfokussierung des PDCA Paradigmas hinaus, da sie die Verbesserung von Entwicklungsprozessen (statt der Verbesserung von Produktionsprozessen) als Gegenstand hat [12]. Weiterhin un-

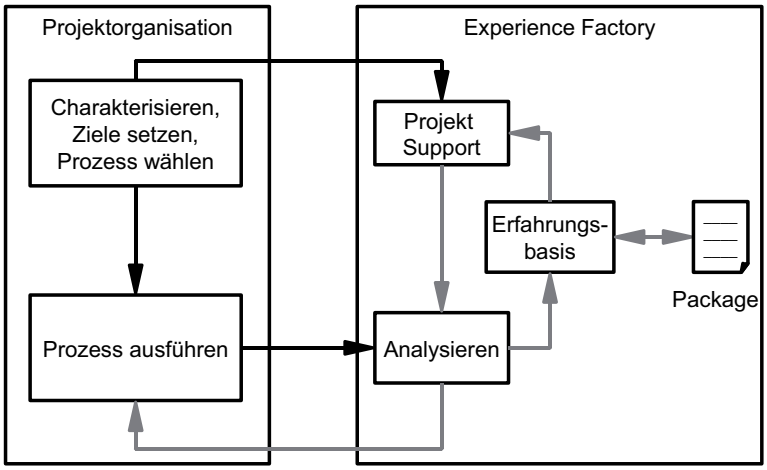


Abbildung 2.2: Informationsflüsse in der Experience Factory als FLOW-Modell, in Anlehnung an [13].

terstützt die Experience Factory typische Verbesserungs-Paradigmen wie TQM, Lean Software Development, CMM [12].

Eine Experience Factory kann auch in Teilprozessen von Projekten einer Projektorganisation eingesetzt werden. Dafür muss der Teilprozess sich sinnvoll in eine Planungs- und Durchführungsphase unterteilen lassen. Dies ist für den Unterprozess „Anforderungsdokumentation“ im Sinne dieser Arbeit gegeben: Planungsgegenstand sind Zielgruppe, Qualitätsziele und angestrebter Formalitätsgrad der Dokumentation. Erfahrungspakete, die der Planung und Durchführung zu Gute kommen, sind heuristische Kritiken.

Für die Erstellung einer erfahrungsbasierten und heuristischen Lösung zur Dokumentation von Anforderungen sind nach dem Experience Factory Paradigma die Informationsflüsse im Projekt, die Erfahrungsflüsse in der Experience Factory und die Schnittstelle zum Austausch von Erfahrungs- und Projektinformation zwischen beiden Systemen relevant.

Kern der Experience Factory ist die Erfahrungsbasis. Diese enthält Erfahrungen in der Form von *Packages* - formalisierte, verallgemeinerte und angepasste Darstellungsformen von Erfahrungen. Heuristische Kritiken können nach dieser Darstellung also als Experience Packages aufgefasst werden.

Im Rahmen des *Project Support* werden nach dem Experience Factory Paradigma für das Projekt nützliche Massgaben, Prozesse, Vorlagen oder Werkzeuge ausgewählt. Im Rahmen des engeren Fokus dieser Arbeit würde in diese Aktivität die Auswahl geeigneter erfahrungsbasierter Werkzeuge und Erfahrungen für ein Projekt fallen.

Ziel dieser Arbeit ist es, Erfahrungsmanagement und Projektdurchführung beim Requirements Engineering noch enger zusammen zu bringen, indem Werkzeuge für das Requirements Engineering durch Konzepte des Erfahrungsmanagements erweitert werden. Um die Dokumentation von Anforderungen zu unterstützen, liegt der Fokus daher auf der unteren Hälfte von Abbildung 2.2. Während der Durchführung werden Arbeitsergebnisse analysiert und gegebenenfalls mit Hilfe der Erfahrungsbasis Feedback an den Nutzer gegeben.

2.2.4 Konstruktives Feedback von Computern

Im Gegensatz zu eher technischen Disziplinen des Software und Systems Engineering unterliegt das Requirements Engineering ganz besonderen Rahmenbedingungen. Dies fängt bereits bei der Erhebung von Anforderungen an, wo die oft vagen und undokumentierten Wünsche und Erwartungen verschiedener Stakeholder berücksichtigt und mit bereits dokumentierten und eher offensichtlichen Anforderungen in Einklang gebracht werden. Diese Aufgabe bezieht ihre inhärente Problematik aus den unterschiedlichen Hintergründen, stillschweigenden Annahmen [132] und bevorzugten Gewohnheiten zur Kommunikation [114] der verschiedenen Stakeholdergruppen. Die Erfahrungen im Requirements Engineering zeigen zudem, dass Stakeholder oft nicht in der Lage sind, ihre Anforderungen zu formulieren [47]. Dies wird sogar noch schwieriger, wenn sich die Stakeholder über technische Realisierungsmöglichkeiten und deren Chancen und Gefahren nicht im Klaren sind.

Auf diese Weise entsteht eine Symmetrie der Unwissenheit [61]: Analysten wissen nicht, welche Anforderungen Stakeholder auf Grund stillschweigenden Wissens [132] nicht nennen, Stakeholder wissen nicht, welche Möglichkeiten eine technische Realisierung ihnen bieten kann.

Interessanterweise gibt es sogar Ansätze, genau diese Unwissenheit auszunutzen. Zum Beispiel beschreibt Berry, wie man mit gezieltem Einsatz von Unwissenheit im Interview auf diese schwer zu erreichende Punkte kommen kann [24]: Durch die Unwissenheit über die Domäne kann der Analyst eine vollständigere Anforderungsspezifikation erreichen, indem er scheinbare Selbstverständlichkeiten hinterfragt. Wichtig ist, dass diese Unwissenheit auch gespielt sein darf.

Ein erfahrener Analyst weiß also, wie mit den Schwierigkeiten des Requirements Engineering umgegangen werden kann. Er hat zudem ein Gespür für mögliche Missverständnisse entwickelt, die ungeklärt bis in die Anforderungsdokumentation gelangt sind. Dabei ergeben sich zwei Probleme:

1. Nicht jede Organisation kann genügend erfahrene Analysten einsetzen.
2. Das Gespür eines erfahrenen Analysten für verborgene Missverständnisse findet nicht immer zuverlässig alle Probleme.

Das erste Problem lässt sich durch systematisches Erfahrungsmanagement abschwächen oder sogar lösen. Bei dem zweiten Problem muss man jedoch weiter gehen. Im Rahmen dieser Arbeit werden heuristische Kritiken genutzt, um Indikationen für Missverständnisse und Probleme in Anforderunsdokumenten automatisch zu finden. Ein wichtiges grundlegendes Konzept ist die DODE-Architektur, die im folgenden Abschnitt beschrieben wird.

Grundlagen der DODE-Architektur

Die DODE-Architektur geht auf Fischer zurück [64]. Er hatte diese Architektur vorgeschlagen, um mit so genannten *Upstream*-Aktivitäten besser umgehen zu können. Mit *Upstream* bezeichnet er die notwendigen Aktivitäten, um von einem gegebenen Problem zu einer guten Spezifikation der zu erstellenden Lösung zu kommen. *Downstream*-Aktivitäten sind dann die Schritte von der Spezifikation bis zu ihrer Umsetzung.

Der große Unterschied ist, dass in den *Upstream*-Aktivitäten das zu lösende Problem weder umfassend definiert, noch fest ist. Abhängig von den möglichen Lösungen kann sich das Ziel sogar noch ändern. Damit ist auch schwer festzustellen, wie gut eine Lösung ist. Fischer schlägt daher vor, Upstream- und Downstream-Aktivitäten besser miteinander zu verknüpfen. Ein Schritt in diese Richtung sind *Domain Oriented Design Environments*, kurz DODE.

Eine DODE besteht aus fünf Komponenten, die eng zusammenarbeiten, um den Nutzer bei Entwurfsaktivitäten zu unterstützen. Abbildung 2.3 gibt einen Überblick über diese Komponenten.

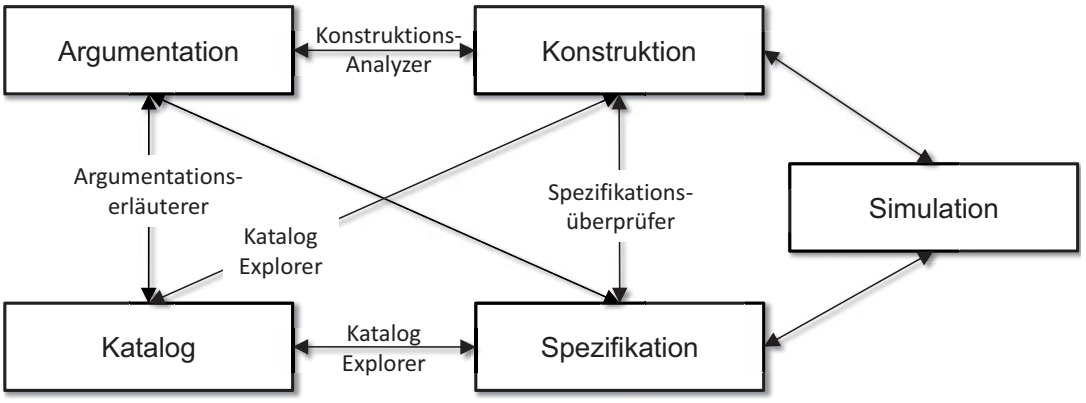


Abbildung 2.3: Konzeptioneller Aufbau eines DODE nach Fischer [64].

Konstruktionskomponente. Der zentrale Teil der DODE-Architektur ist die Konstruktionskomponente, in der die Entwurfsartefakte konstruiert werden. Als

Beispiele nennt Fischer die Erstellung von grafischen Oberflächen oder die Planung einer Küche.

Argumentationskomponente. Die Eingabe (der Inhalt) der Konstruktionskomponente wird in der Argumentationskomponente vom Computer regelbasiert analysiert. Wenn eine der hinterlegten Regeln verletzt wird, äußert die Argumentationskomponente eine Kritik. Nutzer können der Kritik folgen und das Entwurfsartefakt entsprechend verbessern oder dagegen argumentieren. Beide Formen von Feedback können zu einer Verbesserung der Argumentationskomponente führen, eine DODE soll bei der Bedienung *dazulernen*.

Simulationskomponente. Die Simulationskomponente einer DODE soll Nutzern ermöglichen, verschiedene Entwurfsvarianten auszuprobieren und miteinander zu vergleichen. Im Rahmen von Küchenplanern und GUI-Designern ist die Simulation ein gutes Mittel, um solche *wenn-dann* Analysen zu erlauben.

Spezifikationskomponente. Der Entwurf, der in der Konstruktionskomponente erstellt wird, folgt in der Regel einem spezifizierten Ziel (der Spezifikation). Während des Entwurfs wird diese Spezifikation ständig mit den Realisierungsmöglichkeiten konfrontiert. Die Spezifikation muss an den Rahmen dieser Möglichkeiten angepasst werden. Fischer spricht auch von einer Co-Evolution von Spezifikation und Design. Mit Hilfe dieser Komponente kann eine DODE den aktuellen Stand der Spezifikation nachvollziehen. Die Argumentationskomponente soll den aktuellen Entwurf mit dieser Spezifikation vergleichen können. Werden dabei Widersprüche zur Spezifikation erkannt, wird der Nutzer dabei unterstützt, diese aufzulösen - entweder durch Änderung des Entwurfs oder der Spezifikation.

Katalogkomponente. Eine weitere Komponente einer DODE ist der Katalog. Hier werden dem Nutzer fertige Vorlagen oder Entwurfsbausteine zur Wiederverwendung angeboten.

Durch das Zusammenspiel dieser Komponenten ist eine DODE in der Lage, domänenspezifisches Wissen in Entwurfsaktivitäten einfließen zu lassen. Im Rahmen der Argumentationskomponente kann dieses Wissen auf Basis der Nutzerinteraktion erweitert werden. Diese Arbeit baut auf Fischers Konzepten auf und nutzt teilweise ähnliche Architekturen, um Erfahrungen in die Dokumentation von Anforderungen einfließen zu lassen.

2.3 Diskussion und Abgrenzung

Dieses Kapitel hat vor allem eines gezeigt: Es gibt eine erschlagende Vielzahl von Fehlerquellen im Requirements Engineering und es ist keine Silberkugel (verglei-

che Brooks [36][3]) in Sicht, die alle Probleme mit einem Streich löst. Vielmehr ist es so, dass die Lösung für ein Problem ein anderes unter Umständen verschlimmert. So kann eine formale Modellierung der Anforderungen es erleichtern, diese zu verifizieren, es jedoch unmöglich machen, sie beim Kunden zu validieren, wenn dieser die formale Notation unter Umständen nicht versteht.

Diese Arbeit zielt darauf ab, die hier zitierten Grundlagen und Erfahrungen im Requirements Engineering besser nutzbar zu machen.

In Abschnitt 2.1.2 wurde gezeigt, wie die Dokumentation von Anforderungen durch bestimmte Dokumentationsarten und einheitliche Strukturierungsvorgaben unterstützt werden kann. Viele Erfahrungen stecken bereits in solchen Strukturierungsvorgaben. Heuristische Kritiken können jedoch helfen, diese noch besser zu nutzen. So kann die Einhaltung von Strukturierungsvorgaben heuristisch überprüft werden oder eine passende Dokumentationsart heuristisch vorgeschlagen werden.

Ähnliches gilt für die Regeln zur Dokumentation (Abschnitt 2.1.3), die häufig durch Vorlagen oder Prozessvorgaben instrumentalisiert sind. Teilweise sind diese Regeln formalisierbar und dann für heuristisches Feedback nutzbar.

Im Sinne von Abschnitt 2.2 können formalisierte, heuristische Regeln als Erfahrungen aufgefasst und konsequent zur systematischen Verbesserung genutzt werden - das ist zumindest im Rahmen des Requirements Engineering neu. Fischers DODE ist ein Beispiel, mit dem dies möglich ist und das im Rahmen dieser Arbeit zum ersten Mal für das Requirements Engineering angewendet wurde. Die Arbeit untersucht aber auch weniger direkte Arten von heuristischem Feedback zu Anforderungsdokumentation und deren Bezug zum Erfahrungsmanagement im Requirements Engineering.

Die Arbeit leistet damit einen Beitrag, mit dem Organisationen Praktiken des Requirements Engineering kontinuierlich erfahrungsbasiert verbessern können. Gute Requirements Engineering Praktiken sind jedoch weder notwendig noch hinreichend [49]: In ihrem Artikel zeigen Davis und Zowghi diesen Umstand durch folgende Argumentation auf: Wären gute Praktiken beim Requirements Engineering notwendig, dann müsste sich der vollständige Verzicht auf diese Praktiken als nachteilig herausstellen. Dazu lassen sich aber leicht Gegenbeispiele finden. Wären gute Praktiken auf der anderen Seite hinreichend, dann würde es genügen, diese einzusetzen, um erfolgreiches Requirements Engineering durchzuführen.

[3] Brooks unterscheidet zwischen zufälliger und essentieller Komplexität im Software Engineering. Die zufällige, selbst verursachte Komplexität sei durch heutige Methoden und Techniken weitgehend bewältigt. Probleme bei der Erstellung der Software lägen damit hauptsächlich in der Komplexität, die heutigen Systemen innewohnt, begründet. Deshalb wäre keine einzelne technische oder organisatorische Lösung zu erwarten, durch die sich Produktivität und Zuverlässigkeit um eine Größenordnung steigern lassen.

Auch hierzu finden die Autoren Gegenbeispiele. Dennoch erhöhen gute Praktiken die Wahrscheinlichkeit des Erfolgs, ein Umstand, der im Rahmen dieser Arbeit im Kontext der Softwareprojekte in der universitären Lehre sogar nachgewiesen werden konnte (vergleiche Abschnitt 4.3). Die folgenden Kapitel zeigen, dass dies auch für die Konzepte dieser Arbeit gilt.

3 Fallbeispiel: Direktes Feedback mit HeRA

Dieses Kapitel zeigt mit einem Fallbeispiel, wie Heuristiken und Erfahrungen genutzt werden können, um die Dokumentation von Anforderungen zu verbessern. Dazu wird zunächst ein heuristisches Anforderungswerkzeug eingeführt: HeRA (Heuristic Requirements Assistant). Ein Anwendungsszenario verdeutlicht, wie mit Hilfe von HeRA Erfahrungen in die Dokumentation von Anforderungen einfließen.

HeRA ist der wichtigste Forschungsprototyp dieser Arbeit. Viele der hier vorgestellten Konzepte sind mit Hilfe von HeRA evaluiert worden (vergleiche Abschnitt 8.2). Die hier vorgestellte Version von HeRA basiert auf den Masterarbeiten von Crisp [45] und Meyer [120], sowie auf den Bachelorarbeiten von Kitzmann [89], Rumann [141], Szongott [164] und Pilarski [127]. Die Grundidee zu HeRA ist zum ersten Mal 2006 beim Treffen der GI Fachgruppe Requirements Engineering vorgestellt worden [91] und dank des guten Feedbacks konsequent weiter verfolgt worden [98, 128, 101, 104, 106]. Dieses Kapitel basiert teilweise auf einem Beitrag zur International Conference on Software Engineering, 2009 in Vancouver [103].

Abschnitt 3.1 erläutert kurz die Motivation, die zu der Entwicklung von HeRA geführt hat. Abschnitt 3.2 gibt einen Überblick über den Aufbau von HeRA, bevor die einzelnen Feedback-Mechanismen in Abschnitt 3.3 mit Hilfe eines durchgängigen Beispiels erläutert werden. Abschnitt 3.4 fasst die Implikationen von HeRA für diese Arbeit zusammen. Dieses Kapitel dient dazu schon früh ein konkretes Beispiel zu liefern, an dem sich die abstrakteren Teile dieser Arbeit veranschaulichen lassen. Die Konzepte, die hinter HeRA stehen, werden in den darauf folgenden Kapiteln beschrieben. Mit diesem Hintergrund dient HeRA in Kapitel 8 zur Evaluation der Ziele dieser Arbeit.

3.1 Motivation: Feedback bei der Dokumentation von Anforderungen

Requirements Engineering beschäftigt sich mit der Erhebung und Dokumentation der gewünschten Ergebnisse eines Softwareprojekts. Ausgangspunkt ist dabei typischerweise, dass jemand die Idee hat, ein System zu entwerfen oder zu bauen [70]. Von dieser Idee (auch: *Projektvision*) aus startet ein Analyst, indem er nach relevanten Stakeholdern sucht, also Personen, die von dem zu erstellenden System betroffen sind.

Typische Prozesse, wie zum Beispiel das Requirements Abstraction Model (RAM) [76], Volere [139] oder das V-Modell [67, 135], sehen dafür einen Top-Down-Ansatz vor. Dabei werden abstrakte Anforderungen Schritt für Schritt verfeinert, bis eine angemessen konkrete und detaillierte Beschreibung des zu erstellenden Systems erreicht ist.

Der Vorteil dieses Top-Down-Ansatzes ist, dass durch die Dekomposition in kleinere Teile die Komplexität heutiger Software-Systeme beherrschbar bleibt. Eines der inhärenten Probleme des Requirements Engineering (vergleiche Gause und Weinberg [70]), die unterschiedliche Interpretation der ursprünglichen Idee durch die beteiligten Stakeholder und Entwickler, wird auf diese Weise jedoch nicht adressiert. Im Gegenteil, unterschiedliche Interpretationen sind beim Top-Down-Ansatz schwer zu entdecken, wenn Anforderungen zunächst nur auf hohem Abstraktionsniveau diskutiert werden. Dadurch wird die Identifikation von Konflikten verzögert, bis durch Dekomposition ein konkreteres und zugänglicheres Abstraktionsniveau erreicht wird.

Vor diesem Hintergrund entstand die Idee für HeRA. Wenn es möglich ist, die Komplexität durch ein aktives Feedbacksystem in den Griff zu bekommen, dann kann man schon früh zu konkreten Benutzerzielen vordringen und diese dann nachträglich zu abstrakteren Geschäftszielen zusammenfassen. HeRA unterstützt den Analysten dabei, indem es automatisch Informationen zusammenträgt, auf deren Grundlage Konflikte mit anderen Stakeholdern erkannt und Inkonsistenzen aufgedeckt werden können.

HeRA soll dabei wie ein Partner beim Pair-Programming agieren (vergleiche [162]). Das Werkzeug analysiert ständig die Eingabe und vergleicht sie mit anderen bereits dokumentierten Anforderungen im Projekt und einer Erfahrungsbasis. Basierend auf heuristischen Regeln wird daraus dann Feedback generiert, das für den Analysten wertvoll ist. Dies entlastet den Analysten, der nicht mehr das ganze Projekt- und Erfahrungswissen im Hinterkopf haben muss, wenn er neue Anforderungen formuliert.

3.2 Überblick: Feedback-Mechanismen im Zusammenspiel

HeRA basiert auf Fischers DODE-Architektur [64], die in Abschnitt 2.2.4 detailliert beschrieben wird. In HeRA wurde die Idee einer DODE auf die Domäne der Anforderungen angewandt. HeRA setzt die hinter der DODE-Architektur liegenden Ideen weitgehend um. Das Schreiben eines guten Anforderungsdokuments besitzt als Problemstellung alle Charakteristika einer *Upstream*-Aktivität: Die Definition, was eine gute Dokumentation von Anforderungen ist, hängt stark vom Kontext und Inhalt des Projekts ab. Ein Projekt für sicherheitskritische Software wird eine formalere Spezifikation benötigen als eine private Webanwendung. Die

genauen Rahmenbedingungen eines Projekts werden aber erst während der Analysephase festgestellt. Damit stellt sich für den Analysten die Aufgabe, neben den Anforderungen für das Softwareprojekt auch die Rahmenbedingungen zu erfassen und das Dokument dementsprechend auszugestalten. Abbildung 3.1 zeigt den Überblick über HeRAs Komponenten. Folgende Ausprägungen der Komponenten wurde demnach in HeRA integriert:

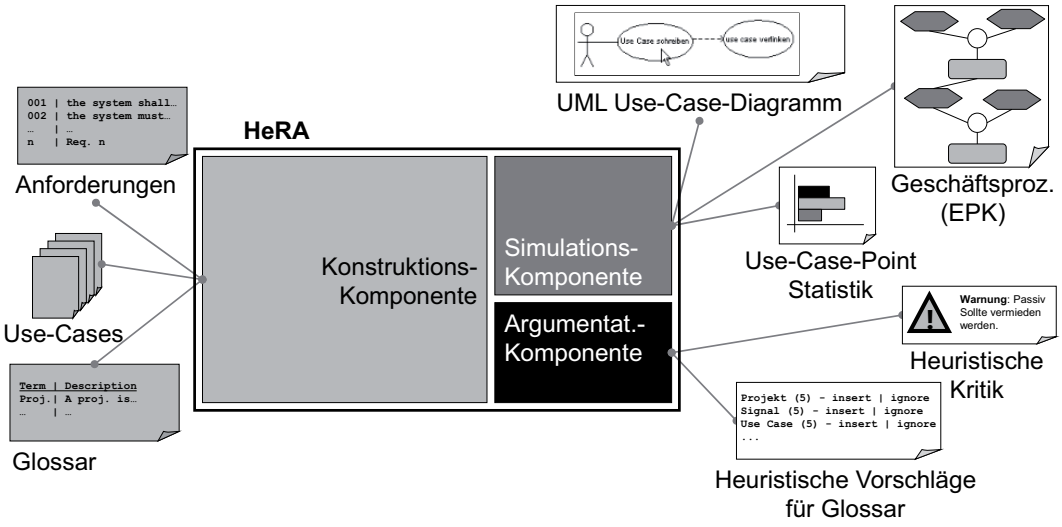


Abbildung 3.1: Der Aufbau von HeRA.

Konstruktionskomponenten. Durch die flexible und erweiterbare Struktur von HeRA ist es recht einfach, neue Konstruktionskomponenten einzurichten. Zur Zeit sind Konstruktionskomponenten zur Erstellung von Use-Cases nach Cockburns Vorlage (siehe Tabelle 2.1 und [42]), Glossar-Begriffen und Anforderungen im Einsatz. Weitere Komponenten zur Konstruktion von Qualitätsmodellen (vergleiche [87]) und Projektrahmenbedingungen existieren in Form von Machbarkeitsstudien, sind aber (noch) nicht einsatzfähig.

Argumentationskomponenten. Argumentationskomponenten für HeRA können ebenfalls flexibel als Erweiterungen eingebunden werden. Zur Zeit existiert ein Kritiksystem (HeRA.Kritik), das die Eingabe analysiert und basierend auf einer erweiterbaren Menge heuristischer Regeln Feedback gibt. Im Rahmen der Konstruktionskomponente für Glossar-Begriffe gibt es eine weitere spezialisierte Argumentationskomponente, die dem Nutzer Glossar-Begriffe vorschlägt (HeRA.Glossar).

Simulationskomponenten. Ziel der Simulationskomponente einer DODE ist es nach Fischer, dem Nutzer Feedback zu geben und ihm eine Abschätzung zu erlauben, welche Auswirkungen eine Entwurfsentscheidung hat. Da HeRA

schon sehr früh im Projekt zum Einsatz kommen soll, schon bei der Erfassung der ursprünglichen Stakeholder-Wünsche, sind die Anforderungen noch nicht formal genug, um eine Simulation zu erlauben (vergleiche Abschnitt 4.2). Trotzdem kann der Versuch, Informationen aus den soeben dokumentierten Anforderungen abzuleiten, wertvolles Feedback über die Vollständigkeit und Integrität der Anforderungen geben. So kann HeRA den Zusammenhang mehrerer Use-Cases darstellen, den implizierten Prozess ableiten oder eine Aufwandschätzung auf Basis des Use-Case-Point-Verfahrens berechnen. Das Feedback hilft dem Analysten festzustellen, ob er die Use-Cases gerade auf sinnvolle Art modelliert oder nicht.

Spezifikationskomponente. Laut Fischer wird bei einer DODE der aktuelle Entwurf mit der Spezifikation verglichen, und geprüft, ob der Entwurf eine gültige Lösung für das vorhandene Problem darstellt. HeRA bezieht sich aber auf die Erstellung einer (Anforderungs-) Spezifikation. Es gibt über Vorlagen hinaus in einem Softwareprojekt damit keine Referenz, mit der man die Inhalte der Konstruktionskomponenten vergleichen könnte. An diese Stelle tritt eine implizit in den Erfahrungen vorhandene Meinung, was eine *gute* Anforderungsdokumentation ist. Außerdem können an einigen Stellen Informationen über die Art des Projekts abgeleitet werden.

Katalogkomponente. Die Katalogkomponente einer DODE soll Wiederverwendung von Entwurfsbausteinen unterstützen. Bei Anforderungen kommt Wiederverwendung zum Beispiel im Rahmen von Produktlinien vor. Für allgemeine Softwareprojekte spielt sie dagegen keine große Rolle. Deshalb wurde kein Katalog mit konkreten Anforderungen angelegt. Statt dessen besitzt HeRA eine Reihe von Vorlagen für Use-Cases und Anforderungen sowie einen Editor, mit dem neue Vorlagen erstellt werden können. Dies erlaubt es, Anforderungsdokumentation in eine bewährte Struktur zu bringen.

HeRA soll den Analysten schon sehr früh im Projekt unterstützen. Auf Basis der ersten Interviews mit zukünftigen Benutzern werden Use-Cases erstellt, um die Interaktion mit dem zu erstellenden System zu modellieren. HeRA bietet zu diesem Zweck eine Vorlage an, die auf den Empfehlungen von Cockburn beruht [42]. HeRA analysiert die Eingabe in diese Vorlage und warnt den Analysten, wenn Mehrdeutigkeiten oder unvollständige Spezifikationen auftreten. Darüber hinaus leitet HeRA UML Use-Case-Diagramme ab, aus denen ersichtlich wird, wie der gerade bearbeitete Use-Case mit anderen Benutzerzielen oder übergeordneten Geschäftszielen zusammenhängt. Ein spezieller Glossar-Assistent kann hinzugezogen werden, um die konsistente Verwendung wichtiger Begriffe der Domäne zu gewährleisten. Ein weiterer Assistent, der in HeRA integriert ist, kann eine Visualisierung des implizierten Geschäftsprozesses erzeugen, indem die Use-Cases basierend auf ihren Vor- und Nachbedingungen aneinander gehängt werden. Dadurch kann festgestellt werden, ob der aktuelle Use-Case gut zum globalen Prozess passt, in den

er eingegliedert ist. Bei Bedarf kann HeRA die Use-Case-Points berechnen und eine darauf basierende Aufwandschätzung für jeden Use-Case anzeigen. All diese Perspektiven können abgeleitet werden, *während* der Use-Case geschrieben wird. Dadurch erhält der Autor direktes Feedback zu seiner Eingabe, die er dann direkt verbessern kann. HeRA wurde bereits erfolgreich in Interviews und Workshops angewandt und hat Analysten bei den folgenden Punkten unterstützt:

1. Benutzerziele werden auf einer Detailstufe erhoben, die es erlaubt, Konflikte zu entdecken.
2. Es wird eine Diskussion ermöglicht, ob der aktuelle Use-Case zu dem angestrebten Geschäftsziel und den bereits dokumentierten Use-Cases passt (basierend auf der Visualisierung als UML Use-Case und Prozessmodell).
3. Wichtige Begriffe werden identifiziert und Konflikte in deren Verwendung werden aufgezeigt.
4. Basierend auf der Aufwandschätzung ist auch eine frühzeitige Diskussion von Prioritäten und Rahmenbedingungen des Projekts möglich.

Durch das direkte Feedback von HeRA wird so ermöglicht, schon sehr früh im Projekt mit der Erhebung von Benutzerzielen zu beginnen und dadurch Inkonsistenzen und Konflikte zu erkennen. Zu beachten ist, dass HeRAs Feedback heuristisch ist. So lassen sich beispielsweise nicht alle Use-Cases als Geschäftsprozess darstellen (vergleiche dazu die Ausführungen von Lübke [109, Seite 126]). In den Fällen, in denen dies aber möglich ist, kann so wertvolles Feedback gewonnen werden [101].

Abbildung 3.2 zeigt, wie die Komponenten von HeRA im konkreten Programm eingebunden sind. Die folgenden Abschnitte erläutern HeRAs Komponenten im Detail.

3.3 HeRA-Komponenten für heuristisches Feedback

Die im vorherigen Abschnitt vorgestellten Bestandteile von HeRA werden nun detailliert und mit einem durchgängigen Beispiel beschrieben.

3.3.1 Kritiksystem für Anforderungen

Anforderungsexperten können die wichtigste Argumentationskomponente von HeRA verwenden, um ihre Erfahrungen aus früheren Projekten als Regeln zu kodieren. In Form von heuristischen Kritiken stehen diese Erfahrungen dann zur Verführung, um zukünftige Analysten bei der Dokumentation von Anforderungen zu unterstützen. HeRA wendet die Regeln auf die Eingabe des Nutzers in die

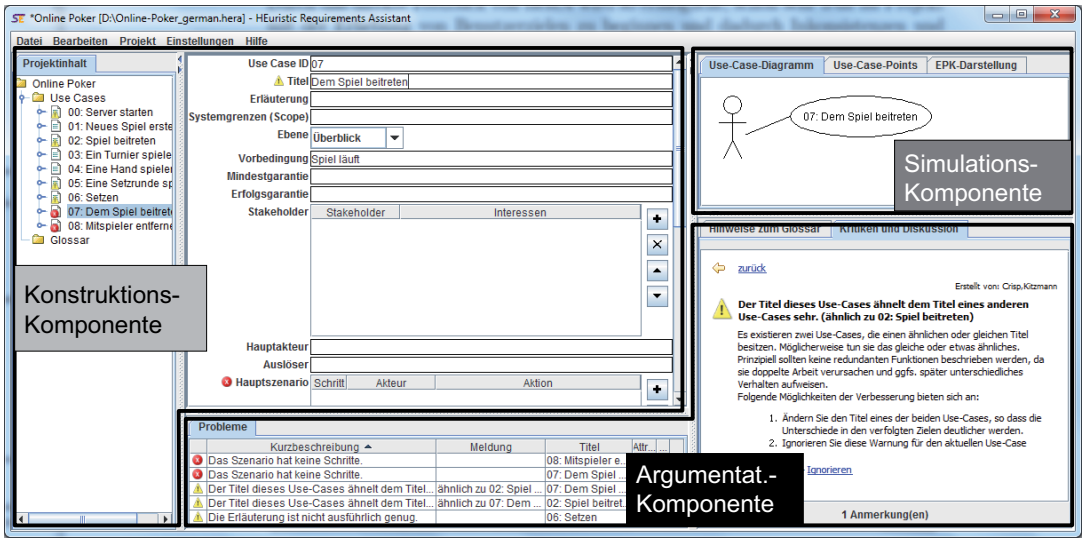


Abbildung 3.2: Screenshot des HeRA-Werkzeugs. Die Struktur aus Abbildung 3.1 ist hervorgehoben.

Konstruktionskomponente an. Basierend auf diesem Mechanismus, im Folgenden auch HeRA.Kritik genannt, kann HeRA dann direktes Feedback zur Eingabe geben.

Als Instantiierung der Argumentationskomponente sind HeRAs Kritiken in der Lage, *Breakdowns* während einer Aktivität, wie dem Tippen von Anforderungen, hervorzurufen (vergleiche Kapitel 5 und [146]): Autoren einer solchen Anforderung werden in ihrer Arbeit unterbrochen und auf ein Problem aufmerksam gemacht, das sie vielleicht übersehen haben. Sie können sich dann entscheiden, ob sie das Problem direkt beheben, zunächst die Arbeit beenden und die Warnung erst später beachten, oder Feedback zu der Warnung geben wollen. Durch das Feedback wird eine Verbesserung der Regelbasis für das nächste Projekt möglich. HeRA erlaubt es zudem, Kritiken zu ignorieren. Wird eine Kritik ignoriert, wird sie in ähnlichen Situationen nicht mehr angezeigt. Der Nutzer kann eine Warnung für den aktuellen Use-Case, dem aktuellen Attribut in allen Use-Cases oder vollständig ignorieren.

Das folgende Beispiel zeigt, wie sich das Kritiksystem auf die Dokumentation von Anforderungen auswirkt:

Beispiel 1: *Direktes computerbasiertes Feedback führt zu besseren Anforderungen*

In einem Projekt haben die Analysten eines Projekts mit allen Stakeholdern des zu erstellenden Software-Systems gesprochen. Dabei haben sie das HeRA-Werkzeug

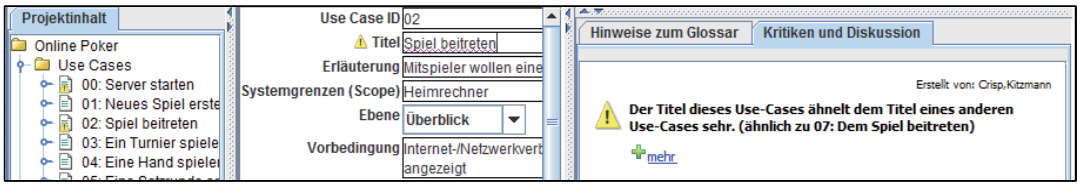


Abbildung 3.3: Die kritikbasierte Argumentationskomponente (rechts) von HeRA warnt, dass der Titel des Use-Cases dem Titel eines anderen Use-Cases zu sehr ähnelt. Die Option *mehr* führt zu einer längeren Beschreibungen und der Möglichkeit Feedback zu geben.

verwendet, um die Interaktionen von Benutzern und System als Use-Cases zu dokumentieren. Je nach Situation kam HeRA dabei direkt im Interview zum Einsatz, oder erst später im Büro.

Die Analysten finden an HeRA besonders die heuristische Unterstützung bei der Formulierung von Use-Cases gut (siehe Abbildung 3.3). Vor allem Analysten, die bisher noch keine Erfahrung mit Use-Cases hatten, profitieren davon, durch HeRA bei der Erstellung angeleitet zu werden. Aber auch erfahrene Analysten, die bereits in anderen Projekten mit Use-Cases gearbeitet haben, finden HeRA hilfreich: Die automatische Identifikation von Mehrdeutigkeiten über so genannte *Weakwords* hilft beispielsweise Flüchtigkeitsfehler zu vermeiden.

Durch HeRAs Unterstützung waren die Analysten in der Lage, einige Fehler zu entdecken, die sonst zu Mehrdeutigkeiten oder Lücken in der Spezifikation geführt hätten. Teilweise mussten sie sogar fehlende Information bei den Stakeholdern nachfragen, während sie die Weakwords durch genauere Information ersetzten. Daher sind die Analysten gerne bereit, Kritiken, die nicht hilfreich waren, zu kommentieren. Dies erlaubt es dem Erfahrungsmanager, das Feedback zu verbessern (siehe Abbildung 3.4).

3.3.2 Argumentationskomponente für Glossare

HeRA.Glossar, eine zweite Argumentationskomponente von HeRA, unterstützt die Nutzer bei der Erstellung eines Glossars [120, 104]. Wie bei dem Kritiksystem wird auch hier die Eingabe analysiert. Dabei werden zwei heuristische Regeln angewendet:

1. *Häufigkeitsheuristik:* Je öfter ein Begriff vorkommt, um so größer ist das Potential dadurch Missverständnisse in die Anforderungsspezifikation ein-

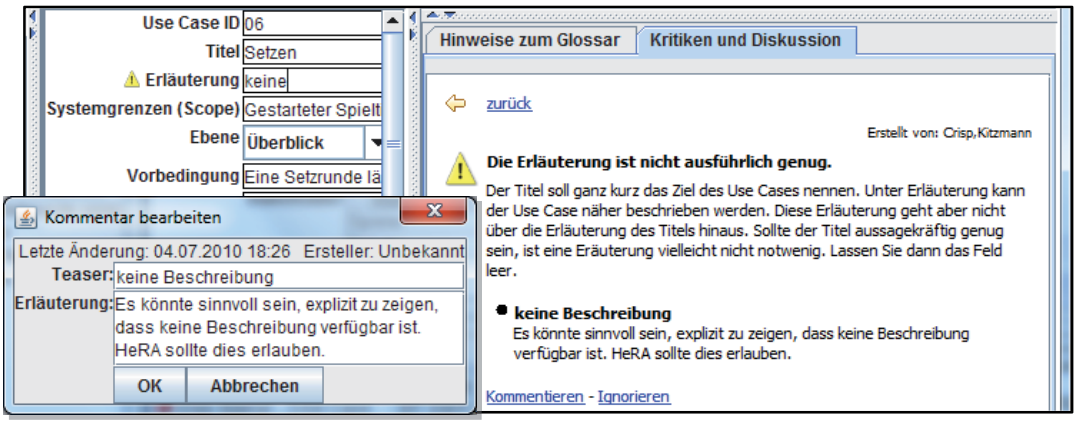


Abbildung 3.4: HeRA erlaubt es, Kritiken zu kommentieren oder zu ignorieren. Kommentare werden in ein einfaches Formular eingetragen und sind dann für alle Nutzer in der Erläuterung der Kritik sichtbar.

zuführen. Womöglich sollte ein solcher Begriff in das Glossar eingefügt werden.

2. *Erfahrungsheuristik:* Ein Begriff, der schon einmal in einem Projekt als wichtiger Begriff erkannt und in ein Glossar eingefügt wurde, ist mit hoher Wahrscheinlichkeit auch für dieses Projekt ein Kandidat für das Glossar.

HeRA.Glossar analysiert die Eingabe und zerlegt diese in einzelne Wörter. Diese werden dann für die Häufigkeitsheuristik gezählt und mit der Datenbank von Begriffen, die bereits in einem anderen Glossar definiert wurden, abgeglichen. Die relativ am häufigsten verwendeten Begriffe, die noch nicht im Glossar stehen, werden dem Nutzer vorgeschlagen.

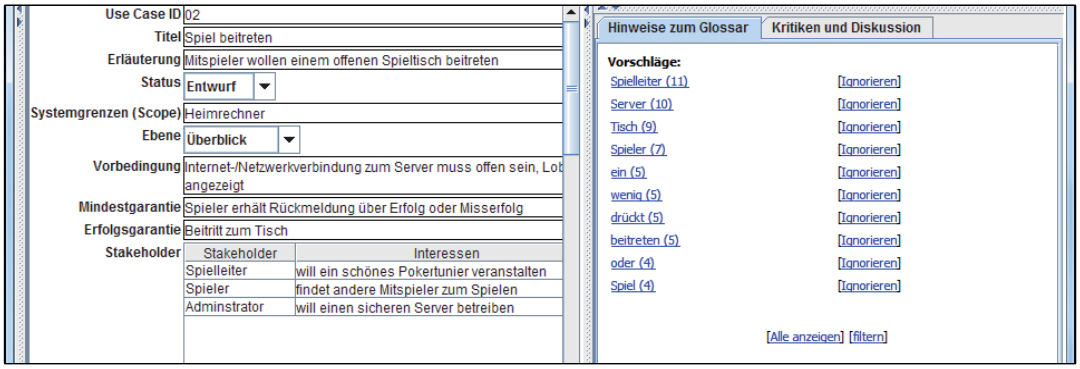


Abbildung 3.5: HeRAs Vorschläge von Begriffen für das Glossar.

Auch hier kann der Nutzer entscheiden, ob er den Vorschlag annimmt und den Begriff in das Glossar aufnimmt, oder ob das Wort keine weitere Erklärung be-

nötigt. Im letzteren Fall kann es auf eine Filterliste gesetzt werden, damit es zukünftig nicht mehr vorgeschlagen wird. Abbildung 3.5 zeigt die Vorschlagsliste in HeRA. Bisher wurden noch keine Filter trainiert. Die Vorschlagsliste enthält daher noch Füllwörter (zum Beispiel "ein"). Diese können mit einem einfachen Klick auf *ignorieren* herausgefiltert werden. Ein Klick auf einen der Begriffe führt dazu, dass dieser in das Glossar aufgenommen wird. Die Zahl neben dem Begriff zeigt an, wie oft dieser Begriff in dem Projekt vorkommt. Je häufiger ein Begriff verwendet wird, desto häufiger kann er auch falsch verwendet werden. Deshalb sortiert HeRA die Vorschläge nach Häufigkeit.

In [104] wird darüber hinaus auch die Hervorhebung von Glossar-Begriffen in den Anforderungen betont. Diese Form des Feedback ist sehr hilfreich, da es bei Autoren und Lesern das Bewusstsein schafft, dass ein markierter Begriff im Rahmen des Projekts eindeutig definiert ist (siehe Abbildung 3.6). In Versuchen konnte beobachtet werden, dass auch dies zu einem *Breakdown* führen kann: Als Leser stellt man sich die Frage, ob der Begriff in der aktuellen Anforderung tatsächlich so gebraucht wird, wie er im Glossar definiert ist.

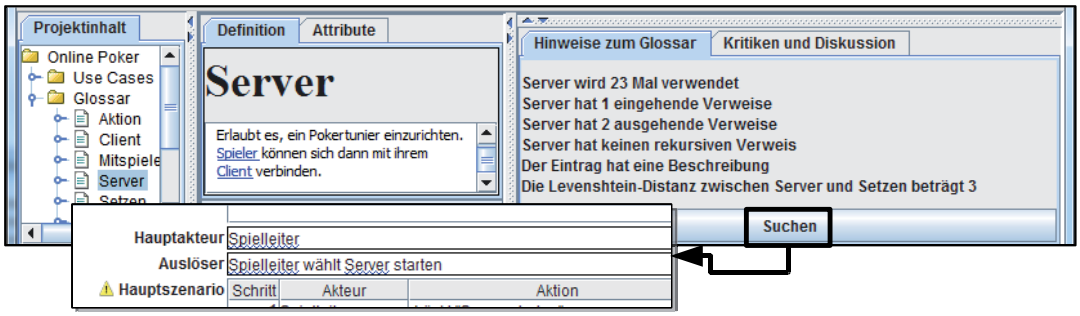


Abbildung 3.6: Ein Glossarbegriff in der Konstruktionskomponente für Glossare. Die Argumentationskomponente zeigt Statistiken zu diesem Begriff. Definierte Begriffe werden in den Anforderungen und Use-Cases hervorgehoben. Jede Verwendung eines Begriffs kann man aus dem Glossar-Editor mit Hilfe der Suche leicht erreichen.

Das folgende Beispiel zeigt, wie sich die Argumentationskomponente für Glossare bei der Dokumentation von Anforderungen nutzen lässt:

Beispiel 2: *Einsatz computerbasierter heuristischer Kritiken bei der Arbeit mit Glossaren*

Eine besondere Herausforderung in einem Softwareprojekt ist die Fachterminologie, die von verschiedenen Stakeholdern verwendet wird. HeRAs Feedback macht

das schnell deutlich, indem es Vorschläge für das Glossar unterbreitet (siehe Abbildung 3.5).

Manche der vorgeschlagenen Begriffe kennen die Analysten nicht, bei anderen sind sie sich nicht sicher, ob sie die Bedeutung des Begriffs genauso sehen, wie die Stakeholder. Aus diesem Grund entscheiden sie sich, mit Hilfe von HeRAs Vorschlägen, ein Glossar aufzubauen. Nachdem die fraglichen Begriffe erst einmal gesammelt wurden, ist nur noch ein Telefonanruf mit einem Kundenvertreter erforderlich, um die zentralen Begriffe zu klären. Obwohl dies einiges an Zeit in Anspruch nimmt, zeigt sich schnell, dass diese Zeit gut investiert ist. Je nach Ursprung und Abteilung wurden einige Begriffe unterschiedlich verwendet.

Jetzt, nachdem das Glossar erzeugt wurde, kann HeRA anzeigen, wo ein Begriff verwendet wird und ihn mit den entsprechenden Anforderungen verlinken (siehe Abbildung 3.6). Dieses Feature erlaubt es, nach Begriffen zu suchen, die inkonsistent verwendet wurden.

3.3.3 Abgeleitetes UML Use-Case-Diagramm

Ein gutes Mittel, sich einen Überblick über eine Menge von Use-Cases und ihrer Beziehungen untereinander zu verschaffen, ist ein UML Use-Case-Diagramm (vergleiche Birk [28]). HeRA.UML stellt eine solche Visualisierung als Simulationskomponente zur Verfügung, um Autoren durch direktes Feedback mehr Überblick zu geben.

Der Fokus liegt dabei darauf, dem Nutzer die unmittelbare Umgebung des gerade bearbeiteten Use-Cases zu visualisieren. Das ist wichtig, denn bei mittelgroßen Systemen mit 50 oder mehr Use-Cases wäre die Übersichtlichkeit eines vollständigen Use-Case-Diagramms fraglich. Zudem haben weite Teile des zu spezifizierenden Systems auch keinen Einfluss auf den aktuellen Use-Case. Damit wird ein anderes mögliches Ziel bei der automatischen Generierung von Use-Case-Diagrammen explizit nicht unterstützt: Die Erzeugung von Diagrammen zum Zweck der Dokumentation. Für dieses Ziel wäre die Vollständigkeit ein wichtiges Kriterium.

Definition 11: ***Direkte Umgebung eines Use-Cases***

Die direkte Umgebung eines Use-Cases u sind alle Use-Cases, die im Hauptszenario, einer Erweiterung oder technischen Variation von u referenziert werden oder u referenzieren sowie der Hauptakteur von u.

Es zeigt sich, dass man die Übersicht über die direkte Umgebung eines Use-Cases mit einem einfachen Algorithmus unterstützen kann. Auch das Layout des

generierten Diagramms gestaltet sich einfach, da tendenziell nur wenige Use-Cases systematisch ausgewählt werden. Der hier skizzierte Algorithmus hat sich in HeRA bewährt:

1. Unterteile die Zeichenfläche in Spalten und Zeilen. Die Breite der Spalten wird so festgelegt, dass sie den UML Use-Case mit dem längsten Namen aufnehmen können.
2. Für jeden Use-Case, in dem der aktuell in HeRA bearbeitete Use-Case referenziert wird: Erzeuge einen UML Use-Case in der nächsten freien Spalte der ersten Zeile.
3. Für den aktuell in HeRA bearbeiteten Use-Case:
   a) Erzeuge einen UML Akteur mit dem Namen des Hauptakteurs und zeichne diesen in die erste Spalte der ersten Zeile.
   b) Erzeuge einen UML Use-Case und setze den Titel des aktuellen Use-Cases als Namen. Zeichne diesen UML Use-Case in die dritte Spalte der selben Zeile.
   c) Zeichne eine waagerechte Assoziation in die Spalte dazwischen, um den UML Akteur mit dem UML Use-Case zu verknüpfen. Füge für jeden Use-Case aus Schritt 2 eine direkte Verbindung zu diesem UML Use-Case ein.
4. Für jeden Schritt im Hauptszenario des aktuell in HeRA bearbeiteten Use-Case:
   a) Wenn in einem Schritt ein weiterer Use-Case referenziert ist, wird dem UML Use-Case-Diagramm eine «include»-Beziehung hinzugefügt.
   b) Der referenzierte Use-Case wird dafür in die fünfte Spalte der ersten Zeile geschrieben, weitere referenzierte Use-Cases entsprechend darunter.
   c) Die «include»-Beziehungen werden dann rechtwinklig in die vierte Spalte gezeichnet.
5. Für jede Erweiterung des aktuell in HeRA bearbeiteten Use-Case: Erzeuge UML Use-Cases und «extend» Beziehungen analog zu den «include» Beziehungen.

Das entstehende Layout ist einfach und in seiner Breite beschränkt. Dadurch eignet es sich gut, um es am rechten oberen Bildschirmrand als Übersichtskarte darzustellen. Abbildung 3.7 zeigt das von HeRA.UML auf Basis des aktuellen Use-Cases abgeleitete UML Use-Case-Diagramm.

Das Resultat ist ein schneller Überblick mit den wichtigsten Informationen zum aktuellen Use-Case (Hauptakteur und Titel des Use-Case) und dessen direktem Umfeld. Durch den einfachen Algorithmus kann das Diagramm sehr schnell erzeugt werden – sogar mehrmals, während ein neuer Titel eingegeben wird. Die referenzierten Use-Cases können über das Übersichtsdiagramm mit einem Doppelklick leicht erreicht werden. Dadurch wird das UML Use-Case-Diagramm ein

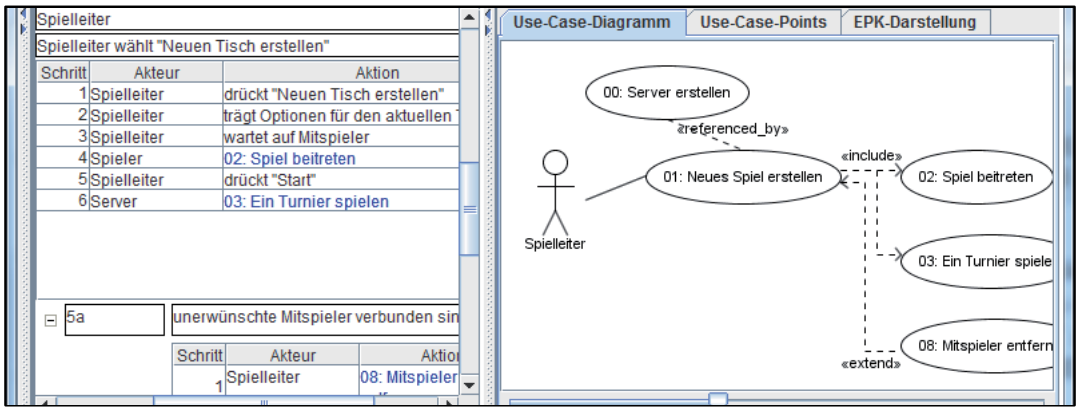


Abbildung 3.7: Generierte UML Sicht auf die Use-Cases in HeRA.

wertvolles grafisches Inhaltsverzeichnis der direkten Umgebung des gerade bearbeiteten Use-Cases.

In [127, 128] wurde untersucht, wie dieser Algorithmus erweitert werden kann. Zum Einen zeigt der erweiterte Algorithmus auch Use-Cases an, in denen der aktuelle Use-Case referenziert wird. Darüber hinaus erlaubt es der erweiterte Algorithmus, Änderungen im generierten Diagramm durchzuführen. Neben der Position der Elemente können Bedingungen an den Erweiterungen, die Titel der Use-Cases und der Name des Hauptakteurs im Diagramm geändert werden. Diese Änderungen werden dann auch für die tabellarischen Use-Cases wirksam.

Dies führt, im Vergleich zum einfachen Algorithmus oben, zu einer erheblich höheren Komplexität, um ein Diagramm automatisch abzuleiten. Die erzeugten Diagramme enthalten potenziell erheblich mehr Use-Cases, dadurch wird das Layout deutlich schwieriger. Zudem sind die Diagramme nicht mehr ohne Weiteres im Hintergrund erstellbar, während der Nutzer tippt. Die Editierbarkeit der Elemente hat sich zudem nicht bewährt, da sie Nutzer daran hindert, schnell zwischen Use-Cases hin und her zu navigieren. Um sowohl Editierbarkeit als auch Navigierbarkeit intuitiv bedienbar anzubieten, muss noch erheblicher Aufwand in die Bedienkonzepte gesteckt werden. Es ist also durchaus möglich, vollständige und syntaktisch korrekte UML Use-Case-Diagramme aus textuellen Use-Cases abzuleiten. Um den Nutzern Feedback zu geben, ist jedoch der einfacher zu bedienende und schnellere Ansatz vorzuziehen.

Das folgende Beispiel zeigt, wie der einfache Algorithmus die Dokumentation von Use-Cases durch computerbasiertes Feedback unterstützt:

Beispiel 3: *Automatisch abgeleitete Use-Case-Diagramme erhöhen die Übersichtlichkeit*

Nachdem sie einige Use-Cases auf Benutzerebene erstellt haben, wollen die Analysten mehr Ordnung in ihre Spezifikation bringen. Dazu gruppieren sie die vorhandenen Use-Cases auf Benutzerebene und weisen sie dem entsprechenden Use-Case auf Geschäftszielebene zu. Dazu werden in den Schritten der Haupt- und Erweiterungsszenarien dieser Use-Cases auf Geschäftszielebene die entsprechenden Use-Cases auf Benutzerebene referenziert.

HeRA leitet schon während dieser Arbeit das entsprechende UML Use-Case-Diagramm ab. Da die Darstellung sich nur auf den aktuellen Use-Case und die dort referenzierten Use-Cases beschränkt, ist sie viel übersichtlicher als die tabellarische Darstellung des textbasierten Use-Cases. Auf einen Blick erhalten die Analysten so Feedback und können entscheiden, ob die dargestellten Use-Cases gut zusammenpassen.

Ein Doppelklick auf einen UML Use-Case im Diagramm führt dazu, dass der entsprechende Use-Case direkt in der Konstruktionskomponente angezeigt wird, so dass die Details nachgelesen werden können.

3.3.4 Darstellung des implizierten Geschäftsprozesses

Sehr oft müssen die von der zu erstellenden Software zu unterstützenden Geschäftsprozesse in der Spezifikation berücksichtigt werden. In einem solchen Fall können Use-Cases auf Benutzerzielebene häufig als Aktivitätsblöcke des Prozesses aufgefasst werden. Die Use-Cases sollten in einem solchen Fall den Geschäftsprozess vollständig abdecken.

Bei einer größeren Menge von Use-Cases wird es schnell unmöglich, die Reihenfolge und Abhängigkeiten zwischen den Use-Cases zu überblicken. Daher bietet HeRA.EPK eine Geschäftsprozess-Visualisierung an. Diese erlaubt im Sinne der Simulationskomponente einer DODE die „was-wenn“-Analyse: Was ergibt sich für ein implizierter globaler Geschäftsprozess, wenn Use-Cases in dieser Form dokumentiert werden.

Der Algorithmus verbindet Use-Cases basierend auf deren Auslöser, Vorbedingungen und Erfolgsgarantien (oder Nachbedingung). Ein Use-Case, der eine bestimmte Bedingung als Vorbedingung benötigt, hängt direkt von einem anderen Use-Case ab, der diese Bedingung als Nachbedingung oder Erfolgsgarantie zusichert. Ein Algorithmus, um eine Menge von Use-Cases als Ereignisgesteuerte Prozesskette (EPK, siehe auch [119]) darzustellen, wurde vollständig in [108, 109] beschrieben. Abbildung 3.8 zeigt ein reales Beispiel aus einem studentischen Projekt.

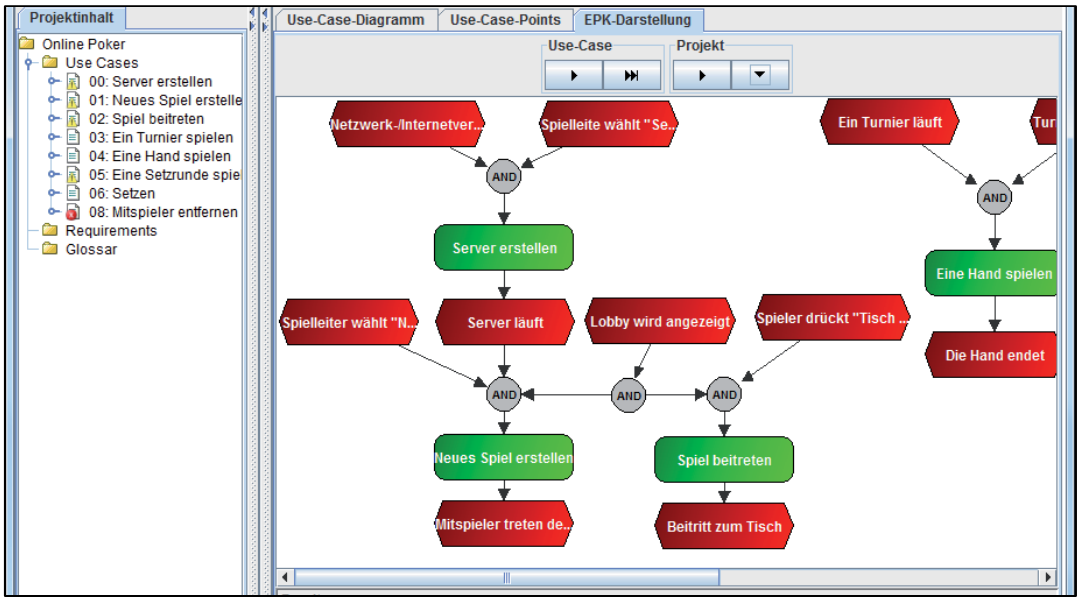


Abbildung 3.8: HeRA kann aus den Auslösern, Vorbedingungen und Erfolgsgarantien der Use-Cases auf Knopfdruck eine EPK ableiten, um den implizierten Geschäftsprozess zu visualisieren. Lücken und Inkonsistenzen fallen mit wenig Übung direkt auf.

Die Visualisierung von Use-Cases als Geschäftsprozess gibt Feedback zum globalen Kontext. Analysten können sehen, ob der globale Kontrollfluss durch die Use-Cases so ist, wie sie es erwarten. Wenn dem nicht so ist, kann dies auf eine Reihe von möglichen Fehlern hindeuten:

Fehlende oder falsche Bedingungen: Auslöser, Vorbedingungen und Erfolgsgarantien eines Use-Cases können falsch sein. Dieser Fehlertyp kann zu einem falschen Verständnis des gewünschten Kontexts beim Entwickler führen und dadurch zu Schwierigkeiten bei der Integration der zu entwickelnden Softwarekomponenten führen. Vor allem die Vorbedingungen sind wichtig, wenn die Aktivitäten in einer bestimmten Reihenfolge durchgeführt werden müssen.

Fehlende Use-Cases: Lücken oder nicht erfüllbare Vorbedingungen in der generierten Geschäftsprozessdarstellung können auf fehlende Use-Cases hindeuten. Wird beispielsweise die Vorbedingung “Lobby wird angezeigt” spezifiziert, dann sollte es auch einen Use-Case mit entsprechender Erfolgsgarantie geben. Auf diese Weise kann die Spezifikation vervollständigt werden.

Abgeleiteter Kontrollfluss passt nicht zum Geschäftsprozess: Widersprüche zwischen dem durch die Use-Cases definierten Kontrollfluss und dem zu unterstützenden Geschäftsprozess resultieren entweder daraus, dass der Ge-

schäftsprozess nicht durchführbare Teile enthält, oder daraus, dass die Use-Cases unpassend sind. Bei beiden Fehlern ist es sehr wertvoll sie frühzeitig zu erkennen.

Darüber hinaus kann der abgeleitete Geschäftsprozess auch genutzt werden, um eine Diskussion der Anforderungen mit verschiedenen Rollen anzuregen (vergleiche [101]).

Das folgende Beispiel verdeutlicht den Nutzen der Möglichkeit, den implizierten Geschäftsprozess schnell ableiten und darstellen zu können:

Beispiel 4: *Die Darstellung des implizierten Geschäftsprozesses offenbart Lücken in den dokumentierten Use-Cases*

Nach der ersten Runde von Interviews will der Projektmanager wissen, wie vollständig die bisher erhobenen Use-Cases bereits sind. Die Analysten wissen, dass einige der Use-Cases schon gut in den zu unterstützenden Geschäftsprozess passen. Sie haben aber auch das Gefühl, dass es noch einige Lücken gibt. Zudem scheinen einige Use-Cases über die ursprüngliche Projektidee hinauszugehen. Sie stoßen deshalb in HeRA schnell die Generierung des Geschäftsprozesses an, um den Kontrollfluss und die Abhängigkeiten zwischen den Use-Cases darzustellen (vergleiche Abbildung 3.8).

Mit etwas Erfahrung lassen sich direkt Stellen identifizieren, an denen Use-Cases fehlen (In Abbildung 3.8: Wie kann jemand einem Spiel beitreten, bevor es gestartet wurde? Wodurch wird erreicht, dass die Lobby dargestellt wird?). Das frühzeitige Wissen über diese Lücken ist wertvoll und erlaubt es, sie durch Rückfragen bei den Stakeholdern zu füllen.

3.3.5 Automatische Berechnung von Use-Case-Points

Die Use-Case-Point-Methode [86] erlaubt es, Projektkosten auf der Basis von Use-Cases abzuschätzen. Es ähnelt den COCOMO und Function Point Methoden und wird auf Basis der Größe des Hauptszenarios und der Anzahl unterschiedlicher Akteure berechnet. Dieser Wert wird mit dem technischen Komplexitätsfaktor (TCF – Technical Complexity Factor) und dem Komplexitätsfaktor für die Projektumgebung (ECF – Environmental Complexity Factor) gewichtet. Die Umrechnung in Personentage geschieht durch den Produktivitätsfaktor, der erfahrungsbasiert bestimmt wird. Zudem werden die Rahmenbedingungen des Projekts mit einbezogen. Abbildung 3.9 zeigt die Eingabemaske für die technische Komplexität in HeRA.

Die Use-Case-Point-Sicht in HeRA ist als Simulationskomponente implementiert (siehe Abbildung 3.1). Genau wie Simulationskomponenten in Fischers ursprüng-

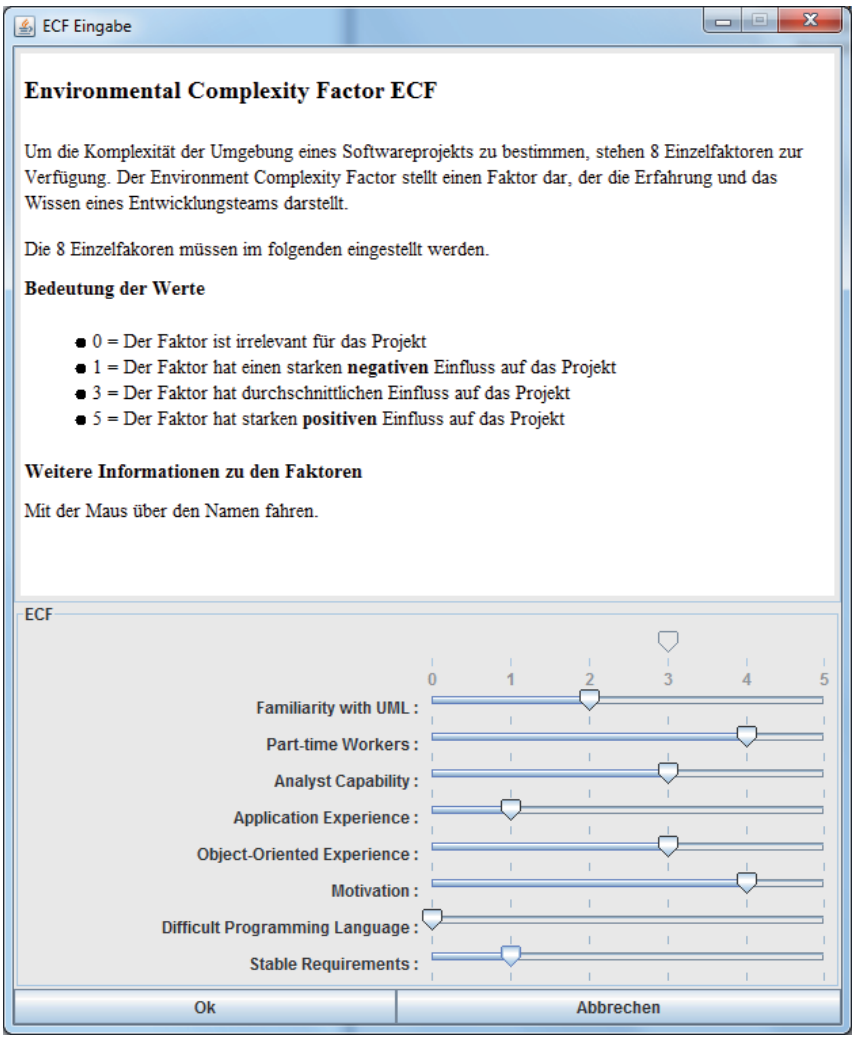


Abbildung 3.9: Für die Berechnung der Use-Case-Points müssen unter anderem Angaben zur technischen Komplexität des Projekts gemacht werden.

lichem Entwurf [64] erlaubt diese Komponente eine *wenn-dann* Analyse auf der Basis des Use-Case-Modells. HeRA ist in der Lage schon sehr früh Schätzungen zu liefern, sogar während die Use-Cases noch spezifiziert werden. Diese Schätzungen sind naturgemäß sehr ungenau. Sie können dennoch für den Analysten wertvoll sein, wenn dieser die Ursache verdächtiger Schätzungen überdenkt. In diesem Fall stehen ihm zwei Möglichkeiten zur Nachbesserung zur Verfügung:

Anpassen des Use-Case-Modells: Durch das Ändern der Abstraktionsebene eines oder mehrerer Use-Cases wird das Ergebnis der Use-Case-Point-Methode stark beeinflusst. Unter Umständen liegt einer der Use-Cases gar nicht auf Benutzerzielebene, sondern auf Unterfunktionsebene.

Ändern der Rahmenbedingungen: Bei der Use-Case-Point-Methode werden bei der Schätzung die Rahmenbedingungen und technische Komplexität des Projekts berücksichtigt. Durch die Änderung dieser Werte (z.B. Beurteilung der Erfahrung des Projektteams oder technischer Risiken), kann der geschätzte Wert ebenso verbessert werden.

Die Anzeige der Use-Case-Points in HeRA (siehe Abbildung 3.10) hilft Autoren die Perspektive der Aufwandschätzung einzunehmen. Nach Trudel und Abran [166] ist dies wertvoll für die Qualität von Anforderungen: Im Rahmen ihrer Forschungsarbeit haben sie die Effektivität von Anforderungs-Reviews je nach Hintergrundwissen der Reviewer gemessen. Dabei wurden hauptberufliche Inspekteure mit Personen verglichen, die hauptsächlich Aufwandsschätzung und Fortschrittskontrolle betreiben. Es wurde gezeigt, dass die Kombination von Reviewern beider Gruppen besonders gute Fehlerentdeckungsraten bringt. Durch HeRA wird es dem Analysten möglich, mit wenig Aufwand aus einem anderen Blickwinkel auf die Anforderungen zu blicken. Oft können dabei bereits vom Autor Auffälligkeiten gefunden und behoben werden, die sonst erst später in einem teuren Review aufgefallen wären.

Das folgende Beispiel zeigt, wie die automatische Berechnung von Use-Case-Points die konsistente Dokumentation von Use-Cases unterstützt:

Beispiel 5: *Automatische Schätzung des Aufwands erlaubt schnelle und häufige Reflexion*

Nachdem einige Use-Cases in HeRA dokumentiert wurden, ist es sinnvoll, in HeRA die Aufwandschätzung berechnen zu lassen. In Abbildung 3.10 ergeben sich aus diesem Feedback einige Fragen: Ist Use-Case 02 wirklich doppelt so teuer wie Use-Case 05? Oder wurde Use-Case 05 einfach noch nicht bis zum gleichen Detailierungsgrad ausgearbeitet? Ist der geschätzte Aufwand von 1260 Arbeitsstunden realistisch? Falls nicht: Stimmt die Einschätzung der technischen Komplexitität und der Umgebungskomplexität?

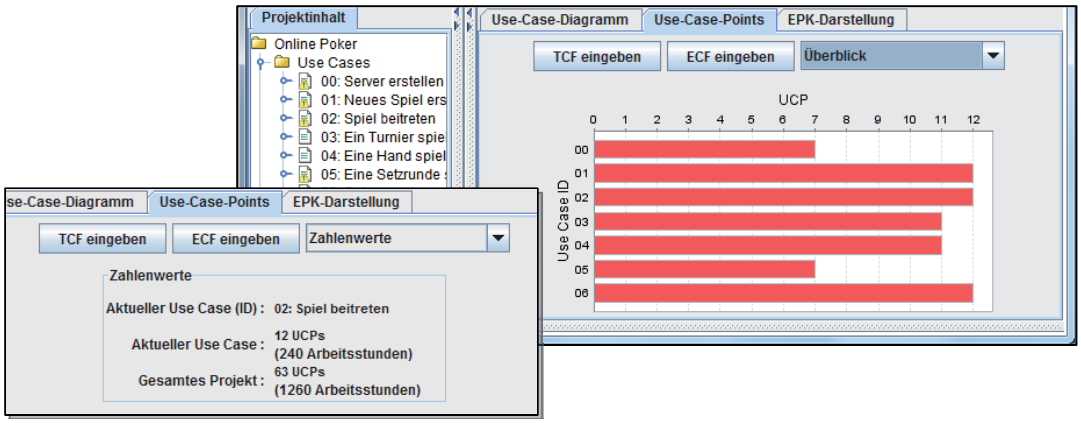


Abbildung 3.10: Über die Use-Case-Point-Metrik gibt HeRA zusätzliches Feedback.

Zur Klärung dieser Fragen empfiehlt es sich, Rücksprache mit Architekten und Projektmanagern zu halten.

3.3.6 Anlegen neuer Erfahrungen

Die bisher vorgestellten Komponenten von HeRA geben Feedback bei der Dokumentation von Anforderungen. Die Komponenten greifen dabei auf Erfahrungen zurück und lassen sich zum Teil auch erfahrungsbasiert verbessern:

Argumentationskomponente für Glossare: Begriffe, die schon einmal in einem anderen Glossar definiert wurden, werden bevorzugt vorgeschlagen. Wird ein Begriff mit Hilfe dieser Komponente in das Glossar eingefügt, wird er auch gleich dieser Erfahrungsbasis hinzugefügt. Zudem werden auch die Filter für die Vorschlaglisten mit der Zeit besser. Wörter, die in einem Projekt ignoriert wurden, werden auch im nächsten Projekte nicht vorgeschlagen.

Automatische Berechnung von Use-Case-Points: Um den Aufwand zu berechnen, wird der Produktivitätsfaktor verwendet. Dieser muss erfahrungsbasiert ermittelt werden, damit die Schätzung genauer wird.

Abgeleitetes UML Use-Case-Diagramm: Für die Nützlichkeit dieser Komponente ist unter anderem die Wahl des richtigen Ausschnitts wichtig. Die Erfahrungen der Nutzer können genutzt werden, um diesen gut zu wählen.

Kritiksystem für Anforderungen: Das Kritiksystem basiert auf einer Menge von heuristischen Kritiken. Nutzer können diese anpassen und sogar neue Kritiken erstellen und so ihre eigenen Erfahrungen in HeRA integrieren.

Diese Möglichkeiten zur erfahrungsbasierten Verbesserung unterscheiden sich zum Teil erheblich. Die Glossar-Komponente *lernt* automatisch, bei den Use-Case-Points ist die Anpassung des Produktivitätsfaktors Teil der Methode. Der angezeigte Ausschnitt der UML Use-Case-Diagramm-Komponente kann nach Feedback durch die Nutzer nur von den HeRA-Entwicklern verbessert werden, die erfahrungsbasierte Verbesserung ist hier also schwierig.

An dieser Stelle soll vor allem noch auf das Hinzufügen neuer Kritiken eingegangen werden, da dies durch den Nutzer von HeRA aktiv geschehen kann.

Für HeRAs Kritiksystem werden die heuristischen Regeln in Javascript[1] formuliert (siehe Abbildung 3.11). Über eine definierte Schnittstelle kann dabei direkt auf das Datenmodell der Konstruktionskomponenten für Use-Cases, Anforderungen und Glossare zugegriffen werden. Es existiert eine Reihe von Basisfunktionen, die verwendet werden können, um kompliziertere Berechnungen durchzuführen. Darüber hinaus gibt es Assistenten, durch die die Erstellung von einfachen Heuristiken weitgehend automatisiert ist. Zusätzlich zu der eigentlichen Heuristik muss eine Beschreibung angegeben werden. Es ist möglich die Heuristik zu parametrisieren. Dies ist bei den meisten bestehenden Heuristiken gemacht worden. So können Nutzer heuristische Kritiken durch einfaches Anpassen der Parameter erweitern. Typische Parameter sind eine Liste von Schlüsselwörtern, deren Auftreten dann in Anforderungen überprüft wird.

Die heuristischen Regeln können zur Laufzeit geändert werden. Das macht es möglich, schnell neue Regeln zu entwickeln, da sie direkt auf einem Beispiel ausprobiert werden können. So wird die Warnung in Abbildung 3.12 angezeigt, sobald der Parameter “bezahlen” oder “internet” in Abbildung 3.11 hinzugefügt wird.

Es gibt einige Arten von typischen Regeln:

Schlüsselwortsuche: Darunter fällt zum Beispiel die Suche nach *Weakwords*[2]. Interessanterweise lässt sich so auch eine recht einfache Heuristik zur Identifikation von Passiv implementieren, indem einfach nach allen Formen von *werden* gesucht wird.

Konsistenz: Regeln, die verschiedene Felder des Use-Cases miteinander abgleichen. Eine Regel in HeRA besagt beispielsweise, dass jeder Akteur eines Use-Cases auch als Stakeholder vorkommen muss.

Analyse der Struktur: Eine Regel dieser Klasse prüft, ob es zu jedem Use-Case, der nicht auf Geschäftszielebene liegt, einen Use-Case auf der nächst höheren

[1] Die Wahl von Javascript ist eher zufällig. Gesucht war eine einfach zu bedienende Skript-Sprache mit ausreichender Mächtigkeit. Bei HeRAs Nutzern hat sich Javascript gut bewährt (vgl. Abschnitt 8.2.4).

[2] Unter Weakwords versteht man Begriffe, die zu Mehrdeutigkeiten in Anforderungen führen können (wie zum Beispiel alle, wichtig, immer, weitgehend, etc.).

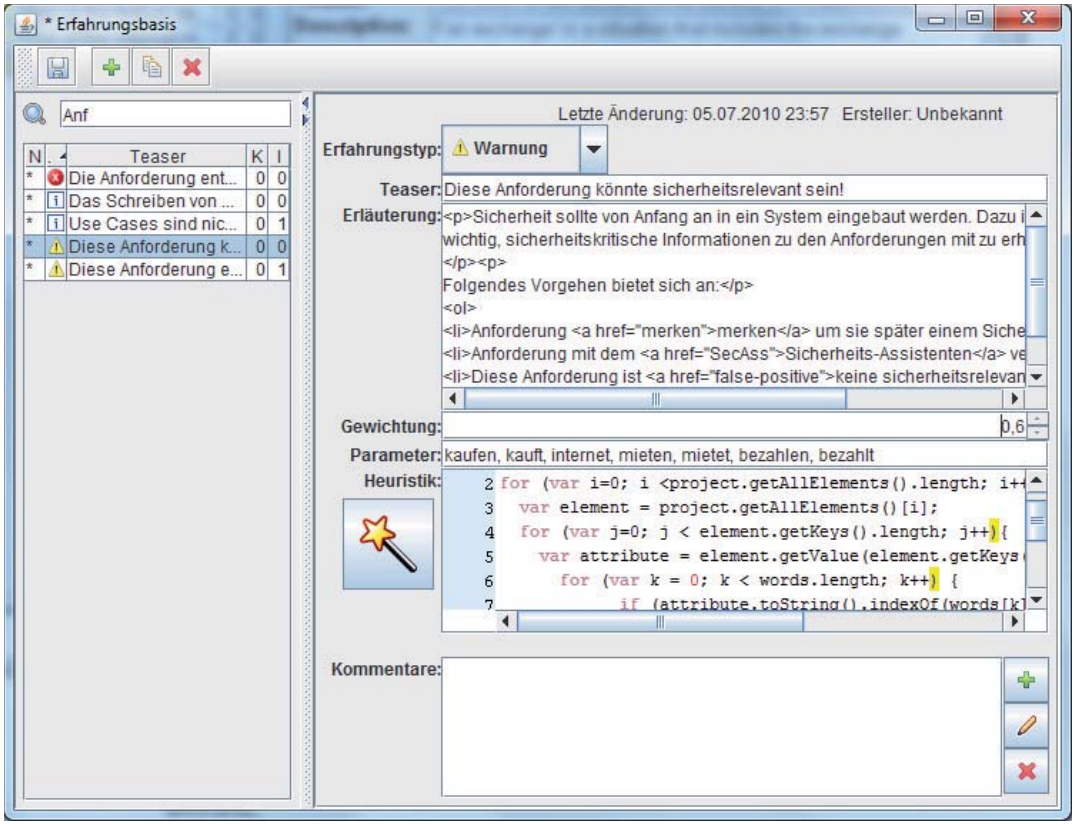


Abbildung 3.11: In HeRAs Erfahrungsbasis können neue Erfahrungen eingegeben werden. Dabei werden alle Aspekte einer Kritik berücksichtigt. Die Beschreibung ist unterteilt in einen kurzen Teaser und eine längere Erläuterung. Die Javascript-Heuristik kann über einen Wizard erzeugt werden.

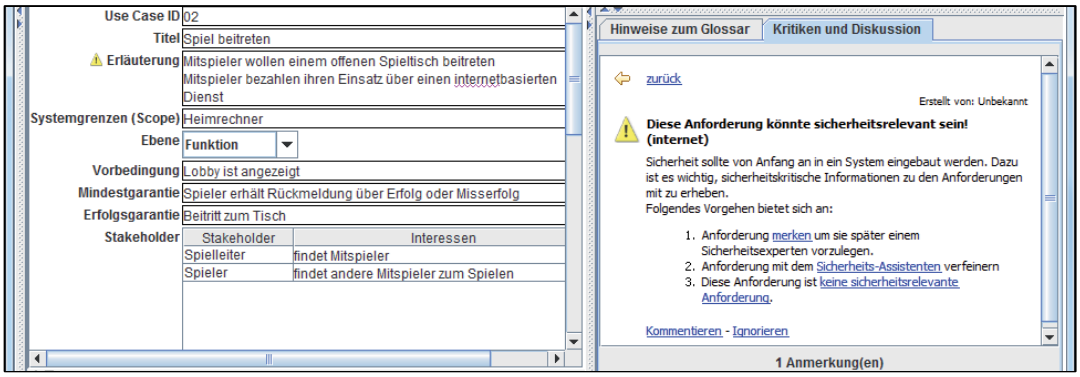


Abbildung 3.12: Die Änderungen an der Erfahrungsbasis werden direkt in HeRA sichtbar. Dies erlaubt es, verschiedene Heuristiken auf einer konkreten Anforderung auszuprobieren.

Ebene gibt, der ihn referenziert. Dadurch wird sichergestellt, dass jeder Use-Case auch durch ein Geschäftsziel motiviert wird.

Damit können Nutzer von HeRA ihre Erfahrungen weitgehend kodieren: Nutzer können beschreiben, dass bestimmte Wörter auf Probleme hindeuten oder nach ihrer Erfahrung nicht benutzt werden sollten. Sie können weiterhin definieren, welche Konsistenzregeln nach ihrer Erfahrung besonders wichtig sind, und wie Anforderungen strukturiert werden sollten.

Das folgende Beispiel zeigt, wie die heuristischen Kritiken von Nutzern angepasst werden können:

Beispiel 6: *Der eingebaute Editor erlaubt es Nutzern die heuristischen Kritiken in HeRA zu ändern*

Nachdem die Analysten von HeRAs Feedback profitiert haben, wollen sie selbst heuristische Kritiken hinzufügen. Neben den allgemeinen Kritiken zu Anforderungen soll HeRA nun auch spezielle Unterstützung zur Erhebung sicherheitskritischer Anforderungen erhalten. Der einfachste Weg dazu ist, HeRAs integrierte *Erfahrungsbasis* zu öffnen und eine ähnliche Kritik zu suchen. So kann beispielsweise die heuristische Kritik zu *Weakwords* dupliziert und die Beschreibung und die Parameter für die Heuristik leicht angepasst werden, damit eine Warnung angezeigt wird, sobald bestimmte Schlüsselwörter verwendet werden. Man kann auch Änderungen an der Implementierung der heuristischen Regel durchführen. In diesem einfachen Fall ist es jedoch nicht nötig den Javascript-Code zu ändern. Zur Kontrolle wechselt man zurück in HeRAs Hauptfenster und fügt eines der Schlüsselwörter in eine Anforderung des gerade geöffneten Projekts ein. Der Analyst kann nun prüfen, ob die Kritik wie gewünscht angezeigt wird.

3.4 Auswirkungen des Fallbeispiels

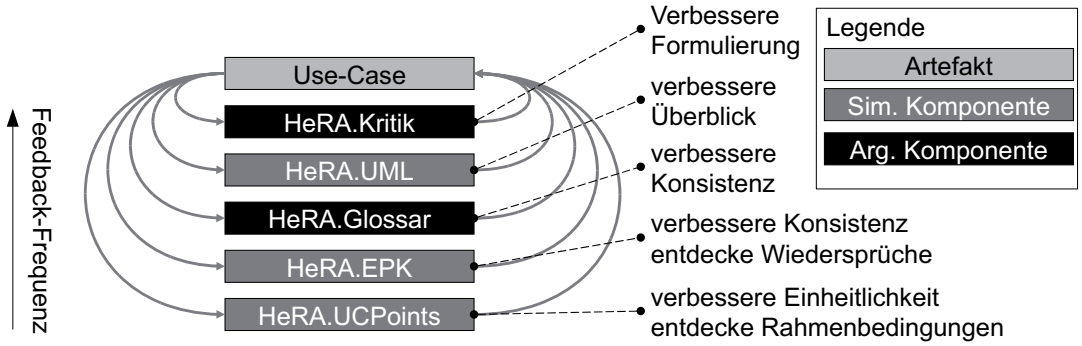


Abbildung 3.13: Computerbasierte Feedbackschleifen in HeRA.

Abbildung 3.13 zeigt die verschiedenen Ebenen, auf denen HeRA Feedback gibt. Über die Kritiken wird direktes Feedback zur Eingabe gegeben, vergleichbar mit einer automatischen Rechtschreibprüfung. Nachdem ein Use-Case bearbeitet wurde, bilden die Vorschläge für das Glossar eine weitere Feedback-Schleife. Ein generiertes UML Use-Case-Diagramm verbessert den Überblick über die modellierten Use-Cases. Falls gewünscht, können die Analysten noch weiteres Feedback anfordern: Eine EPK kann aus einer gegebenen Menge von Use-Cases abgeleitet werden und den implizierten Geschäftsprozess grafisch darstellen. Durch die Use-Case-Point-Schätzung kann zudem der Aufwand, der durch die aktuelle Use-Case-Spezifikation impliziert wird, berechnet werden. Die Geschwindigkeit des Feedbacks fällt von innen nach außen in Abbildung 3.13. Die positiven Effekte dieser Feedbackschleifen werden in Kapitel 8 evaluiert.

Dieses Fallbeispiel zeigt, dass heuristisches und erfahrungsbasiertes Feedback hilfreich ist, um bessere Anforderungsdokumente zu erstellen. In den folgenden Kapiteln werden die Hintergründe dazu näher untersucht. Kapitel 4 zeigt, welche Rollen durch diese Konzepte unterstützt werden und welchen direkten Nutzen Softwareprojekte daraus ziehen können. Zudem wird gezeigt, welche Art von Anforderungsdokumentation sich auf diese Weise verbessern lässt und dass gute Anforderungsdokumentation ein wichtiger Erfolgsfaktor im Projekt ist. Kapitel 5 untersucht die Erfahrungskreisläufe und zeigt, was Organisationen und Individuen mit Hilfe von Feedback bei der Anforderungsdokumentation lernen können. Darauf aufbauend wird dann in Kapitel 8 nachgewiesen, dass dieser Ansatz in verschiedenen Einsatzszenarien funktioniert.

4 Problemfeld: Anforderungsdokumentation

Diese Arbeit beschäftigt sich mit der Verbesserung der Dokumentation von Anforderungen auf der Basis von Erfahrungen und Heuristiken. In diesem Kapitel wird das Problemfeld der Arbeit beschrieben, also die Aspekte der Dokumentation von Anforderungen, die den Rahmen dieser Arbeit definieren. Dies ist eine wichtige Grundlage für die darauf aufbauenden Kapitel. Dokumentation von Anforderungen wird im Rahmen dieser Arbeit wie folgt definiert:

Definition 12: *Anforderungsdokumentation*

Unter *Dokumentation von Anforderungen* versteht man sowohl das Resultat (die Dokumente, in denen Anforderungen enthalten sind), als auch die Tätigkeit und den Prozess ihrer Erstellung.

Diese mehrdeutige Definition spiegelt wieder, dass Erfahrungen und Heuristiken bereits bei der Erstellung von Dokumenten einfließen, um bessere Dokumente zu erhalten. Die Verbesserung von Anforderungsdokumentation berührt also sowohl die Aktivität als auch das Resultat.

Um diese Mehrdeutigkeit vermeiden zu können, legen die folgenden beiden Definitionen sich jeweils auf Aktivität- und Produktbeschreibung fest.

Definition 13: *Anforderungen dokumentieren*

Die Aktivität, Anforderungen in einem Dokument festzuhalten.

Definition 14: *Anforderungsdokument*

Das Dokument, das Anforderungen enthält.

In Abschnitt 4.1 werden Rollen und Nutzertypen charakterisiert, die mit Anforderungsdokumenten in Softwareprojekten in Berührung kommen. Um eine Verbesserung von Anforderungsdokumentation zu erreichen, muss man Qualitätssicherer auf eine andere Art und Weise unterstützen als Analysten. Zudem ist die persönliche Erfahrung des Nutzers ein entscheidendes Merkmal, genau wie dessen Bereitschaft, die erfahrungsbasierten Konzepte dieser Arbeit zu nutzen.

In Abschnitt 4.2 werden die im Rahmen dieser Arbeit wesentlichen Aspekte von Anforderungsdokumentation beschrieben: Reife und Formalität von Dokumenten.

Beide Aspekte können auf einer Skala abgebildet werden, in die sich mehr oder weniger formale oder reife Dokumente einordnen lassen. Dies dient der Abgrenzung der Ergebnisse dieser Arbeit zu anderen Arbeiten: Die Klasse reifer und formaler Anforderungsdokumente ist nicht der Kontext dieser Arbeit. Für diese Klasse existieren mit Simulation und Model-In-the-Loop (MIL) bereits mächtige Mechanismen, um gute Qualität zu erreichen [129, 35]. Im Gegensatz dazu sollen die Konzepte in dieser Arbeit auf dem Weg zu derart reifer und formaler Anforderungsdokumentation helfen: Heuristiken können sinnvoll auf unvollständigen und informalen Dokumenten angewendet werden.

Basierend auf den Konzepten der Reife und Formalität von Anforderungsdokumenten wird in Abschnitt 4.3 experimentell gezeigt, dass die Qualität von Anfordungsdokumenten einen starken Einfluss auf den Projekterfolg hat. Dies ist die zentrale Motivation für die vorliegende Arbeit: Um die Chancen auf einen Projekterfolg zu erhöhen, soll die Anforderungsdokumentation verbessert werden.

Abschnitt 4.4 fasst die Ergebnisse dieses Kapitels zusammen.

4.1 Verbesserung von Anforderungsdokumentation

Ziel dieser Arbeit ist die Verbesserung der Dokumentation von Anforderungen. Dafür ist eine präzise Definition von Verbesserung nötig.

Definition 15: ***Verbesserung von Anforderungsdokumentation***

1. Verbessern der Nützlichkeit eines Anforderungsdokuments.
2. Senken der Kosten der Erstellung eines Anforderungsdokuments, ohne die Nützlichkeit des Dokuments zu senken.

In Abschnitt 4.1.1 wird gezeigt, welche Rollen von guter Anforderungsdokumentation profitieren. Es liegt Nahe, die Verbesserung von Anforderungsdokumentation aus der Perspektive dieser Rollen zu beurteilen. Wie gut diese Verbesserung gelingen kann, hängt auch vom Typ des Nutzers ab. Charakteristika von Nutzern werden in Abschnitt 4.1.2 diskutiert.

4.1.1 Betroffene Rollen

Requirements Engineering ist für verschiedene Rollen in einem Projekt eine wichtige Voraussetzung und Grundlage (z.B. Projektmanager, Architekten, Entwickler, Tester, Qualitätsbeauftragter). Diese Rollen profitieren jedoch nicht allein von Verbesserungen beim Requirements Engineering: Der Kunde profitiert, indem die

Wahrscheinlichkeit steigt, dass sich das Projekt auszahlt. Der zukünftige Nutzer profitiert, indem das zu erstellende Produkt nützlich ist und nicht zuletzt profitiert auch der Analyst von der besseren Unterstützung seiner Arbeit. Dieser Abschnitt gibt einen Überblick über die Aspekte, die für jede dieser Rollen wichtig sind (zusammengefasst in Tabelle 4.1).

Royce [140] hat durch seine Arbeiten die Sicht auf die grundlegenden Aktivitäten in einem Softwareprojekt entscheidend geprägt. Bekannt geworden ist vor allem das oft kritisierte Wasserfallmodell. Das wichtige Ergebnis dieser Arbeiten ist die Unterteilung von Softwareprojekten in prinzipiell zu unterscheidende Aktivitäten: Die Analyse, der Entwurf, die Implementierung, das Testen und die Inbetriebnahme. Neuere Ansätze wie das V-Modell XT [81], eXtreme Programming [17], Spiralmodell [30] und Scrum [154] geben mehr Details vor, ändern die Abfolge oder Vorgehensweise der Phasen. Sie alle lassen sich aber mit den Aktivitäten aus dem Modell von Royce beschreiben. Um die typischen Rollen eines Softwareprojekts zu definieren, gibt Tabelle 4.1 zu jeder Phase nach Royce eine hauptverantwortliche Rolle an. Aus dem Requirements Engineering kommen noch drei wesentliche zusätzliche Rollen: Der Kunde (bezahlt das Projekt), der Nutzer (Nutznießer der Ergebnisse des Projekts) und andere Stakeholder (weitere, durch das Projekt betroffene Personen). Letztere haben in der Regel einen sehr spezifischen Bezug zu Anforderungsdokumentation und sind daher in Tabelle 4.1 nicht berücksichtigt. Besonderes Augenmerk verdienen zudem noch Rollen, die die Durchführung des Projekts wesentlich sind [148]: Der Projektmanager und die Qualitätssicherung. Letztere Rolle kann als Verallgemeinerung des Qualitätsbeauftragten gesehen werden und hebt sich durch die kontinuierliche und durchgängige Begleitung des Projekts ab. Die dritte Spalte in 4.1 gibt an, wie diese Rollen mit Anforderungsdokumentation in Berührung kommen.

Im Mittelpunkt steht das Dokumentieren von Anforderungen durch den Analysten. Um die Dokumente zu erstellen muss der Analyst Anforderungen zunächst erheben. Dafür ist die Mitarbeit des Kunden sowie der späteren Nutzer erforderlich. Der Analyst muss die so erhobenen Anforderungen zudem verstehen. Dieses Ziel teilt er mit Architekten und Entwicklern. Für die konsistente Dokumentation müssen Widersprüche und Konflikte gelöst werden. Je nach Vorgehensmodell kann die Dokumentation formal oder inhaltlich geprüft werden. Tracing und Change Management schlagen sich selbst wieder in geeigneter Dokumentation nieder. Da Requirements Management nicht Fokus der Arbeit ist werden Tracing und Change Management hier nicht weiter vertieft.

Aus Sicht der Rollen, die von Anforderungsdokumentation profitieren, kann nun beurteilt werden, was eine Verbesserung darstellt. Tabelle 4.2 zeigt Ziele und typische Kostentreiber im Requirements Engineering. Ein Werkzeug, das bei der Unterstützung der Elicitation ansetzt, müsste demnach zu besserer Vollständigkeit der Anforderung führen oder die Zeit beim Kunden bei gleichem Ergebnis verkürzen, um eine Verbesserung zu erreichen.

Tabelle 4.1: Stakeholder von Anforderungsdokumentation: Grundlegende Rollen eines Softwareprojekts (nach Projektphasen in [140]; zusätzliche Phasen sind *kursiv gesetzt*).

Phase	**Rolle**	**Bezug zur Anforderungsdokumentation**
Analyse	Analyst	Erstellt das Anforderungsdokument im Rahmen der Anforderungsanalyse.
Design	Architekt	Erstellt einen passenden Entwurf auf Basis der Anforderungsdokumentation; wichtig sind Prioritäten und nicht-funktionale Anforderungen, um fundierte Entwurfsentscheidungen zu fällen.
Implementierung	Entwickler	Setzt die dokumentierten Anforderungen um; wichtig ist, dass Details und Behandlung von Ausnahmefällen spezifiziert sind.
Test	Qualitätsbeauftragter	Testet die Software auf Basis der dokumentierten Anforderungen; wichtig ist, dass Abnahmekriterien für alle Anforderungen vorliegen.
Abnahme	*Kunde*	Prüft, ob das Anforderungsdokument alle geforderten Anforderungen enthält; prüft, ob das Produkt alle geforderten Anforderungen umsetzt; wichtig ist, dass die Anforderungen für den Kunden verständlich und unmissverständlich spezifiziert sind.
Nutzung	*Nutzer*	Gibt Feedback, ob ein System mit den dokumentierten Anforderungen im Betrieb nützlich wäre; wichtig ist, dass die Anforderungen konkret und für den Nutzer verständlich und unmissverständlich spezifiziert sind.
Projektbegleitend	*Projektmanager*	Plant, überprüft und steuert die Erstellung der Anforderungsdokumentation; plant, überprüft und steuert das restliche Projekt auf Basis der dokumentierten Anforderungen; wichtig ist, dass die Anforderungen sich gut verwalten lassen.
Projektbegleitend	*Qualitätssicherung*	Überwacht die Qualität der zu erstellenden Anforderungsdokumente und darauf aufbauender Artefakte.

Tabelle 4.2: Nutzen und Kostentreiber bei den Aktivitäten der Requirements Analysis und des Requirements Management.

Aktivität	**Ziel der Dokumentation**	**Kostentreiber**
Elicitation	Anforderungen vollständig erheben, trotz stillschweigender Anforderungen und *Symmetry of Ignorance**	Kundenzeit
Interpretation	Zusammenhänge, Implikationen verstehen	Fehlinterpretationen
Negotiation	Widersprüche finden und auflösen	Lange unentdeckte Widersprüche, Kundenzeit
Documentation	Gute, persistente Sicherung der Anforderungen	Umfang, Mehrdeutigkeit, Unangemessen f. Zielgr.
Verification	Fehlerfreie Anforderungs-dokumentation	Dauer des Reviews, Umfang der Doku., geringe Formalität der Anf.
Validation	Richtige Anforderungen dokumentiert	Für Kunden unverständl. Anf., Kundenzeit, Umfang
Tracing	Nachvollziehbarkeit	Pflege, falsche und fehlende Links
Change Management	Kontrolle über Änderungen	schleichendes Wachsen durch unpräzise Anforderungen

* *Symmetry of Ignorance* bezeichnet die gegenseitige Unwissenheit zwischen Entwickler und Fachexperten [61].

Tabelle 4.3: Nutzen und Kosten von Requirements Engineering für Aktivitäten des Projektmanagement und der Qualitätssicherung.

Aktivität	**Ziel der Dokumentation**	**Kostentreiber**
Planen	Solider, belastbarer Projektplan	Lange Meetings, unsyst. Dokumentation von Anf.
Messen (RE)	Guter Überblick über Fortschritt des RE	Unklarer Status der Anforderungen
Messen (Projekt)	Projektfortschritt über erfüllte Kundenwünsche	Unsystematische Dokumentation
Steuern (Projekt)	Richtige Entscheidung unter Unsicherheit	Mangelhafte Entscheidungsgrundlage
Validation abh. Artefakte	Architektur und Umsetzung passend zu Anforderungen	Mangelnde Referenzierbarkeit von Anforderungen
Abnahmetests	Entwickeltes System entspricht Anforderungen	Mangelnde Testbarkeit der Anforderungen

Entsprechend kann auch definiert werden, was aus Sicht des Projektmanagements und des Qualitätsbeauftragten die Nützlichkeit von Anforderungsdokumentation ausmacht. Tabelle 4.3 zeigt die Ziele von Dokumentation aus Sicht eines Projektmanagers oder Qualitätsbeauftragten. Beispielsweise macht es systematische Dokumentation von Anforderungen leichter, den Aufwand für ein Projekt einzuschätzen, da Anforderungen leichter eingeschätzt werden können. Die Möglichkeit, automatisch Berichte zu erzeugen, würde die Kosten für die Aufwandschätzung senken.

Interessant ist hier, dass höhere Formalität die Verifikation unterstützt, während für die Validierung mit dem Kunden eine bessere Verständlichkeit nötig ist (siehe auch Abschnitt 4.2). Eine Verbesserung könnte in diesem Fall die Unterstüzung dabei sein, diesen scheinbaren Widerspruch aufzulösen. Dies kann beispielsweise gelingen, indem formale Modelle automatisch aus den natürlichsprachlichen Anforderungen abgeleitet werden (wie zum Beispiel in [7]).

4.1.2 Typen von Nutzern

Für die Diskussion des Verbesserungspotenzials für Anforderungsdokumentation von heuristischen und erfahrungsbasierten Ansätzen ist neben den betroffenen Rollen auch eine Diskussion der Personen wichtig. In diesem Abschnitt wird ein

Modell potenzieller Nutzer gegeben, an denen erfahrungsbasierte und heuristische Werkzeuge ausgerichtet werden können. Wesentliche Merkmale des Modells sind die Eigenschaften *Erfahrung* und *Wohlwollen*.

Erfahrung von Nutzern

Heuristische Kritiken dienen dazu, Erfahrungen im RE zu nutzen. Damit ist die Erfahrung, die ein Nutzer selbst mitbringt, ein wichtiges Unterscheidungsmerkmal: Einem Neuling kann man auch mit grundlegendem Wissen noch weiterhelfen, einen sehr erfahrenen Analysten würde man mit solchen Trivialitäten eher abschrecken.

Erfahrene Nutzer sollten vor allem in die Lage versetzt werden, ihre Erfahrungen auch einzusetzen. Eine gute Aufbereitung vorhandener Information kann hier sehr hilfreich sein. Bevormundung durch stark interpretierendes heuristisches Feedback wird eher auf Ablehnung stoßen.

Erfahrene Nutzer sind für alle erfahrungsbasierten Ansätze interessant, weil sie eine wertvolle Quelle für neue Erfahrungen sind. Dazu müssen sie dem Ansatz positiv gegenüberstehen.

Einstellung

Neben der Erfahrung des Nutzers ist zudem wichtig, ob dieser Kritik von einem Computersystem offen gegenübersteht, oder dies ablehnt. Dies hängt von vielen unterschiedlichen Dingen ab:

Grundeinstellung des Nutzers. Manche Nutzer stehen heuristischen Kritiken grundsätzlich skeptischer gegenüber als andere. Dies sei am Beispiel der automatischen Rechtschreibprüfung illustriert, die in immer mehr Texteditoren standardmäßig aktiviert ist. Manche Nutzer arbeiten sehr gerne mit der automatischen Rechtschreibprüfung Andere Nutzer stört es, dass sie dadurch in ihrem Arbeitsablauf unterbrochen werden.

Jemanden, der solches pro-aktives Feedback ablehnt, sollte man die Möglichkeit geben, dieses Feedback abzuschalten und erst bei Bedarf zu aktivieren. Die Präsentation der Kritik muss sich also entweder an den Bedürfnissen und Vorlieben der Nutzer orientieren, oder sich zumindest daran anpassen lassen.

Historie und Kontext des Nutzers. Das direkte Umfeld des Nutzers kann ebenfalls einen großen Einfluss auf die Akzeptanz heuristischer Kritiken haben.

- *Motivation des Nutzers.* Will der Nutzer überhaupt gute Arbeitsergebnisse abliefern? Um welchen Preis?
- *Zeitdruck.* Kann der Nutzer gute Anforderungsdokumente erstellen, oder lässt dies der aktuelle Zeitplan womöglich gar nicht zu? Ein Nutzer, der sich hohem Zeitdruck ausgesetzt fühlt, hat vermutlich nicht die Ruhe, sich mit computerbasiertem Feedback zu beschäftigen, selbst wenn dieses seine Arbeit beschleunigen kann.
- *Vertrauen.* Vertraut der Nutzer darauf, dass die heuristischen Kritiken ausschließlich eingesetzt werden, um ihn bei der Erstellung guter Arbeitsergebnisse zu unterstützen? Wenn der Nutzer befürchten muss, dass die Ergebnisse heuristischer Prüfung in seine Bewertung durch einen Vorgesetzten eingeht, so wird er entsprechenden Ansätzen skeptisch bis ablehnend gegenüberstehen.
- *Nützlichkeit der Kritik.* Wenn der Nutzer bereits von heuristischer Kritik profitiert hat, wird dies seine Haltung gegenüber entsprechenden Ansätzen vermutlich positiv beeinflussen. Andersherum kann unnütze Kritik auch das Vertrauen zerstören. Wieviel unnützes Feedback für einen Nutzer erträglich ist, hängt auch wieder mit der Präsentation zusammen, zum Beispiel wie leicht es dem Nutzer fällt, dieses zu ignorieren.

Klassifikation von Nutzern

Abbildung 4.1 zeigt die Eigenschaften *Erfahrung* und *Grundeinstellung* in einer Portfolio-Darstellung. In jedem Quadrant ist ein Stereotyp dargestellt, der als Platzhalter für Nutzer mit dieser Charakteristik dient. Den größten Nutzen bringen erfahrungsbasierte und heuristische Ansätze dem *Neuling* im linken, oberen Bereich (wohlwollend und unerfahren): Diese Nutzer stehen heuristischen Ansätzen aufgeschlossen gegenüber. Sie können von dem Feedback profitieren und viel lernen. In manchen Fällen können die heuristischen Kritiken während diesem Lernprozess auch um neue Erfahrung erweitert werden.

Nutzer im rechten oberen Quadrant sind erfahren und stehen heuristischen Ansätzen positiv gegenüber, hier repräsentiert durch den *Trainer*. Das Verbesserungspotenzial ist hier geringer als beim *Neuling*, weil man dem *Trainer* nicht mehr so viel Neues bieten kann. Wenn der *Trainer* sich mit erfahrunsbasiertem und heuristischem Feedback auseinandersetzt, besteht dafür jedoch hohes Verbesserungspotenzial an den heuristischen Kritiken. Ansätze, die das Erfahrungswissen des *Trainers* aufnehmen können, haben bei dieser Nutzerklasse hohes Potenzial.

Der *alte Hase* im rechten unteren Quadranten wird durch heuristisches und erfahrungsbasiertes Feedback ebenfalls nicht viel Neues lernen. Da der *alte Hase*

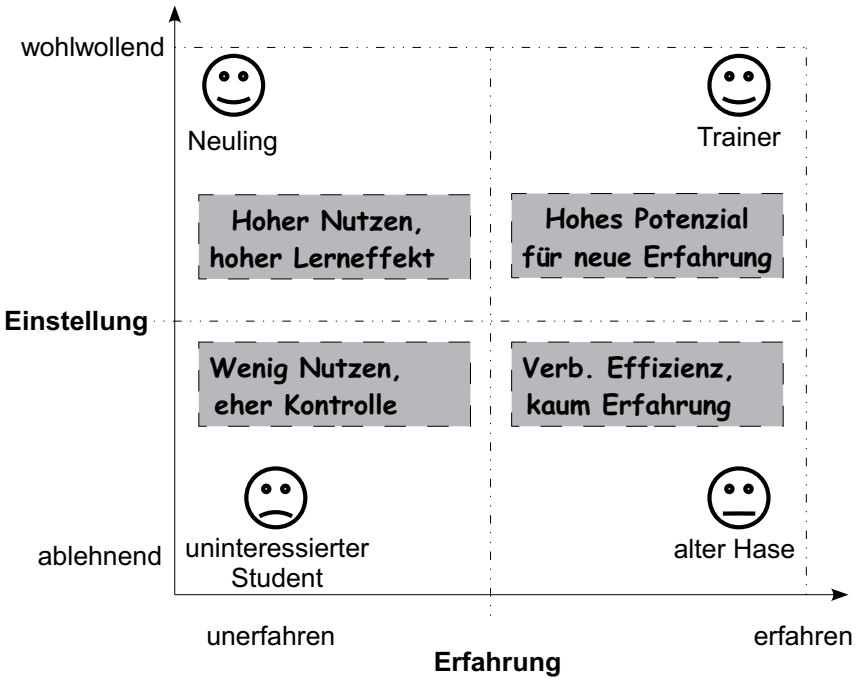


Abbildung 4.1: Klassifikation von Nutzern: Stereotypen und Potenzial für erfahrungsbasierte und heuristische Ansätze im Requirements Engineering.

diesen Ansätzen eher ablehnend gegenübersteht, wird es kaum möglich sein, von seinen Erfahrungen zu lernen. Der Nutzen von heuristischem Feedback liegt hier vor allem in der Steigerung der Effizienz, in dem dieser Nutzerklasse Arbeit abgenommen oder erleichtert wird.

Der *uninteressierte Student* bringt wenig Erfahrung für die Dokumentation von Anforderungen mit, hat aber auch kein Interesse dies zu ändern. Heuristisches Feedback wird von dieser Nutzerklasse ignoriert, kann aber von Betreuern zur Kontrolle und Fehlersuche verwendet werden. Der Versuch, hier neue Erfahrungen zu gewinnen, ist zum Scheitern verurteilt und kann die Qualität der Erfahrungsbasis gefährden, wenn neue Erfahrungen beispielsweise aus der Beobachtung des Verhaltens des uninteressierten Studenten abgeleitet werden.

Die Komponenten von HeRA zielen auf verschiedene Gruppen von Nutzern ab. HeRA.Kritik ist vor allem auf die linken und oberen Quadranten ausgerichtet. Das liegt zum Einen an dem Inhalt der Kritiken, zum Anderen an der Art der Präsentation. *Alte Hasen* könnten sich hier leicht bevormundet fühlen.

HeRA.Glossar kann für *Neulinge*, *Trainer* und *alte Hasen* hilfreich sein: Diese Komponente erleichtert es dem alten Hasen, Glossarbegriffe anzulegen und gibt

dem Neuling Orientierung. Der *uninteressierte Student* kann durch zufälliges Hinzufügen von Begriffen die Erfahrungsheuristik dieser Komponente schwächen.

HeRA.EPK ist eher für Nutzer in den beiden rechten Quadranten hilfreich, weil für die Interpretation Erfahrung nötig ist. Das gilt auch für die Use Case Points. Das abgeleitete UML Use-Case-Diagramm schadet in keinem der Quadranten, da es weder aufdringlich noch kompliziert ist.

4.2 Modell der Anforderungsdokumentation

Um zu untersuchen, wie Anforderungsdokumentation mit Erfahrungen und dem Einsatz von Heuristiken verbessert werden kann, müssen Dokumenttypen von Anforderungen genauer betrachtet werden. Im Requirements Engineering finden zum Teil sehr unterschiedliche Dokumenttypen Verwendung und nicht alle Typen können gleich gut durch heuristische Regeln unterstützt werden. Dieser Abschnitt führt den Grad der Formalität und Reife von Dokumenten als relevantes Unterscheidungsmerkmal ein. Damit werden in diesem Abschnitt folgende Ziele verfolgt:

1. *Definition relevanter Eigenschaften der auf Basis von Erfahrungen und Heuristiken zu verbessernden Anforderungsdokumente:* Über die in den Grundlagen gezeigten Eigenschaften von Anforderungsdokumenten hinaus sind Formalität und Reife notwendig, um die Ansatzpunkte dieser Arbeit zu beschreiben.
2. *Abgrenzung zu formaler Verifikation:* Formale Verifikation benötigt reife und formale Dokumente. Erfahrungen und Heuristiken sind bereits nützlich, bevor die dafür erforderliche Reife und Formalität erreicht wird.

Der Formalitätsbegriff geht auf die Forderungen von Pohl zurück, dass alle Anforderungen konform zu den Dokumentationsvorschriften dokumentiert sein müssen [131]. Diese Dokumentationsvorschriften schreiben eine mehr oder weniger formale Beschreibung von Anforderungen vor. Die beiden Formalitätsstufen *natürlichsprachliche Anforderungen* und *modellbasierte Anforderungen* aus [131] sind hier allerdings nicht ausreichend, um das Problemfeld dieser Arbeit zu beschreiben. Zum Einen sind aus Sicht der heuristischen Überprüfbarkeit kaum relevante Unterschiede zwischen stark strukturierten natürlichsprachlichen Anforderungen und Modellen zu finden. Zum Anderen muss diese Formalität im Laufe des Projekts erst erreicht werden. Es reicht also nicht, die Formalität vorzuschreiben, die das fertige Anforderungsdokument haben soll. Vielmehr muss im Rahmen dieser Arbeit auch der Weg beschrieben werden, auf dem diese Formalität erreicht wird. Erfahrungen und Heuristiken unterstützen schließlich die Aktivität der Erstellung eines Anforderungsdokuments.

4.2.1 Formalität von Anforderungsdokumenten

In vielen Projekten gibt es ein zentrales Anforderungsdokument für das zu entwickelnde System. In diesem Dokument werden alle Anforderungen für das Projekt gesammelt. Typischerweise besteht ein solches Dokument aus einer Einleitung, die die Projekt-Vision enthält und einen Überblick gibt. Zudem gibt es Abschnitte für funktionale Anforderungen, Qualitätsanforderungen, Anforderungen an die Benutzerschnittstelle und weitere Typen von Anforderungen (vergleiche zum Beispiel [3, 139, 81]).

Die individuelle Erfahrung eines Autors ist bei der Erstellung dieser Inhalte generell hilfreich. Prozess- und dokumentzentrierte Vorgehensmodelle profitieren zudem von strukturierten Dokumenten und Vorlagen (Templates). Über verwendete Vorlagen können Erfahrungen gut in neue Projekte einfließen. Dabei kann durch erfahrungsbasiertes Tailoring die Struktur oder durch Bearbeitungshinweise der Inhalt verbessert werden [102].

Die Art und Weise, wie Anforderungen dokumentiert werden, hängt zum Einen vom gewählten Vorgehensmodell und der verwendeten Vorlage, zum Anderen von der Art der Anforderungen ab. Je nach Vorgehensmodell liegen für Anforderungsdokumente sehr unterschiedliche Vorlagen vor (siehe Abbildung 4.3 für eine Beispiel-Vorlage). Auch die Art der darin dokumentierten Anforderungen hat einen großen Einfluss auf die Form der Dokumentation. So werden funktionale Anforderungen in der Regel ganz anders dokumentiert als Qualitätsanforderungen. Gute Vorlagen definieren diese Form für jeden Typ von Anforderungen.

In der Regel werden Anforderungen in einem Dokument mittels einer Sammlung verschiedener Modellarten dokumentiert. Dabei lassen sich in gängigen Vorlagen durchaus Gemeinsamkeiten und Standardbestandteile identifizieren [3, 139, 81]. Üblicherweise unterscheidet sich der Formalitätsgrad der einzelnen Abschnitte solcher Dokumente sehr stark. Die Projekt-Vision ist naturgemäß deutlich abstrakter und wird daher weniger stark formalisiert als die Beschreibung der funktionalen Anforderungen.

Um ein Maß für die Formalität von Anforderungsdokumenten zu erhalten, muss zunächst deren Vergleichbarkeit hergestellt werden. Für ein solches allgemeines Maß der Formalität bräuchte man eine Normalform für Anforderungsdokumente, in der sich jeder Aspekt eindeutig zuordnen lässt. Die Aspekte dieser Normalform lassen sich theoretisch folgendermaßen festlegen:

Für jede existierende Anforderungsdokumentvorlage werden die dort addressierten Aspekte analysiert. Kann ein Aspekt nicht eindeutig einem bereits identifizierten relevanten Aspekt der Normalform zugeordnet werden, gibt es zwei Möglichkeiten: Ist er noch gar nicht in den Aspekten der Normalform enthalten, wird er aufgenommen. Überschneidet er sich mit einem oder mehr Aspekten der Normal-

form, müssen diese Aspekte weiter verfeinert werden, bis sie eindeutig aufeinander abzubilden sind. Der Vergleich der Formalität von Dokumenten mit unterschiedlichen Vorlagen kann dann stattfinden, indem für jeden Aspekt verglichen wird, wie formal dieser beschrieben wird.

Inhalte eines Anforderungsdokuments in dieser imaginären Normalform beschreiben jeweils einen Aspekt von Anforderungen. Die Formalität der Beschreibung eines Aspekts in einem Dokument kann mit der Formalität der Beschreibung des selben Aspekts in einem anderen Dokument verglichen werden. Schwieriger ist dies, wenn die zu vergleichenden Dokumente unterschiedliche Vorlagen verwenden. In diesem Fall kann es sein, dass sich Aspekte nicht eindeutig aufeinander abbilden lassen. Als Ausweg bleibt, die Aspekte feiner zu fassen, so dass sie sich eindeutig aufeinander abbilden lassen.

Auf Grundlage dieses Gedankenmodells kann nun ein grobes Metamodell für Anforderungsdokumente aufgebaut werden. Dieses ist in Abbildung 4.2 dargestellt. Das Modell zeigt, dass ein Anforderungsdokument aus mehreren Aspekten besteht. Jeder Aspekt wird durch eine Anforderungsbeschreibung dokumentiert: Entweder durch textuelle Anforderungen oder durch ein Anforderungsmodell.

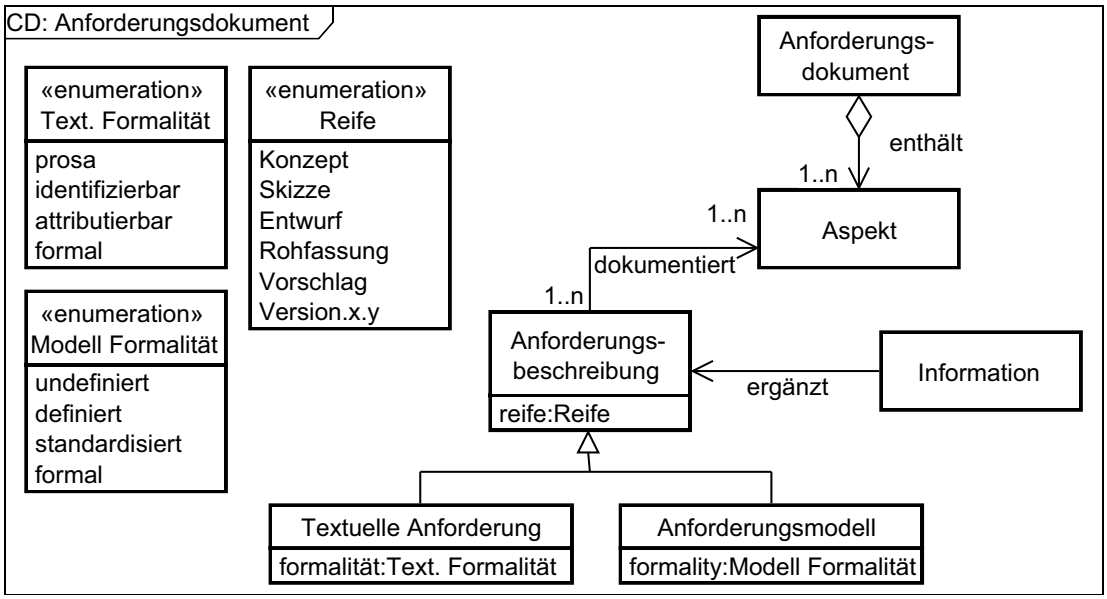


Abbildung 4.2: Modell eines Anforderungsdokuments.

Die Beschreibung eines Aspekts kann als Modell aufgefasst werden. Relevante Eigenschaften des Aspekts des zu erstellenden Systems (z.B. funktionale Anforderungen) sollen festgehalten werden. Um die Formalität dieser Beschreibungen (oder Modelle) zu charakterisieren, macht es Sinn, grafische und textuelle Modelle

zu unterscheiden. Textuelle Beschreibungen können in verschiedenen Formalitätsstufen vorliegen.

- *Prosa.* Ein Textabschnitt zu einem Aspekt enthält eine oder mehrere nicht weiter voneinander isolierte Anforderungen.
- *Identifizierbar.* In der Beschreibung eines Aspekts sind Anforderungen vereinzelt und eindeutig identifizierbar.
- *Attributiert.* Anforderungen zu einem Aspekt besitzen Attribute mit Metainformationen. Sie sind nach Attributwert sortierbar und filterbar.

Formalität von grafischen Modellen von Anforderungsaspekten lässt sich wie folgt charakterisieren:

- *Undefiniert.* Gerade in den frühen Phasen eines Softwareprojekts, wie bei der Erhebung von Anforderungen, kommen häufig Skizzen ohne definierte Syntax und Semantik zum Einsatz.
- *Definiert.* Syntax und Semantik des Modells sind festgelegt.
- *Standardisiert.* Syntax und Semantik sind über das Anforderungsdokument hinaus in einem Standard vorgegeben, wie zum Beispiel in der UML Spezifikation [124, 125].

Die Standardisierung erlaubt es, heuristische Regeln auf Basis der Syntax zu definieren und über den Rahmen des aktuellen Dokuments hinaus zu verwenden. Prinzipiell ist dies auch bei textuellen Anforderungsbeschreibungen erwünscht. Heuristische Regeln funktionieren nur dann, wenn die Attribute, auf die sie sich beziehen, auch vorhanden sind. Bei textuellen Anforderungen ist die Haupthürde jedoch die Attribute maschinenlesbar aus dem Text zu extrahieren, so dass heuristische Regeln über Werkzeug- und Vorlagen-Grenzen hinweg nur schwer wiederverwendbar sind.

Sowohl für textuelle als auch grafische Modelle gibt es eine vierte Formalitätsstufe:

- *Formal.* Ein Modell ist so formal, dass es sich als Grundlage für Transformationen und Simulationen eignet. Textuelle Beschreibungen folgen formalen Regeln (wie zum Beispiel der Z Notation [83]).

Der Grad der Formalität von Anforderungsdokumenten wird in Tabelle 4.4 auf einer vierstufigen Ordinalskala dargestellt. Die erste Spalte gibt den Formalitätsgrad an, die zweite und dritte Spalte die entsprechende Formalitätsklasse bei Anforderungen, bzw. Modellen und die vierte Spalte gibt eine Beschreibung und Beispiele für Dokumente des jeweiligen Formalitätsgrads an.

Mit Hilfe der Skala in Tabelle 4.4 kann die Formalität einzelner Aspekte zweier Anforderungsdokumente miteinander verglichen werden. Im Rahmen dieser Ar-

Tabelle 4.4: Einordnung von Anforderungsbeschreibungen in Formalitätsgrade.

Grad	Textuell	Grafisch	Beschreibung
1	Prosa	Undefiniert	Textuelle Anforderungen sind im Fließtext beschrieben, Modelle haben keine definierte Syntax. Die Semantik kann im besten Fall aus dem Kontext abgeleitet werden.
2	Identifizierbar	Definiert	Textuelle Anforderungen sind strukturiert und einzeln identifizierbar. Modelle haben eine definierte Syntax. Die Semantik ist zumindest teilweise definiert.
3	Attributiert	Standardisiert	Spezielle Attribute tragen Semantik für textuelle Anforderungen, Syntax und Semantik der Modelle ist in einem Standard festgelegt. Beispiele hierfür sind Use-Cases, Anforderungs-Datenbanken und UML Modelle.
4	Formal	Formal	Syntax und Semantik sind so beschrieben, dass sie vom Computer ausgewertet werden können. In diese Klasse fallen zum Beispiel Z-Spezifikationen und UML Modelle, die gezielt für Simulationen und Transformationen erstellt wurden.

beit reicht der Vergleich der Aspekte auch aus: Sollen beispielsweise Erfahrungen und Heuristiken genutzt werden, um Autoren bei der Erstellung von Use-Cases zu unterstützen, reicht es, den Formalitätsgrad der Use-Case-Beschreibungen zu kennen. Um aber abschätzen zu können, wie weit eine solche Unterstützung bei der Erstellung der gesamten Anforderungsdokumentation gehen kann, ist es hilfreich, das Maß der Formalität auf eine einzige Kennzahl pro Dokument zu verdichten. Da diese Kennzahl nur dem Überblick dient, wird hier mit dem Median über alle Aspekte die einfachste Form der Aggregation, die bedeutungsvolle Werte liefert, gewählt.

Abbildung 4.3 zeigt exemplarisch den Grad der Formalität einzelner Abschnitte von Anforderungsdokumenten aus dem Softwareprojekt an der Leibniz Universität Hannover. Die in dieser Lehrveranstaltung angefertigten Anforderungsdokumente werden an verschiedenen Stellen in dieser Arbeit zur Evaluation verwendet.

Abbildung 4.4 ordnet einige gebräuchliche Beispiele von Anforderungsdokumenten in dieses Formalitätsmodell für Dokumente ein. Die Abbildung stellt außerdem dar, in welchen Bereichen computerbasierte Unterstützung geboten werden kann. Formale Modelle (Formalitätsgrad 4 nach Tabelle 4.4) können simuliert werden [35]. Auf diese Weise ist bei geeigneten Simulationsmodellen sogar eine teilweise

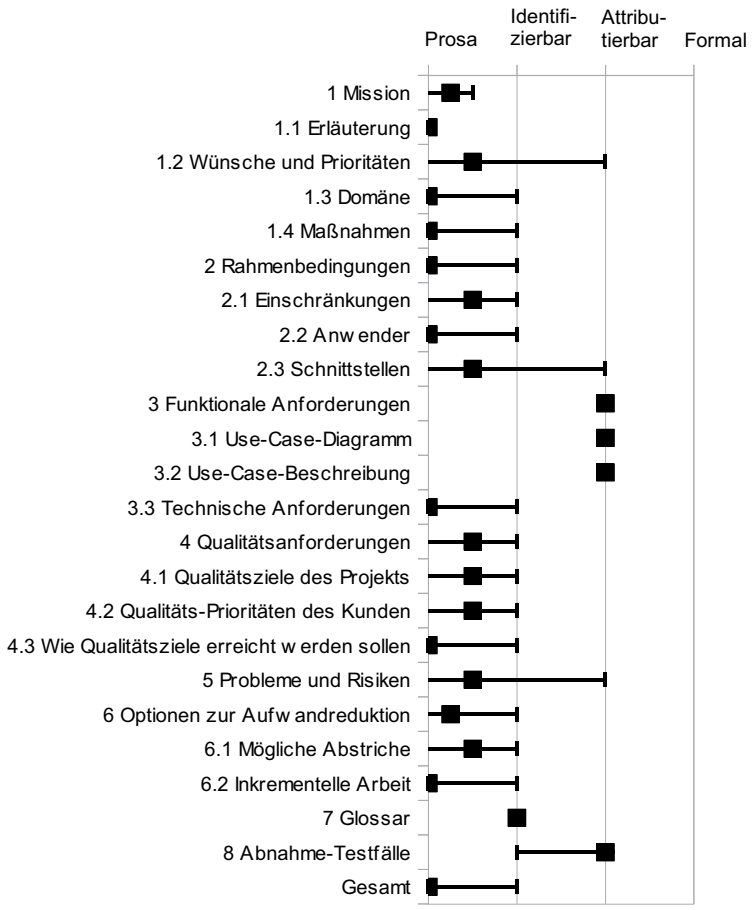


Abbildung 4.3: Formalität der Anforderungsdokumente im studentischen Softwareprojekt an der Leibniz Universität Hannover. Für jeden Abschnitt der Vorlage ist der Median der Formalität sowie der minimale und maximale Wert gegeben.

Validierung möglich, indem gezeigt wird, ob das zu erstellende System das gegebene Problem lösen kann. Verifikation mit Hilfe von Model-Checking ist bei formalen und teilweise auch bei attributierten Dokumenten möglich.

Heuristiken können eingesetzt werden, wenn zu überprüfende Anforderungen identifiziert werden können, oder eine geeignete Definition von Syntax und Semantik vorliegt (also ab Formalitätsgrad 2).

Bei Modellen lassen sich ab dem Formalitätsgrad 3 spezielle Model-Checker einsetzen, die den Heuristiken in der Regel überlegen sind [40]. Je nachdem, wie streng die Modelle der definierten Syntax und Semantik folgen, eignen sich allerdings auch hier Heuristiken, um Modelle zu überprüfen [19]. Bei stark textlastigen und

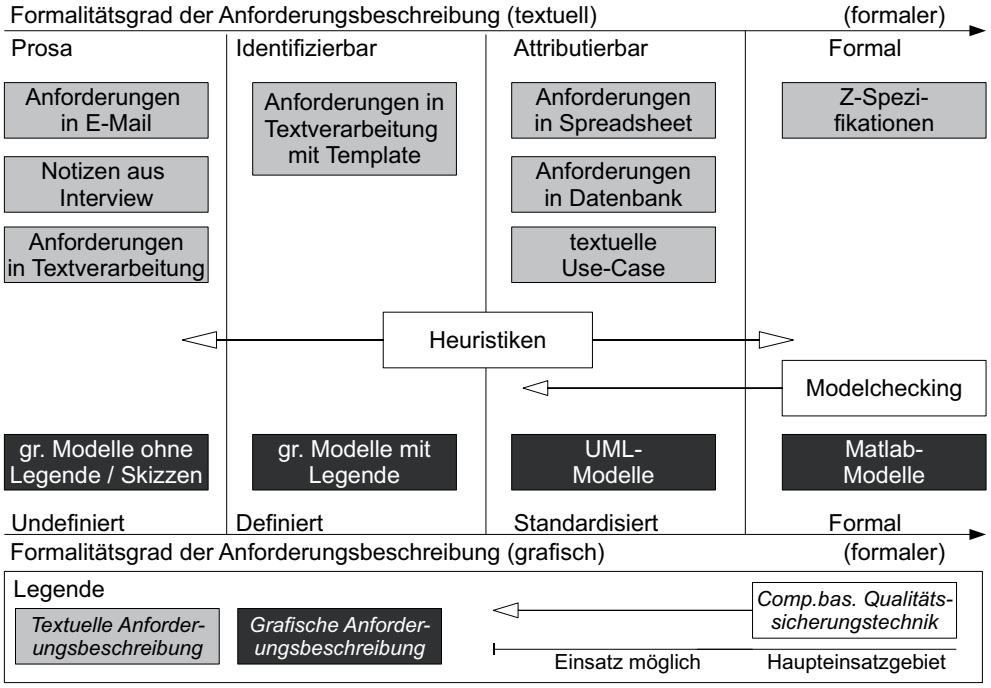


Abbildung 4.4: Beispiele für unterschiedlich formale Anforderungsdokumente.

attributierbaren Anforderungsdokumenten, beispielsweise Use-Cases, lassen sich Heuristiken optimal einsetzen, da die Verarbeitung des Texts durch die strukturierenden Attribute vereinfacht wird, Model-Checker aber auf Grund der natürlichen Sprache nicht richtig zum Einsatz gebracht werden können.

4.2.2 Reife von Anforderungsdokumenten

Dokumente im Software Engineering durchlaufen verschiedene Reifestufen. Diese variieren nach Vorgehensmodell. Tabelle 4.5 zeigt ein grundlegendes Reifegradmodell nach Zuser et al. [178], bei dem zwischen optionalen und erforderlichen Reifestufen unterschieden wird. Demnach hat jedes Dokument mit dem *Entwurf* eine Reifestufe, in der das Dokument zwar noch Inkonsistenzen aufweisen kann, der spätere Inhalt des Dokuments jedoch bereits abzusehen ist. Später im Lebenszyklus wird der Entwurf weiterentwickelt zu einem *Vorschlag* für die Finalversion. Dieser Vorschlag ist die Grundlage für die Managemententscheidung und das Dokument ist bereits in einer endgültigen Fassung. Fällt die Entscheidung für diesen Vorschlag, entsteht eine offizielle *Version* und das Dokument geht von der Erstellungsphase in die Verwendungsphase.

Als Beispiel sollen hier erneut die Anforderungsdokumente aus dem Softwareprojekt dienen. Abbildung 4.5 zeigt die Reifestufen der Anforderungsdokumentation

Tabelle 4.5: Reifegradmodell von Dokumenten nach Zuser et al. [178].

Reifestufe	**Beschreibung**	**Optional?**
Konzept	Struktur (Überschriften), eventuell Stichworte	Optional
Skizze	Unvollständige Texte, nicht konsistent, formfrei	Optional
Entwurf	Vollständige Erstfassung, eventuell noch inkonsistent (Inhalt, Form)	Erforderlich
Rohfassung	Inhaltlich korrekte Fassung; Form wird wahrscheinlich noch geändert	Optional
Vorschlag	Endgültige Fassung, Form und Inhalt korrekt und konsistent; Beschlussvorlage	Erforderlich
Version.x.y	Geprüfte und freigegebene / beschlossene Version; Unterliegt Änderungsmanagement	Erforderlich

über einem typischen Verlauf der Analysephase. Das dargestellte Softwareprojekt folgt dem Wasserfallmodell. Dazu wird ein einziges Anforderungsdokument als Ganzes betrachtet. In inkrementellen Projekten würde man die verschiedenen Anforderungsbeschreibungen einzeln auf die erforderliche Reifestufe bringen, so dass diese bereits in Entwurf und Implementierung gehen können, bevor andere Anforderungsbeschreibungen in gleicher Weise ausgearbeitet werden.

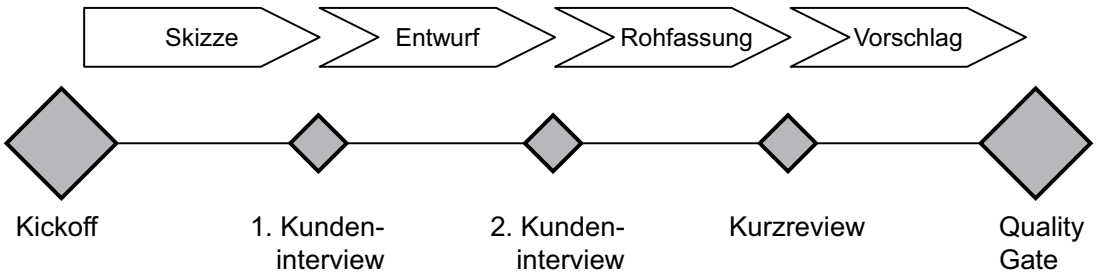


Abbildung 4.5: Reifestufen im Softwareprojekt bei typischem Projektverlauf.

Für die Qualitätssicherung leiten sich aus dem Reifegradmodell nach Zuser et al. einige Rahmenbedingungen ab [178]:

Definition 16: *Verifikation*

Verifikation ist der Versuch nachzuweisen, dass das Produkt korrekt ist, also das Ergebnis einer Aktivität seinen durch vorherige Aktivitäten gestellten Vorgaben entspricht.

(nach SWEBOK [4])

Für die *Verifikation* wird also eine Referenz benötigt, gegen die man prüfen kann [148]. Dies ist normalerweise ein Dokument, das nach dem Vorgehensmodell dem zu verifizierenden Dokument logisch voraus geht. Ein solches Dokument kann bei Anforderungsdokumenten nicht unbedingt vorausgesetzt werden, da Anforderungsdokumente in vielen Prozessen die ersten offiziell geforderten Dokumente darstellen. Häufig wird daher bei Anforderungen gegen ein Metamodell oder eine Vorlage geprüft, da durch diese zumindest Vorgaben an die Syntax und Struktur gegeben sind. Anforderungen müssen also nach diesen Vorgaben (Vorlage oder Metamodell) syntaktisch richtig dokumentiert werden. Daher muss für eine sinnvolle Verifikation die Form konsistent sein. Nach dem Reifestufenmodell ist dies schlimmstenfalls erst bei dem *Vorschlag* der Fall.

Definition 17: *Validierung*

Validierung ist der Versuch nachzuweisen, dass das richtige Produkt erzeugt wird, also das Ergebnis einer Aktivität seinen spezifischen beabsichtigten Zweck erfüllt.

(nach SWEBOK [4])

Die *Validierung* wird bei Anforderungsdokumenten als inhaltliche Prüfung aufgefasst. Ein Review durch den Kunden gilt hier als das Mittel der Wahl [139]. Eine solche inhaltliche Prüfung ist erst dann sinnvoll, wenn der Inhalt des zu prüfenden Dokuments ernstgemeint ist. Nach dem Reifestufenmodell in Tabelle 4.5 ist dies verlässlich frühestens in der *Rohfassung* der Fall, falls diese Reifestufe vorgesehen ist.

Aus diesen Überlegungen ergibt sich, dass nach dem Reifestufenmodell die Validierung mit dem Kunden bereits sinnvoll möglich sein kann, bevor Verifikation in Frage kommt. Da die Involvierung des Kunden aufwendiger als eine interne Prüfung ist, wird in der Praxis normalerweise aber genau andersherum vorgegangen: Erst wenn ein Dokument laut Verifikation in Ordnung ist, lässt man es für die (teurere) Validierung zu.

4.2.3 Lebenszyklus von Andorderungsdokumenten

Mit Formalität und Reifestufe von Anforderungsdokumenten stehen zwei wichtige Aspekte zur Charakterisierung von Dokumenten im Rahmen dieser Arbeit zur Verfügung. In diesem Abschnitt werden diese beiden Aspekte verwendet, um den Lebenszyklus von Anforderungsdokumenten zu modellieren. Anforderungsbeschreibungen sind zu Anfang eins Projekts noch vage und werden mit zunehmender Reife immer formaler.

Projekte beginnen mehr oder weniger weit auf der linken Seite des Formalitätsmodells in Abbildung 4.4 (vergleiche Pohl [131]). Typische Beispiele für Anfor-

derungsbeschreibungen des Formalitätsgrads 1 sind Ideen-Skizzen als Foto von einem Whiteboard oder auf einer Serviette sowie E-Mails oder Visions-Folien, durch die eine Projektidee kommuniziert wird. Diese sind eine gute Grundlage für die Reifestufe *Konzept*. In seltenen Fällen kommen kleinere Projekte mit diesem Formalitätsgrad aus. In der Regel wird man aber versuchen, die Anforderungen durch Interviews und Workshops genauer zu erfassen und zu strukturieren. Falls für das Projekt ein formales Requirements Management (also Änderungsmanagement und Tracing) erforderlich ist, bietet sich eine attributierte Datenbank an (Formalitätsgrad 3). Bei sicherheitskritischen Projekten wird man auch formale Modelle einführen, um schon früh hohe Sicherheitsstandards garantieren zu können (Formalitätsgrad 4).

Projekte, die auf Basis vorhandener Anforderungsdokumente entstehen, starten häufig schon in der Mitte oder rechts in Abbildung 4.4. Dies ist das Ziel von Produktlinien und Wiederverwendung von Anforderungen. So werden beispielsweise bei der Daimler AG stabile Anforderungen früherer Projekte systematisch wiederverwendet [77], wodurch schon früh im Projekt ein hoher Formalitätsgrad erreicht wird (siehe auch Abschnitt 8.3).

Je nach Vorgehensmodell variiert die Anzahl der Anforderungsdokumente. So werden im V-Modell mit dem Lasten- und das Pflichtenheft zwei wichtige Anforderungsdokumente genannt [81]. Sowohl Lastenheft als auch Pflichtenheft geben eine ganzheitliche Sicht auf die Anforderungen eines Projekts. Damit lassen sie sich von Struktur und Inhalt mit ähnlichen Dokumenten aus dem RUP-Vorgehensmodell [139] oder den Vorschlägen aus dem IEEE Standard 830–1998 [3] vergleichen. Diese Dokumente haben gemeinsam, dass der Formalitätsgrad der darin enthaltenen Anforderungs-Aspekte sehr heterogen ist. Demnach enthält ein Anforderungsdokument mit dieser Struktur unterschiedliche Typen von Anforderungen (funktionale Anforderung, Qualitätsanforderungen, etc.) auf verschiedenen Abstraktionsebenen (Überblick bis Detail).

Beispiel 7: *Formalität bei wachsender Reife eines Anforderungsdokuments*

In einem Projekt P sollen die funktionalen Anforderungen mit Use-Cases dokumentiert werden. Für die Reifestufe *Konzept* des Artefakts *funktionale Anforderungen* beginnt man in P mit einer formlosen Beschreibung von Benutzerzielen (Formalitätsgrad: Prosa). Für die *Skizze* der funktionalen Anforderungen wird für jedes Benutzerziel ein kurzer Titel und eine Beschreibung erarbeitet. Die Anforderungen sind damit einzeln identifizierbar. Für die Reifestufe *Entwurf* werden diese Anforderungen mit Hilfe eines UML Use-Case-Diagramms geordnet (Formalitätsgrad *Standardisiert*). Für den *Vorschlag* der funktionalen Anforderungsdokumentation werden die UML Use-Cases zu vollständigen tabellarischen Use-Cases ausformuliert. Der Verlauf der Formalität der funktionalen Anforderungen in P ist in Abbildung 4.6 zusammen mit einem Beispielverlauf für ein innovatives Projekt P' (links) und ein reifes Standardprojekt P" (rechts) dargestellt.

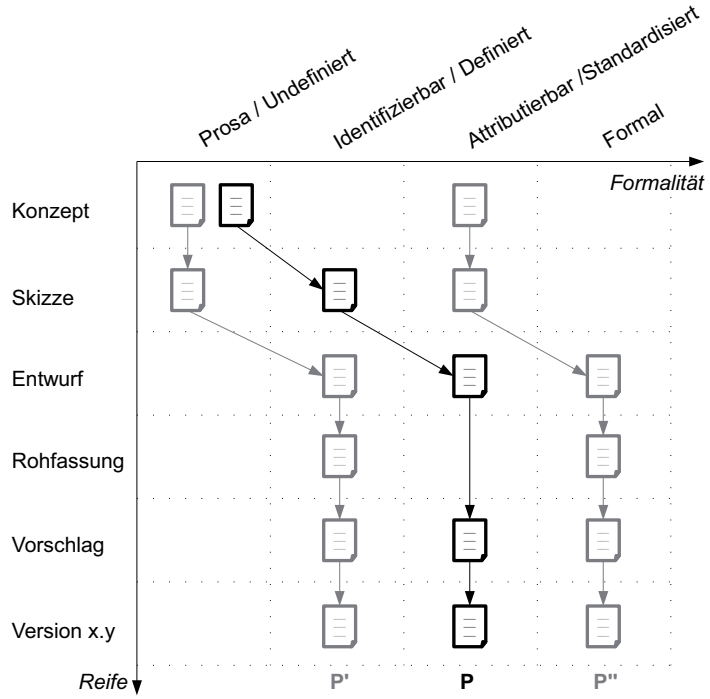


Abbildung 4.6: Beispielverläufe der Formalität von Anforderungsdokumenten.

Für ein Projekt sind also die Formalitätsgrade des Anforderungsdokuments in der Reifestufe Konzept sowie der darauf aufbauenden Reifestufen charakteristisch. Sowohl Formalität als auch Reife eines Dokuments hat einen großen Einfluss auf die möglichen Maßnahmen zur Qualitätssicherung. Damit Anforderungsdokumente verifizierbar sind, müssen sie eine definierte Form haben. Zusätzlich müssen sie inhaltlich in einer guten Form sein, damit sich der Aufwand der Verifikation lohnt. Viele Verifikationsmethoden (Simulation, Model-Checking, formaler Beweis) benötigen zudem eine gewisse Vollständigkeit der Informationen. Daher kann Verifikation frühestens im Laufe der Erstellung einer Rohfassung ab dem Formalitätsgrad identifizierbar/definiert angewandt werden.

Um ein Anforderungsdokument validieren zu können, muss es mindestens eine Form haben, die einem Stakeholder übergeben werden kann. Zudem müssen die Inhalte weit genug beschrieben sein, damit sich eine Validierung lohnt. Unsystematisch als Fließtext dokumentierte Anforderungen erschweren dem Kunden die Validierung genauso, wie schwer verständliche Formalismen. Validierung ist daher sinnvoll auf einer Rohfassung möglich, die vor allem leicht verständlich sein sollte. Diese Verständlichkeit ist in den Formalitätsstufen Prosa und Formal eher einge-

schränkt. Abbildung 4.7 illustriert diesen Zusammenhang. Die Bereiche, in denen Dokumente für Verifikation und Validierung gut geeignet sind, sind als Flächen hervorgehoben. Die Grenzen zwischen den Formalitäts- und Reifestufen sind teilweise schlecht greifbar. Wenn eine der Flächen genau in einer Formalitäts- oder Reifestufe endet, soll dies andeuten, dass diese Form der Qualitätssicherung in Einzelfällen auch in dieser Stufe möglich ist.

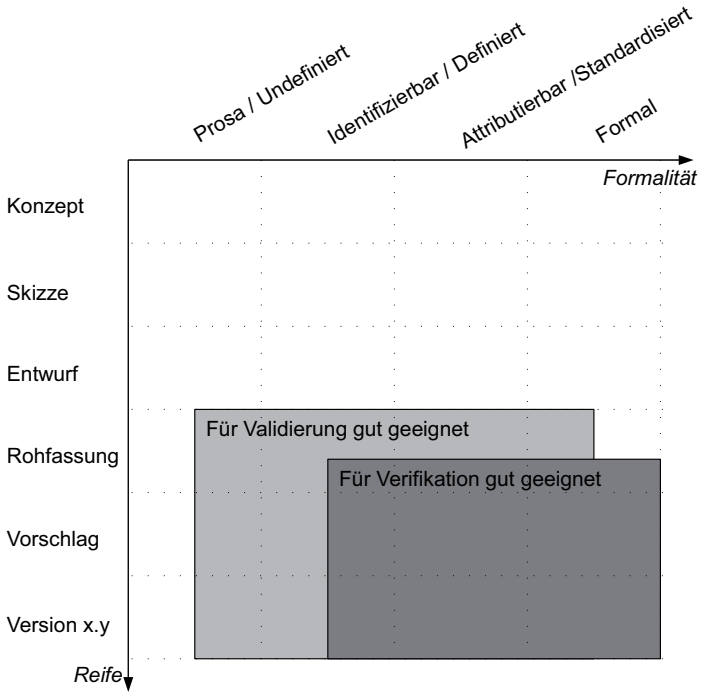


Abbildung 4.7: Qualitätssicherung nach Reifestufen und Formalitätsgraden.

Validierung kann zur Qualitätssicherung also bereits früher und auf unformaleren Dokumenten eingesetzt werden als Verifikation. Das steht eher im Widerspruch zum normalen Vorgehen, bei dem man Artefakte eines Softwareprojekts zunächst in einen möglichst guten Stand bringt, bevor man den Auftraggeber damit behelligt. Pragmatisch behilft man sich hier, indem Einzelteile der Anforderungsdokumente schnell in eine entsprechende Form gebracht werden. Auch können Anforderungen vorübergehend in eine Form gebracht werden, die die Kommunikation mit dem Kunden erleichtert, wie zum Beispiel Oberflächenprototypen. Dabei entsteht allerdings das Problem, dass der Gesamtzusammenhang verloren geht und die Konsistenz zwischen den verschiedenen Einzelteilen und abgeleiteten Formen sichergestellt werden muss. Im Rahmen dieser Arbeit wird untersucht, wie Erfahrungen und Heuristiken in diesem Kontext eingesetzt werden können, um die Konsistenz zwischen verschiedenen Artefakten sicherzustellen und Anforderungsdokumente schon vor der Verifizierbarkeit heuristisch zu prüfen.

4.3 Experiment: Relevanz guter Anforderungsdokumentation

Grundannahme dieser Arbeit ist, dass gute Dokumentation von Anforderungen eine Voraussetzung für den Projekterfolg ist. Eine Verbesserung von Anforderungsdokumentation erhöht daher die Aussichten auf einen Projekterfolg. In diesem Abschnitt wird diese Grundannahme am Beispiel von 16 studentischen Softwareprojekten nachgewiesen. Dabei wird wie folgt vorgegangen:

1. Auf Basis der Grundlagen der Qualitätseigenschaften von Anforderungsdokumenten in Abschnitt 2.1.4 wird in Abschnitt 4.3.1 ein Maß für die *Qualität von Anforderungsdokumenten* mit Hilfe der Goal-Question-Metric-Methode definiert.
2. Zudem wird ein Maß für *Projekterfolg* auf Basis eines Fragebogens an den Kunden definiert.
3. Auf Basis der Definition dieser Maße werden konkrete be- oder widerlegbare Hypothesen aufgestellt.
4. Die *Qualität von Anforderungsdokumenten* und der *Projekterfolg* werden auf Basis der definierten Maße erhoben (Abschnitt 4.3.2).
5. Durch statistische Analyse wird der Nachweis des Zusammenhangs zwischen *Qualität von Anforderungsdokumenten* und *Projekterfolg* durchgeführt (Abschnitt 4.3.3).
6. Die Ergebnisse werden in Abschnitt 4.3.4 mit verwandten Arbeiten verglichen und ihre Gültigkeit wird Abschnitt 4.3.5 diskutiert.

Auf Grundlage des Zusammenhangs zwischen Qualität der Anforderungsdokumente und dem Projekterfolg wird es möglich, Projekte in Schieflage frühzeitig zu identifizieren. Für Organisationen, die mehrere Projekte parallel durchführen, ist dies wertvoll, um rechtzeitig reagieren zu können: Gefährdete Projekte können unterstützt werden, bevor es zu spät ist, aussichtslose Projekte können abgebrochen werden, bevor sie zu viele Ressourcen verbrauchen. Dieses Ziel ist auch im Kontext der untersuchten studentischen Projekte relevant. Jeweils acht dieser Projekte fanden im gleichen Semester statt. Alle Projekte eines Semesters erreichen das Ende der Analysephase gleichzeitig. Um in diesem Kontext die Projekte zu identifizieren, die eine intensive Betreuung besonders nötig haben, ist es nötig, die Qualität der Anforderungsspezifikation effizient zu ermitteln.

Die in diesem Abschnitt dargestellten Ergebnisse sind teilweise in der Bachelorarbeit von El Boustani entstanden [31] und wurden im Rahmen der Requirements Engineering Konferenz präsentiert [97]. Die dahinter liegende empirische Methodik und der Vergleich der Ergebnisse mit anderen Arbeiten wurden ebenfalls veröffentlicht (siehe [94]). Über diese Veröffentlichungen hinaus wird hier die statistische Aussagekraft des Experiments untersucht.

4.3.1 Messziele

Bei der Anwendung von Metriken ist ein häufiger Fehler, dass nur die Faktoren gemessen werden, die leicht zu erfassen sind. Dieses Vorgehen führt häufig zu Ergebnissen, die nicht hilfreich sind, um festzustellen, ob die Qualitätsziele erreicht wurden.

Im Gegensatz dazu schreibt die Goal-Question-Metric-Methode (GQM) einen Top-Down Ansatz für zielorientiertes Messen vor. Die grundlegenden Schritte sind die Definition der Messziele, deren genauere Beschreibung durch Hypothesen (zum Beispiel durch Abstraction Sheets), das Ableiten von Fragestellungen und schließlich das Ableiten von Metriken, mit denen diese Fragen beantwortet werden können (siehe [167]).

Für dieses Experiment werden Abstraction Sheets genutzt, um Hypothesen über den Zusammenhang zwischen Qualität von Anforderungsdokumenten und Projekterfolg aufzustellen. Diese können nach der Datenerhebung bestätigt oder widerlegt werden. In diesem Abschnitt wird das Vorgehen nur skizziert. Die eigentlichen Abstraction Sheets und Details der Metriken sind in [31] zu finden.

Abbildung 4.8 gibt einen Überblick über die Ziele und Metriken zur Anforderungsqualität. Das Ziel ist die Qualität von Anforderungsspezifikationen systematisch zu messen. Dabei gibt es zwei Unterziele: Messung der *formalen Anforderungsqualität* und Messung der *inhaltlichen Anforderungsqualität*. Der Begriff der *formalen Anforderungsqualität* bezieht sich auf Formulierungsregeln, wie in [142] (z.B. vollständig definierte Prozesswörter, Vermeidung unvollständiger Vergleiche, usw.). Im Gegensatz dazu beschreibt der Begriff *inhaltliche Anforderungsqualität* Aspekte, die Interpretation erfordern. Um zum Beispiel zu entscheiden, ob ein Anforderungsdokument (in der Folge auch Spezifikation genannt) ein Qualitätsmodell enthält oder nicht, muss in der Anforderungsspezifikation nach Qualitätszielen gesucht und entschieden werden, ob diese ausreichend detailliert und konkret beschrieben wurden. Die inhaltliche Anforderungsqualität ist weiter unterteilt in allgemeine (hellgrau) und für nicht funktionale Anforderungen spezifische Teile: a) technische, b) bedienungsspezifische und c) qualitätsspezifische Aspekte.

Metrik für Projekterfolg

Neben den Zielen für Anforderungsqualität muss auch der Projekterfolg systematisch erfasst werden. Dazu wurde zu jedem Projekt der Kunde mit Hilfe eines kurzen Fragebogens gebeten, den Projekterfolg auf der folgenden Skala zu bewerten:

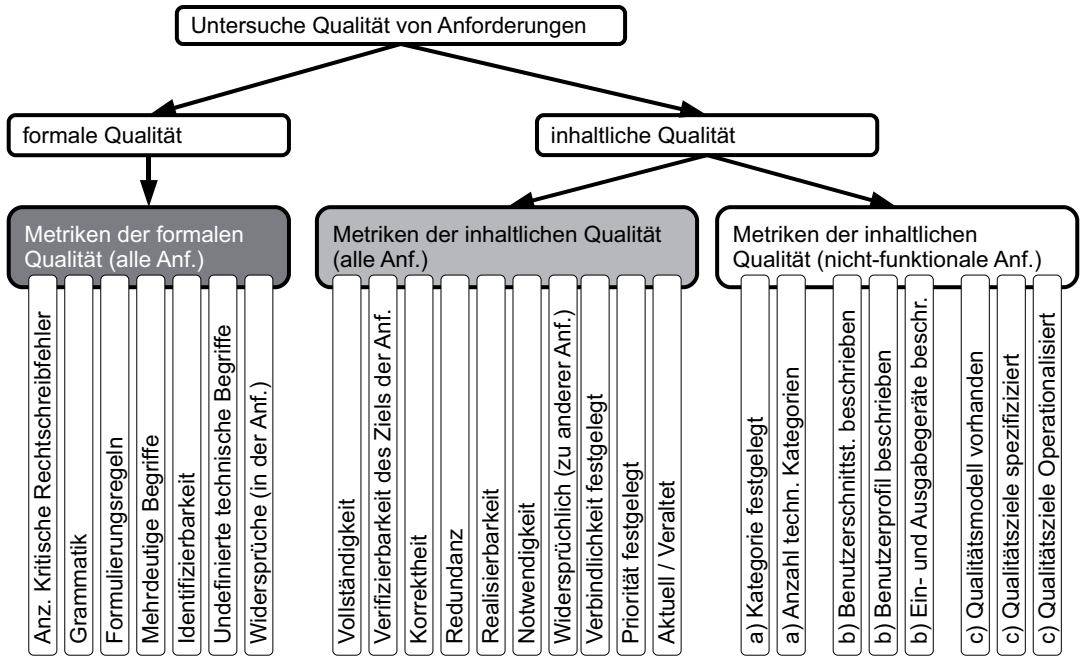


Abbildung 4.8: Messziele und Metriken für Anforderungsqualität.

++ Die Projektergebnisse (z.B. die Software) werden auf die geplante Weise verwendet.

\+ Die Projektergebnisse könnten auf die ursprünglich gedachte Weise verwendet werden, aber es gibt bessere Lösungen, um die Ziele des Kunden zu erreichen.

\- Projekte dieser Kategorie haben einige Kundenziele verfehlt. Der Kunde glaubt, dass diese Ziele innerhalb von einem Monat Nacharbeit erreichbar wären.

-- Diese Kategorie besteht aus Projekten, die keine funktionierende Software geliefert haben. Diese Projekte haben den Abnahmetest nicht bestanden und der Kunde glaubt nicht, dass es Sinn macht, das Projekt fortzuführen.

Diese Definition von Projekterfolg weicht von typischen Definitionen (wie zum Beispiel im Chaos Report [2]) ab: Die hier untersuchten studentischen Projekte können weder Zeit noch Budget überschreiten, weil sie am Ende des Semesters automatisch enden. Wenn sie dann nicht liefern können oder für den Kunden nicht nützlich sind, gilt das Projekt im Rahmen dieses Experiments als gescheitert. Nur Projekte in Kategorie ++ gelten im Rahmen dieser Studie als erfolgreich (vergleiche Diskussion in Abschnitt 4.3.4, Tabelle 4.13).

Hypothesen

Auf Basis der Definition der Maße von Anforderungsqualität und Projekterfolg kann die Grundannahme *gute Anforderungsdokumentation ist Voraussetzung für den Projekterfolg* zu folgenden Hypothesen konkretisiert werden:

Hypothese 1 Projekte mit hoher gemessener Qualität der Anforderungsspezifikation haben mit hoher Wahrscheinlichkeit Erfolg (Kategorie ++).
Einflussfaktoren: Es wird ein Zusammenhang zwischen formal meßbaren Qualitätsaspekten und der Qualität des Inhalts einer Anforderungsspezifikation vermutet (wie in [171] beschrieben). Außerdem kann eine Anforderungsspezifikation mit hoher gemessener Qualität auch auf ein gut organisiertes Projektteam hindeuten, das eine erhöhte Chance hat am Ende wertvolle Software zu liefern.
Hypothese gilt als bestätigt, wenn es einen oberen Schwellenwert gibt, so dass mehr als 75% der Projekte, die eine bessere gemessene Qualität in ihrer Anforderungsspezifikation aufweisen, in Kategorie ++ fallen.

Hypothese 2 Projekte mit niedriger gemessener Qualität der Anforderungsspezifikation haben mit hoher Wahrscheinlichkeit keinen Erfolg. (Kategorie +, -, oder --).
Einflussfaktoren: Durch eine schlechte Anforderungsspezifikation gerät das Projekt in Gefahr, da eine verlässliche und persistente Sammlung der Anforderungen fehlt. Projektgruppen, die eine schlechte Anforderungsspezifikation erzeugen, haben unter Umständen auch noch andere Probleme. So können beispielsweise Projektteilnehmer gegeneinander arbeiten oder ein schlechtes Zeitmanagement haben. Solche Schwierigkeiten können sich über den Projektverlauf multiplizieren.
Hypothese gilt als bestätigt, wenn ein unterer Schwellenwert gefunden werden kann, so dass mindestens 75% Projekte deren gemessene Qualität unter diesem Schwellenwert liegen, scheitern (+, - oder --).

Zusammengefasst ergibt sich die Frage nach dem Zusammenhang zwischen Qualität der Anforderungsspezifikation und Projekterfolg:

Hypothese 3 Es besteht ein statistisch relevanter Zusammenhang (Korrelation) zwischen der Qualität der Anforderungsspezifikation und dem Projekterfolg.
Einflussfaktoren: Erfolgreiche Projekte (++) haben im Schnitt eine deutlich bessere Spezifikation als nicht erfolgreiche Projekte (+, - oder --).
Hypothese gilt als bestätigt, wenn es eine *starke Korrelation* zwischen den Variablen *gemessene Qualität der Anforderungsspezifikation* und *Projekterfolg* gibt.

Die Korrelation zwischen zwei Variablen x und y bestimmt man über den Korrelationskoeffizienten r [113]. Nach Cohen spricht man von einer starken Korrelation, wenn $0{,}5 < r \leq 1{,}0$ ist [43].

4.3.2 Durchführung

Dieser Abschnitt beschreibt den Untersuchungsgegenstand (studentische Projekte) und die Durchführung des Experiments.

Untersuchungsgegenstand

Untersuchungsgegenstand dieses Experiments waren 16 Softwareprojekte, die im Rahmen der Lehre an der Leibniz Universität Hannover durchgeführt worden sind. Diese Softwareprojekte sind Teil des Curriculums des Bachelorstudiengangs Informatik. Alle Teilnehmer haben grundlegende Kenntnisse in Programmiersprachen, Datenstrukturen und Algorithmen sowie in Software Engineering und Projektmanagement. Teilnehmer mit Kenntnissen aus weiterführenden Lehrveranstaltungen, wie Datenbanken, künstliche Intelligenz oder Requirements Engineering werden gleichmäßig auf die Projektgruppen verteilt.

Jede Projektgruppe besteht aus 5 Mitgliedern. Die Studenten müssen innerhalb der Gruppe einen Projektleiter wählen, der die Gruppe nach außen vertritt. Die Projekte dauern ein Semester (4 Monate) und Studenten verbringen durchschnittlich 16 Stunden pro Woche mit dem Projekt.

Eines der Lehrziele bei dem Softwareprojekt ist es, die Studenten in einem realistischen Softwareprojekt Erfahrungen sammeln zu lassen. Deshalb gibt es in jedem Projekt einen Kunden mit echtem Interesse an dem zu erstellenden Produkt. Für dieses Experiment ist das echte Interesse des Kunden wichtig, um hinterher den Projekterfolg feststellen zu können. Studenten sind es gewohnt Fragen ausführlich mit den Betreuern von Lehrveranstaltungen zu diskutieren. Da dies in realen Projekten in der Regel nicht möglich ist, wurde zusätzlich die Zeit begrenzt, in der die Studenten Interviews mit dem Kunden durchführen oder mit einem Betreuer technische Fragen diskutieren dürfen.

Für das Softwareprojekt ist ein strikter Wasserfall-Prozess vorgeschrieben. Die Studenten starten mit einer Analysephase, gehen dann in die Entwurfsphase über und implementieren schließlich die Software. Am Ende begutachtet der Kunde die Software in einem Abnahmetest. In der Regel wird die Software dann akzeptiert, da gute Leistungen und Aufwand der Studenten in dem Projekt honoriert werden. Deshalb ist es wichtig, den Erfolg der Software unabhängig von diesem Abnahmetest zu definieren.

Untersuchungsdurchführung

Die Qualität der Anforderungsspezifikation wird am Ende der Analysephase gemessen, also *nach* der Entscheidung auf Basis des Vorschlags (→ Reifestufen-

modell). An dieser Stelle wird eine sogenannte *Baseline* der Anforderungsspezifikation erstellt, damit ein definierter Stand als Grundlage für die Entwurfsphase vorliegt. Die untersuchten Spezifikationen haben also die Reifestufe *Version 1.0*. Die Formalität der Dokumentation ist recht gering, geht aber an einigen Stellen deutlich über Prosa hinaus. Vor allem die funktionalen Anforderungen sind recht formal dokumentiert (Formalitätsgrad *attributierbar*, vergleiche Abbildung 4.3).

Für die Erhebung quantitativer Daten gab es im Rahmen dieses Experiments grundlegende Rahmenbedingungen. Da mit der hier vorgestellten Methode bis zu 16 Anforderungsspezifikationen in kürzester Zeit parallel begutachtet werden sollten und dafür nur sehr begrenzte Ressourcen zur Verfügung standen, musste die Zeit für die Datenerhebung auf unter 180 Minuten pro Anforderungsspezifikation begrenzt werden.

Ziel des Experiments war es, aus der Analyse des Anforderungsdokuments auf die Wahrscheinlichkeit eines Projekterfolgs oder Fehlschlags zu schließen. Dabei wurden ausschließlich Metriken verwendet, die *ohne Hintergrundwissen* über das konkrete Projekt erhoben werden können. Die Perspektive des Kunden auf die Anforderungsqualität wurde daher nicht berücksichtigt (z.B. wurde nicht aus Sicht des Kunden überprüft, ob die Anforderungen vollständig sind).

Ein zweites, untergeordnetes Ziel dieses Experiments war es, eine Methode bereitszustellen, mit der frühzeitig gefährdete Projekte identifiziert werden können. Dazu müssen Projekte mit schlechten Anforderungsspezifikationen schnell identifiziert werden können. Um die Geschwindigkeit der Messung zu erhöhen, wurde für die Bewertung der Anforderungsspezifikationen ein spezielles Werkzeug programmiert.

Mit diesem Werkzeug gestaltet sich die Analyse wie folgt: Der komplette Text einer Anforderungsspezifikation wird kopiert und in das Werkzeug eingefügt. Das Werkzeug spaltet den Text in einzelne Sätze auf und fragt den Begutachter für jeden Satz, ob es sich dabei um eine funktionale, technische, Bedienbarkeits- oder Qualitäts-Anforderung handelt. Je nach Klassifikation werden diese Anforderungen dem Begutachter danach zu jeder Metrik aus Abbildung 4.8 vorgelegt. Abbildung 4.9 zeigt das Vorgehen zur Bewertung einer Anforderungsspezifikation.

Für einige Metriken liefert das Werkzeug heuristische Hilfestellung. Zum Beispiel werden über eine Volltextsuche Kandidaten für widersprüchliche Anforderungen gesucht.

Nach [113] sollen Metriken die folgenden Eigenschaften erfüllen:

Einfachheit: Der Aufwand zur Interpretation sollte angemessen sein. Daher werden im Rahmen des Experiments Ergebnisse als Prozent- oder Zahlenwert angegeben.

Eignung (Validität): Es sollte einen sinnvollen Zusammenhang (*eine hinreichend*

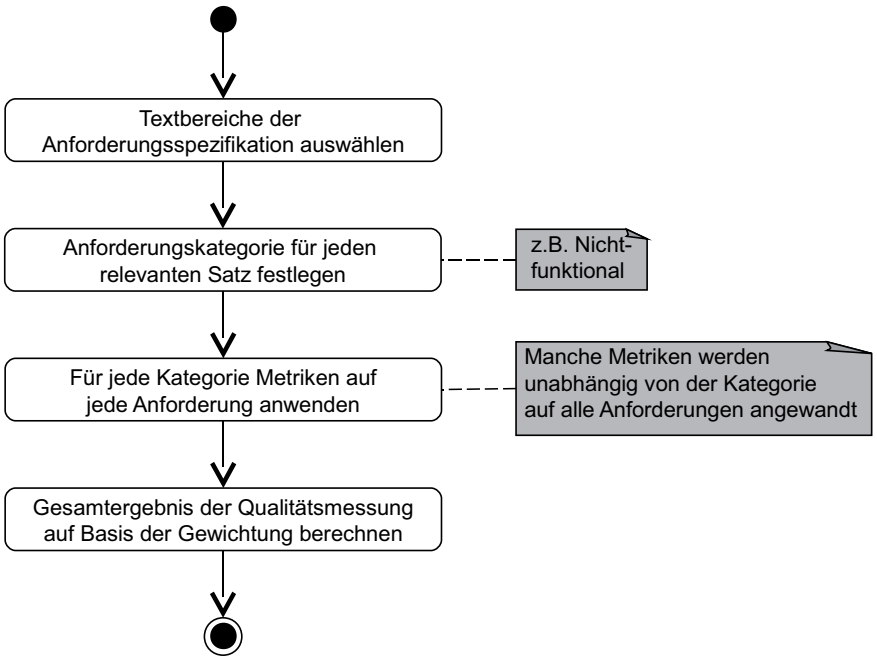


Abbildung 4.9: Vorgehen zur Bewertung der Qualität einer Anforderungsspezifikation.

starke Korrelation [113]) zwischen der Metrik und der zu messenden Eigenschaft geben. Da im Rahmen dieses Experiments die Metriken über die GQM-Methode hergeleitet wurden, ist diese Eigenschaft erfüllt.

Stabilität: Eine Metrik sollte gegenüber unterbewusster Manipulation stabil sein. Die Metriken wurden so weit wie möglich objektiv messbar ausgewählt. Die Stabilität im Rahmen dieses Experiments wird in Abschnitt 4.3.5 diskutiert.

Rechtzeitigkeit: Die Metrik sollte rechtzeitig erhoben werden können, um im zu unterstützenden Prozess hilfreich zu sein. Dies ist im Rahmen dieses Experiments gegeben, da der Zusammenhang zwischen der Qualität von Anforderungsdokumenten und Projekterfolg nachträglich hergestellt wird. Im Projekt wären die hier vorgestellten Metriken ebenfalls rechtzeitig, da die Messung zum ersten Quality Gate stattfindet, somit also noch Einfluss auf das Projekt genommen werden kann [66].

Analysierbarkeit: Die Messwerte sollen einfach zueinander in Relation gesetzt werden können. Voraussetzung dafür sind numerische Wertebereiche und geeignete Skalentypen. Die Analysierbarkeit wird im Abschnitt 4.3.3 im Rahmen der Diskussion der Ergebnisse gezeigt.

Wiederholbarkeit: Es sollte ein objektives Messkriterium gegeben sein und es sollte eine subjektive Beeinflussung der Messung ausgeschlossen sein. Dies ist in diesem Experiment nur teilweise erfüllt, weil einige Metriken subjektive Faktoren beinhalten (vergleiche Abschnitt 4.3.5).

Tabelle 4.6 gibt ein Beispiel für eine Metrik.

Tabelle 4.6: Beispiel der Metrik *Verifizierbarkeit des Ziels der Anf.* aus Abbildung 4.8.

Verifizierbare Ziele der Anforderungen	$\frac{\text{Anzahl der verifizierbaren Aspekte der Anforderungen}}{\text{Anzahl aller Aspekte von Anforderungen}}$
Einfachheit:	Gegeben, da Ergebnis als Prozentzahl vorliegt.
Eignung (Validität):	Gegeben, da Metrik über GQM dem Messziel *inhaltliche Qualität von Anforderungsdokumentation* zugeordnet wurde.
Stabilität:	Teilweise gegeben. Die Entscheidung, ob ein Aspekt einer Anforderung verifizierbar ist kann von Person zu Person unterschiedlich ausfallen.
Rechtzeitigkeit:	Gegeben, da sowohl das Messziel des Experiments unterstützt wird, als auch eine Reaktion im Rahmen des Projekts möglich wäre.
Analysierbarkeit:	Gegeben, da die normierten Messdaten zweier Projekte miteinander vergleichbar sind.
Wiederholbarkeit:	Teilweise gegeben. Die Entscheidung, ob ein Aspekt einer Anforderung verifizierbar ist, kann von Person zu Person unterschiedlich ausfallen.

4.3.3 Ergebnisse

Dieser Abschnitt beschreibt die Ergebnisse des Experiments. Zunächst werden die erhobenen Daten verdichtet und präsentiert. Danach wird die Bedeutung der Ergebnisse hinsichtlich der Hypothesen diskutiert.

Deskriptive Statistik

Abbildung 4.10 zeigt einen Ausschnitt der Daten, die im Rahmen der Messung erhoben wurden. Diese Daten decken 16 Softwareprojekte ab, die in zwei aufeinander folgenden Jahren durchgeführt wurden. Die Farben der Säulen beziehen sich auf die Teile des Zielbaums in Abbildung 4.8.

Grau: Ergebnisse von Metriken, die für alle Anforderungen gelten und die zu dem Messziel *formale Anforderungsqualität* gehören.

Tabelle 4.7: Gewichtung der Qualitätsaspekte von Anforderungen.

Qualitätsaspekt	Gewichtung (w)
Korrektheit	10
Realisierbarkeit	10
Konsistenz	10
Gültig und aktuell	9
Prüfbarkeit	8
Verstehbarkeit	5
Notwendigkeit	3
Vollständigkeit	2
Eindeutigkeit	2
Bewertbarkeit	1
Klassifizierbarkeit	1

Hellgrau: Ergebnisse von Metriken, die für alle Anforderungen gelten und dem Messziel *inhaltliche Anforderungsqualität* zugeordnet sind.

Weiß: Ergebnisse von Metriken, die ausschließlich bei nicht-funktionalen Anforderungen gelten und dem Messziel *inhaltliche Anforderungsqualität* zugeordnet sind.

Um die Qualität von Anforderungsspezifikationen unterschiedlicher Projekte miteinander vergleichen zu können, muss ein Gesamtwert pro Anforderungsspezifikation berechnet werden. Daher wurden die Qualitätsaspekte in diesem Experiment so gewählt, dass sie auf weitverbreiteten Standards (IEEE Std 830-1998 [3]) und pragmatischen Erweiterungen (Rupp [142]) dazu basieren. Für jeden Qualitätsaspekt wird eine Gewichtung angegeben, das den Einfluss auf die Gesamtqualität wiederspiegelt. Die Gewichte wurden auf Basis der Vorschläge von Davis gewählt [48].

Sei $m_i(srs)$ die Metrik i, angewandt auf alle Anforderungen der Anforderungsspezifikation srs und w_i das Gewicht des Qualitätsaspekts aus Tabelle 4.7, mit dem Metrik i assoziiert ist. Die Verbindung zwischen einer Metrik und einem Qualitätsaspekt leitet sich aus dem GQM-Baum ab. Der Gesamtwert der Qualität einer Anforderungsspezifikation wird damit folgendermaßen berechnet:

$$f(srs) = \sum_i w_i * m_i(srs) \tag{4.1}$$

Abbildung 4.10 zeigt die derart aggregierten Ergebnisse. Niedrige Werte (unter 40) sind ein Zeichen für schlechte Qualität der Anforderungsspezifikation, mit einem hohen Risiko für einen Fehlschlag des entsprechenden Projekts. Werte über 44 deuten auf hohe Qualität hin. Interessanterweise entsprechen diese Werte dem Bauchgefühl des jeweiligen Betreuers eines Projekts.

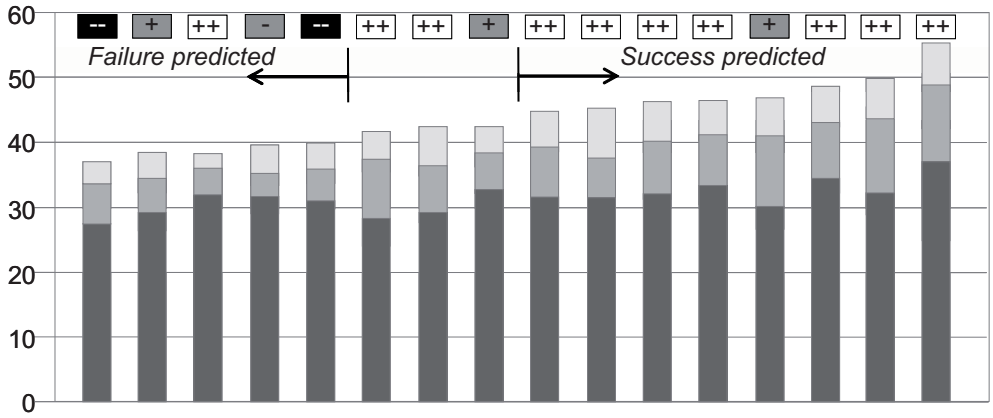


Abbildung 4.10: Ergebnisse der Qualitätsmessung bei 16 Softwareprojekten. Die Symbole oben zeigen, wie das Projekt tatsächlich ausgegangen ist.

Als Nebeneffekt konnte im Rahmen der Untersuchung beobachtet werden, dass selbst einfache Formulierungsrichtlinien (zum Beispiel Anforderungsschablonen [142] oder Use-Case-Vorlagen [42]) zu einer starken Verbesserung der Anforderungsqualität führen. Solche Richtlinien lassen einfach weniger Spielraum für Fehler.

Im Folgenden bezeichnen ++, +, - und -- jeweils die Mengen der Projekte, die entsprechend abgeschlossen haben. Tabelle 4.8 zeigt, dass Projekte der Kategorie ++ im Durchschnitt deutlich bessere Qualitätswerte erreichen, als die anderen Projekte.

Tabelle 4.8: Deskriptive Statistik zu den Ergebnissen aus Abbildung 4.10. Messgruppe gibt die Klassen an, für deren Projekte die Qualitätsergebnisse in der jeweiligen Zeile aggregiert wurden.

Messgruppe	n	Durchschn.	Std	Median	Min	Max
++ ∪ + ∪ - ∪ --	16	44,0	4,96	43,6	37,0	55,4
++	10	45,9	4,77	45,8	38,3	55,4
+ ∪ - ∪ --	6	40,7	3,55	40,0	37,0	46,9

Abbildung 4.11 visualisiert die Daten aus Tabelle 4.8 und gibt so ein erstes Gefühl dafür, wie Qualität der Anforderungsspezifikation und Projekterfolg zusammenhängen. Auf der linken Seite von Abbildung 4.11 ist in einem Scatterplot dargestellt, welche Qualitätswerte Projekte verschiedener Erfolgsklassen annehmen. Es fällt auf, dass für Projekte aus - und -- besonders schlechte Anforderungsqualität gemessen wurde, während Projekte aus + fast so gut sind wie Projekte

aus ++. Das mag daran liegen, dass überwiegend politische Gründe dazu geführt haben, dass die Ergebnisse dieser Projekte nicht wie geplant verwendet werden. Grund für den Misserfolg ist daher nicht die fehlende inhaltliche Qualität. Würde man die Mengen ++ und + zusammenfassen, wäre das Ergebnis noch deutlicher.

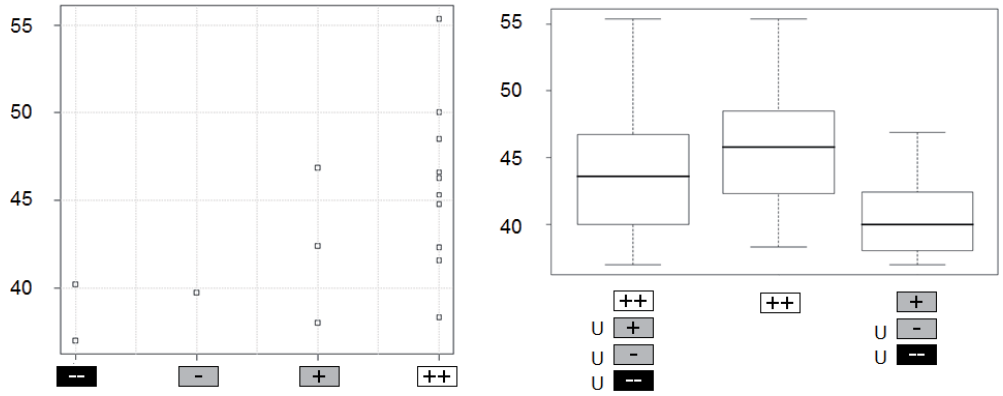


Abbildung 4.11: Scatterplot der Projektergebnisse (links) und Darstellung der deskriptiven Statistik aus Tabelle 4.8 (rechts). Ergebnisse der Projekte in Kategorie ++ unterscheiden sich deutlich von Projekten anderer Kategorien.

Auf der rechten Seite von Abbildung 4.11 sind durchschnittliche, maximale und minimale Qualität sowie das obere und untere Quartil für verschiedene Gruppen von Projekten in einem Boxplot dargestellt. Es zeigt sich, dass erfolgreiche Projekte überdurchschnitlich gute Spezifikationen haben und nicht erfolgreiche Projekte deutlich schlechtere Spezifikationen haben.

Diskussion der Hypothesen

Basierend auf den Ergebnissen in Abbildung 4.10 und 4.11 können nun die Hypothesen aus Abschnitt 4.3.1 untersucht werden. Ziel des Experiments war es, die Abhängigkeit zwischen hoher Qualität einer Anforderungsspezifikation und Erfolg des Projekts zu untersuchen. Die Ergebnisse bestätigen unsere Hypothesen:

Hypothese 1 ist bestätigt: 87,5 % der Projekte, die mehr als 44 Punkte erreichten, fallen in Kategorie ++. Die Ergebnisse in Abbildung 4.10 gehen sogar noch darüber hinaus: Alle Projekte, die mehr als 44 Punkte erreicht haben, haben gute Ergebnisse geliefert (Kategorie ++ oder +).

Hypothese 2 ist bestätigt: 80 % der Projekte unter dem unteren Schwellenwert von 40 Punkten waren nicht erfolgreich, nur ein erfolgreiches Projekt (Ka-

tegorie ++) erreichte einen Punktestand unter diesem Schwellenwert. Alle Projekte, die ungenügende Ergebnisse lieferten (Kategorie - und --) haben weniger als 40 Punkte erreicht.

Untersuchungen an 24 weiteren Projekten aus der selben Lehrveranstaltung bestätigen dieses Ergebnis. Im Gegensatz zu den Ergebnissen in Abbildung 4.10 zeigen sie darüberhinaus, dass Projekte, deren Qualität zwischen den beiden Schwellenwerten liegen, ein geringeres, aber immer noch relevantes Risiko für einen Fehlschlag haben. Basierend auf diesen zusätzlichen Ergebnissen ist nicht anzunehmen, dass der obere Schwellenwert tiefer gesetzt werden kann.

Hypothese 3 ist bestätigt: Statistische Standardtests zeigen einen Unterschied der durchschnittlichen Qualität zwischen erfolgreichen und nicht erfolgreichen Projekten von 12,8 %. Dieser Unterschied ist signifikant ($p = 0{,}027 < \alpha = 0{,}05$), aber mit einem eher großen Konfidenzintervall bei den Unterschieden: 0,69 – 9,71. Um eine bessere Aussage treffen zu können, müssen also mehr Projekte analysiert werden. Der Effekt ist allerdings groß (standardisierte Differenz, oder *Cohens d*: $d = 1,27$). Die Korrelation gilt mit $r = 0{,}52$ als stark, womit Hypothese 3 bestätigt ist. Mit $r^2 = 0,27$ kann man folgern, dass die gemessene Anforderungsqualität den Projekterfolg immerhin zu 27 % beeinflusst, nicht genug, um Qualität als verlässliche Vorhersage für den Projekterfolg zu verwenden.

Konsequenz der Ergebnisse für diese Arbeit

Die Untersuchung des Einflusses von messbarer Anforderungsqualität auf den Projekterfolg zieht eine weitere Konsequenz nach sich. Es gibt innerhalb eines Anforderungsdokuments sichtbare Indikatoren, die auf inhaltliche Probleme in der Dokumentation von Anforderungsaspekten hindeuten. Abbildung 4.12 zeigt diesen Zusammenhang. Die hier verwendeten Metriken zielen darauf ab, diese Indikatoren zu erfassen. Dabei sind die untersuchten Aspekte der Anforderungsspezifikation funktionale Anforderungen, nicht-funktionale Anforderungen, technische Anforderungen und Anforderungen an die Bedienbarkeit.

Definition 18: *Indikator*

Ein Indikator besteht aus einer Metrik und einem definierten Wertebereich. Liegt die Maßzahl der Metrik im definierten Wertebereich, so ist eine Indikation gegeben.

Indikatoren werden im Rahmen der Arbeit genutzt, um Probleme in dokumentierten Aspekten heuristisch zu entdecken. Beim Einsatz solcher Heuristiken muss

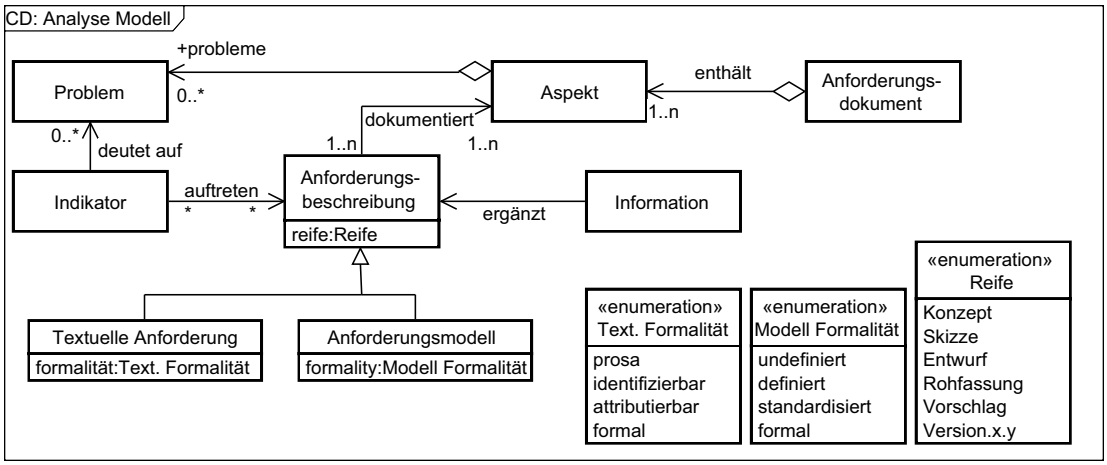


Abbildung 4.12: Modell der heuristischen Analyse von Anforderungsdokumenten.

bedacht werden, dass es auch möglich ist, Indikationen zu vermeiden, ohne das Problem zu lösen.

4.3.4 Vergleichbare Studien

Die Qualität von Requirements Engineering wurde auch in anderen Studien untersucht [56, 85, 123, 68, 158]. Die Ergebnisse dieser Studien waren eine wertvolle Grundlage für dieses Experiment, da sie es erlaubten, die Softwareprojekte unserer Studenten mit industriellen Softwareprojekten zu vergleichen. Damit dieses Experiment in ähnlicher Weise für Andere nützlich ist, werden hier die Ergebnisse miteinander verglichen.

Forsberg [68] hat die Beziehung zwischen der Zeit, die in einem Projekt für Requirements Engineering verwendet wird und dem Projekterfolg verglichen. Demnach ist es ratsam, ca. 20 % der Projektzeit mit Requirements Engineering zu verbringen. Diesen Ergebnissen kann im Rahmen des Experiments hier nichts hinzugefügt werden, da die Studenten in den Projekten nicht selbst bestimmen können, wieviel Zeit sie für Requirements Engineering aufbringen. Alle Projekte verbringen grob 20 % ihrer Zeit mit der Anforderungsanalyse. Da diese Zeitspanne fest vorgegeben ist, ist es erst möglich eine Verbindung zwischen der Qualität der Anforderungsspezifikation und Projekterfolg zu sehen.

So und Berry haben untersucht, wie Prozess-Änderungen das Requirements Engineering verbesserten [158]. Dazu wurden zwei aufeinander folgende Versionen eines großen Software-Produkts miteinander verglichen. Das ermutigende Ergebnis ist, dass sich mehr Aufwand beim Requirements Engineering auszahlt. Der Nach-

weis basiert auf der geringeren Anzahl von Bug-Reports und Änderungswünschen. Auch diese Ergebnisse können durch das Experiment hier nicht ergänzt werden, weil der Entwicklungsprozess in den studentischen Projekten keinen Änderungsmanagementprozess beinhaltet. Außerdem werden nach Projektende auch keine Bugs mehr verfolgt. Aus diesem Grund weicht die Definition des Projekterfolgs zu sehr von [158] ab. Davon abgesehen wird auch in diesem Experiment bestätigt, dass gutes Requirements Engineering die Kundenzufriedenheit erhöht.

Olsson et al. [123] haben das Verhältnis zwischen funktionalen und nicht-funktionalen Anforderungen bei Sony-Ericsson untersucht. Demnach sind etwa 40 % der spezifizierten Anforderungen nicht-funktional. Dieses Ergebnis ist sehr interessant für die Vergleichbarkeit studentischer und industrieller Softwareprojekte: Es zeigt einen quantifizierbaren Unterschied zwischen den studentischen Projekten und industriellen Softwareprojekten, der unabhängig von der Projektgröße ist. Tabelle 4.9 stellt das entsprechende Verhältnis für die Anforderungen der untersuchten Projekte dar. Der Unterschied ist mit 10 Prozentpunkten geringer als erwartet. Die nicht-funktionalen Anforderungen in den studentischen Projekten sind jedoch oft schlecht spezifiziert (zum Beispiel gibt es so gut wie keine testbar spezifizierten Qualitätsanforderungen).

Tabelle 4.9: Anteil der nicht-funktionalen Anforderungen an allen Anforderungen.

Anteil der technischen Anforderungen	10 %
Anteil der Bedienbarkeitsanforderungen	2,8 %
Anteil der Qualitätsanforderungen	17,2 %
Anteil der nicht-funktionalen Anforderungen	30%

Kamata und Tamai [85] untersuchen das Verhältnis zwischen der Qualität einer Anforderungsspezifikation und Projekterfolg. Im Gegensatz zu dem Ansatz in dem hier vorliegenden Experiment verwenden Kamata und Tamai Daten, die aus normalen Qualitätssicherungsaktivitäten abgeleitet wurden. Qualitätsbeauftragte beurteilen jeden Abschnitt einer Anforderungsspezifikation basierend auf 100 Kriterien. Der Ansatz scheint dabei auf subjektiv eingeschätzten Kriterien und der Erfahrung der Qualitätsbeauftragten zu basieren. Auf Grund der Reife der Organisation können die Evaluationsergebnisse dennoch als wiederholbar eingestuft werden.

Die Qualität einer Anforderungsspezifikation wird berechnet, indem die Bewertungen Anforderungsartefakten zugeordnet werden (hier: Unterabschnitte der Anforderungsspezifikation nach [3]). Dies erlaubt es Kamata und Tamai kritische Unterabschnitte für den Projekterfolg zu identifizieren. Interessant ist der Ansatz, das Qualitätsprofil auf Basis einer weitverbreiteten Struktur für Anforderungsspezifikationen zu berechnen, da dies die Vergleichbarkeit erleichtert.

Um die Ergebnisse des hier vorliegenden Experiments mit den Ergebnissen von Kamata und Tamai zu vergleichen, wurden mehrere Schritte unternommen:

1. *Fundamentale Unterschiede der beiden Ansätze beschreiben:* Im Gegensatz zu [85] bezieht sich dieses Experiment vor allem auf formale Aspekte. Es wurde darauf geachtet, Metriken so objektiv wie möglich zu wählen, anstatt auf erfahrene Qualitätsbeauftragte zu setzen. Außerdem zielt das Experiment hier darauf ab, die Gesamtqualität zu untersuchen. Das Hauptergebnis ist dasselbe: eine starke Korrelation zwischen der Qualität von Anforderungsspezifikationen und Projekterfolg. Dabei ist die Definition von Projekterfolg unterschiedlich: Kamata und Tamai definieren Projekterfolg nach [2] unabhängig von der Akzeptanz des Kunden als *in Zeit und Budget*. Abbildung 4.13 zeigt eine Abbildung dieser Definition auf die hier verwendete Definition.
2. *Ergebnisse beider Ansätze aufeinander abbilden und vergleichen:* Um die Ergebnisse aus Abschnitt 4.3.3 mit denen von Kamata und Tamai zu vergleichen, müssen die Ergebnisse jeder einzelnen Metrik auf Werte zwischen 0 und 1 normiert werden. Einige der inhaltsspezifischen Metriken können direkt auf die vorgeschlagene Struktur abgebildet werden. Die anderen Metriken messen die Anforderungsspezifikation als Ganzes. Daher musste jeder Befund zu so einer Metrik nach dem Abschnitt gefiltert werden, in dem er auftritt. Danach müssen die Abschnitte der studentischen Anforderungsspezifikationen auf die Abschnitte einer Spezifikation nach [3] abgebildet werden.

Basierend auf dieser Abbildung können die Profile von erfolgreichen und gescheiterten Projekten mit entsprechenden Profilen aus [85] verglichen werden. Das Ziel dabei war, zu überprüfen, ob auch in den studentischen Anforderungsspezifikationen kritische Bereiche identifiziert werden können.

Wie in Abbildung 4.14 gezeigt wird, wurden dabei keine guten Ergebnisse erreicht. Der Hauptgrund dafür ist, dass die Metriken des hier vorgestellten Experiments die Qualität einzelner Abschnitte nicht fair messen, zum Beispiel sind einige Bereiche nur durch sehr wenige Metriken abgedeckt.

Viele der Metriken betreffen die Abschnitte 1.3 und 3.2 nach [3], aber nur die allgemeinen Metriken der formalen Anforderungsqualität treffen auch auf Abschnitt 3.7 zu. Dennoch scheint es sich basierend auf diesem Vergleich zu bestätigen, dass bei den studentischen Projekten genauso wie in [85] nicht alle Abschnitte gleich viel zum Projekterfolg beitragen.

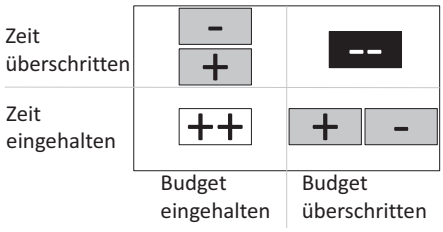


Abbildung 4.13: Vergleich der hier gewählten Definition (erfolgreiche Projekte sind in Kategorie ++) von Projekterfolg mit Standarddefinitionen (erfolgreiche Projekte sind in Zeit und Budget) [2, 85].

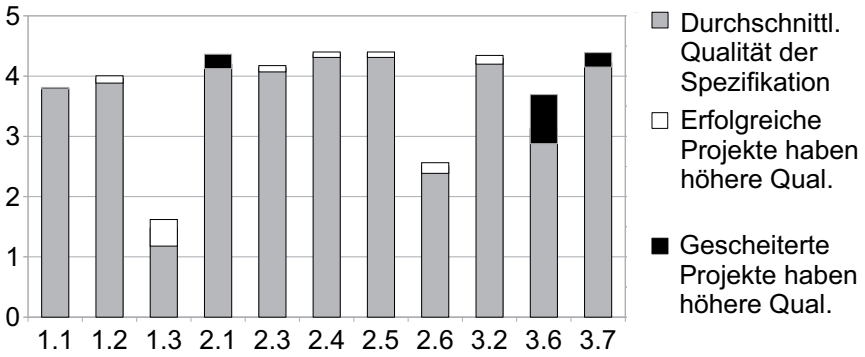


Abbildung 4.14: Qualität stud. Anforderungsspezifikationen abgebildet auf Abschnitte der IEEE-Vorlage [3]. In manchen Abschnitten waren gescheiterte Projekte im Durchschnitt besser als erfolgreiche.

4.3.5 Diskussion der Validität

Nach Wohlin et al. [173] muss sich jede empirische Studie mit bestimmten Gefährdungen der Validität auseinandersetzen. Dies dient auch der Vergleichbarkeit der Ergebnisse, da je nach Situation entschieden werden kann, ob diese Gültigkeit besitzen.

Validität des Aufbaus

Im Rahmen der Validität des Aufbaus werden Probleme diskutiert, die die Abbildung der theoretischen Konstrukte auf die untersuchten Variablen des Experiments betreffen.

Im Rahmen dieses Experiments sind zwei Probleme zu beachten. Da alle Messungen von einer Person durchgeführt worden sind, können sich die Messungen mit der Zeit verschoben haben, da die Person immer vertrauter mit dem Mess-

prozess wurde. Das zweite Problem betrifft die Ermüdung. Das Problem, dass der Messende mit der Zeit ermüdet und sich dadurch die Ergebnisse unnachvollziehbar verzerren, ist hier sehr relevant. Die Wahrscheinlichkeit dafür, dass dieses Problem eingetreten ist, wird aber als eher gering eingestuft, da nur eine Anforderungsspezifikation pro Tag vermessen wurde.

Interne Validität

Die interne Validität betrifft die Möglichkeit, korrekte Schlüsse auf die Kausalität der beobachteten Zusammenhänge aus den erhobenen Daten zu ziehen.

Um zu entscheiden, ob der Projekterfolg wirklich an der Qualität der Anforderungsspezifikation und nicht nur an der besseren Leistung einzelner Gruppen liegt, wäre es sinnvoll gewesen, die Anforderungsspezifikationen nach der Analysephase zwischen den Gruppen zu tauschen. Der Aufbau dieses Experiments ist jedoch stark durch den Aufbau der studentischen Projekte beeinflusst, die darin untersucht wurden. Unabhängig von dem Experiment sollten Studenten darin lernen, ein vollständiges Softwareprojekt durchzuführen. Deshalb war ein solcher Tausch nicht möglich. Damit ist das eigentliche Ziel des Experiments, einen kausalen Zusammenhang zwischen Qualität der Anforderunsgsdokumente und Projekterfolg zu zeigen, formal nicht zu erreichen. Das Experiment zeigt zunächst nur eine Korrelation und es ist durchaus möglich, dass die beiden Variablen *Qualität der Anforderungsdokumente* und *Projekterfolg* kausal von einer nicht untersuchten dritten Variablen abhängen (zum Beispiel individuelle Stärke der beobachteten Studenten).

Nach dem *Gesetz von Boehm* (vergleiche Endres und Rombach [57]) ist ein kausaler Zusammenhang jedoch wahrscheinlich: Boehm hat gezeigt, dass Fehlerkosten steigen, je länger fehlerhafte Artefakte im Entwicklungsprozess verbleiben [29]. Dies liegt auch daran, dass im Laufe des Projekts weitere Artefakte auf den fehlerhaften Artefakten aufbauen. Behebt man den Fehler erst spät, muss man auch die darauf aufbauenden Artefakte überprüfen und gegebenenfalls verbessern. Schließlich sind auch Folgefehler nicht auszuschließen.

Externe Validität

Die Diskussion der externen Validität betrifft die Möglichkeit über den Geltungsbereich der empirischen Studie hinaus Schlüsse zu ziehen, um zum Beispiel Ergebnisse in die Industrie zu transferieren.

Ein Problem kann hier sein, dass alle untersuchten Projekte in der Lehre an einer Universität durchgeführt worden sind. Dies könnte die Übertragbarkeit der Ergebnisse in industrielle Softwareprojekte gefährden. Methodisch macht es kei-

nen Unterschied, ob studentische oder industrielle Softwareprojekte auf Basis der hier vorgestellten Metriken beurteilt werden. Der Umstand, dass alle untersuchten Projekte einen sehr ähnlichen Hintergrund haben, erhöht hingegen die Qualität der Daten, da Störfaktoren minimiert werden. Die Validität der Schlusfolgerung ist dadurch sogar höher, als wenn Industrieprojekte untersucht worden wären.

Validität der Schlussfolgerung

Mit der Validität der Schlussfolgerung ist die Möglichkeit gemeint, statistisch signifikante Schlüsse aus einer empirischen Studie zu ziehen.

Ein kleineres Problem betrifft die statistische Aussagekraft der Ergebnisse. 16 Spezifikationen wurden von einer Person intensiv begutachtet. Die beträchtliche Zahl von weiteren 24 Spezifikationen wurde von anderen Personen analysiert. Die hier präsentierten Ergebnisse wurden dabei bestätigt.

Das Hauptproblem bei der Validität der Schlussfolgerung ist die Verlässlichkeit der Messungen. Da einige Messungen auf subjektiver Beurteilung beruhen, ist eine gewisse Verzerrung der Ergebnisse wahrscheinlich. Im Rahmen des Experiments wurde das Problem dadurch abgeschwächt, dass alle Messungen von der gleichen Person durchgeführt wurden (nicht Autor dieser Arbeit). Auf diese Weise wurden Messungen verschiedener Projekte konsistent interpretiert. Um ähnliche Messungen verteilt durch verschiedene Personen durchführen zu können, müssen mit der Zeit Messrichtlinien eingeführt werden, um den Einfluss subjektiver Beurteilungen zu reduzieren.

Diskussion der Wiederholbarkeit. Ein wichtiger Einflussfaktor auf die Validität der Schlussfolgerung ist die Wiederholbarkeit: Wie beeinflussen unterschiedliche Assessoren das Ergebnis? Um dies zu untersuchen, wurden im Rahmen des Experiments vier Personen gebeten, das selbe Projekt zu bewerten. Die Ergebnisse einer Person wichen drastisch von den anderen ab, wie Abbildung 4.15 zeigt.

Daher wurden alle unterschiedlich klassifizierten Anforderungen analysiert. Die Hauptgründe für Abweichungen waren:

1. Falsch-Positive, z.B. Anforderungen, die als passiv klassifiziert wurden, in Wirklichkeit jedoch aktiv waren.
2. Interpretationsspielraum, z.B. Identifikation von Prozesswörtern.
3. Subjektive Maße, z.B. Verständlichkeit, Redundanz und technische Begriffe.

Der Effekt von Falsch-Positiven und Interpretationsspielraum kann weiter reduziert werden, indem Assessoren besser geschult werden, Listen mit Schlüsselwörtern erhalten oder von heuristischen Werkzeugen unterstützt werden. Außerdem

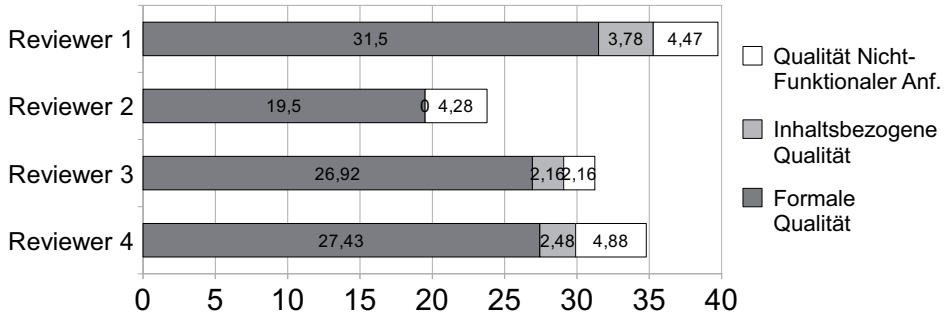


Abbildung 4.15: Die Bewertung der Wiederholbarkeit zeigt Raum für Verbesserung.

sind diese Fehler oft systematisch: Wenn eine Person ein Projekt mehrmals bewertet, wird das Ergebnis mehr oder weniger dasselbe sein.

Ein weiteres Problem ist die lange Dauer der Bewertung. Die Bewertung einer typischen Anforderungsspezifikation dauerte 120 bis 180 Minuten. Darüber hinaus sind einige Metriken recht komplex, wie zum Beispiel die Identifikation von Inkonsistenzen zwischen verschiedenen Anforderungen.

Basierend auf diesen Problemen mit der Wiederholbarkeit entstehen auch Bedenken zur Gültigkeit des Ergebnisses des Experiments, dem Einfluss der Qualität von Anforderungsspezifikationen auf den Projekterfolg. Auf Grund der sehr unterschiedlichen Bewertung des Referenzprojekts durch unterschiedliche Begutachter gibt es keinen absoluten Schwellenwert für die Qualität einer Spezifikation. Allerdings haben alle Gutachter die Referenzspezifikation als schlecht beurteilt. Es ist davon auszugehen, dass alle Reviewer eine gute Spezifikation besser bewerten würden. Dadurch ergibt sich eine relative Ordnung der Projekte und damit immerhin relative Schwellenwerte.

Tabelle 4.10 zeigt die Typen von Messfehlern, die im Rahmen des Experiments identifiziert wurden. Die erste Spalte gibt den Fehlertyp an und die zweite Spalte zeigt ein typisches Beispiel dazu. Die dritte Spalte schlägt Strategien vor, wie diese Messfehler in künftigen Messungen adressiert werden können.

4.4 Zusammenfassung: Problemfeld Anforderungsdokumentation

In diesem Kapitel wurde das Problemfeld der Arbeit beschrieben: Anforderungsdokumentation. Dazu wurden die relevanten Eigenschaften von Anforderungsdo-

Tabelle 4.10: Überblick der Messfehler.

	Typ	**Beispiel**	**Verbesserungsstrategie**
1	Falsch-Positive	Passiv *gefunden*, wo aktiv *verwendet* war.	• Schulungen / Training • Liste mit Schlüsselwörtern • Unterstützung durch heuristische Werkzeuge
2	Interpretationsspielraum	*Identifizieren von Prozesswörtern*	
3	Subjektive Maße	*Verständlichkeit*	• Effekte quantifizieren und limitieren

kumentation hergeleitet und modelliert. Dieser Abschnitt fasst die wichtigsten Ergebnisse zusammen:

- Formalität und Reife von Anforderungsdokumenten
- Einfluss der Qualität von Anforderungsdokumenten auf Projekterfolg
- Zusammenhang interner (inhaltlicher) und externer (messbarer) Qualität von Anforderungsdokumentation

Letztlich lassen sich so heuristische Regeln motivieren, die formale, messbare Qualitätsaspekte überprüfen und gegebenenfalls Feedback geben. Dieses Feedback dient zur Verbesserung der Anforderungsdokumentation, und damit zur Verbesserung der Chance auf Projekterfolg.

Anforderungsqualität begünstigt Projekterfolg. Im vorliegenden Experiment wurden zwei Ziele erreicht: Zum Einen wurde gezeigt, dass die Qualität von Anforderungsspezifikationen mit formalen Kriterien vergleichbar gemessen werden kann. Dies ist im Kontext der Arbeit wichtig, da diese formalen Kriterien als Basis für heuristischen Regeln dienen können. Zum Anderen wurde gezeigt, dass eine starke Korrelation zwischen guter Qualität einer Anforderungsspezifikation und dem Erfolg des Projekts besteht. Die Verbesserung von Anforderungsdokumentation lohnt sich also.

Darüber hinaus war es im Kontext der untersuchten Softwareprojekte möglich, eine untere Schranke für die Qualität zu bestimmen. Projekte, deren Spezifikationen schlechter sind, werden mit hoher Wahrscheinlichkeit fehlschlagen.

Mit dieser Art von Messung steht für Softwareprojekte ein mächtiges Hilfsmittel zur Verfügung. Je nach Ergebnis kann man entscheiden, ob ein Softwareprojekt weiter machen darf, oder ob die Spezifikation deutlich überarbeitet werden muss. Wenn die Qualität besonders schlecht ist, sollte innerhalb des Projekts nach weiteren Problemen gesucht werden. Eine Softwarefirma hat in diesem Zusammenhang eine weitere Option: Zu riskante Projekte können schon früh gestoppt werden.

Messbare Indikatoren deuten auf inhaltliche Probleme. Für den Einfluss der Anforderungsqualität auf den Projekterfolg lassen sich zwei mögliche Begründungen identifizieren. Auf der einen Seite könnte es sein, dass schlecht geschriebene Anforderungen während Entwurf und Implementierung missverstanden werden. Dagegen spricht, dass in den beobachteten Projekten Anforderungsspezifikation, Entwurf und Implementierung von den gleichen Personen durchgeführt wurde, die Kenntnis der Anforderungen haben. Auf der anderen Seite könnte man nach Rupp [142] und DeMarco [51] wie folgt argumentieren: Die Autoren der Anforderungsspezifikation sind durchaus in der Lage, einen gut strukturierten und formal guten Text zu verfassen. Wenn sich Indikationen mangelnder Qualität häufen, dann liegt das häufig daran, dass die Autoren das Problem nicht vollständig verstehen oder es ungelöste Widersprüche in den Anforderungen gibt. Wegen dieses Zusammenhangs ist es mit formalen Metriken möglich, inhaltliche Probleme zu finden.

Der Ansatz, auf Basis formaler Aspekte die Qualität von Anforderungsdokumentation zu messen, ist hilfreich, da Befunde auf gefährliche Stellen in der Spezifikation hinweisen, die dann in der Validierung besonders untersucht werden können. Auf diese Weise kann die Effizienz von Reviews gesteigert werden. Entsprechende Hinweise können aber auch konstruktiv genutzt werden, um den Autoren der Dokumentation auf ein mögliches Problem hinzuweisen. Für diese Arbeit ergeben sich damit folgende Auswirkungen:

Nützlichkeit: Da bereits formal messbare Qualität von Anforderungen einen Einfluss auf den Projekterfolg hat, sind Ansätze, die diese Qualität verbessern nützlich.

Umsetzbarkeit: Gerade die formalen Qualitätsmetriken lassen sich automatisiert messen. Dadurch eignen sie sich als Grundlage für Heuristiken.

5 Ansatz: Erfahrungsbasiertes Requirements Engineering

In Kapitel 4 wurde gezeigt, dass qualitativ gute Anforderungen großen Einfluss auf den Projekterfolg haben. Es ist jedoch nicht einfach, gute Anforderungen zu schreiben. Autoren von Anforderungsdokumentation müssen sich eine Reihe von Gewohnheiten und einen professionellen Schreibstil aneignen, um Mehrdeutigkeiten zu vermeiden. Bereits der Erwerb und die Verbreitung solcher Fähigkeiten und Erfahrungen ist eine Herausforderung. In dieser Situation liefert Feedback durch *heuristische Kritiken* und *abgeleitete Modelle* wertvolle Unterstützung.

Definition 19: *Feedback*

Feedback (auch: Rückkopplung oder Rückmeldung) ist in dieser Arbeit

(i) ein Mechanismus, der einen Teil der eingehenden Informationen analysiert, aufbereitet oder bewertet und das Ergebnis zu deren Quelle zurückführt,

(ii) die zurückgeführten Informationen nach (i).

Erfahrungen über das Schreiben guter Anforderungen können als *heuristische Kritik* formalisiert werden. Werden solche heuristischen Kritiken in ein Werkzeug für das Requirements Engineering integriert, können Anforderungsdokumente automatisch heuristisch überprüft werden. So erhalten Autoren konstruktives Feedback (sogenannte Kritiken), wenn eine Erfahrung anwendbar ist.

Abgeleitete Modelle können es einem Nutzer erlauben, Anforderungsdokumente aus einer anderen Perspektive zu betrachten. Der Nutzer erhält so wichtiges Feedback über die gerade bearbeiteten Anforderungen und kann seinen Dokumentationsstil daran ausrichten und verbessern.

Beide Feedbackarten helfen Nutzern, Wissen und Erfahrung auf ihre aktuelle Aufgabe zu transferieren und durch Anwendung zu erlernen. Dieses Kapitel beschreibt zunächst, wie auf Basis heuristischer Kritiken das Schreiben besserer Anforderungsdokumente erlernt werden kann – sowohl von den Individuen einer Organisation, als auch von der Organisation als Ganzem. Abschnitt 5.1 wiederholt kurz die Grundlagen lernender Software erstellender Organisationen (LSO) und stellt sie in den Zusammenhang von Heuristiken und erfahrungsbasierten Werkzeugen des Requirements Engineering. Danach zeigt Abschnitt 5.2 wie heuristische Kritiken für das Erfahrungsmanagement einer LSO eingesetzt werden

können, insbesondere für das Lernen neuer Erfahrung. Abschnitt 5.3 beschreibt, welche Möglichkeiten abgeleitete Modelle in diesem Zusammenhang bieten: Dieses Konzept erlaubt es vor allem erfahreneren Nutzern eine neue Perspektive auf die dokumentierten Anforderungen einzunehmen und ihre Erfahrungen besser einzubringen.

5.1 Grundlagen des Lernens in Organisationen

In Abschnitt 2.2.1 wurden die Grundlagen lernender Software erstellender Organisationen für diese Arbeit gelegt. Die wesentlichen Definitionen, auf denen diese Arbeit aufbaut, werden hier kurz wiederholt:

Erfahrung. Erfahrung ist ein Drei-Tupel, bestehend aus einer Beobachtung, einer diesbezüglichen Emotion und einer daraus abgeleiteten Schlussfolgerung (oder Hypothese) (vergleiche Schneider [151]).

LSO. Eine lernende Software erstellende Organisation (LSO) pflegt die folgenden Aspekte organisatorischen Lernens: (1) Lernen der Individuen der Organisation, (2) Organisationsweite Sammlung von Wissen und Erfahrung und (3) Kultivieren einer organisationsweiten Infrastruktur für den Austausch von Wissen und Erfahrungen (vergleiche Schneider [151]).

Erfahrungsmanagement. Wenn Erfahrung systematisch erzeugt, evaluiert, verwaltet, überarbeitet, verteilt und angewendet wird, um ein Problem zu lösen, wird dies als Erfahrungsmanagement bezeichnet (vergleiche Schneider [151]).

Das Konzept der heuristischen Kritiken soll das Erfahrungsmanagement in einer LSO im Rahmen des Requirements Engineering unterstützen. Die Idee dabei ist, durch die Integration der heuristischen Kritiken in Werkzeuge des Requirements Engineering wichtige Bausteine der benötigten Infrastruktur für das Erfahrungsmanagement zu liefern und den Austausch von Erfahrungen einer LSO zu verbessern. Ein solches Werkzeug wird im Folgenden *erfahrungsbasiertes Werkzeug* genannt:

Definition 20: *Erfahrungsbasiertes Werkzeug*

Ein erfahrungsbasiertes Werkzeug unterstützt mindestens einen der folgenden Aspekte einer LSO:

Lernen: Erzeugen von neuen Erfahrungen.

Evaluieren: Evaluieren bestehender Erfahrungen.

Verwalten: Verteilung, Aktualisierung und Überarbeitung von vorhandenen Erfahrungen.

Anwenden: Nutzen von Erfahrung in einem speziellen Kontext.

Nach der Definition unterstützt ein *erfahrungsbasiertes Werkzeug* mindestens einen Aspekt des *Erfahrungsmanagement*. Zusätzlich können erfahrungsbasierte Werkzeuge eine domänspezifische Aktivität unterstützen. Handelt es sich dabei beispielsweise um eine Requirements Engineering Aktivität, heißt das entsprechende Werkzeug in dieser Arbeit *erfahrungsbasiertes RE-Werkzeug*.

Im Rahmen dieser Arbeit liegt der Fokus auf dem *Lernen* und *Anwenden* von Erfahrung und Wissen. In den folgenden beiden Abschnitten werden zwei Arten von erfahrungsbasierten RE-Werkzeugen im Detail diskutiert. In Abschnitt 5.2 werden *heuristische RE-Werkzeuge* beschrieben, die Anforderungsdokumente auf Basis heuristischer Kritiken analysieren und dem Nutzer Feedback geben. In Abschnitt 5.3 werden *Perspektivwechselnde RE-Werkzeuge* behandelt. Diese helfen ihren Nutzern, Erfahrungen anzuwenden, in dem sie es ermöglichen, dokumentierte Anforderungen aus einem anderen Blickwinkel zu betrachten.

Beispiel 8: *Erfahrungsbasiertes RE-Werkzeug: HeRA*

HeRA (vergleiche Kapitel 3) ist ein erfahrungsbasiertes RE-Werkzeug, weil es die Analyse und Dokumentation von Use-Cases (Requirements Engineering Aktivitäten) und zusätzlich das Anwenden und Lernen von Erfahrungen unterstützt.

HeRA verfügt sowohl über Eigenschaften heuristischer RE-Werkzeuge (das Kritiksystem für Anforderungen und Glossare) als auch perspektivwechselnder RE-Werkzeuge (durch das Ableiten von UML-Diagrammen, EPKs oder Use-Case-Point-Schätzungen).

5.2 Heuristiken und Erfahrungen

Heuristiken spielen eine große Rolle bei effizienten Algorithmen zu Optimierungs- und Suchproblemen. Für diese Probleme sind oft keine effizienten Algorithmen bekannt. Eine nahezu optimale Lösung, die in vertretbarer Zeit berechnet werden kann, ist in diesem Fall besser, als keine Lösung angeben zu können. Dementsprechend wird der Begriff *Heuristik* in der Informatik allgemein wie folgt definiert:

Definition 21: *Heuristik, allgemein*

In der Informatik versteht man unter einer *Heuristik* Techniken, die gute (beinahe optimale) Lösungen mit angemessenen Berechnungskosten suchen. Dabei garantieren *Heuristiken* weder die Brauchbarkeit noch die Optimalität einer Lösung. In vielen Fällen kann für eine brauchbare Lösung nicht angegeben werden, wie Nahe sie der optimalen Lösung kommt.

(nach Roussell [143])

Diese Definition gilt für Heuristiken zur Verbesserung von Anforderungsdokumentation. Im Rahmen dieser Arbeit wird der Begriff Heuristik allerdings eingeschränkter verwendet. Heuristiken stellen einen Zusammenhang zwischen Indikatoren in Anforderungsdokumenten und Problemen her (vergleiche Abbildung 4.12, Abschnitt 4.4). Daher kann die Definition hier noch weiter geschärft werden:

Definition 22: *Heuristische Regel zur Verbesserung von Anforderungsdokumentation*

Eine heuristische Regel zur Verbesserung von Anforderungsdokumentation stellt einen Zusammenhang zwischen Indikatoren und einem bestimmten Problem in Anforderungsdokumenten her. Sie beschreibt, wie Indikationen (automatisch) in einem Anforderungsdokument nachgewiesen werden können und unter welchen Umständen (Kombination von Indikationen) von einem Problem ausgegangen werden soll. Die heuristische Regel garantiert nicht, dass *a)* das Problem vorliegt, wenn die Kombination von Indikationen vorliegt und *b)* das Problem nicht vorliegt, wenn die Kombination von Indikationen nicht vorliegt.

Die Situation, dass laut einer heuristischen Regel ein Problem vorliegt, weil eine Kombination von Indikationen darauf hindeutet, wird im Folgenden als *feuern* der heuristischen Regel bezeichnet.

Die heuristische Regel garantiert also weder eine optimale Lösung (alle Vorkommen eines bestimmten Problems in einem Anforderungsdokument), noch eine brauchbare Lösung (tatsächliche Vorkommen eines bestimmten Problems in einem Anforderungsdokument) zu finden. Dennoch können heuristische Regeln bei der Dokumentation von Anforderungen nützlich sein. Das hängt teilweise von der Qualität der heuristischen Regel ab, also wie oft tatsächlich ein Problem vorliegt, wenn die Regel feuert (Precision) und wie selten ein Problem vorliegt, wenn die Regel nicht feuert (Recall). Teilweise hängt der Nutzen einer heuristischen Regel aber auch davon ab, wie sie eingesetzt wird. Auch eine schlechte Regel kann unter Umständen besser sein, als gar keine Unterstützung.

Fischer spricht im Kontext der DODE auch von Kritiken [62]. In diesen domänenorientierten Entwicklungsumgebungen sind Kritiken computergeneriertes Feed-

back. Es wird erzeugt, indem die aktuelle Lösung in der Entwicklungsumgebung mit Standard-Lösungen in der Erfahrungsbasis verglichen wird (siehe Abschnitt 2.2.4). Das Konzept der computerbasierten Kritik wird hier auf Basis des Heuristik-Begriffs wie folgt definiert:

Definition 23: ***Heuristische Kritik***

Als heuristische Kritik wird computerbasiertes Feedback zu einer Aktivität oder einem Arbeitsergebnis bezeichnet. Grundlage der Kritik ist eine Erfahrung. Eine heuristische Kritik besteht aus

a) einer heuristischen Regel, die vom Computer angewendet werden kann,

b) einer Kritikalität,

c) einer bedeutungsvollen und konstruktiven Meldung.

Sie kann zudem noch eine ausführliche Beschreibung der ursprünglichen Beobachtung aus der zugrunde liegenden Erfahrung enthalten. Wenn die heuristische Regel (a) feuert, wird die kritische Formulierung der Beobachtung (c) dem Durchführenden der Aktivität oder dem Autoren des Arbeitsergebnisses als Feedback gegeben. Die Darstellung des Feedbacks soll sich an der Kritikalität (b) orientieren.

Um heuristische Kritiken zur erfahrungsbasierten Verbesserung der Dokumentation von Anforderungen einzusetzen, muss der Zusammenhang zwischen Heuristiken und Erfahrungen geklärt werden. Abbildung 5.1 zeigt eine Erfahrung wie oben definiert im Zusammenhang von beobachtbaren Indikatoren und Problemen. Die Emotion dient dazu, die Beobachtung zu werten. Interessante Erfahrungen im Kontext dieser Arbeit betreffen zu vermeidende Probleme. Die Hypothese stellt den vermuteten Zusammenhang zwischen einem (nicht notwendigerweise automatisch messbaren) Indikator und einem Problem her.

Tabelle 5.1 zeigt ein Beispiel für eine Erfahrung und eine daraus abgeleitete heuristische Kritik. Ziel ist es, dass die heuristische Regel (als Bestandteil (a) der heuristischen Kritik) die Anwendbarkeit einer Erfahrung feststellen kann. Die heuristische Regel wird dabei aus der Hypothese oder Schlussfolgerung einer Erfahrung abgeleitet. Wenn eine *ähnliche* Situation beobachtet werden kann, feuert die heuristische Regel. Da die Heuristik auf der Hypothese oder Schlussfolgerung aus einer Erfahrung beruht, kann sie schon auf Grund dieser Vorlage nicht als allgemeingültig angesehen werden.

Beispiel 9: *Implementierung heuristischer Kritiken in HeRA*

HeRA (siehe Kapitel 3) verwaltet heuristischen Kritiken in XML (siehe Listing 5.1). Jede heuristische Kritik erhält eine eindeutige ID, damit Änderungen und

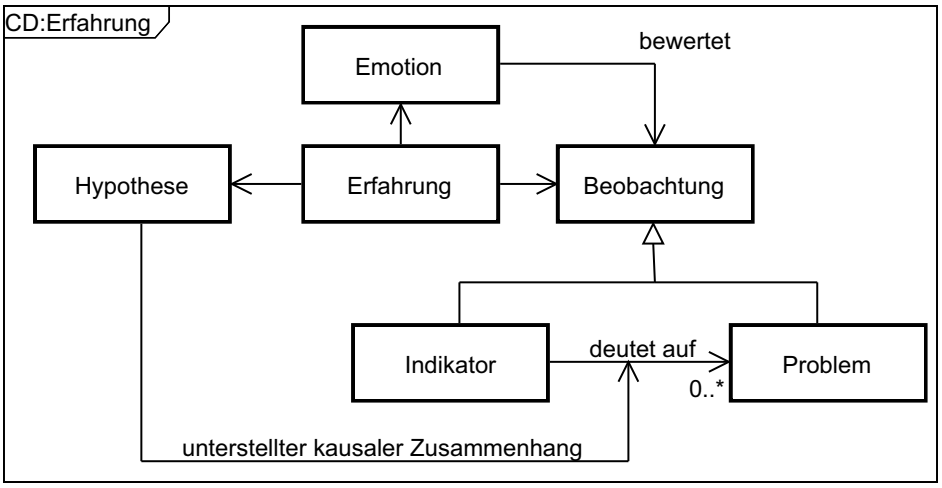


Abbildung 5.1: Modell einer Erfahrung im Kontext von Problemen und Indikatoren von Anforderungsdokumentation.

Ergänzungen verwaltet werden können. Typ und Gewicht einer Kritik geben die Kritikalität an. Die heuristische Regel (als Javascript) ist ebenfalls in dem XML-Fragment enthalten, hier aber zur besseren Lesbarkeit in Listing 5.2 ausgelagert.

Listing 5.1: Beispiel einer heuristischen Kritik in HeRA.

```
<experience type="warning" id="1182252237392345267" weight="0.8">
  <teaser>Diese Anforderung enth&auml;lt passiv.</teaser>
  <fulltext>
    In passiven S&auml;tzen wird oft das Subjekt ausgelassen. Bei Anforderungen
      muss jedoch stets klar sein, wer etwas tut. Formulieren Sie den Satz um und
      verwenden Sie dabei aktiv.
  </fulltext>
  <parameter>
    wird, wurde, worden, werden, gewesen, werde, wirst, werdet
  </parameter>
  <heuristic>...</heuristic>
  <comments />
  <ignorecomments />
</experience>
```

Beispiel 10: *Implementierung heuristischer Regeln in HeRA*

Listing 5.2 zeigt eine Implementierung der heuristischen Regel zu der Kritik aus Tabelle 5.1 und Listing 5.1. Die Parameter aus Listing 5.1 stehen in der Javascript-Umgebung als Array zur Verfügung, genau wie die Datenstruktur `Projekt`. Diese Implementierung verwendet einen recht einfachen Indikator dafür, dass ein Pro-

Tabelle 5.1: Gegenüberstellung von Erfahrung und heuristischer Kritik.

Aspekt der Erf.	**Erfahrung**	**Heuristische Kritik**	**Aspekt der Heur. K.**
(i) Beobachtung	Anforderung wurde missverstanden, weil wir nicht definiert hatten, wer zuständig ist (Passiv)	Wenn Passiv in einer Anforderung entdeckt wird...	*(a) heuristische Regel*
(ii) Emotion	Wir haben eine Woche gebraucht, um das Modul zu überarbeiten – nur wegen einer schlampigen Formulierung!	...zeige eine Warnung...	*(b) Kritikalität*
(iii) Schlußfolgerung / Hypothese	Wir sollten Passiv vermeiden, *weil* immer genau beschrieben werden sollte, wer für eine Aktion verantwortlich ist!	...die den Nutzer auffordert, Aktiv zu formulieren und die Verantwortlichkeit zu definieren	*(c) bedeutungsvolle und konstruktive Meldung*

blem vorliegt. Als Metrik wird die Anzahl verschiedener Formen von *werden* in bestimmten Feldern eines Use-Cases gemessen, der Wertebereich des Indikators ist > 0. Wird also mindestens eine Form von *werden* gefunden, liegt nach dieser heuristischen Regel eine Indikation vor und es soll von dem Problem ausgegagen werden, das in dem Use-Case der Verantwortliche für eine Aktion ausgelassen wurde. HeRA reagiert darauf, indem eine Warnung mit dem Teaser aus Listing 5.1 angezeigt wird.

Listing 5.2: Beispiel einer heuristischen Regel in HeRA.

```
var attributes = ["title", "description", "precondition", "minimumGuarantee", "
    successGuarantee" , "trigger"];
var usecases = project.getAllUseCases();
for (var i = 0; i < usecases.length; i++) {
  for (var j = 0; j < attributes.length; j++) {
    for (var k = 0; k < parameter.length; k++) {
      if (usecases[i].getValue(attributes[j]).indexOf(parameter[k])>=0) {
        contextList.addContext(parameter[k], project, UseCases[i],
        attributes[j]); break;
      }
    }
  }
}
```

Heuristiken kommen in der Informatik immer dann zum Tragen, wenn die Berechnung der idealen Lösung eine zu hohe Komplexität (Speicherbedarf / Laufzeit) hat, oder gar nicht möglich ist. Im Falle des Umgangs mit natürlicher Sprache spielt auch die Schwierigkeit eine Rolle, überhaupt ein eindeutiges Ergebnis zu berechnen, da natürliche Sprachen in der Regel kontextsensitiv sind. Im Rahmen dieser Arbeit sollen heuristische Kritiken genutzt werden, um Erfahrungen zu dokumentieren. An dieser Stelle müssen daher vereinfachte heuristische Regeln zugelassen werden, wenn sie dadurch schnell (ohne lange Entwicklungszeit) eingesetzt werden können. Dies erlaubt es eine geringere Verlässlichkeit heuristischer Kritik in Kauf zu nehmen und dafür schnell nützliche Kritiken auf Basis von Erfahrungen zur Verfügung stellen zu können. Nicht optimale Regeln können dann im Einsatz verbessert, unbrauchbare Regeln aussortiert werden. Erweist sich eine heuristische Kritik als besonders wertvoll, kann ihre heuristische Regel später durch einen komplexeren Algorithmus ersetzt werden.

Es gibt zwei konzeptionelle Unsicherheiten, die Einfluss auf die Verlässlichkeit heuristischer Regeln haben:

1. Die Schlussfolgerung, die im Rahmen einer Erfahrung aus einer Beobachtung gezogen wird, muss nicht allgemein richtig sein.
2. Wenn die heuristische Regel aus dieser Schlussfolgerung abgeleitet wird, darf sie vereinfacht werden.

Im folgenden Abschnitt wird die Verlässlichkeit sowie andere wichtige Aspekte von heuristischen Kritiken und deren Einfluss auf das Lernen näher betrachtet.

5.2.1 Modell des heuristikbasierten Lernens

Das Erlernen neuen Wissens und neuer Erfahrungen kann in einer LSO gemäß Definition auf Seite 34 auf drei Gebieten erfolgen:

1. Lernen der Individuen.
2. Organisationsweites Sammeln von Wissen und Erfahrungen.
3. Verbesserung der Infrastruktur für den Austausch von Wissen und Erfahrung.

Erfahrungsbasierte RE-Werkzeuge sind Teil einer Verbesserung der Infrastruktur für den Austausch von Wissen und Erfahrung. Dieser Abschnitt stellt dar, wie diese Werkzeuge das individuelle Lernen und das organisationsweite Sammeln von Wissen und Erfahrungen unterstützen. Eine Zusammenfassung des Modells wurde auf der Requirements Engineering Konferenz 2009 vorgestellt [106].

Schneider bezeichnet Erfahrung als eine Form des Wissen, die *involviert sein* erfordert ("Experience needs Involvement" [151]). Damit eine Person Erfahrung sammeln kann, muss sie in der entsprechenden Domäne aktiv sein.

Nach Schön hat diese Person das Wissen aber nur genau in dem Moment parat, wenn es angewendet wird ("Reflection in Action" [146]). Fragt man hingegen erst später, was der wesentliche Schritt der Aktivität war, bekommt man keine geeignete Antwort mehr. Vielmehr muss der Praktiker während seiner Arbeit unterbrochen werden, um ihm Gelegenheit zur Reflexion der letzten Aktivität zu geben. Dieser Vorgang wird bei Schön *Breakdown* genannt [146].

Ein *Breakdown* kann dabei zu zwei wünschenswerten Effekten führen. Nachdem die handelnde Person die letzte Aktion überdenken konnte, kann sie diese entweder für unzulänglich erkennen und zurücknehmen oder als zielführend erkennen. In beiden Fällen lernt die Person eine Hypothese für die akute Situation, die zum *Breakdown* geführt hat. Ohne diese Unterbrechung wäre diese neue Erfahrung nicht möglich gewesen. Eine Form der Werkzeugunterstützung beim individuellen Lernen kann die Herbeiführung eines *Breakdowns* sein.

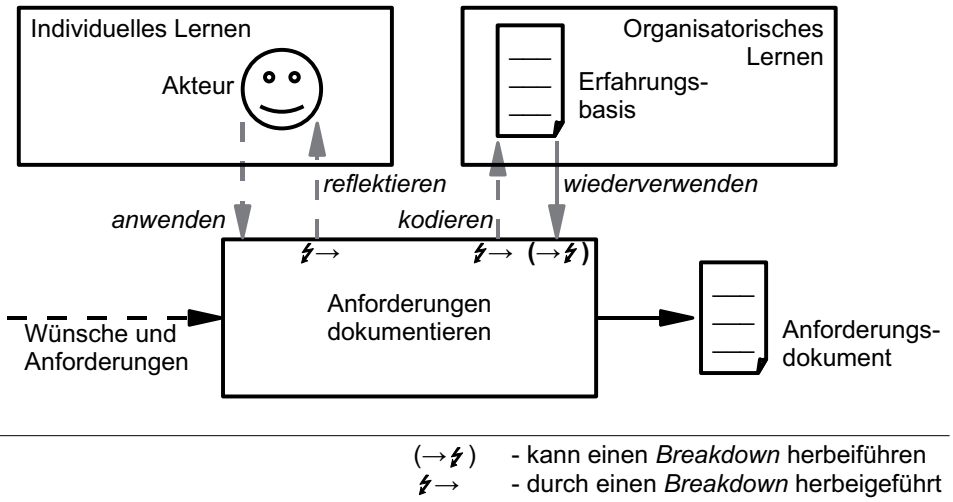


Abbildung 5.2: Informationsflüsse des individuellen und organisatorischen Lernens durch heuristisches Feedback beim Dokumentieren von Anforderungen.

Abbildung 5.2 zeigt diese beiden wichtigen Aspekte des Lernens und wie sie durch Feedback heuristischer Kritiken unterstützt werden. Individuelles Lernen (links) und organisatorisches Lernen (rechts) finden während der Aktivität *Anforderungen dokumentieren* (unten) statt.

Blitze stehen für einen *konstruktiven Breakdown*. Hier wird also die eigentliche Arbeit, das Dokumentieren von Anforderungen, unterbrochen, um dem Autoren die Gelegenheit zu geben, über die Tätigkeit zu reflektieren [146, 64].

Individuelles Lernen: Reflektieren und Anwenden.

Selbst einem gut ausgebildeten Analysten können unter Druck Fehler unterlaufen. In solchen Fällen ist es gut einen Mentor zu haben, der konstruktives Feedback gibt, bis sich aus theoretischem Wissen gute Angewohnheiten entwickelt haben. Heuristische Kritiken können solches Feedback geben.

Beispiel 11: *Reflektieren und anwenden*

Eine heuristische Regel feuert, da sie Passiv in einer Anforderung identifiziert hat. Das erfahrungsbasierte RE-Werkzeug markiert die Anforderung gemäß der Kritikalität der heuristischen Kritik und stellt die Meldung als Warnung dar. Dies kann zu einem Breakdown (*wiederverwenden* in Abbildung 5.2) führen. In diesem Fall hält der Autor inne und überfliegt die gerade spezifizierte Anforderung. Er ärgert sich, dass er unterbewusst schon wieder eine passive Formulierung gewählt hat (*wiederverwenden* in Abbildung 5.2) und korrigiert den Fehler (*anwenden* in Abbildung 5.2). Bei der nächsten Anforderung achtet er bewusst darauf, sie im Aktiv zu formulieren.

Reflexion. Wenn eine heuristische Regel feuert, erhält der Analyst Feedback in Form einer heuristischen Kritik. Dadurch wird er in seiner Aufgabe unterbrochen und erhält die Chance, über die Aktion zu reflektieren, die er gerade durchgeführt hat. Heuristische Kritiken erleichtern also Lernen durch Reflexion, wenn sie direkt auf den Eingaben des Analysten angewandt werden.

Anwenden. Ein gut ausgebildeter Analyst kennt Regeln, nach denen gute Anforderungen geschrieben werden (zum Beispiel nach Rupp [142]). Dieses Wissen bringt er mit ein, während er Anforderungen dokumentiert. Unter Druck ist das allerdings nicht immer einfach. Erinnerungen und Warnungen helfen in so einem Fall, das Wissen anzuwenden und zu wiederholen. Auf diese Weise wird aus abstraktem Wissen eine Angewohnheit, während zugleich eine bessere Spezifikation entsteht.

Organisatorisches Lernen: Wiederverwenden und Kodieren.

Wie in Tabelle 5.1 und den Listings 5.1 und 5.2 gezeigt, können Erfahrungen in Form von heuristischen Kritiken kodiert werden. Für eine LSO ist dies besonders wertvoll:

Wiederverwenden. Basierend auf heuristischen Regeln können mit Hilfe von Computern Situationen gefunden werden, die der Beobachtung entsprechen, welche zu der ursprünglichen Erfahrung geführt hat. Je nach der ursprünglich berichteten Emotion zeigt eine mehr oder weniger deutliche Fehlermeldung potentielle Verbesserungen auf. So können die Erfahrungen der Organisation leicht angewendet werden.

Kodieren. Um Erfahrungen derart leicht anwendbar zu machen, müssen sie geeignet kodiert werden. Darüber hinaus müssen Mechanismen vorgesehen werden, die Erfahrungen um neue Erkenntnisse zu erweitern. Dies ist auch deshalb notwendig, weil heuristische Kritiken nicht immer richtig liegen. Darüberhinaus sind sie auch nicht immer anwendbar.

Beispiel 12: *Passiv in Bedingungen*

Passiv ist in Bedingungen durchaus akzeptabel: *Wenn die Datei gespeichert wurde, ...* In diesem Fall ist es egal, wer für das Speichern der Datei verantwortlich ist.

Auf den *Breakdown* durch eine heuristische Kritik folgt in der Regel die Reflexion des Nutzers. Wird dabei erkannt, dass die Kritik im konkreten Fall nicht anwendbar ist, kann der Analyst die heuristische Kritik verfeinern und zum Beispiel definieren, dass diese nicht für Bedingungen gilt. Auf diese Weise wird der Erfahrungsbasis der Organisation eine neue Erfahrung zugefügt.

Beispiel 13: *Verfeinerung von heuristischen Kritiken in HeRA*

HeRA (vergleiche Kapitel 3) sieht für diese Verfeinerung mehrere Möglichkeiten vor. Die einfachste Möglichkeit ist, eine heuristische Kritik zu kommentieren. Diese Kommentare werden dann mit der Kritik zusammen angezeigt und können den Nutzen erheblich steigern.

Ist eine heuristische Kritik nicht sinnvoll, kann der Analyst diese auch *ignorieren*. Dazu muss er in ein Formular eintragen, unter welchen Umständen die Kritik nicht angezeigt werden soll. Ein optionaler Kommentar erleichtert es, diese Umstände später in die heuristische Regel einfließen zu lassen.

Etwas mehr Zeit nimmt es in Anspruch, kleinere Änderungen an der heuristischen Kritik vorzunehmen. HeRA erlaubt es, den Teaser, die Erläuterung oder optionale Parameter der heuristischen Regel anzupassen (wie zum Beispiel die Schlüsselwortliste der Passiv-Kritik in Listing 5.1). Zudem sind auch Änderungen an der heuristischen Regel direkt möglich. Der Analyst könnte `precondition`

aus der Liste der Attribute in Listing 5.2 entfernen, damit die Kritik bei der Vorbedingung nicht mehr angezeigt wird.

Schließlich ist es möglich, neue heuristische Kritiken anzulegen oder die heuristischen Regeln grundlegend zu ändern.

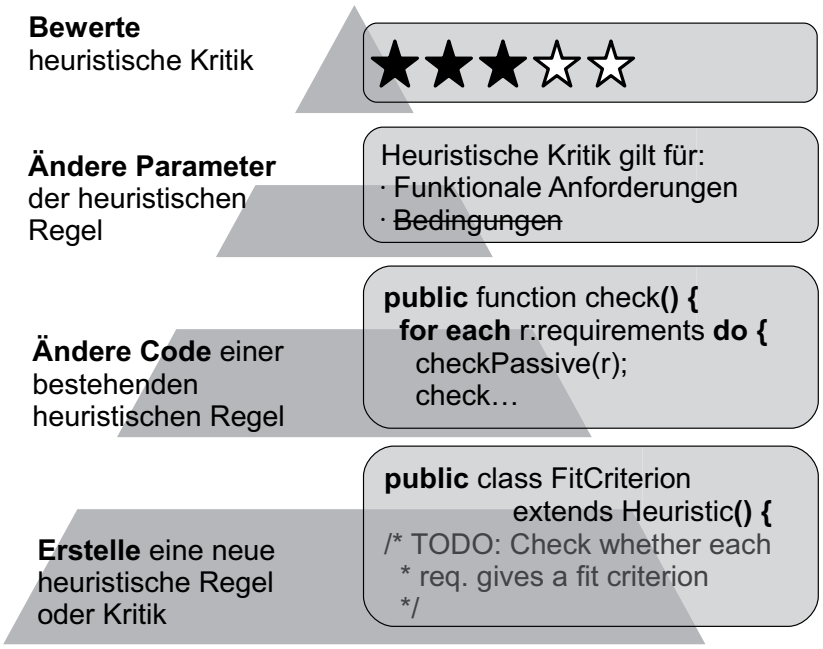


Abbildung 5.3: Die Feedback-Pyramide zeigt verschiedene Ebenen, auf denen ein Nutzer Feedback geben kann. Diese sind mit unterschiedlich viel Aufwand verbunden.

Abbildung 5.3 zeigt die verschiedenen Ebenen, auf denen Nutzer Feedback geben können. Die Fläche der Pyramide symbolisiert den Aufwand, den der Nutzer hat, daneben sind Beispiele dargestellt. Die einfachste Form des Feedbacks, die Bewertung einer heuristischen Kritik, ist in HeRA zur Zeit nur mit Hilfe eines Kommentars möglich. Eine solche Komfortfunktion für die Meinungsäußerung der Nutzer kann ab einer gewissen Größe von Nutzerstamm und Erfahrungsbasis wertvolle Dienste für die Verwaltung der heuristischen Kritiken leisten, da keine als freier Text formulierten Bewertungen der Nutzer ausgewertet werden müssen.

Durch die Kodierung von Erfahrungen als heuristische Kritiken liegen die Erfahrungen zudem in einer gut handhabbaren Granularität vor.

5.2.2 Eigenschaften heuristikbasierten Lernens

Die beiden vorherigen Abschnitte haben gezeigt, dass Erfahrung als heuristische Kritik kodiert werden kann, um so individuelles und organisatorisches Lernen zu unterstützen. Beim Einsatz heuristischer Kritiken müssen deren spezielle Eigenschaften berücksichtigt werden. So erfordert die Auswertung heuristischer Regeln mehr oder weniger Interpretation und die Ergebnisse sind nur zu einem bestimmten Grad verlässlich. Heuristische RE-Werkzeuge können auf Basis der verwendeten Regeln mehr oder weniger Autorität haben und verschieden stark auf konstruktive *Breakdowns* setzen. Im Folgenden werden diese Eigenschaften definiert und erläutert.

Die Grundlage für heuristikbasiertes Lernen auf der individuellen Ebene ist das Zusammenspiel zwischen *Breakdown*, *Reflexion* und der anschließenden *Ergebnissicherung*. An diesen drei Punkten können heuristische Werkzeuge ansetzen.

Eine wichtige Eigenschaft heuristischer RE-Werkzeuge ist die Fähigkeit, einen konstruktiven Breakdown herbeizuführen und so individuelles Lernen zu unterstützen. Das heuristische RE-Werkzeug gibt dazu Feedback, mit dem Ziel, den Nutzer zu unterbrechen. Dieses Ziel kann mehr oder weniger ausgeprägt sein. Dies soll an den folgenden drei Beispielen erläutert werden.

Beispiel 14: *Schwacher* Breakdown *in HeRA*

Nachdem ein Nutzer ein Subsystem mit Use-Cases spezifiziert hat, veranlasst er sein Werkzeug (reaktiv) ein Ablaufdiagramm (EPK) aus den Use-Cases abzuleiten (Feedback). Bei Durchsicht des Ergebnisses fallen dem Nutzer Ungereimtheiten auf („Das sieht gar nicht so aus, wie ich es mir vorgestellt habe...“) und der Nutzer beschließt, dem nachzugehen (Breakdown). Scheinbar sind Vor- und Nachbedingungen der Use-Cases nicht konsistent zu einander (Reflexion).

Beispiel 15: *Mittlerer* Breakdown *in HeRA*

Während der Nutzer einen Use-Case schreibt, wird vom heuristischen Werkzeug parallel (pro-aktiv) ein UML-Use-Case-Diagramm abgeleitet (Feedback). Als sein Blick auf das generierte Diagramm fällt, bemerkt der Nutzer zufällig, dass es einen ähnlichen Use-Case schon gibt (Breakdown).

Beispiel 16: *Starker* Breakdown *in HeRA*

Während der Nutzer einen Use-Case schreibt, wird vom heuristischen Werkzeug automatisch (pro-aktiv) festgestellt, dass es bereits einen ähnlichen Use-Case gibt. Das System zeigt eine Warnung, markiert beide Use-Cases und unterbricht damit den Nutzer (Breakdown). Dieser muss nun nachvollziehen, ob die Warnung berechtigt ist (Reflexion) und entsprechend reagieren.

Bei der Entwicklung von heuristischen Werkzeugen ist Fingerspitzengefühl gefragt, damit ein solches Werkzeug als hilfreich und nicht als störend empfunden wird. Diese Bedienbarkeitsaspekte werden in Kapitel 4.1.2 an Hand der Nutzertypen diskutiert. Hier ist hingegen wichtig, in wie fern die Ausprägungen heuristischen Feedbacks in obigen Beispielen heuristikbasiertes Lernen unterstützen. Im ersten Fall gibt ein reaktives System Feedback, dass durch den Nutzer analysiert werden muss. Im zweiten Fall wird das Feedback vom System pro-aktiv bereitgestellt, es muss aber immer noch vom Nutzer analysiert werden. Im dritten Fall analysiert das pro-aktive System selbst und konfrontiert den Nutzer mit dem Ergebnis.

In allen drei Fällen geht es um die Fähigkeit eines erfahrungsbasierten Werkzeugs, einen *Breakdown* herbeizuführen. Diese Fähigkeit wird durch zwei Eigenschaften heuristischer RE-Werkzeuge erreicht: Einerseits muss interpretiert werden, wann eine Situation einen *Breakdown* erfordert, andererseits muss der Zeitpunkt für den *Breakdown* bestimmt werden.

Definition 24: *Interpretierend*

Ein heuristische Werkzeug ist interpretierend, wenn es zu einer Situation Daten sammelt, auswertet und dem Nutzer eine Bewertung präsentiert, die über die direkt vorliegenen Daten hinaus geht. Ein heuristisches Werkzeug ist

- *schwach interpretierend*, wenn es vorhandene Daten aufbereitet, die Bewertung jedoch dem Nutzer überlässt,
- *mittelstark interpretierend*, wenn es die aufbereiteten Daten sortiert oder stark gefiltert darstellt und so eine bestimmte Interpretation nahe legt,
- *stark interpretierend*, wenn es basierend auf aufbereiteten Daten eine Bewertung vornimmt.

Folgendes Beispiel verdeutlicht die Eigenschaft *interpretierend* eines heuristischen Werkzeugs am Beispiel von HeRA (siehe Kapitel 3):

Beispiel 17: *Interpretierendes Feedback in HeRA*

In HeRA unterscheiden sich die verschiedenen Feedback-Zyklen recht stark im Grad der Interpretation. Die Generierung von EPKs ist nur schwach interpretierend, es bleibt dem Nutzer überlassen, Konsistenzfehler zu finden. Die schwache Interpretation liegt in der Annahme, das Use-Cases und EPKs zusammenpassen müssen.

Der Vorschlag von verwendeten Begriffen für das Glossar stellt eine mittlere Interpretation dar, der Nutzer wird durch die spezielle Sortierung dazu gedrängt,

Begriffe in das Glossar aufzunehmen.

Das Kritiksystem für Anforderungen ist stark interpretierend. Das Feedback sagt beispielsweise deutlich aus, dass eine zu kurze Beschreibung oder gleichlautende Titel zweier Use-Cases ein Problem darstellen.

Definition 25: ***Pro-Aktiv***

Die Pro-Aktivität eines heuristischen Werkzeugs beschreibt, wie stark dieses Werkzeug den Zeitpunkt des Feedbacks selbst bestimmt. Ein heuristisches Werkzeug ist

- *schwach pro-aktiv* oder *reaktiv*, wenn es nur auf direkte Anforderung des Nutzers aktiv wird,
- *mittelstark pro-aktiv*, wenn es auf bestimmte Nutzer-Aktionen reagiert,
- *stark pro-aktiv*, wenn es selbstständig den Zeitpunkt des Feedbacks festlegt.

Die Pro-Aktivität soll wieder an Hand von HeRA verdeutlicht werden:

Beispiel 18: *Pro-Aktivität in HeRA*

Die Berechnung der Use-Case-Points geschieht nur dann, wenn der Nutzer dies fordert, ist also reaktiv. Das UML Use-Case-Diagramm wird immer dann aktualisiert, wenn sich an den Use-Cases ein im Diagramm abzubildendes Element ändert und ist daher mittel pro-aktiv. Das Kritiksystem für Anforderungen ist stark pro-aktiv: Das Feedback wird leicht zeitverzögert gegeben, aber nur dann, wenn der Nutzer im zu kritisierenden Kontext arbeitet.

Ein pro-aktives und interpretierendes heuristisches Werkzeug ist in der Lage, selbst darüber zu entscheiden, auf welche Weise Feedback gegeben wird. Wird die Situation auf Basis einer heuristischen Kritik interpretiert, kann aus deren Kritikalität abgeleitet werden, wie stark der Nutzer unterbrochen werden soll. Die Möglichkeiten fangen bei dezenten Hinweisen auf weitere Information am Rande des Bildschirms, die leicht ignoriert und übersehen werden können, an. Ein Beispiel dafür ist das UML Use-Case-Diagramm in HeRA, in dem es farblich keine Änderung gibt. Das Kritiksystem für Anforderungen bindet die Aufmerksamkeit des Nutzers schon deutlich stärker, in dem es an verschiedenen Stellen am Rande des Blickfelds deutliche farbliche Änderungen vornimmt, wenn eine Warnung angezeigt wird. Keines der in dieser Arbeit untersuchten heuristischen RE-Werkzeuge würde jedoch dem Nutzer den Fokus rauben und eine Warnung in einem Fehlerdialog anzeigen.

Die Pro-Aktivität und Interpretationsfähigkeit eines heuristischen Werkzeugs hat also Einfluss darauf, wie gut es einen *Breakdown* herbeiführen kann. Dadurch kann der Nutzer dazu gebracht werden zu reflektieren. Das Feedback, das zum *Breakdown* geführt hat, entspricht der Beobachtung in einer Erfahrung. Durch positives oder negatives Feedback wird beim Nutzer eine Emotion erzeugt. Während der Reflexion kommt der Nutzer zu einer Schlussfolgerung (bzw. Hypothese). Basiert das Feedback selbst auf Erfahrung, kann die zugrunde liegende Hypothese dabei entweder unterstützt oder abgelehnt werden, indem sie durch das konkret beobachtete Negativbeispiel widerlegt wird. Die Erfahrung hinter dem heuristischen Feedback kann aber dennoch wertvoll bleiben, so lange sie *meistens* relevant ist (vergleiche [151]). Um die Verlässlichkeit von heuristischen Regeln bewerten zu können, werden die aus dem Information Retrieval bekannten Maße für *Recall* und *Precision* verwendet (vergleiche [10]):

Definition 26: *Recall einer heuristischen Kritik*

i) Der Recall einer heuristischen Kritik ist das Verhältnis der Anzahl der Fälle, in denen ihre heuristische Regel tatsächlich feuert, zur Anzahl der Fälle, in denen die zugrunde liegende Erfahrung anwendbar ist.

ii) Wird eine Erfahrung durch eine Gruppe von heuristischen Kritiken abgebildet, so ist der Recall der Gruppe von heuristischen Kritiken das Verhältnis der Anzahl der Fälle, in denen mindestens eine der heuristischen Regeln feuert zur Anzahl der Fälle, in denen die zugrunde liegende Erfahrung anwendbar ist.

Kiyavitskaya et al. haben Anforderungen an Werkzeuge zur Identifikation und zur Messung von Mehrdeutigkeiten in natürlicher Sprache beschrieben [90]. Sie fordern, dass solche Werkzeuge einen Recall von 100 % haben müssen. Sonst wären solche Werkzeuge nicht hilfreich, da Analysten dennoch den gesamten Text durchforsten müssten, um die unentdeckten Probleme zu finden.

Im Zusammenhang dieser Arbeit ist zu beachten, dass von einem Gutachter, der in einem Dokument die Anwendbarkeit von Erfahrungen überprüfen soll, in der Regel auch nicht alle Vorkommen entdeckt werden. In vielen Fällen ist der Recall eines menschlichen Reviewers sogar sehr niedrig (vergleiche die stark abweichenden Ergebnisse in Abschnitt 4.3.5 sowie das Abschneiden der Gutachter in Abschnitt 8.3, deren Recall in den meisten Fällen unter 60 % lag). Eine heuristische Kritik kann daher auch bei niedrigem Recall nützlich sein, so lange sie einem menschlichen Gutachter überlegen ist oder ihm zumindest Arbeit abnimmt.

Aus dieser Beobachtung und der Definition von Recall ergeben sich zwei Schwierigkeiten:

1. Es ist nicht objektiv feststellbar, wann eine zugrunde liegende Erfahrung anwendbar ist.
2. Es ist schwer, Werte für niedrigen, mittleren und hohen Recall anzugeben.

Auf diese Probleme wird teilweise bei der Evaluation in Kapitel 8 eingegangen. Als Faustregel werden mehrere Gutachter benötigt, die gemeinsam die Anwendbarkeit einer Erfahrung überprüfen und ein mehr oder weniger einheitliches Votum abgeben. Von einem hohem Recall kann man ab 90 % sprechen, ein mittlerer Recall liegt zwischen 60 % und 90 %. Selbst bei niedrigem Recall sollte man versuchen, besser zu liegen, als eine Kontrollregel, die zufällig feuert.

Definition 27: *Precision einer heuristischen Kritik*

Die Precision einer heuristischen Kritik ist das Verhältnis der Anzahl der Fälle, in denen ihre heuristische Regel feuert und die zugrunde liegende Erfahrung anwendbar ist, zur Gesamtzahl der Fälle, in denen die heuristische Regel feuert.

Kiyavitskaya et al. fordern, dass die Precision hoch liegt [90]. Auf diese Weise wird die Anzahl der Rückmeldungen, die ein Analyst genauer prüfen muss, deutlich reduziert. Als Faustregel kann man von einer hohen Precision sprechen, wenn diese über 80 % liegt. Analog zum Recall kann eine heuristische Kritik aber auch bei sehr geringer Precision wertvoll sein, wenn dem Nutzer unter vielen irrelevanten und falschen Warnungen gelegentlich auch eine zutreffende Warnung präsentiert wird. In diesem Fall sollte jedoch der Recall sehr hoch liegen (nahe 100 %), damit alle vorhandenen Probleme durch die Warnungen auch abgedeckt sind. Nur dann lohnt sich der Aufwand für den Nutzer, die Rückmeldungen manuell durchzugehen (vergleiche Evaluation von HeRA.Glossar in Abschnitt 8.2.2).

Recall und Precision einer heuristischen Kritik sind auf Bezug der zugrunde liegenden Erfahrung definiert. Bei der Evaluation der Autorität eines heuristischen Werkzeugs muss man daher festlegen, wie man mit Fehlern 1. Art (ein Problem wird erkannt, obwohl es nicht vorliegt) und 2. Art (ein Problem wird nicht erkannt, obwohl es vorliegt) der Erfahrungen umgeht. Im Beispiel Passiv kann man der heuristischen Kritik nicht zur Last legen, wenn sie bei Passiv feuert, auch wenn der Akteur bekannt ist. Ebenfalls kann man zur Berechnung nicht heranziehen, ob sie in Fällen feuert, in denen der Akteur zwar nicht bekannt ist, aber kein Passiv vorliegt.

Unabhängig von Fehlern beim Ableiten heuristischer Kritiken aus einer Erfahrung kann bereits die Hypothese (oder Schlussfolgerung) einer Erfahrung unzuverlässig sein. Dies soll am Beispiel der Hypothese zur Erfahrung aus Tabelle 5.1 erläutert werden:

> *Wir sollten Passiv vermeiden,* weil *immer genau beschrieben werden sollte, wer für eine Aktion verantwortlich ist!*

Diese Hypothese liefert für die folgenden Beispiele Fehler:

1. Die Datei wird vom Server automatisch gespeichert.
2. Das Speichern der Datei geschieht automatisch.

Im ersten Fall liegt ein Fehler der 1. Art vor: Die Anforderung ist Passiv, der Fall wird durch die Hypothese als schlecht eingestuft. Dennoch ist definiert, wer für die Aktion verantwortlich ist (der Server).

Im zweiten Fall liegt ein Fehler der 2. Art vor: Die Anforderung ist nicht Passiv, der Fall ist durch die Hypothese nicht abgedeckt. Dennoch ist nicht klar, wer für die Aktion verantwortlich ist (Sowohl Server als auch Client können für das Speichern der Datei verantwortlich sein).

Definition 28: *Verlässlichkeit*

Die Verlässlichkeit einer Hypothese wird definiert über die relative Anzahl der Beobachtungen, in denen sie zutrifft. Zu ihrer Charakterisierung müssen sowohl die Anzahl der Fehler 1. Art (Hypothese trifft zu, aber die Erfahrung ist in diesem Kontext nicht richtig), als auch die Anzahl der Fehler 2. Art (Hypothese trifft nicht zu, aber die Erfahrung wäre anwendbar) berücksichtigt werden.

Heuristische Kritiken werden auf Basis von Erfahrungen erstellt. Ein heuristisches Werkzeug kann daher nicht verlässlicher sein, als die zu Grunde liegenden Hypothesen. Hinzu kommt eine zusätzliche Ungenauigkeit aus möglichen Zugeständnissen an die Komplexität der Heuristik. Die sehr einfache heuristische Regel zur Identifikation von Passiv aus Listing 5.2 basiert zum Beispiel auf der Suche nach allen möglichen Formen von *werden*. Diese Heuristik kann sehr schnell implementiert werden, auch von versierten Nutzern eines heuristischen Werkzeugs. Durch die Einfachheit entstehen aber zusätzlich weitere Fehler der 1. Art (Futur: *Das System wird die Datei automatisch speichern.*) und der 2. Art (Zustandspassiv: *Die Datei ist gespeichert.*).

Definition 29: *Autorität*

Der Grad, zu dem sich Nutzer eines erfahrungsbasierten RE-Werkzeuges auf dessen Feedback verlassen, wird in dieser Arbeit als Autorität bezeichnet.

Diese Definition hat zwei Facetten. Zum Einen kann die Berücksichtigung von Feedback heuristischer Werkzeuge vorgeschrieben sein. Eine Warnung muss berücksichtigt werden, ob sie auf ein tatsächliches Problem hindeutet oder nicht.

Zum Anderen muss sich ein heuristisches Werkzeug diese Autorität durch hohe Verlässlichkeit verdienen. Bei geringer Verlässlichkeit sinkt das Vertrauen in das Werkzeug. Außerdem können viele Fehlalarme die Bedienbarkeit deutlich reduzieren, vor allem, wenn deren Berücksichtigung vorgeschrieben ist. Weist das Feedback andererseits fast immer auf tatsächliche Probleme hin, so kann deren Berücksichtigung problemlos vorgeschrieben werden.

Verlässlichkeit von Hypothesen und Autorität heuristischer Werkzeuge ist demnach eng mit *Recall* und *Precision* verknüpft. Als kombiniertes Maß für beide Eigenschaften kommt das F-Maß in Frage (vergleiche [10]):

$$F = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{5.1}$$

Zu beachten ist, dass dieses Maß die Autorität nur indirekt beschreibt. Unerfahrene Nutzer verlassen sich unter Umständen auch auf unzuverlässiges Feedback. Die Autorität beschreibt, wie ein heuristisches Werkzeug eingesetzt werden soll. Bei der Entwicklung eines Werkzeugs mit hoher Autorität ist unbedingt darauf zu achten, dass die F-Maße der verwendeten heuristischen Regeln diesem Ziel gerecht werden. Als Faustregel kann man von einem hohen F-Maß sprechen, wenn dieses über 0,85 liegt.

Nach *Breakdown* und *Reflexion* findet die Ergebnissicherung statt. Zum Einen sollte die Reflexion zu einer Verbesserung bei der unterbrochenen Aktivität führen. Dies ist Gegenstand der Evaluation der heuristischen Werkzeuge in Kapitel 8. Zum Anderen führt die Reflexion auch dazu, dass neue Erfahrungen gewonnen werden. Um organisatorisches Lernen zu fördern, ist es sinnvoll, diese Erfahrungen als heuristische Kritiken zu kodieren und in die Werkzeuge zu integrieren.

Definition 30: *Lernfähigkeit*

Die Eigenschaft *Lernfähigkeit* eines erfahrungsbasiertes RE-Werkzeugs bezeichnet dessen Fähigkeit, neue Erfahrungen aufzunehmen oder bestehende Erfahrungen zu verfeinern. Ein heuristisches Werkzeug besitzt

- *geringe Lernfähigkeit*, wenn die Erweiterung der Erfahrungsbasis eher Aufgabe von Experten ist,
- *mittlere Lernfähigkeit*, wenn es spezielle Unterstützung der Nutzer zur Erfassung neuer Erfahrungen anbietet,
- *starke Lernfähigkeit*, wenn es die heuristischen Regeln dynamisch durch Beobachtung des Nutzerverhaltens anpassen kann.

Das Lernen innerhalb eines erfahrungsbasierten RE-Werkzeugs kann automatisch geschehen, indem beispielsweise heuristische Regeln durch Anpassung von Ge-

wichten oder anderen Formen maschinellen Lernens verfeinert werden. Hier ist aber auch die Unterstützung bei der Erfassung von Erfahrung gemeint. Wird beispielsweise ein *Breakdown* herbeigeführt, so bietet ein lernfähiges erfahrungsbasiertes Werkzeug seinem Nutzer die Möglichkeit, die Ergebnisse der Reflexion in die Erfahrungsbasis einzupflegen.

Das Spektrum der Lernfähigkeit heuristischer Werkzeuge erstreckt sich über folgende Beispiele:

1. **Keine Lernfähigkeit:** Ein heuristisches RE-Werkzeug, das keine Lernfähigkeit besitzt, beschränkt sich ganz auf die anderen Aspekte. Typischerweise muss der Erfahrungsschatz im Quellcode angepasst werden.
2. **Kommentierbar:** Ein kommentierbares heuristisches RE-Werkzeug stellt seinen Nutzern eine Möglichkeit zur Verfügung, Feedback zu heuristischen Kritiken zu geben. Dieses Feedback muss dann nachträglich in die Erfahrungsbasis eingearbeitet werden.
3. **Nutzerinteraktiv:** Ein heuristische RE-Werkzeuge, das Nutzerinteraktion erlaubt, läßt eine Änderung des heuristischen Erfahrungsschatzes während der Verwendung durch den Nutzer zu. So können heuristische Kritiken bei neuen Erfahrungen direkt angepasst werden.
4. **Nutzerbeobachtend:** Ein heuristisches RE-Werkzeuge ist nutzerbeobachtend, wenn es seine Verwendung durch den Nutzer analysiert und dadurch automatisch lernt.

Beispiel 19: *Lernfähigkeit durch Nutzerbeobachtung*

Ein Beispiel für ein nutzerbeobachtendes heuristisches Werkzeug ist HeRAs Kritiksystem für Glossarbegriffe (siehe Abschnitt 3.3.2). Das System beobachtet, wie der Nutzer mit der Schnittstelle interagiert. Insbesondere lernt das System alle Begriffe, die dem Glossar hinzugefügt werden. Im nächsten Projekt werden Begriffe unter anderem dann für das Glossar vorgeschlagen, wenn das System sie bereits kennt. Wenn der Nutzer sich nun die Vorschlagsliste zeigen lässt, wird ihm gar nicht bewusst, dass er gerade auf Erfahrungen aus alten Projekten zugreift.

Heuristische Kritiken eignen sich als Medium, um Erfahrungen zu kodieren. Sie haben den Vorteil, dass die Beobachtung einer Erfahrung als heuristische Regel kodiert ist. Dadurch können ähnliche Situationen, in denen die Erfahrung möglicherweise anwendbar ist, automatisch identifiziert werden.

Damit die individuell erworbenen Erfahrungen der Organisation zur Verfügung stehen, sind einige zusätzliche Schritte erforderlich. Um organisationsweit zu lernen, muss dazu zusätzlich noch ein zentrales Archiv von heuristischen Kritiken

gepflegt werden. Ein Mechanismus dazu ist im Rahmen der Bachelorarbeit von Rumann entstanden [141]. Neben der technischen Infrastruktur ist es allerdings besonders wichtig, dass es einen Verantwortlichen für die Verwaltung und Pflege von Erfahrungen gibt. Die Verwaltung von Erfahrungen und der Lebenszyklus einer Erfahrung, der sich daraus ergibt, sind Bestandteile des nächsten Abschnitts.

5.2.3 Lebenszyklus von Erfahrung

Im vorherigen Abschnitt wurde ein Modell für das Erlernen neuer Erfahrungen auf Basis von heuristischen Kritiken gegeben. Dabei wurde gezeigt, wie heuristische Kritiken *Breakdowns* erzeugen und so die Entstehung neuer Erfahrungen fördern. Es wurde argumentiert, dass auch organisatorisches Lernen durch die vorgestellten Mechanismen unterstützt wird. In diesem Abschnitt wird ein weiterer wesentlicher Aspekt organisatorischen Lernens untersucht - die Verwaltung auf diese Weise entstandener Erfahrungen.

Dieser Abschnitt ist nach Demings Plan-Do-Check-Act Kreislauf aufgebaut [52]. Demings Ansatz erlaubt die erfahrungsbasierte Prozessverbesserung. Abbildung 5.4 zeigt einen Überblick der erfahrungsbasierten Verbesserung der Dokumentation von Anforderungen mit Hilfe von Heuristiken.

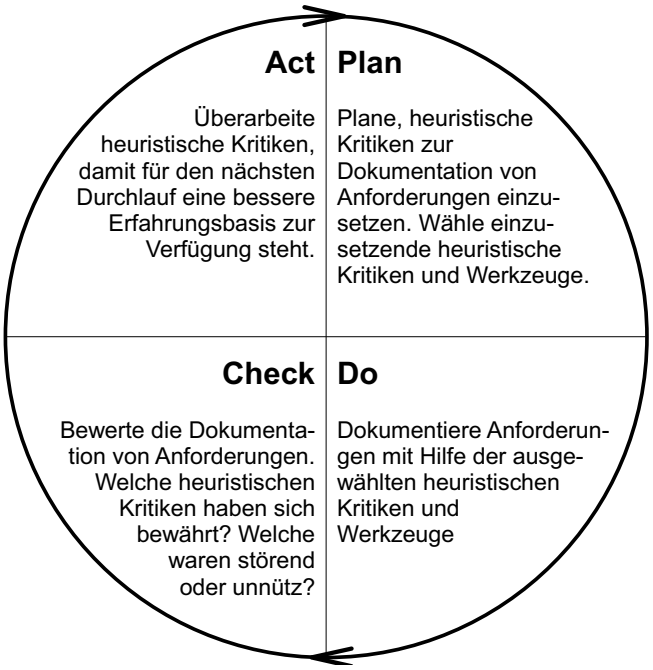


Abbildung 5.4: Erfahrungsbasierte Verbesserung der Dokumentation von Anforderungen mit Hilfe von heuristischen Kritiken (vgl. Deming [52]).

In der ersten Phase (*Plan*) wird geplant, wie vorgegangen werden soll. Im Rahmen dieser Arbeit bedeutet das zu planen, wie erfahrungs- und heuristikbasiert Anforderungsdokumentation erstellt werden kann. Dies beinhaltet insbesondere die Auswahl der heuristischen Kritiken (der Erfahrungen) und der heuristischen RE-Werkzeuge.

In der zweiten Phase (*Do*) wird der Prozess gemäß dem geplanten Vorgehen durchgeführt. Hier werden also die vorher ausgewählten heuristischen Kritiken angewandt, um qualitativ hochwertige Anforderungsdokumente zu erstellen.

In der dritten Phase (*Check*) wird das Vorgehen bewertet. Hier bedeutet das, festzustellen, welche heuristischen Kritiken sich bewährt haben und welche unnütz oder sogar störend sind.

Schließlich werden in der vierten Phase (*Act*) die entsprechenden Schlüsse aus dem gelernten gezogen. Heuristische Kritiken werden überarbeitet, damit für den kommenden Durchlauf eine bessere Erfahrungsbasis für die Auswahl (*Plan*) zur Verfügung steht.

Im Folgenden werden diese Phasen an einem durchgängigen Beispiel erläutert.

Initiale Quellen für Erfahrungen und heuristische Kritiken

Wenn noch keine heuristische Kritiken vorliegen, müssen zunächst Quellen für Erfahrungen identifiziert werden. Dazu bieten sich zum Beispiel Erfahrungen aus alten Projekten an.

Beispiel 20: *Breakdown und Reflexion*

Am Ende der Anforderungsanalyse von Projekt A sollen die Abnahmetests geschrieben werden. Bei der Anforderung “Die Datei wird gespeichert” bleibt der Analyst hängen *(Breakdown)*: Wer speichert die Datei? Tut das der Nutzer oder wird das automatisch von dem System durchgeführt? Der Analyst geht noch einmal die Notizen der Interviews durch, aber ohne Erfolg. Nachfragen im Team bringt nur widersprüchliche Meinungen zu Tage: Nach dem ersten Interview hatte man das als selbstverständlich betrachtet und deshalb im zweiten Interview nicht hinterfragt. Die *Beobachtung* ist also: In dieser Anforderung sind wichtige Informationen nicht enthalten, weil Passiv verwendet wurde. Durch die entstandenen Risiken und zusätzlichen Aufwände ist die damit verbundene *Emotion* sehr negativ. Das Team von Projekt A stellt in der Nachbesprechung die *Hypothese* auf, dass Passiv generell schlecht für die Dokumentation von Anforderungen ist.

Andere Quellen für Anforderungen sind Bücher oder wissenschaftliche Beiträge. Diese liefern oft gut begründete Hypothesen und Vorschläge, wie darauf zu reagie-

ren ist. Praktische Bücher enthalten zudem bewährte Strategien aus der Praxis. Diese *Best Practices* sind ein guter Startpunkt, um geeignete Erfahrungen zu identifizieren.

Wurden geeignete Erfahrungen identifiziert, müssen diese als heuristische Kritiken kodiert werden.

Beispiel 21: *Kodierung*

Das erfahrungsbasierte RE-Werkzeug hilft bereits bei der Erfassung der Erfahrung. Schnell gibt der Analyst die Erfahrung in ein dafür vorgesehenes Formular ein. Dann fährt er mit seiner Arbeit fort. Später greift ein Erfahrungsmanager die so notierte Erfahrung auf. Zu der Hypothese lässt sich mit Hilfe der Assistenten im erfahrungsbasierten RE-Werkzeug und ein bisschen Übung schnell eine entsprechende heuristische Regel implementieren.

Auf diese Weise konnten im Rahmen der Entwicklung von HeRA ein Großteil der Richtlinien für die Dokumentation von Use-Cases aus den Standard-Werken wie Cockburns *Writing Effective Use Cases* [42] und *Patterns for Effective Use Cases* von Adolph et al. [5] übernommen werden [45, 89, 91, 98].

Auswahlphase von Erfahrungen und heuristischen Kritiken

In der Planungs- und Auswahlphase gilt es, für ein konkretes Projekt die geeigneten Erfahrungen auszuwählen. Durch die Kodierung als heuristische Kritiken stehen bereits Erfahrungen für den Einsatz in heuristischen Werkzeugen zur Verfügung. Für den effektiven Einsatz in einem Softwareprojekt kommt es nun darauf an, heuristische Kritiken auszuwählen, die für ihre Nutzer nützlich und angemessen sind (vergleiche Abschnitt 4.1.2).

Fischer nennt diese Phase *Seeding* und meint damit, dass in einem erfahrungsbasierten Ansatz eine kritische Menge nützlicher Erfahrungen gesät werden muss [64]. Bereits diese initiale Erfahrungssammlung muss im Projektteam als nützlich betrachtet werden, da der erfahrungsbasierte Ansatz sonst nicht verwendet wird. Und nur, wenn die Erfahrungen genutzt werden, ist damit zu rechnen, dass das Projektteam bereit ist, neue Erfahrungen zur Verfügung zu stellen.

Anwendungsphase von Erfahrungen und heuristischen Kritiken

Wenn im Planungsschritt eine gute Erfahrungsbasis ausgewählt wurde, stellt sich immer noch das Problem, diese Erfahrungen im Projektalltag auch zu verwenden. Folgende vier Schritte müssen erfolgen, um von einer Erfahrung zu profitieren.

1. Erkennen, dass zu einem gegebenen Problem Erfahrungen existieren. Hier spielt der *Breakdown* erneut eine wichtige Rolle: Einem Nutzer muss erst bewusst werden, dass eine Aktivität womöglich besser (effektiver oder effizienter) durchgeführt werden kann.
2. Auffinden einer relevanten Erfahrung. Die Suche nach relevanten Erfahrungen ist keinesfalls trivial. Damit ein erfahrungsbasierter Ansatz verwendet wird, muss es einfacher sein, eine passende Erfahrung zu finden, als die Aktivität ohne Erfahrungsunterstützung durchzuführen.
3. Anwenden einer Erfahrung. Problematisch ist hier, dass sich die Hypothese oder Schlussfolgerung auf einen anderen Kontext bezieht. Dadurch wird gegebenenfalls eine Transferleistung bei der Anwendung nötig.
4. Das Resultat der Anwendung einer Erfahrung erfassen. Der Einsatz einer Erfahrung ist eine hervorragende Chance, diese zu verfeinern und mehr zu lernen.

Heuristische Werkzeuge spielen ihre Stärken vor allem bei den ersten beiden Schritten aus, indem sie automatisch Situationen erkennen, in denen eine Erfahrung anwendbar ist und diese dann auch automatisch auffinden können. Dies erleichtert es, das Wissen einer Organisation wieder zu verwenden.

Beispiel 22: *Wiederverwenden*

Sobald eine Anforderung von nun an in Passiv formuliert wird, zeigt das erfahrungsbasierte RE-Werkzeug eine Warnung an: Passiv ist für die Dokumentation von Anforderungen ungeeignet. Bei Bedarf ist auch die ursprüngliche Beobachtung abrufbar.

Die Automatisierung dieser wesentlichen Schritte erleichtert die Wiederverwendung nach Abbildung 5.2 deutlich. Die Organisation als solche wird in der Anwendung ihres Wissens besser.

Beispiel 23: *Anwenden einer Erfahrung*

In einem parallel laufenden Projekt B beginnt eine Analystin gerade mit der Dokumentation der Anforderungen. An mehreren Stellen fallen ihr Hinweise auf, dass eine Anforderung im Passiv formuliert ist. Zunächst sind diese Warnungen störend und ärgerlich, aber nach Durchsicht der Beobachtung und der markierten Anforderungen beschließt sie, die meisten Formulierungen zu überarbeiten. An einigen Stellen wäre Passiv nicht weiter schlimm gewesen, aber an einer Stelle fehlt tatsächlich wichtige Information.

Idealerweise unterstützt ein erfahrungsbasiertes Werkzeug über die Wiederverwendung von Erfahrungen hinaus auch das Erbeuten neuer Erfahrungen.

Beispiel 24: *Bewertung der Erfahrung*

Schließlich kommt die Analystin an eine Vorbedingung zu einem Use Case: "Die Datei wurde gespeichert." Es ist egal, wer die Datei gespeichert hat, aber der nächste Schritt darf nur ausgeführt werden, wenn die Datei gespeichert worden ist. Die Analystin fügt der Erfahrung einen Kommentar hinzu, der diese Beobachtung wiedergibt. Zudem teilt sie dem erfahrungsbasierten Werkzeug mit, dass diese Erfahrung in Vorbedingungen nicht mehr angezeigt werden soll. Da die Erfahrung aber ansonsten nützlich war, gibt sie noch eine positive Bewertung an. Wichtig ist, dass diese Funktionen leicht erreichbar und bedienbar sind.

In dem Beispiel wird deutlich, wie Erfahrungen während der Verwendung in einem erfahrungsbasiertem Werkzeug wachsen. Fischer nennt diese Phase *Evolutionary Growth* [64].

Bewertungsphase von Erfahrungen und heuristischen Kritiken

Will man auf Basis von Erfahrungen und Heuristiken lernen, muss man festhalten, was sich bewährt hat und was nicht. In der Bewertungsphase ist das Ziel der LSO, herauszufinden, welche Erfahrungen sich besonders bewährt haben und welche eher ungeeignet waren.

Der LSO steht dabei eine gute Quelle für die Erfassung des Resultats der Anwendung von Erfahrungen zur Verfügung: Nutzer haben heuristische Kritiken verwendet, kommentiert oder deaktiviert. Eine weitere wichtige Grundlage für die Bewertung von heuristischen Kritiken können Statistiken über die Nutzung sein, wenn diese während der Anwendungsphase erfasst wurden. Da die gute Qualität der Erfahrungsbasis ein Erfolgsfaktor für einen erfahrungsbasierten Ansatz ist, sollte unbedingt ein verantwortlicher Erfahrungsmanager für diese Aufgabe bestimmt werden.

Beispiel 25: *Bewerten von Erfahrungen*

Für die gemeinsame Erfahrungsbasis der Projekte A und B ist ein Erfahrungsmanager zuständig. Dieser hat dem Projektmanager von Projekt B schon bei der Auswahl geeigneter Erfahrungen und heuristischer Kritiken unterstützt. Nun ist er sehr gespannt, wie sich die heuristischen Kritiken in Projekt B bewährt haben. Neben seiner normalen Tätigkeit überfliegt er daher regelmäßig die Kommentare. Zwischendurch konnte er so sogar einige Missverständnisse ausräumen und falsch

verstandene Kritiken klären. Jetzt, in der heißen Phase kurz vor der Abnahme der Anforderungsspezifikation in Projekt B, hält er sich allerdings zurück. Er nimmt sich aber vor, nach der Abnahme die Anforderungsanalysten zu ihrer Meinung über die heuristischen Kritiken zu befragen, da diese erfahrungsgemäß kurz vor Abgabe weniger Erfahrungen kommentieren.

Diese Erkenntnisse müssen nun genutzt werden, um die Erfahrungsbasis zu verbessern. So steht für das nächste Projekt eine bessere Grundlage zur Auswahl der Erfahrungen zur Verfügung.

Konsolidierungsphase von Erfahrungen und heuristischen Kritiken

Nach einiger Zeit muss der Erfahrungsmanager die gewachsene Erfahrungsbasis überarbeiten. Auf Basis der Bewertungen aus der Phase zuvor entscheidet der Erfahrungsmanager, heuristische Kritiken zu verfeinern, zu entfernen oder hinzuzufügen. Zusätzlich überarbeitet er auch Texte von Beobachtung und Schlussfolgerung der heuristischen Kritiken.

- Heuristische Kritiken, die von vielen Nutzern deaktiviert wurden, sollten aus der Erfahrungsbasis entfernt werden, oder mit einem Hinweis versehen werden, dass sie für diese Art der Projekte ungeeignet sind.
- Heuristische Kritiken mit vielen Kommentaren müssen vermutlich vom Text her überarbeitet werden. Zum Teil können Kommentare auch Erfahrungen enthalten, aus denen neue heuristische Kritiken abgeleitet werden können.
- Heuristische Kritiken, die häufig gut bewertet wurden, sollten ausgezeichnet werden, damit sie auch für andere Projekte bevorzugt zur Verfügung gestellt werden.

Die Überarbeitung von Erfahrungen und ihre Auswahl für das nächste Projekt bezeichnet Fischer als *Re-Seeding* [64]. Damit schließt sich der Kreis und die verbesserte Erfahrungsbasis steht zur Unterstützung des nächsten Projekts zur Verfügung.

Werkzeuge zur Anforderungsdokumentation, die Erfahrungen und heuristische Kritiken nutzen, können also einen erheblichen Beitrag zum Lernen in einer Organisation beitragen. Bisher wurde hauptsächlich diskutiert, wie dies auf Basis heuristischer Kritiken geschehen kann. Im folgendem Abschnitt werden weitere Mechanismen diskutiert, mit denen erfahrungsbasierte RE-Werkzeuge ihre Nutzer unterstützen können.

5.3 Perspektivwechsel und Erfahrungen

Textuelle Anforderungen beschreiben das zu erstellende System. Für Kunden, Entwickler und Analysten werden aber immer wieder Fragen rund um die Anforderungen wichtig, die auf Basis der rein textuellen Anforderungen nur schwer zu beantworten sind.

- Wie wird das System hinterher aussehen? Beschreiben die Anforderungen ein sinnvoll und angenehm nutzbares System? Dies ist insbesondere dem Kunden oft unklar, vor allem, wenn die Notation formaler wird.
- Wie gliedert sich eine bestimmte Anforderung in den Gesamtablauf? Eine Anforderung kann in ihrem Kontext des Anforderungsdokuments sinnvoll und richtig erscheinen, aber widerpricht womöglich Anforderungen an anderer Stelle.
- Sind die Stakeholder-Wünsche im Rahmen des Projektbudgets überhaupt umsetzbar?

Erfahrene Analysten klären solche Fragen frühzeitig. Schließlich sind das wichtige Schritte der Interpretation und Negotiation:

- Kundenfeedback zu einem Oberflächenprototyp ist oft wertvoller als Kundenfeedback zu einer textuellen Beschreibung.
- Wird das System später von verschiedenen Rollen in mehreren Arbeitsschritten bedient, so müssen die Schritte zueinander passen. Ein Artefakt, das in einem Schritt begonnen wird, muss in einem weiteren Schritt weiterverarbeitet werden. Vorbedingungen für einen Schritt müssen durch einen anderen Schritt erst geschaffen werden.
- Um zu einer geeigneten Priorisierung der Anforderungen durch den Kunden zu kommen, müssen diesem die Kosten transparent gemacht werden. Häufig verbergen sich hinter stark unterschiedlichen Aufwandsschätzungen auch unterschiedliche Interpretationen einer Anforderung.

Aus diesem Grund werden in der Anforderungsanalyse neben den Anforderungen häufig Oberflächenprototypen, Geschäftsprozessmodelle, Aufwandschätzungen und ähnliche anforderungsbezogene Artefakte angefertigt. Dies sind in der Regel mühsame Vorgänge, die einige Zeit in Anspruch nehmen. Die Möglichkeit, diese Artefakte werkzeuggestützt und teilweise automatisch ableiten zu können, ist daher wertvoll.

5.3.1 Abgeleitete Perspektiven und Erfahrungen

Im Rahmen dieser Arbeit ist zu klären, wie das werkzeuggestützte oder automatische Ableiten von neuen Perspektiven das Lernen, Evaluieren, Verwalten oder Anwenden von Erfahrungen unterstützt.

Definition 31: *Perspektivwechselnd*

Die Fähigkeit eines erfahrungsbasierten RE-Werkzeugs, andere Perspektiven aus einem gegebenen Artefakt abzuleiten, wird in dieser Arbeit *Perspektivwechselnd* genannt. Je nach Ausprägung ist das Werkzeug:

- *stark perspektivwechselnd.* Das Werkzeug ist in der Lage, zwei Perspektiven zu verknüpfen, so dass Nutzer Änderungen in beiden Perspektiven durchführen können und diese in die jeweils andere Perspektive überführt werden.
- *mittel perspektivwechselnd.* Das Werkzeug ist in der Lage, eine zweite Perspektive auf ein gegebenes Primär-Artefakt zu liefern. Änderungen sind nur an dem Primär-Artefakt möglich oder werden bei der nächsten Ableitung überschrieben.
- *nicht perspektivwechselnd.*

Genau wie in Abschnitt 5.2.1 wird auch hier zwischen individuellem und organisatorischem Lernen unterschieden.

Individuelles Lernen

Nach Abschnitt 5.2.1 helfen heuristische Kritiken, Anforderungen im Detail zu verbessern. Dazu werden Indikatoren genutzt, um Befunde zu identifizieren. Der Vorteil ist, dass Erfahrungen als heuristische Kritik kodiert und einfach wieder verwendet werden können. Damit ist ein gutes Mittel gegeben, das Dokumentieren von Anforderungen in einer Organisation zu verbessern.

Es gibt aber durchaus Grenzen. Nicht alle Erfahrungen können ohne Schwierigkeiten als heuristische Kritik kodiert werden und bis ein heuristisches RE-Werkzeug genauso wertvolle Erfahrungen bereitstellen kann wie ein erfahrener Analyst ist eine lange evolutionäre Wachstumsphase der Erfahrungsbasis nötig.

Statt dessen ist es in vielen Fällen sinnvoller, erfahrene Analysten dabei zu unterstützen, ihren individuellen Erfahrungsschatz effektiv einzusetzen.

Dazu muss der Analyst im richtigen Moment die richtigen Informationen bekommen. Die Fähigkeit schnell eine andere Perspektive auf eine Menge dokumentierter Anforderungen einnehmen zu können, ist eine Lösung dazu.

Organisatorisches Lernen

Ableiten von Perspektiven geschieht durch heuristische Regeln. Welche Perspektiven man benötigt, erfährt man von erfahrenen Analysten. Die Kodierung dieser Erfahrungen in perspektivwechselnde Werkzeuge ist wesenlich aufwendiger als bei heuristischen RE-Werkzeugen, hilft aber dennoch, das Wissen einer lernenden Software erstellenden Organisation anzuwenden.

Anleitung zur Erstellung eines perspektivwechselnden Werkzeugs. Nach folgendem Vorgehen kann eine LSO Werkzeuge für den Perspektivwechsel erstellen, um die Anwendung von Erfahrungen zu erleichtern:

1. Identifikation zweier relevanter Perpektiven, in der Regel auf Basis einer Erfahrung.
2. Bezug zwischen beiden Perpektiven herstellen. Dabei ist auf geeignete Abstraktion zu achten.
3. Entscheidung: Gibt es eine Primär-Perspektive oder müssen beide Perspektiven weiter bearbeitet und ineinander überführt werden können?
4. Entscheidung: Pro-Aktivität oder nicht? Bei einem pro-aktiven Werkzeug sollte die abgeleitete Perspektive sehr einfach gehalten werden, damit der Nutzer von seiner eigentlichen Aktivität nicht zu sehr abgelenkt wird.
5. Entscheidung: Autoritätsgrad. Welche Qualität muss die abgeleitete Perspektive haben? Dient sie nur dem Überblick oder wird sie Bestandteil der Anforderungsdokumentation?

Der folgende Abschnitt liefert Beispiele für dieses Vorgehen und illustriert einige Werkzeuge für den Perspektivwechsel.

5.3.2 Beispiele

Ausgangspunkt für die Erstellung eines perspektivwechelnden Werkzeugs sei folgende Erfahrung (nach Schneider [150]).

Beispiel 26: *Erfahrung: Zusammenhang zwischen Bedienschnittstelle und Funktion*

Beobachtung: Stakeholder tun sich schwer damit, sich abstrakte Funktionen in Use-Cases vorzustellen. Bei Oberflächenprototypen können sie viel besseres Feedback geben.

Emotion: Gut: Das kann man nutzen!

Hypothese: Use-Cases sollten mit Hilfe von Oberflächen-Prototypen bei späteren Nutzern validiert werden. Besser noch: schon im Interview sollte man Oberflächenprototypen parallel zu Use-Cases erstellen und diskutieren.

Aus dieser Erfahrung kann nun ein erfahrungsbasiertes Werkzeug für den Perspektivwechsel abgeleitet werden. Das Werkzeuge *FastFeedback* erlaubt es auf einem Tablet-PC, parallel zu Use-Cases Oberflächen-Skizzen zu erstellen. Elemente dieser Zeichnung können mit den Schritten im Hauptszenario verknüpft werden. *FastFeedback* erlaubt eine Simulation der Oberfläche: Der spätere Nutzer kann auf Basis der Oberflächen-Skizzen zu jedem Schritt des Use-Cases angeben, wie er mit der Oberfläche interagieren will. *FastFeedback* setzt also Schritte eines Use-Cases in Bezug auf Teile einer Oberflächen-Skizze und erlaubt es, die Bedienung zu simulieren.

FastFeedback soll Änderungen in beiden Perspektiven zulassen. Der Nutzer setzt dazu Elemente beider Perspektiven (Use-Case-Schritte, Bedienelemente) zueinander in Beziehung. Die Simulation der Schritte startet auf expliziten Wunsch des Analysten, FastFeedback ist also nicht pro-aktiv. FastFeedback interpretiert die vorliegenden Daten auch nicht besonders stark, die heuristische Regel postuliert einzig, dass zu den Schritten eines Use-Cases in der Regel eine Interaktion mit der Bedienschnittstelle gehört. Dennoch erlauben diese Werkzeuge vor allem im Interview einen *Breakdown*: Durch die Visualisierung der Zusammenhänge werden Interviewender und Interviewter in die Lage versetzt, Missstände zu erkennen, anzusprechen und damit den Fortlauf des Gesprächs zu unterbrechen. Dies wird durch einen Perspektivwechsel erreicht, wie in Tabelle 5.2 veranschaulicht wird. Die Skizzen werden im weiteren Verlauf der Anforderungsanalyse wiederverwendet. Selbst wenn in der finalen Version der Anforderungsspezifikation eine sauberer gezeichnete Oberflächenskizze enthalten ist, ist die Autorität von FastFeedback hoch einzustufen. Schneider argumentiert, dass sich so große Zeitersparnisse erzielen lassen, indem Use-Cases direkt bei der Erfassung validiert werden können [149].

Neben FastFeedback kann auch HeRA als perspektivwechselndes RE-Werkzeug aufgefasst werden. Dazu sind drei entsprechende Komponenten in HeRA integriert. Diese wurden in Kapitel 3 bereits kurz vorgestellt. Tabelle 5.2 beschreibt diese Ansätze kurz, diese werden im Folgenden in Bezug auf das Lernen durch perspektivwechselnde Werkzeuge gesetzt.

HeRA.UML: Dieser Perspektivwechsel erhöht den Überblick des Analysten [45]. In der tabellarischen Darstellung eines Use-Cases liegt der Fokus vor allem auf den Details. Der Nutzer sieht, welche Stakeholder Interesse an diesem Use-Case haben und welche anderen Use-Cases notwendig sind, um das Benutzerziel des aktuellen Use-Cases zu erreichen. Durch den werkzeuggestützten Perspektivwechsel

Tabelle 5.2: Perspektivwechsel durch abgeleitete Sichten.

Ansatz	Persp. A	Fokus A	Persp. B	Fokus B
FastFeedback	Use-Case	Fokus auf funktionale Anforderungen	Oberflächen-Prototyp	Fokus auf Verwendung des Systems
HeRA.UML	Use-Case	Fokus auf Arbeitsschritte zum erreichen eines Benutzerziels	UML Use-Case-Diagramm	Fokus auf Benutzer- und Geschäftsziele in einem Teilbereich
HeRA.EPK	Use-Case	Fokus auf funktionale Anforderungen	Ereignisgesteuerte Prozesskette	Fokus auf Geschäftsprozess und Zusammenspiel der Use-Cases.
HeRA. UCPoints	Use-Case	Fokus auf funktionale Anforderungen	Aufwandschätzung (Use-Case-Points)	Komplexität und Ausgewogenheit des zu erstellenden Systems

wird es möglich, den aktuell bearbeiteten Use-Case im Kontext relevanter anderer Use-Cases zu sehen. Dies hilft dabei, verschiedene Use-Cases gut gegeneinander abzugrenzen. Zudem unterstützt es den Autoren dabei, die Use-Cases Top-Down zu erstellen und zunächst aus der Perspektive des Use-Case-Diagramms die notwendigen Use-Cases eines Teilsystems zu identifizieren, bevor diese in der tabellarischen Perspektive ausgearbeitet werden. Dieses Vorgehen schlägt unter anderem Birk vor [28], vergleiche auch [127, 128].

HeRA.EPK: Dieser Perspektivwechsel stellt die in einem Use-Case beschriebenen Ziele eines Nutzers in den Zusammenhang des zu unterstützenden Geschäftsprozesses. Aus einer Menge von Use-Cases kann automatisch eine Ereignisgesteuerte Prozesskette abgeleitet werden [101]. Diese zeigt den Prozess, der durch die von den späteren Benutzern erhobenen Anforderungen impliziert wird. Aus Prozesssicht fallen in der Regel aber noch ganz neue Anforderungen und Rahmenbedingungen auf. Auch können Lücken identifiziert werden, die durch zusätzliche Use-Cases gefüllt werden sollten (vergleiche Evaluation in Abschnitt 8.2.1).

HeRA.UCPoints: Die Use-Case-Point-Methode erlaubt eine Aufwandschätzung auf Basis der Use-Cases im Stile der Function-Point-Methode. Dadurch kann der

Analyst die Use-Cases aus Sicht eines Projektleiters betrachten. Interessant ist hier wieder der Vergleich mehrerer Use-Cases. Idealerweise sind diese vom Aufwand ausgeglichen. Gibt es hingegen große Abweichungen, sind die Use-Cases vermutlich schlecht zugeschnitten. Dies macht es in der Planung später schwer die Aufgaben aufzuteilen.

5.3.3 Nutzen von perspektivwechselnden Werkzeugen

Persektivwechselnde Werkzeuge unterstützen das Dokumentieren von Anforderungen auf verschiedene Weisen:

1. *Erfahrungsaustausch.* Durch die unterschiedlichen Perspektiven können die verschiedenen Rollen bei der Anforderungsanalyse zusammengebracht werden. So gelingt es, die jeweiligen Erfahrungen dieser Rollen (zum Beispiel Kunde, Analyst, Nutzer, Geschäftsprozessanalyst) mit einzubringen. Dadurch lernen alle Beteiligten. Diese Form des Lernens wird auch *Sozialisation* genannt (vergleiche Nonaka und Takeuchi [121]).
2. *Rollengerechte Aufbereitung von Informationen.* Der Austausch ist umso ergiebiger, je unterschiedlicher der Erfahrungshintergrund ist. Allerdings besteht dann die Gefahr, dass ein gemeinsames Verständnis fehlt. Perspektivwechselnde Werkzeuge erlauben es daher, die Informationen für die verschiedenen Rollen geeignet aufzubereiten.
3. *Plausibilitätsprüfung.* Es reicht bereits aus, wenn eine Person eine Tätigkeit aus der Perspektive verschiedener Rollen betrachtet (vergleiche Arbeiten zu perspektivbasierten Inspektionen, z.B. Basili et al. [15]). Dadurch gelingt es teilweise, die Betriebsblindheit zu überwinden.

Die Fähigkeit, benötigte Perspektiven automatisch ableiten zu können, beschleunigt zunächst einmal die Arbeit mit Anforderungen. Dazu sind die Anforderungen an den Ableitungsmechanismus jedoch sehr hoch.

Für die *was-wenn* Analyse ist eine solch qualitativ hochwertige abgeleitete Perspektive nicht notwendig [64]. Es reicht, dem Nutzer ein ungefähres Bild davon zu geben, wie sich das gerade bearbeitete Artefakt in der anderen Perspektive darstellt. Vielmehr ist ein guter Abstraktionsmechanismus wichtig, der Einzelheiten versteckt und relevante Eigenschaften der Anforderungen hervorhebt.

Die Möglichkeit, Anforderungen sehr schnell aus einer anderen Perspektive betrachten zu können, versetzt den Analysten in die Lage, seine eigenen Erfahrungen optimal einzusetzen. Je nachdem, ob ein pro-aktives perspektivwechselndes Werkzeug vorliegt oder nicht, unterscheidet sich das Vorgehen aus Sicht des Lernmodells.

Ein pro-aktives perspektivwechselndes Werkzeug stellt die abgeleitete Sicht selbstständig zur Verfügung. Fällt dem Nutzer dabei eine Ungereimtheit auf, so hat das Werkzeug einen *Breakdown* herbeigeführt. Der Nutzer gerät in die Reflexion und überdenkt womöglich sein Vorgehen.

Ein nicht pro-aktives Werkzeug wird vom Nutzer routinemäßig verwendet, um die Plausibilität einer Modellierungsentscheidung oder Anforderungsinterpretation zu prüfen. Entweder geschieht dies an bestimmten Stellen im Vorgehen oder wann immer aus Sicht des Nutzers Klärungsbedarf besteht. Das Werkzeug erlaubt es seinem Nutzer, die fraglichen Anforderungen schnell aus unterschiedlichen Perspektiven zu betrachten und so zu überprüfen, ob alles in Ordnung ist.

5.4 Zusammenfassung: Lernmodell

In diesem Kapitel wurde der Ansatz erläutert, mit dem die Dokumentation von Anforderungen verbessert werden soll: Erfahrungen und Heuristiken. Erfahrung hilft dabei, Aufgaben effektiver und effizienter zu erfüllen, zum Beispiel indem Fehler vermieden werden. Im Sinne von Kapitel 4 führt dies dazu, dass bessere Dokumentation mit weniger Aufwand erstellt werden kann.

In Abschnitt 5.2 wurde der Zusammenhang zwischen einer Erfahrung und ihrer Kodierung als heuristische Kritik hergestellt. Kern dabei ist, auf Basis der Hypothese oder Schlussfolgerung einer Erfahrung eine heuristische Regel zu erstellen. Mit dieser heuristischen Regel kann automatisch ermittelt werden, ob die Erfahrung anwendbar ist. Charakteristisch für eine Heuristik ist, dass die damit berechnete Lösung weder brauchbar noch optimal sein muss. Es kann also weder davon ausgegangen werden, dass die heuristische Regel immer feuert, wenn die dazugehörige Erfahrung anwendbar ist (*Recall*), noch darf angenommen werden, dass die dazugehörige Erfahrung immer anwendbar ist, wenn die heuristische Regel feuert (*Precision*). Ob eine Heuristik dennoch nützlich ist, hängt nicht nur von ihrer Güte ab, sondern auch davon, wie sie in einem erfahrungsbasiertem Werkzeug eingesetzt wird. Heuristische Kritiken sind ein mächtiges Mittel zur Kodierung von Erfahrungen, welches sowohl individuelles Lernen, als auch organisatorisches Lernen unterstützt. In Abschnitt 5.2.1 wurde modelliert, wie Lernen auf Basis heuristischer Kritiken funktioniert. Daraus wurden wichtige Eigenschaften für heuristischen RE-Werkzeuge abgeleitet. Diese sind mehr oder weniger *pro-aktiv*, *interpretierend*, *verlässlich*, *autoritär* und *lernfähig*.

Die Kodierung von Erfahrungen als heuristische Kritik hat einige Vorteile für eine LSO. Eine heuristische Kritik ist konkret, da sie sich auf durch eine heuristische Regel identifizierbare Situationen bezieht. Es ist leicht festzustellen, ob sie anwendbar ist. Durch den stark schematischen Aufbau ist eine heuristische Kritik leicht zu verwalten. Abschnitt 5.2.3 gibt ein Modell des Lebenszyklus von Erfah-

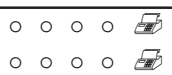
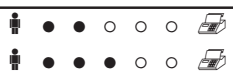
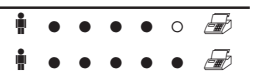
rungen und heuristischen Kritiken bei der Dokumentation von Anforderungen auf Basis des PDCA-Zyklus nach Deming [52].

Neben dem Feedback durch heuristische Kritiken wurde ein zweiter Mechanismus vorgestellt, bei dem das Feedback auf abgeleiteten Modellen basiert. Durch den Perspektivwechsel wird es erleichtert, Erfahrungen aus einem anderen Bereich auf den aktuellen Kontext anzuwenden. So erleichtert ein aus Use-Cases abgeleiter Geschäftsprozess die Sicht des Kunden, der ein Geschäftsziel verfolgt und die Sicht des Nutzers, der eine lokale Aufgabe lösen muss, zu verschmelzen. Ein Dialog dieser Rollen kann zu einem Erfahrungsaustausch (Sozialisation) führen. Abschnitt 5.3 führt daher die Fähigkeit *Perspektivwechselnd* als weitere Eigenschaft von erfahrungsbasierten Werkzeugen ein.

Tabelle 5.3 fasst die wesentlichen Eigenschaften erfahrungsbasierter Werkzeuge für das Requirements Engineering zusammen. Die Tabelle gibt darüber hinaus Hinweise, wie die Ausprägung einer Eigenschaft für ein erfahrungsbasiertes Werkzeug charakterisiert werden kann.

Aus der Bewertung aller vier Eigenschaften für ein erfahrungsbasiertes Werkzeug lässt sich ein Profil erstellen. Dieses Profil ist die Grundlage zur Validierung der in diesem Kapitel vorgestellten Konzepte und wird in Kapitel 6 eingeführt. Die letzte Zeile in Tabelle 5.3 zeigt grafisch mit gefüllten und leeren Kreisen, wie die Ausprägungen der Eigenschaften in den Profilen erfahrungsbasierter Werkzeuge dargestellt werden. Mehr leere Kreise zeigen an, dass der Nutzer erfahrungsbasierter Werkzeuge mehr Verantwortung und Aufgaben behält, gefüllte Kreise zeigen, dass das erfahrungsbasierte RE-Werkzeug mehr Verantwortung übernimmt oder stärker eingreift.

Tabelle 5.3: Lernmodell für erfahrungsbasierte RE-Werkzeuge.

	Ausprägung		
Aspekt	gering (0-1)	mittel (2-3)	stark (4-5)
Autorität	Hinweise des Werkzeugs treffen selten zu	Hinweise des Werkzeugs treffen öfter zu als nicht	Hinweise des Werkzeugs treffen fast immer zu und sind verpflichtend
Pro-Aktiv	Werkzeug wird nur auf direkte Aufforderung des Nutzers aktiv	Werkzeug reagiert auf bestimmte Nutzer-Aktionen	Werkzeug bestimmt Zeitpunkt des Feedbacks selbstständig
Interpretierend	Feedback durch Darstellung vorhandener Daten, Nutzer bewertet	Werkzeug legt durch Filterung, Sortierung u.ä. eine bestimmte Interpretation nahe	Werkzeug nimmt auf Basis aufbereiteter Daten eine Bewertung vor
Lernfähigkeit	Erweiterung der Erfahrungsbasis eher Aufgabe von Experten	Spezielle Unterstützung der Nutzer bei der Erfassung von Erfahrungen	Dynamisches Anpassen der heuristischen Kritiken nach Beobachtung des Nutzerverhaltens
Perspektivwechselnd	Werkzeug stellt keine weitere Perspektive bereit (0) oder höchstens ein abstrakteres Inhaltsverzeichnis (1)	Werkzeug kann eine zweite Perspektive auf ein gegebenes Primär-Artefakt ableiten	Werkzeug kann zwei Perspektiven miteinander verknüpfen und Änderungen einer Perspektive in die jeweils andere überführen
Darstellung im Profil			

6 Problemstellung: Bessere Anforderungsdokumentation lernen

In den vorherigen Kapiteln wurden die Grundlagen beschrieben (Kapitel 2), ein Beispiel für ein erfahrungsbasiertes RE-Werkzeug gegeben (Kapitel 3) und die Konzepte beschrieben, mit denen Requirements Engineering und Erfahrungsmanagement im Rahmen dieser Arbeit zusammengebracht werden sollen (Kapitel 4 und 5). Der konzeptionelle Beitrag dieser Arbeit, also die beiden Forschungsfelder zusammenzubringen, ist damit erfolgt und wird in Abschnitt 6.1 zusammengefasst.

Auf Grundlage dieser Zusammenfassung wird dann in Abschnitt 6.2 die Zielsetzung für den zweiten großen Beitrag dieser Dissertation beschrieben: Der systematischen Evaluation des Potenzials von Erfahrungen und Heuristiken zur Verbesserung der Dokumentation von Anforderungen.

6.1 Aufbau und Profil erfahrungsbasierter RE-Werkzeuge

Zur besseren Übersicht werden zunächst die statischen Modelle zu Anforderungsdokumentation (Kapitel 4), Erfahrungen und Heuristiken (Kapitel 5) in Abschnitt 6.1.1 zusammengeführt. In Abschnitt 6.1.2 werden prinzipielle Möglichkeiten der physischen Verteilung dargestellt. Dadurch ergeben sich konzeptionell sehr unterschiedliche heuristische RE-Werkzeuge in Bezug auf die Verwaltung von Erfahrungen, den Bezug zur Dokumentation von Anforderungen und den Einfluss auf den Dokumentationsprozess.

6.1.1 Statisches Modell

Abbildung 6.1 fasst in einem Klassendiagramm die Zusammenhänge zwischen Anforderungsdokumenten und erfahrungsbasierten RE-Werkzeugen zusammen. Aus Kapitel 4 stammt das Modell für Anforderungsdokumente. Anforderungsdokumente beinhalten verschiedene Aspekte, die jeweils durch textuelle oder modellbasierte Beschreibungen dokumentiert sind. Diese Beschreibungen unterscheiden sich unter anderem durch ihre Formalität und Reife. Beide Merkmale sind wesentlich für die Entscheidung, ob heuristische Verfahren zur Verbesserung der Anforderungsdokumentation sinnvoll eingesetzt werden können - heuristische Verfahren

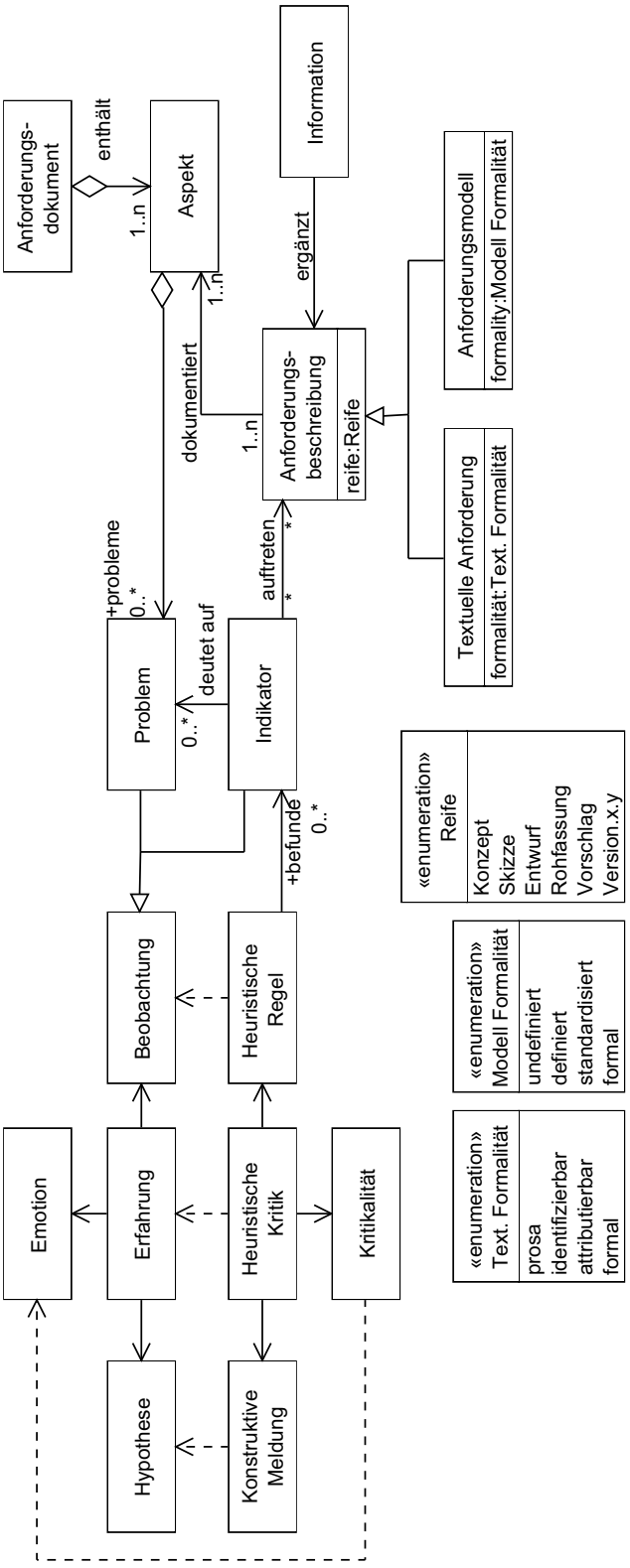


Abbildung 6.1: Zusammenhang zwischen Anforderungsdokumenten, heuristischen Kritiken und Erfahrungen.

sind schon bei geringerer Reife und Formalität hilfreich, als formale Verifikationsverfahren.

Dieser Umstand wird in Abbildung 6.1 durch das Konzept *Indikator* modelliert. Ein Indikator kann auf einer Anforderungsbeschreibung (unter anderem von einer heuristischen Regel) ausgewertet werden und kann auf Probleme hindeuten.

Sowohl Probleme als auch Indikatoren können als Beobachtung aufgefasst werden. Eine solche Beobachtung kann zum Beispiel im Rahmen einer Erfahrung zusammen mit einer Emotion gemacht werden. Die Hypothese dieser Erfahrung stellt dann den Zusammenhang zwischen einem Indikator und einem Problem her. In Kapitel 5 wurde beschrieben, wie heuristische Regeln aus einer solchen Hypothese abgeleitet werden können, um zur Beobachtung ähnliche Situationen automatisch zu identifizieren. Heuristische Regeln dienen dazu, Indikatoren automatisch auszuwerten und helfen, als Bestandteil einer heurisitschen Kritik, bei der erfahrungsbasierten Verbesserung von Anforderungen.

Heuristische und perspektivwechselnde Werkzeuge sind in dem Klassendiagramm nicht abgebildet. Diese Werkzeuge nutzen heuristische Regeln, wie in Kapitel 5 beschrieben, um organisatorisches Lernen zu unterstützen und dem Nutzer zu besserer Anforderungsdokumentation zu verhelfen. Prinzipiell ist dafür eine Möglichkeit nötig, Anforderungsbeschreibungen zu laden, durch heuristische Regeln zu analysieren und dem Nutzer das Analyseergebnis zu präsentieren. Der folgende Abschnitt diskutiert verschiedene mögliche Ausprägungen dieser Kernfunktionen.

6.1.2 Verteilungsmodell

Die physikalische Verteilung von Komponenten zeigt wichtige Unterscheidungsmerkmale erfahrungsbasierter RE-Werkzeuge auf. Die Frage, auf welchen Geräten und in welchem physikalischen Kontext die Komponenten eines erfahrungsbasierten Werkzeugs ausgeführt werden hängt eng mit den vorgesehenen Eigenschaften des Werkzeugs und mit dem geplanten Einsatzszenario zusammen. Dieser Abschnitt erläutert, welchen Einfluss diese Überlegungen auf die Architektur von erfahrungsbasierten RE-Werkzeugen haben.

Prinzipiell lassen sich die folgenden Komponenten voneinander abgrenzen: Die Verwaltung von Erfahrungen, die Analyse von Dokumenten, um andere Sichten abzuleiten oder heuristisches Feedback zu geben, und die Dokumentation von Anforderungen. Die folgenden Abschnitte erläutern den Einfluss dieser Komponenten auf die physikalische Verteilung.

Verwaltung von Erfahrungen

Kern eines erfahrungsbasierten Werkzeugs ist eine Erfahrungsbasis, die als heuristische Kritiken kodierte Erfahrungen enthält. Die Art dieser Erfahrungen kann sehr unterschiedlich sein. Unterstützt die heuristische Regel vor allem das Ableiten anderer Sichten aus einem Anforderungsdokument, so ist sie vermutlich recht komplex. Änderungen kommen eher selten vor, so dass die Regel direkt in Werkzeuge integriert werden kann, die auf einer lokalen Arbeitsstation laufen. Bei Änderungen an der Regel müssen dann natürlich alle Arbeitsstationen aktualisiert werden.

Dienen heuristische Kritiken andersherum zu einem regen Austausch von Erfahrungen, so müssen sie zwangsläufig unabhängig von der Installation auf einer Arbeitsstation verwaltet werden. Eine projektübergreifende Erfahrungsbasis müsste dann von allen Clienten erreichbar sein. Je nach Einsatzgebiet ist das nicht immer möglich, so dass sich Clienten erst bei der nächsten Verbindung mit der Erfahrungsbasis synchronisieren können.

Trennung von Projektaktivität und Erfahrungsnutzung

Die in Kapitel 5 vorgestellten Konzepte erlauben eine Reihe von sehr unterschiedlichen erfahrungsbasierten Werkzeugen. Werden heuristische Werkzeuge analytisch eingesetzt, so ist es prinzipiell möglich, sie vollständig von der Aktivität *Dokumentation von Anforderungen* zu trennen. Das Ergebnisdokument wird dann analysiert. Dies erlaubt es, die Ergebnisse der heuristischen Prüfung zentral zu filtern und zu priorisieren, bevor sie in den Verbesserungsprozess eingehen. Ein Vorteil ist, dass die Rollentrennung zwischen Qualitätsmanagern und Autoren aufrechterhalten werden kann und Anforderungsdokumente nicht von ihrem Autor überprüft werden, der bei seinem eigenen Erzeugnis womöglich betriebsblind ist. Die heuristische Prüfung sollte dementsprechend im Rahmen des Qualitätsmanagement stattfinden. Sie kann entweder auf einem zentralen Server angeboten werden oder in spezielle Werkzeuge für das Qualitätsmanagement eingebunden sein.

Werden hingegen erfahrungsbasierte Werkzeuge konstruktiv eingesetzt, so ist das direkte Feedback an den Autor wichtig. Eine möglichst enge Verzahnung von Dokumentation und heuristischem Feedback kann durch Integration beider Aktivitäten in ein Werkzeug erreicht werden. Die heuristische Analyse und das Ableiten von Modellen muss in diesem Fall auf dem gleichen physikalischen Gerät wie die Dokumentations-Aktivität geschehen.

Verfügbarkeit und Änderbarkeit von Anforderungsdokumenten

Je nach Einsatzszenario werden Anforderungen von einem Verantwortlichen oder von einer Gruppe von Personen editiert. Je höher die Änderungshäufigkeit, desto größer ist der Bedarf, Anforderungen durch eine ganze Gruppe von Personen anpassen zu können. In diesem Fall kann es sinnvoll sein, die Anforderungen an zentraler Stelle zu editieren, z.B. mit Hilfe eines Wikis (siehe auch [34, 96]).

Muss andererseits ein formaler Änderungsmanagementprozess eingehalten werden, so wird man die Berechtigung zum Ändern von Anforderungen zentral verwalten wollen. In diesem Fall empfiehlt sich eine zentrale Verwaltung von Änderungswünschen und schreibgeschützten Anforderungen, und eine Client-seitige Lösung zur Änderung von Anforderungen mit ausgeprägtem Rechtesystem.

6.2 Zielsetzung und Forschungsfragen

In Abschnitt 4.3 wurde gezeigt, dass ein Zusammenhang zwischen messbaren Qualitätseigenschaften und Projekterfolg besteht. Mit heuristischen Kritiken (Abschnitt 5.2) kann eine LSO dies nutzen, indem auf Basis messbarer Eigenschaften von Anforderungsdokumenten Feedback gegeben wird. Es wurde zudem gezeigt, wie sich dieser Mechanismus im Kontext des Lernprozesses einer LSO nutzen lässt. Diese Konzepte zeigen, wie sich das Ziel dieser Arbeit prinzipiell erreichen lässt:

Ziel: Die Dokumentation von Anforderungen auf Basis von Erfahrungen und Heuristiken verbessern.

Die Arbeit untersucht, wie Organisationen auf Basis messbarer Qualitätseigenschaften systematisch lernen können, Anforderungen besser zu dokumentieren. Abbildung 6.2 zeigt Teilziele, mit denen dieses Ziel im Rahmen dieser Arbeit erreicht wird. Ein heuristischer, erfahrungsbasierter Ansatz muss mindestens eines dieser Ziele unterstützen. Diese Arbeit liefert Evaluationsbeispiele für alle drei Teilziele.

Die Teilziele 1 und 2 in Abbildung 6.2 betreffen das Erfahrungsmanagement einer Organisation. Wird das Erfahrungsmanagement verbessert, können Erfahrungen besser genutzt werden, um die Dokumentation zu verbessern. Die Erfahrungen sollen im Rahmen dieser Arbeit als heuristische Kritiken kodiert werden. Neben der besseren Unterstützung des Erfahrungsmanagements dienen diese heuristischen Kritiken auch direkt der Verbesserung der Dokumentation von Anforderungen. Dieser Aspekt ist durch Teilziel 3 abgedeckt.

Die drei Teilziele werden nun in Anlehnung an die GQM-Methode weiter heruntergebrochen. Dies erlaubt eine Evaluation der Konzepte in Kapitel 8. Im Vergleich zu klassischem Vorgehen nach der GQM-Methode sind die Ziele auf höherer

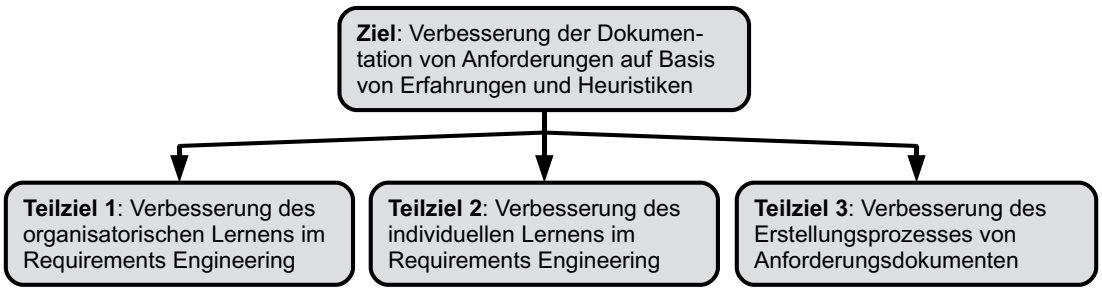


Abbildung 6.2: Der Zielbaum zerlegt das Hauptziel dieser Arbeit in Teilziele: individuelles und organisatorisches Lernen sowie die Verbesserung des Erstellungsprozesses von Anforderungsdokumenten.

Abstraktionsebene. Erst in Kapitel 8 werden konkrete Ziele in den Evaluationsszenarien abgeleitet. Die Mechanismen der GQM-Methode helfen jedoch, die Ziele dieser Arbeit genau zu definieren.

Tabelle 6.1 zeigt die Zielfacetten der Teilziele. Dabei werden die Qualitätsaspekte des organisatorischen und individuellen Lernens aus dem Lernmodell (vgl. Abbildung 5.2) und die Qualitätsaspekte der Verbesserung des Erstellungsprozesses aus der Definition in Abschnitt 4.1 abgeleitet. Dabei liegt der Fokus auf den technischen Aspekten der Konzepte. Der Einfluss sozialer Aspekte, wie zum Beispiel durch das Nutzerprofil in Abschnitt 4.1, werden hingegen nicht evaluiert, sondern als fix angenommen: Jede Beispielimplementierung zur Evaluation in Kapitel 8 ist auf die Nutzertypen ausgerichtet, von denen sie eingesetzt wird.

Tabelle 6.1: Zielfacetten dieser Arbeit.

Teilz.	**Zweck**	**Qualitätsaspekt**	**Betrachtungsgegenstand**	**Perspektive**
1	Verbessere	*Kodierung* und *Wiederverwendung* von Erfahrungen	organisatorisches Lernen	Erfahrungsmanager
2	Verbessere	*Reflexion* und *Anwendung* von Erfahrungen	individuelles Lernen	Analyst
3	Verbessere	*Erstellungskosten* und *Qualität*	Anforderungsdokumentation	Projektintern (Projektleiter, Qualitätsbeauftragter)

Eine Evaluation des Teilziels 2 zum individuellen Lernen ist schwierig, da sich das durch Breakdown, Reflexion und Anwendung entstandene Wissen nur schwer beobachten oder messen lässt. Zu beobachten sind vielmehr die Auswirkungen des individuellen Lernprozesses: Helfen erfahrungsbasierte RE-Werkzeuge Individuen, bessere Anforderungsdokumente zu erstellen? Bei der konkreten Evaluation ist also der Zusammenhang der Teilziele 2 und 3 besonders interessant. Werden mit Hilfe erfahrungsbasierter RE-Werkzeuge bessere Anforderungsdokumente erstellt als mit nicht erfahrungsbasierten RE-Werkzeugen (Teilziel 3), so kann je nach Kontext der Evaluation indirekt auf bessere Reflexion und Anwendung individueller Erfahrung geschlossen werden (Teilziel 2).

In den folgenden Abschnitten werden die Ziele mit Hilfe von Abstraction Sheets aus der GQM-Methode [33, 167] verfeinert. Aus den Abstraction Sheets lassen sich die Fragen aus der Kombination der Qualitätsaspekte und der Einflussfaktoren ableiten. Nicht alle Kombinationen sind zur Erreichung des GQM-Ziels relevant, die wichtigen Kombinationen werden in den Einfluss- und Ausgangsshypothesen diskutiert.

In Abschnitt 6.2.4 folgt eine Diskussion der Informations- und Erfahrungsflüsse rund um erfahrungsbasierte RE-Werkzeuge. In dieses Informationsflussmodell werden die Forschungsfragen aus der Problemstellung in Abschnitt 6.2 eingeordnet.

6.2.1 Teilziel 1 – Lernen in der Organisation

Ein wesentlicher Bestandteil der Verbesserung von Anforderungsdokumentation auf Basis von Erfahrungen ist, dass neue Erfahrungen entstehen und alte Erfahrungen verbessert werden. Je besser der Erfahrungsschatz einer Organisation, um so besser kann die Dokummentation auf Basis von diesen Erfahrungen verbessert werden. Tabelle 6.2 zeigt das GQM Abstraction Sheet zu diesem Ziel.

Für dieses Teilziel sind Lernfähigkeit und Autorität erfahrungsbasierter RE-Werkzeuge die Schlüssel-Eigenschaften. Eine besondere Rolle spielt die Akzeptanz der Nutzer, die eine Grundvoraussetzung ist, um die Konzepte dieser Arbeit zu untersuchen. Es stellen sich also die folgenden Fragen:

Q 1.1. Akzeptieren die Nutzer das erfahrungsbasierte Werkzeug und schätzen sie es als nützlich ein (abgeleitet von EH 1.1)?

Q 1.2. Sind die Nutzer in der Lage, Erfahrungen (heur. Kritiken) zu bewerten (abgeleitet aus AH 1.3 und EH 1.2)?

Q 1.3. Sind die Nutzer in der Lage, Erfahrungen (heur. Kritiken) zu ändern oder neu anzulegen (abgeleitet aus AH 1.3 und EH 1.3)?

Tabelle 6.2: Abstraction Sheet für das Teilziel 1 “organisatorisches Lernen”.

Zweck	**Qualitätsaspekt**	**Betrachtungs-gegenstand**	**Perspektive**
Verbessere	Kodierung und Wieder-verwendung von Erfahrungen	organisatorisches Lernen	Erfahrungs-manager

Qualitätsaspekte	**Einflussfaktoren**
QZ 1.1. Menge bewerteter, überarbeiteter, neuerstellter Erfahrungen (Kodierung)	**EF 1.1.** Lernfähigkeit des erfahrungsbasierten Werkzeugs
QZ 1.2. Menge wieder verwendeter Erfahrungen	**EF 1.2.** Autorität des erfahrungsbasierten Werkzeugs
	EF 1.3. Nutzer schätzt erfahrungsbasiertes Werkzeug als nützlich ein
Ausgangshypothesen	**Einflusshypothesen**
AH 1.1. Nutzer bewerten eher schlechte / störende Erfahrungen	**EH 1.1.** Wenn das erfahrungsbasierte Werkzeug als nützlich empfunden wird, sind Nutzer eher bereit, Erfahrungen zu bewerten, zu überarbeiten oder neu anzulegen, aber auch sie zu nutzen
AH 1.2. Nutzer passen Erfahrungen selten an, selbstständig lernende erfahrungsbasierte Werkzeuge können aus der Nutzerbeobachtung lernen	**EH 1.2.** Nutzer sind eher bereit, Erfahrungen zu bewerten, anzupassen oder neue Erfahrungen anzulegen, wenn dies sehr einfach ist
AH 1.3. Nutzer sind in der Lage, neue Erfahrungen anzulegen und tun dies in Sondersituationen	**EH 1.3.** Mittlere Lernfähigkeit (Assistenten und Feedback-Möglichkeiten) erlauben es, neue Erfahrungen und Erfahrungsänderungen zu kodieren
AH 1.4. Nutzer wenden Erfahrungen in der Regel an	**EH 1.4.** Starke Lernfähigkeit (Beobachtung des Nutzerverhaltens) erlaubt das Ableiten von Bewertungen und Verbesserungen von Erfahrungen
	EH 1.5. Hohe Autorität des erfahrungsbasierten Werkzeugs führt dazu, dass mehr Erfahrungen wiederverwendet werden

Q 1.4. Kann das erfahrungsbasierte RE-Werkzeug aus der Beobachtung von Nutzerverhalten Erfahrung ableiten (abgeleitet aus AH 1.2 und EH 1.4)?

Q 1.5. Lernt die Organisation durch den Einsatz erfahrungsbasierter Werkzeuge mehr neue Erfahrungen als ohne (abgeleitet aus AH 1.1-3 und EH 1.1-4)?

Q 1.6. Werden durch den Einsatz erfahrungsbasierter RE-Werkzeuge mehr Erfahrungen aus der Erfahrungsbasis der Organisation angewandt als ohne (abgeleitet aus AH 1.4 und EH 1.5)?

Bei der Evaluation werden konkrete erfahrungsbasierte Werkzeuge in Evaluationsszenarien eingesetzt. Dabei werden zu den Fragen je nach konkreter Situation geeignete Metriken ausgewählt.

6.2.2 Teilziel 2 – Individuelles Lernen

Der zweite wichtige Teil des Erfahrungsmanagements sind die Erfahrungen der Individuen der Organisation. Nach dem Lernmodell dieser Arbeit (vgl. Abbildung 5.2) sind hier Reflexion und Anwendung von Erfahrungen zu untersuchen. Tabelle 6.3 zeigt, wie dieses Ziel mit den Eigenschaften erfahrungsbasierter Werkzeuge zusammenhängt. Dabei wird die Perspektive des Analysten eingenommen, da dieser während der Dokumentation von Anforderungen unterstützt werden soll.

Zu zeigen ist also, ob Nutzer den Perspektivwechsel, den erfahrungsbasierte RE-Werkzeuge durch abgeleitete Modelle erlauben, nützlich finden. Zudem muss untersucht werden, ob Breakdowns bei Nutzern verursacht werden können und beim Lernen helfen. Zu beachten ist, dass je nach Typ des Nutzers und Autorität des Werkzeugs die gewünschten Effekte ausbleiben können. Für die Evaluation ergeben sich dadurch die folgenden Fragen:

Q 2.1. Wird die Menge der neuen Erfahrungen durch den Einsatz eines proaktiven erfahrungsbasierten RE-Werkzeugs beim Analysten erhöht (abgeleitet aus AH 2.1 / EH 2.1)?

Q 2.2. Wird die Menge der neuen Erfahrungen durch den Einsatz eines interpretierenden erfahrungsbasierten RE-Werkzeugs erhöht (abgeleitet aus AH 2.1 / EH 2.2)?

Q 2.3. Erlaubt es die Fähigkeit eines erfahrungsbasierten RE-Werkzeugs zum Perspektivwechsel dem Nutzer mehr Erfahrung anzuwenden? (abgeleitet aus AH 2.2 / EH 2.3)?

Q 2.4. Nimmt der Nutzer je nach Autorität des erfahrungsbasierten Werkzeugs, seiner eigenen Erfahrung und seines Typs kritisches Feedback des Werkzeugs überhaupt an (abgeleitet aus AH 2.2 / EH 2.4)?

Tabelle 6.3: Abstraction Sheet für das Teilziel 2 “individuelles Lernen”.

Zweck	**Qualitätsaspekt**	**Betrachtungsgegenstand**	**Perspektive**
Verbessere	Reflexion und Anwendung von Erfahrungen	individuelles Lernen	Analyst

Qualitätsaspekte	**Einflussfaktoren**
QZ 2.1. Menge der neuen Erfahrungen durch Reflexion	**EF 2.1.** Pro-Aktivität des erfahrungsbasierten Werkzeugs
QZ 2.2. Menge der angewandten Erfahrungen	**EF 2.2.** Interpretation des erfahrungsbasierten Werkzeugs
	EF 2.3. Erfahrungsbasiertes Werkzeug erlaubt einen Perspektivwechsel
	EF 2.4. Autorität des erfahrungsbasierten Werkzeugs sowie Erfahrung und Typ des Nutzers
Ausgangshypothesen	**Einflusshypothesen**
AH 2.1. Nutzer lernen durch erfahrungsbasiertes Werkzeug Anforderungen besser zu dokumentieren	**EH 2.1.** Pro-Aktivität kann zu einem *Breakdown* während der Dokumentation von Anforderungen führen und so die Reflexion auslösen
AH 2.2. Nutzer erstellen mit Hilfe eines erfahrungsbasierten Werkzeugs bessere Anforderungsdokumente	**EH 2.2.** Interpretation durch das erfahrungsbasierte Werkzeug hilft dem Nutzer bei der Einschätzung, ob eine Aktion gut oder schlecht war – Grundvoraussetzung, um zu lernen.
	EH 2.3. Der Perspektivwechsel erlaubt es dem Nutzer, die Auswirkungen seiner Aktionen zu verstehen und seine Erfahrung anzuwenden
	EH 2.4. Mangelnde Autorität stört das Vertrauen des Nutzers und schwächt den Lerneffekt ab. Je nach eigener Erfahrung, Routine und persönlichen Typs ist der Nutzer nicht offen für kritisches Feedback

6.2.3 Teilziel 3 – Verbesserung der Anforderungsdokumentation

Schließlich wird im Rahmen dieser Arbeit untersucht, wie genau die Verbesserung von Anforderungsdokumenten auf Basis von Erfahrungen und heuristischem Feedback funktioniert. Das Abstraction Sheet in Tabelle 6.4 gibt dazu Aufschluss.

Zu untersuchen ist also, ob Anforderungsdokumente, die mit Hilfe von erfahrungsbasierten Werkzeugen erzeugt wurden, weniger Fehler enthalten als vergleichbare Dokumente, die ohne diese Unterstützung erstellt wurden. Dabei soll der Overhead durch die Verwendung eines erfahrungsbasierten Werkzeugs geringer sein, als der zu erreichende Nutzen. Die folgenden Fragen ergeben sich zu Teilziel 3 aus dem Abstraction Sheet in Tabelle 6.4:

Q 3.1. Gibt es einen Zusammenhang zwischen messbaren, formalen Qualitätseigenschaften und inhaltlichen Problemen (abgeleitet aus AH/EH 3.1)?

Tabelle 6.4: Abstraction Sheet für das Teilziel 3: "Verbesserung von Dokumentation".

Zweck	**Qualitätsaspekt**	**Betrachtungsgegenstand**	**Perspektive**
Verbessere	Erstellungskosten und Qualität	Anforderungsdokumentation	Projektintern

Qualitätsaspekte	**Einflussfaktoren**
QZ 3.1. Anzahl formaler Fehler	**EF 3.1.** Zusammenhang zwischen a) messbaren Indikatoren, b) formalen Problemen und c) inhaltlichen Problemen
QZ 3.2. Anzahl inhaltlicher Fehler	**EF 3.2.** Autorität des erfahrungsbasierten RE-Werkzeugs
QZ 3.3. Erstellungskosten	**EF 3.3.** Pro-Aktivität des erfahrungsbasierten RE-Werkzeugs
QZ 3.4. Kosten der Qualitätssicherung	
Ausgangshypothesen	**Einflusshypothesen**
AH 3.1. Anzahl formaler und inhaltlicher Probleme korrelieren positiv	**EH 3.1.** Anzahl formaler und inhaltlicher Probleme korrelieren positiv, erfahrungsbasierte RE-Werkzeuge können (relevante) formale Probleme auf Basis messbarer Indikatoren aufzeigen
AH 3.2. Erfahrungsbasierte Werkzeuge helfen, weniger formale Fehler zu machen	**EH 3.2.** Hohe Autorität erfahrungsbasierter RE-Werkzeuge erlaubt es vorzuschreiben, dass identifizierte formale Probleme behoben werden. Dies führt zu weniger Fehlern
AH 3.3. Erfahrungsbasierte Werkzeuge helfen, Missverständnisse frühzeitig zu finden und auszuräumen	**EH 3.3.** Pro-aktive erfahrungsbasierte Werkzeuge erlauben es, konstruktiv Details zu berücksichtigen, deren nachträgliche Verbesserung zu teuer ist
AH 3.4. Erstellungskosten steigen nur marginal durch Verwendung erfahrungsbasierter RE-Werkzeuge	**EH 3.4.** Autorität des erfahrungsbasierten Werkzeugs erlaubt hohen Automatisierungsgrad und senkt so die Kosten der Qualitätssicherung
AH 3.5. Kosten der Qualitätssicherung sinken durch Verwendung erfahrungsbasierter RE-Werkzeuge	

Q 3.2. Führt der Einsatz erfahrungsbasierter RE-Werkzeuge mit hoher Autorität dazu, dass Anforderungsdokumente weniger formale Fehler enthalten (abgeleitet aus AH/EH 3.2)?

Q 3.3. Führt der Einsatz erfahrungsbasierter RE-Werkzeuge mit hoher Pro-Aktivität dazu, dass Anforderungsdokumente weniger Fehler enthalten (abgeleitet aus AH/EH 3.3)?

Q 3.4. Erhöhen sich die Erstellungskosten von Anforderungsdokumenten durch den Einsatz erfahrungsbasierter RE-Werkzeuge (abgeleitet aus AH 3.4)?

Q 3.5. Sinken die Kosten für Qualitätssicherung von Anforderungsdokumenten durch den Einsatz erfahrungsbasierter RE-Werkzeuge (abgeleitet aus AH 3.5 und EH 3.4)?

6.2.4 Erfahrungsflüsse bei der Anforderungsdokumentation

Über die statische Sicht aus Abschnitt 6.1.1 hinaus werden in diesem Abschnitt die dynamischen Zusammenhänge der Verbesserung von Anforderungsdokumentation mit erfahrungsbasierten RE-Werkzeugen dargestellt. Dazu dienen Informationsflussmodelle nach der FLOW-Methode [152].

Erfahrungsbasierte Werkzeuge nach Kapitel 5 können zusammen mit dem Lebenszyklus von Erfahrungen als *Experience Factory* aufgefasst werden (vergleiche Abschnitt 2.2). An Stelle der Organisation eines Projekts tritt das Requirements Engineering. Bei der Zielsetzung für diesen Teilprozess findet die Auswahl erfahrungsbasierter RE-Werkzeuge und der darin zu verwendenden heuristischen Kritiken statt. Heuristische Kritiken dienen als Experience Package und erleichtern die Analyse der Anforderungsdokumentation, wenn erfahrungsbasierte RE-Werkzeuge verwendet werden. Abbildung 6.3 veranschaulicht diese Zusammenhänge.

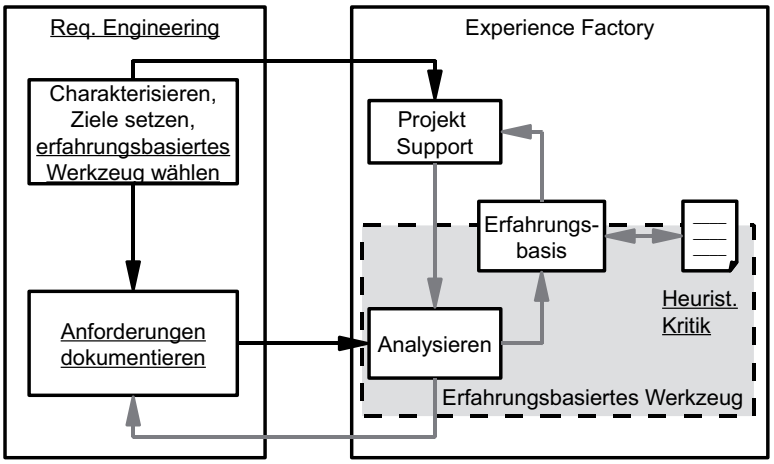


Abbildung 6.3: Rolle von erfahrungsbasierten RE-Werkzeugen (hellgrau hinterlegt) in einer Experience Factory. Änderungen zu Abbildung 2.2 sind unterstrichen. Erfahrungsbasierte RE-Werkzeuge werden in die Erfahrungsbasis mit eingebunden und unterstützen die Analyse, erfordern aber nach wie vor eine gute Einbindung in die Organisation.

Für die Planung des Teilprojekts *Requirements Engineering* ist es erforderlich, Ziele zu setzen. Der *Projekt Support* kann zum Beispiel dabei helfen, angemessene Qualitätsziele für Dokumentation von Anforderungen festzulegen und erfahrungsbasierte RE-Werkzeuge auszuwählen. Mit Hilfe der Projektcharakteristiken kann für diese ein projektspezifisches *Seeding* durchgeführt werden, indem Erfahrungen für die Analyse bereitgestellt werden.

Die Erfahrungen in der Erfahrungsbasis liegen als heuristische Kritiken *verpackt* vor. Die heuristischen Regeln vereinfachen die Analyse von Anforderungsdokumenten. Da diese Aktivität im Rahmen dieser Arbeit besonders wichtig ist, werden die Informationsflüsse bei der durch erfahrungsbasierte RE-Werkzeuge unterstützten Anforderungsdokumentation in Abbildung 6.4 detaillierter dargestellt. Dabei sind zwei prinzipielle Informationsflüsse zu unterscheiden:

1. Flüsse von Projektinformationen: Anforderungen fließen von Anforderungsquellen in Anforderungsdokumente. Dazu werden sie von einem Nutzer mit Hilfe von RE-Werkzeugen (z.B. Texteditoren) dokumentiert. Projektinformationsflüsse sind im Diagramm schwarz gezeichnet.
2. Flüsse von Erfahrungen: Erfahrungen fließen über heuristische Kritiken zum Nutzer, um diesen bei seiner Aufgabe zu unterstützen. Zudem kann der Nutzer seine Erfahrung zurückfließen lassen, zum Beispiel, wenn er in einem abgeleiteten Modell einen Missstand erkennt. Erfahrungsinformationsflüsse sind im Diagramm grau gezeichnet.

Zudem werden die beiden Typen von erfahrungsbasierten RE-Werkzeugen unterschieden: heuristische Werkzeuge, die Feedback mit Hilfe von heuristischen Kritiken geben und perspektivwechselnde Werkzeuge, die Feedback mit Hilfe von abgeleiteten Perspektiven geben. Letztere verwenden ebenfalls heuristische Regeln, um eine neue Perspektive aus dem Anforderungsdokument zu erzeugen. Dieses Feedback ist als Projektinformationsfluss gekennzeichnet, weil es lediglich eine andere Sicht auf die Projektinformationen enthält. Der Nutzer wird dadurch in die Lage versetzt, seine eigene Erfahrung einzusetzen, indem er Auffälligkeiten in der abgeleiteten Sicht analysiert.

Die Evaluationsziele aus Abschnitt 6.2 sind in Abbildung 6.4 den entsprechenden Informationsflüssen zugeordnet:

Teilziel 1. Zu untersuchen ist, ob es prinzipiell möglich ist, dass Erfahrungen vom Nutzer eines erfahrungsbasierten Werkzeugs über die Analyse in die Erfahrungsbasis zurückfließen und welchen Einfluss die Lernfähigkeit von erfahrungsbasierten Werkzeugen darauf hat.

Teilziel 2. Zu untersuchen ist hier, ob Nutzer durch die Verwendung von erfahrungsbasierten Werkzeugen lernen. Dabei ist auch der Einfluss der Autorität, Pro-Aktivität, Interpretationsleistung und der Möglichkeit zum Perspektivwechsel von erfahrungsbasierten Werkzeugen zu beachten.

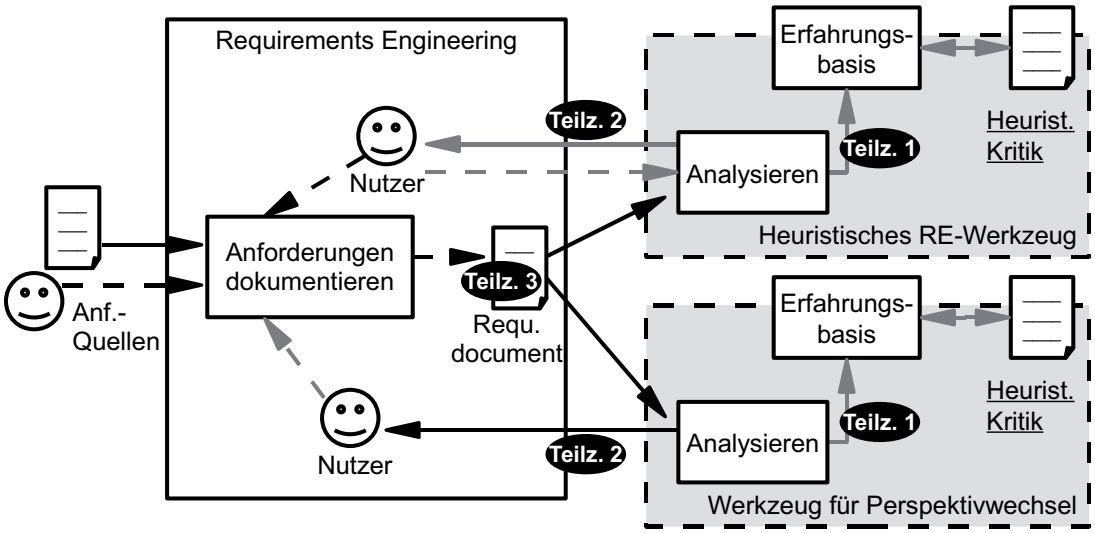


Abbildung 6.4: FLOW Modell der Informationsflüsse erfahrungsbasierter Werkzeuge. Die schwarzen Ovale ordnen die Ziele dieser Arbeit den Informationsflüssen zu, an denen sie evaluiert werden.

Teilziel 3. Zu untersuchen ist, ob Anforderungsdokumente bei Verwendung geeigneter erfahrungsbasierter RE-Werkzeuge tatsächlich qualitativ besser werden, als wenn diese nicht zum Einsatz kommen. Dabei ist der Einfluss der Reife und der Formalität der Anforderungsdokumente zu beachten, sowie der Projektkontext und die persönliche Erfahrung der Nutzer.

Die Informationsflüsse sind also durch die konkreten Eigenschaften der betrachteten Anforderungsdokumente, erfahrungsbasierten Werkzeuge und Nutzertypen charakterisiert. Je nach konkretem Einsatzszenario wird dieses Informationsflussmodell verfeinert, um die entsprechenden Besonderheiten zu reflektieren (vergleiche Abschnitt 2.2.2). Prinzipiell sind die folgenden Informationskreisläufe wichtig:

Erfahrungskreislauf

Der Erfahrungskreislauf ist wichtig, damit eine Verbesserung stattfinden kann. Neue Erfahrungen sollen so in das System einfließen, alte Erfahrungen sollen verbessert werden. Abbildung 6.5 zeigt ausschließlich den Erfahrungskreislauf aus Abbildung 6.4.

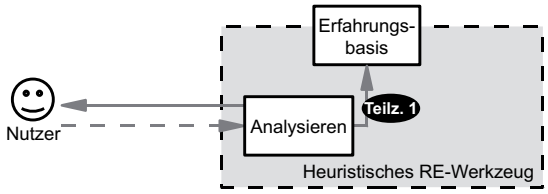


Abbildung 6.5: Der Erfahrungskreislauf erfahrungsbasierter RE-Werkzeuge aus Abbildung 6.4.

Entsprechend der Experience Factory sind Nutzer und Dokumenationsaktivität teil des Requirements Engineering. Die heuristischen Kritiken gehören zur Experience Factory. Über die Analyse fließen Erfahrungen zum Nutzer und vom Nutzer zurück in die Erfahrungsbasis. Wie in allen erfahrungsbasierten Ansätzen ist es wichtig, nicht mit einer leeren Erfahrungsbasis zu starten. Zum *Seeding* [64] können zum Beispiel Hinweise aus der Literatur dienen. Neue Erfahrungen können dann bei der Anwendung alter Erfahrungen gewonnen werden. Sie liegen dann zum Beispiel in Form von Kommentaren zu heuristischem Feedback oder als Beobachtung des Nutzerverhaltens vor. In einem speziellen Arbeitsschritt werden die heuristischen Kritiken auf Basis dieser Erfahrungen erstellt oder verbessert. Das Ergebnis ist eine verbesserte Menge von heuristischen Kritiken, mit deren Hilfe das heuristische Werkzeug die aktuelle Eingabe überprüfen kann.

Asynchrones heuristisches Feedback

Mit Hilfe der heuristischen Kritiken wird Feedback für den Nutzer des erfahrungsbasierten Werkzeugs generiert. Der Nutzer muss nicht unbedingt der Analyst sein. Abbildung 6.6 zeigt eine Verfeinerung der Aktivität *Anforderungen dokumentieren*. Hier werden heuristische Kritiken verwendet, um asynchron Feedback zu geben, beispielsweise um fertige Anforderungsspezifikationen zu analysieren. Die Rolle *Nutzer* wird durch einen Qualitätsbeauftragten übernommen. Das Dokument *Befunde* zeigt genauer, wie das Feedback des Qualitätsbeauftragten beim Dokumentieren von Anforderungen genutzt wird. Zusammen mit dem *Analyst* wird so die Aktivität *Anforderungen dokumentieren* verfeinert.

Das heuristische Analyse-Werkzeug erhält als Eingabe mit externen Programmen erstellte Anforderungen und generiert mit Hilfe der heuristischen Kritiken ein schriftliches Gutachten mit Befunden. Dies ähnelt dem Resultat eines Reviews. Ein Qualitätsbeauftragter sichtet diese Befunde, priorisiert sie und sortiert Fehlmeldungen aus. Die restlichen Befunde werden dann zur Verbesserung dem zuständigen Analysten wieder zugewiesen. Die überarbeitete Fassung des Anforderungsdokuments kann dann erneut in den Verbesserungsprozess einfließen. So

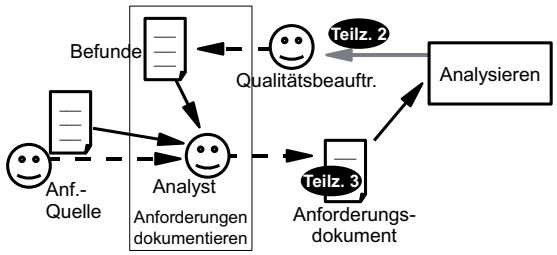


Abbildung 6.6: Asynchrones heuristisches Feedback dient zur analytischen Verbesserung von Zwischenergebnissen der Anforderungsdokumente (Verfeinerung des allgemeinen Flusses in Abbildung 6.4).

werden die heuristischen Kritiken zur analytischen Qualitätssicherung eingesetzt. Das Feedback geschieht asynchron zur Dokumentation der Anforderungen.

Direktes heuristisches Feedback

Im Gegensatz zum asynchronen heuristischen Feeback dient der Informationskreislauf des direkten heuristischen Feedbacks zur Verbesserung der gerade bearbeiteten Anforderungen. Abbildung 6.7 zeigt diesen Informationskreislauf als Alternative zu dem Informationsfluss in Abbildung 6.6. Auch hier liegt nach der FLOW-Methode eine Verfeinerung des allgemeinen Flusses in Abbildung 6.4 vor (vergleiche Abschnitt 2.2.2).

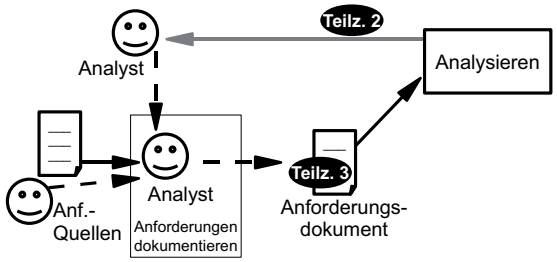


Abbildung 6.7: Direktes heuristisches Feedback dient zur konstruktiven Verbesserung während der Dokumentation von Anforderungen (Verfeinerung des allgemeinen Flusses Abbildung 6.4).

Ein Analyst arbeitet mit dem Werkzeug, um Anforderungen zu dokumentieren. Er kann mit bereits vorhandenen Use-Cases und Anforderungen starten und diese mit Hilfe des Werkzeugs weiter verarbeiten. Die Eingabe wird mit Hilfe von heuristischen Kritiken analysiert. Als Ergebnis präsentiert das Werkzeug Feedback in Form von Warnungen und Verbesserungsvorschlägen. Dies kann beim Analysten

einen *Breakdown* auslösen, ihn also dazu bringen, seine Eingabe zu überdenken. In Abbildung 6.7 ist dies durch einen Informationsfluss vom Analysten zum Analysten modelliert, durch den auch die Erhaltung der FLOW Signatur der Aktivität *Anforderungen dokumentieren* aus Abbildung 6.4 leichter erkennbar wird.

Der Analyst kann das Feedback ignorieren, zurückweisen oder akzeptieren. Diese Entscheidung ist der Einstieg in oben genannten Erfahrungskreislauf. Falls das Feedback akzeptiert wird, fließen neue Informationen in die Anforderungsdokumentation, die ihrerseits durch die heuristischen Kritiken analysiert werden. Dadurch ist der Informationskreislauf des *direkten heuristischen Feedbacks* geschlossen.

Feedback durch abgeleitete Perspektiven

Viele Fehler und Inkonsistenzen sind das Resultat mangelnder Übersicht. Das automatische Ableiten einer anderen Perspektive auf die Anforderungen kann helfen, die Übersicht zu erhöhen (z.B. in Form eines abgeleiteten Use-Case-Diagramms) und Inkonsistenzen aufzudecken.

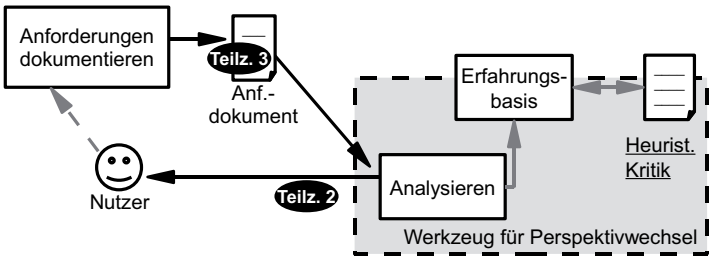


Abbildung 6.8: Der Informationskreislauf um abgeleitete Modelle für einen Perpektivwechsel aus Abbildung 6.4.

Der Nutzer kann nun die ursprünglichen Anforderungen überarbeiten, so dass die daraus generierte Perspektive eher seinen Erwartungen entspricht. Abbildung 6.8 zeigt diesen Informationskreislauf als Ausschnitt aus Abbildung 6.4. Im Gegensatz zu heuristischem Feedback muss der Nutzer also selbst *Indikationen* für mögliche Probleme finden.

6.3 Zusammenfassung: Problemstellung und Ziele der Arbeit

In diesem Kapitel wurde die Problemstellung aus Kapitel 1 mit Hilfe der Grundbegriffe aus den Kapiteln 2, 4 und 5 präzisiert. Die konkrete Forschungsfrage lautet:

Wie kann eine Organisation auf Basis messbarer Qualitätseigenschaften systematisch lernen, Anforderungen besser zu dokumentieren?

Um diese Frage zu beantworten, wurden konkrete Messziele aufgestellt und auf Basis von Abstraction Sheets der GQM-Methode und der in Abschnitt 6.1 zusammengefassten Konzepte dieser Arbeit präzisiert. Dies entspricht den folgenden Forschungsfragen:

Frage 1	Sind erfahrungsbasierte RE-Werkzeuge in der Lage, organisatorisches Lernen zu unterstützen?
Frage 2	Sind erfahrungsbasierte RE-Werkzeuge in der Lage, individuelles Lernen zu unterstützen?
Frage 3	Helfen erfahrungsbasierte RE-Werkzeuge, bessere Anforderungsdokumentation zu erstellen?

Zudem wurden in diesem Kapitel die verschiedenen Eigenschaften von Anforderungsdokumenten und erfahrungsbasierten RE-Werkzeugen zusammengefasst. Unterschiedliche Belegungen dieser Eigenschaften sind Eingangsvariablen für die Evaluation der Messziele im folgenden Kapitel.

7 Beispielimplementierungen: erfahrungsbasierte RE-Werkzeuge

Da in dieser Arbeit die auf heuristischem Feedback basierende werkzeuggestützte Nutzung von Erfahrungen evaluiert werden soll, werden für die Evaluation spezielle *Beispielimplementierungen* benötigt, die bestimmte Aspekte erfahrungsbasierter RE-Werkzeuge umsetzen.

Definition 32: *Beispielimplementierung*

Eine Beispielimplementierung eines erfahrungsbasierten RE-Werkzeugs ist auf ein Evaluationsszenario abgestimmt und setzt bestimmte Aspekte des Referenzmodells um.

Beispielimplementierungen sind unter anderem die einzelnen Komponenten von HeRA (vergleiche Kapitel 3). Die unterschiedlichen Ausprägungen erfahrungsbasierter RE-Werkzeuge spiegeln sich in den Einflussfaktoren der Abstraction Sheets in Abschnitt 6.2 wieder.

Abschnitt 7.1 führt einen Steckbrief für erfahrungsbasierte RE-Werkzeuge ein. Dieser Steckbrief hilft, die wesentlichen Eigenschaften eines erfahrungsbasierten RE-Werkzeugs kompakt darzustellen und es so zu charakterisieren.

Im Rahmen der Vorgaben aus Kapitel 6.1 werden in den darauf folgenden Abschnitten konkrete Beispielimplementierungen für die Evaluation mit Hilfe des Steckbriefs kurz charakterisiert. Das Kapitel schließt mit einem kurzen Überblick, der Unterschiede zwischen den eingeführten Beispielimplementierungen verdeutlicht.

7.1 Werkzeugsteckbrief

Voraussetzung für die Evaluation sind eine Reihe von erfahrungsbasierten RE-Werkzeugen, mit denen die hier vorgestellten Konzepte abgedeckt werden. Dieser Abschnitt liefert einen Steckbrief zur kompakten Klassifikation von erfahrungsbasierten Werkzeugen: Alle in dieser Arbeit vorgestellten erfahrungsbasierten RE-Werkzeuge werden mit Hilfe dieses Steckbriefs kurz skizziert und gegebenenfalls zusätzlich erläutert (siehe Abschnitt 7). Tabelle 7.1 veranschaulicht das Schema des Steckbriefs.

Tabelle 7.1: Struktur des Steckbriefs erfahrungsbasierter RE-Werkzeuge.

<Erfahrungsbasiertes RE-Werkzeug>, Steckbrief		
Name:	<kurzer Bezeichner mit inhaltlichem Bezug>	<**Visualisierung** der Charakteristika *Einstellung* und *Erfahrung* der anvisierten Nutzer gemäß Abbildung 4.1>
Typ:	<heuristisches oder perspektivwechelndes RE-Werkzeug>	
Beschreibung:	<1-2 Sätze, welche Rolle dieses Werkzeug zu welchem Zweck einsetzt.>	
Nutzerprofil:	<Rolle und Ziel nach Kapitel 4.1.>	
Einsatzgebiet		
Reife:	<Einschätzung der Dokument-Reife nach Kapitel 4.2, für die sich dieses Werkzeug einsetzen lässt.>	<**Visualisierung** der anvisierten Dokument-Formalität und -Reife gemäß Abbildung 4.7 im Vergleich zur Validier- und Verifizierbarkeit>
Formalität:	<Einschätzung der Dokument-Formalität nach Kapitel 4.2, für die sich dieses Werkzeug einsetzen lässt.>	
Beschreibung:	<Beschreibung der Dokumente, auf die das Werkzeug angewendet wird.>	
Eigenschaften		
Autorität:	<Einschätzung der Autorität des Werkzeugs nach Kapitel 5.>	<**Visualisierung** der Eigenschaften von erfahrungsbasierten RE-Werkzeugen nach Tabelle 5.3 von ○○○○○ bis ●●●●●>
Pro-Aktivität:	<Einschätzung der Pro-Aktivität des Werkzeugs nach Kapitel 5.>	
Interpretierend:	<Einschätzung der selbstständigen Interpretation durch das Werkzeug nach Kapitel 5.>	
Lernfähig:	<Einschätzung der Lernfähigkeit des Werkzeugs nach Kapitel 5.>	
Perspektivwechselnd	Einschätzung der Unterstützung für Perspektivwechsel durch das Werkzeug nach Kapitel 5.>	

Im Kopf des Steckbriefs erhält jedes Werkzeug einen Namen. Der Typ zeigt an, ob Feedback mittels heuristischer Kritiken oder abgeleiteten Sichten gegeben wird. Eine kurze Beschreibung des Werkzeugs und ein kurzer Hinweis auf den zu unterstützenden Nutzertyp gibt zusätzliche Informationen. Diese Informationen werden auf Basis von Abbildung 4.1 visualisiert, in dem das vorausgesetze Wohlwollen und die vorausgesetzte Erfahrung des Nutzers für dieses Werkzeug eingetragen sind.

Der Bereich *Einsatzgebiet* bezieht sich auf die Konzepte aus Kapitel 4 und beschreibt die Reife und Formalität geeigneter Dokumente für dieses Werkzeug. Normalerweise handelt es sich dabei um einen bestimmten Bereich, in dem Reife und Formalität liegen müssen. Eine Erläuterung der Anforderungsdokumente, die mit diesem Werkzeug bearbeitet werden, ergänzt die Beschreibung des Einsatzgebiets. Eine grafische Darstellung visualisiert das Einsatzgebiet in einer Kopie von Abbildung 4.4. Die hellgraue Fläche zeigt, in welchem Bereich sinnvoll validiert werden kann, die dunkelgraue Fläche zeigt, in welchem Bereich sinnvoll verifiziert werden kann. Die schraffierte Fläche zeigt, in welchem Bereich das erfahrungsbasierte Werkzeug sinnvoll eingesetzt werden kann.

Schließlich wird das erfahrungsbasierte Werkzeug im unteren Teil des Steckbriefs nach den Konzepten aus Kapitel 5 charakterisiert. Bemerkungen geben Aufschluss darüber, wie die Bewertung des jeweiligen Aspekts zustande kommt. Gefüllte und leere Kreise visualisieren die Eigenschaften des Werkzeugs. Je mehr gefüllte Kreise eine Eigenschaft hat, um so mehr Verantwortung übernimmt das erfahrungsbasierte RE-Werkzeug. Dies ist durch das Computersymbol auf der rechten Seite symbolisiert. Je mehr leere Kreise eine Eigenschaft hat, um so mehr ist der Nutzer gefordert.

Die Beschreibungen in dem Steckbrief geben zudem Aufschluss auf die Verteilung der Komponenten gemäß Abschnitt 6.1.2.

7.2 HeRA (Heuristic Requirements Assistant)

HeRA wurde als Plattform konzipiert, mit der neue Konzepte für erfahrungsbasierte RE-Werkzeuge schnell umgesetzt werden können. In Kapitel 3 wurde diese Plattform mit den gebräuchlichsten Erweiterungen bereits vorgestellt. In diesem Abschnitt werden drei Erweiterungen mit unterschiedlichen Schwerpunkten für die Evaluation in Bezug zu den Konzepzen der Kapitel 4, 5 und 6 gesetzt:

- *HeRA.EPK:* Um den implizieten Geschäftsprozess einer Reihe von Use-Cases darzustellen, werden Ereignisgesteuerte Prozessketten abgeleitet. An dieser Erweiterung wird der Nutzen abgeleiteter Modelle für den Perspektivwechsel evaluiert.

- *HeRA.Glossar:* Die Argumentationskomponente HeRA.Glossar unterstützt die Erstellung von Glossaren. Die Besonderheit dieser Erweiterung ist das Ableiten von Erfahrungen durch die Beobachtung des Nutzerverhaltens.
- *HeRA.Kritik:* Das Kritiksystem für Anforderungen dient dazu, den Nutzer schon bei der Erstellung von Anforderungen Ratschläge und Hinweise zu geben, damit das zu erstellende Anforderungsdokument eine hohe Qualität erreicht. Die Besonderheit an dieser Erweiterung zu HeRA ist die hohe Pro-Aktivität und Interpretation durch die enthaltenen heuristischen Kritiken. Zudem erlaubt es das Kritiksystem seinen Nutzern, heuristische Regeln zu bearbeiten.

Da diese drei Erweiterungen sehr unterschiedliche Schwerpunkte hinsichtlich der unterstützten Nutzer- und Dokumenttypen sowie in den Ausprägungen der Eigenschaften erfahrungsbasierter RE-Werkzeuge haben, werden sie hier einzeln in Testumgebungen eingesetzt.

7.2.1 HeRA.EPK

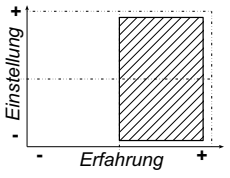
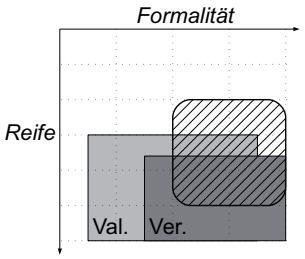
Die EPK-Erweiterung für HeRA (vergleiche Abschnitt 3.3.4) ist entstanden, um die Anforderungsanalyse für Service-Orientierte Architekturen (SOA) zu unterstützen [101]. In SOA-Projekten ist die Analyse des Geschäftsprozess noch wichtiger als in normalen Softwareprojekten. Dennoch werden in solchen Projekten die meisten Anforderungsdokumente normaler Softwareprojekte weiterhin benötigt. Dies führt zu einer Menge von Anforderungsmodellen, in denen Informationen dupliziert werden (wie zum Beispiel in Use-Cases und Geschäftsprozessen). Das Problem ist, dass sich diese Modelle häufig widersprechen, da sie in der Verantwortung unterschiedlicher Rollen liegen und unterschiedliche Stakeholder adressieren. Dadurch entsteht ein hoher Aufwand für die Synchronisation solcher Modelle.

Inkonsistenzen und Widersprüche in verschiedenen Anforderungsmodellen sind jedoch eine wichtige Quelle für neue oder bislang unentdeckte Anforderungen. Die EPK-Erweiterung in HeRA erlaubt es, sowohl die Use-Case- als auch die Geschäftsprozess-Perspektive einzunehmen. So können Widersprüche früh identifiziert und geklärt werden. Sowohl die Geschäftsprozessmodelle als auch die Use-Cases können auf diese Weise verbessert werden. Tabelle 7.2 gibt das Profil dieses erfahrungsbasierten RE-Werkzeugs in Form eines Steckbriefs.

7.2.2 HeRA.Glossar

Die Glossar-Erweiterung wurde als Bestandteil von HeRA bereits in Abschnitt 3.3.2 eingeführt. Sie basiert auf der Masterarbeit von Meyer [120]. An dieser Stelle werden daher nur kurz die verwendeten Heuristiken motiviert und beschrieben.

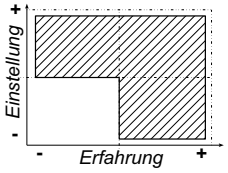
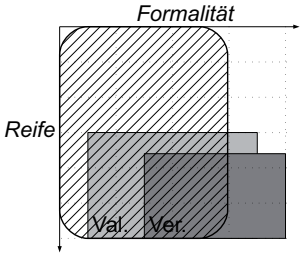
Tabelle 7.2: HeRA.EPK, Steckbrief eines perspektivwechselnden Werkzeugs.

HeRA.EPK, Steckbrief		
Name:	HeRA.EPK	
Typ:	erfahrungsbasiertes RE-Werkzeug	
Beschreibung:	Leitet aus einer Menge von Use-Cases eine EPK ab, die den durch die Use-Cases implizierten Geschäftsprozess visualisiert.	
Nutzerprofil:	Analyst prüft regelmäßig, ob die Use-Cases zusammen Sinn ergeben. Dafür ist Erfahrung nötig.	
Einsatzgebiet		
Reife:	vom Entwurf bis zum Vorschlag	
Formalität:	attributiert bis formal	
Beschreibung:	HeRA.EPK nutzt die Bedingungen und Auslöser eines Use-Cases um eine EPK abzuleiten [108]. Diese Attribute müssen explizit modelliert sein. Die EPKs dienen dazu, Inkonsistenzen im Entwurf zu finden und in den folgenden Versionen der Dokumente zu vermeiden.	
Eigenschaften		
Autorität:	mittel (3)	●●●○○
Pro-Aktivität:	schwach (1), in Editor integriert	●○○○○
Interpretierend:	schwach (1), Darst. vorhandener Daten	●○○○○
Lernfähig:	nein (0), erfordert neue Algorithmen	○○○○○
Perspektivwechselnd:	stark (4), die neue Perspektive erlaubt es, die Sichtweise eines *Business Analyst* einzunehmen [169]	●●●●○

Glossare dienen in Anforderungsspezifikationen dazu, eine allgemeine Basis für Definitionen wichtiger Begriffe zu bilden. Dennoch werden viele mehrdeutige Begriffe nicht in einem Glossar definiert, häufig auf Grund der stillschweigenden Annahme, dass alle Beteiligten sich über die Bedeutung im Klaren und einig sind. Solche Missverständnisse verursachen hohe Risiken in Projekten, vor allem, weil sie so schwer zu entdecken sind.

Um die Diskussion und schließlich die Definition solch gefährlicher Begriffe zu erreichen, wird ein Auslöser benötigt. Als Erweiterung zu HeRA ist das kontextsensitive Kritiksystem für Glossarbegriffe in der Lage, heuristisch gefährliche Begriffe zu entdecken und hervorzuheben [120, 104]. Dazu greift die Erweiterung auf zwei einfache aber mächtige Heuristiken zurück: Die Häufigkeitsheuristik findet wichtige und daher oft verwendete Begriffe. Die Erfahrungsheuristik entdeckt

Tabelle 7.3: HeRA.Glossar, Steckbrief einer Argumentationskomponente für Glossare.

HeRA.Glossar, Steckbrief		
Name:	HeRA.Glossar	
Typ:	heuristisches RE-Werkzeug	
Beschreibung:	Direktes Feedback bei der Dokumentation von Anforderungen hilft Nutzern gute Glossare zu erstellen.	
Nutzerprofil:	Analyst dokumentiert Anf. Er ist nicht unerfahren *und* ablehnend.	
Einsatzgebiet		
Reife:	Einsatz über den gesamten Projektverlauf sinnvoll möglich.	
Formalität:	Text. Prosa (1) möglich, identifizierbar (2) bis attributierbar (3) empfohlen	
Beschreibung:	Das Kritiksystem analysiert die Eingabe und schlägt dem Nutzer Begriffe vor, die in einem Glossar definiert werden sollten. Der Nutzer kann diese Vorschläge schnell zur späteren Bearbeitung dem Glossar hinzufügen und mit der Dokumentation von Anforderungen fortfahren.	
Eigenschaften		
Autorität:	gering (2), viele Vorschläge werden ignoriert	●●○○○
Pro-Aktivität:	hoch (4), aktuelles Feld wird bei Änderung analysiert	●●●●○
Interpretierend:	mittel (3), grammatikalische Beugung wird ausgefiltert	●●●○○
Lernfähig:	automatisch (5), Jeder Begriff, der in ein Glossar aufgenommen wird, bereichert die Erfahrungsheuristik	●●●●●
Perspektivwechselnd:	gering (1), lediglich Überblick über verwendete Begriffe	●○○○○

durch den Vergleich mit alten Glossaren Begriffe, die bereits früher wichtig genug waren, um in einem Glossar definiert zu werden. Dies erlaubt es, Glossare aus alten Projekten wiederzuverwenden, um Begriffe vorzuschlagen, die auch für das Glossar des aktuellen Projekts interessant sein könnten. Tabelle 7.3 gibt den Steckbrief dieses heuristischen RE-Werkzeugs.

Als Methode, um eine gemeinsame Basis in Softwareprojekten zu erstellen sind Glossare weitgehend akzeptiert (z.B. [27, 116, Seite 99]). Der Zweck von Glossaren ist es, Begriffe aus der projektspezifischen Domäne zu definieren und ein gemeinsames Verständnis zu erlangen. Die Nützlichkeit von Glossaren in diesem Kontext wurde bereits oft berichtet (z.B. [27, 116, Seite 99]).

Nicht nur in studentischen Softwareprojekten kann man oft beobachten, dass vor allem komplizierte Fachbegriffe in das Glossar einer Anforderungsspezifikation aufgenommen werden. Es ist verständlich und wichtig, wenn fachfremde Software-Entwickler unvertraute Begriffe an zentraler Stelle erläutern. Oft führen aber gerade die gebräuchlichen Begriffe zu Missverständnissen, wenn sie im Kontext eines Projekts unterschiedlich verwendet werden. Ein Beispiel für diesen Effekt ist das Wort *Projekt*. Die meisten Entwickler nehmen zunächst an, zu wissen, wie der Begriff *Projekt* definiert ist (*ein gemeinsames Unterfangen, häufig verbunden mit Forschung oder Entwurf, das sorgfältig geplant wird um ein gegebenes Ziel zu erreichen* [134]). In einem Softwareprojekt war dieser Begriff jedoch abweichend als *Container für die Daten, die im zu entwickelnden Projekt erzeugt werden* definiert.

Dieses Szenario ist deshalb so gefährlich, weil Entwickler und Kunden denken, dass sie ein gemeinsames Verständnis des Begriffs haben. Weder Kunde noch Entwickler sind sich ihrer gegensätzlichen Definitionen des Begriffs bewusst. Daher kann nicht erwartet werden, dass sie dieses Problem ohne Hilfe von außen lösen. Der einzige Weg, diese versteckte Mehrdeutigkeit aufzudecken, ist es, beide Parteien mit der Inkonsistenz ihrer angenommenen Bedeutungen eines Begriffs zu konfrontieren.

Das Kritiksystem für Glossarbegriffe dient dazu, solche potentiell gefährlichen Begriffe zu entdecken und an den Analysten zu melden. Dieser kann nun prüfen, ob diese Begriffe von allen Stakeholdern konsistent interpretiert werden. Im Beispiel oben könnte ein Analyst überrascht sein, dass der Begriff *Projekt* für das Glossar vorgeschlagen wird. Diese Überraschung kann den Impuls (*Breakdown*) geben, zu klären, ob alle Beteiligten sich über eine einzige Bedeutung des Begriffs einig sind.

Um dem Nutzer interessante Begriffe für das Glossar vorzuschlagen, muss zunächst definiert werden, was einen Begriff für das Glossar interessant macht. Die Glossar-Erweiterung folgt dabei zwei Grundannahmen:

- **Annahme 1:** *Häufigkeit ist wichtig.* Wenn ein Begriff oft verwendet wird, ist er wahrscheinlich relevant für das Glossar, weil er oft falsch verwendet werden kann.
- **Annahme 2:** *Erfahrung ist wichtig.* Wenn ein Begriff in einem alten Projekt dem Glossar hinzugefügt wurde, sollte dieser vermutlich im aktuellen Glossar auch definiert werden.

Es ist empirisch belegt, dass Menschen wichtige Begriffe öfter schreiben, um sie zu klären oder ihre Bedeutung zu unterstreichen [115]. Dadurch ist die Häufigkeitsheuristik ein wichtiges und starkes Mittel um Begriffe zu identifizieren, die für ein Projekt wichtig sind. Einfaches Zählen von Begriffen reicht jedoch nicht aus um gute Ergebnisse zu erzielen. Ein Wort kann in verschiedenen Formen im Text verwendet werden. Ohne eine Form der Normalisierung würde der Algorithmus jede Form einzeln vorschlagen. Die Begriffe *Projekt* und *Projekte* haben dieselbe Bedeutung, aber wenn sie buchstäblich verglichen werden, sind sie unterschiedlich. Daher müssen alle Wörter zunächst in eine definierte Grundform überführt werden. Die bekannteste Methode dafür ist Stemming (z.B. Porter-Stemming [133]) oder Lemmatisierung. Diese Verfahren führen jedoch dazu, dass die Grundform kein grammatikalisch korrektes Wort mehr ist. Im natürlichsprachlichen Kontext einer Anforderungsspezifikation ist dies ein Nachteil. Daher verwendet die Glossarerweiterung eine modifizierte Rechtschreibprüfung und die Wörterbücher von OpenOffice.org, um die Grundform von gebeugten Wörtern abzuleiten. Dies bringt einen weiteren Vorteil mit sich: Weil die Lemmatisierung von der Sprache unabhängig ist, funktioniert dieser Algorithmus in jeder Sprache, für die es ein Wörterbuch gibt. Durch diese Methode erkennt der Algorithmus die Wörter *Projekt* und *Projekte* als einen im Glossar zu definierenden Begriff.

Die Erfahrungsheuristik geht von der Annahme aus, dass das Entdecken von für das Glossar relevanten Begriffen die größte Herausforderung bei der Erstellung hochwertiger Glossare ist. Wurde ein relevanter Begriff einmal entdeckt, so kann später von dieser Erfahrung profitiert werden, indem Begriffe, die bereits einmal in einem Glossar definiert waren immer wieder vorgeschlagen werden. In einem solchen Fall ist ein Begriff kein stillschweigendes Risiko mehr, sondern als ein Begriff identifiziert worden, der in einem Glossar erläutert werden sollte. Aus diesem Grund werden solche Begriffe in der Vorschlagsliste ganz oben angezeigt, sogar vor Begriffen, die nach der Häufigkeitsheuristik wichtiger sind (öfter vorkommen).

Damit die Vorschlagsliste für den Nutzer hilfreich ist muss die Anzahl der vorgeschlagenen Begriffe begrenzt werden. Werden zu viele Begriffe vorgeschlagen, wird der Nutzer eher abgelenkt als unterstützt. Daher wird die Vorschlagsliste auf verschiedene Arten gefiltert:

- *Top-10.* Dem Nutzer werden im Normalfall nur die wichtigsten 10 Begriffe angezeigt. Der Nutzer kann aber auch wählen, dass mehr Begriffe angezeigt werden.

- *Keine Dopplung.* Begriffe, die bereits im Glossar stehen, werden nicht erneut angezeigt.
- *Rote Liste.* Der Nutzer kann unwichtige Begriffe ausfiltern. Die meisten dieser sogenannten Stop-Wörter werden mittlerweile bereits herausgefiltert, der Nutzer kann zusätzlich mit regulären Ausdrücken verhindern, dass IDs oder Seitenzahlen vorgeschlagen werden.
- *Kontext.* Es werden nur Begriffe vorgeschlagen, die im aktuellen Kontext (z.B. gerade bearbeiteter Use-Case) auch vorkommen.

Idealerweise verwendet der Nutzer das Kritiksystem für Glossarbegriffe während er Anforderungen in HeRA dokumentiert. Dabei hält er hin und wieder inne und überfliegt die Vorschlagsliste (zum Beispiel nachdem ein Use-Case vollständig dokumentiert wurde). Mit einem Klick auf das Wort wird dieses dem Glossar hinzugefügt. Der Editor bleibt jedoch in seinem Kontext, so dass der Nutzer direkt weiterarbeiten kann. Zu einem späteren Zeitpunkt wird er das Glossar bearbeiten und allen Begriffen eine Definition hinzufügen.

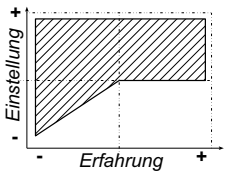
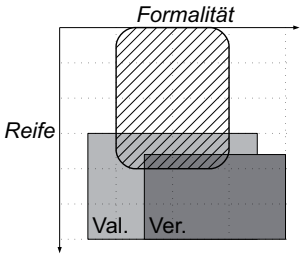
7.2.3 HeRA.Kritik

Obwohl in der Literatur zahlreiche Anleitungen zu finden sind, wie Anforderungen optimal dokumentiert werden sollten, finden sich in der Praxis viele Beispiele, die diesen Regeln nicht entsprechen. Selbst Experten haben oft nicht alle Regeln vor Augen und während der Analysephase eines Projekts selten die Zeit für ausgedehnte Literaturrecherchen. Entscheidend ist auch, dass man einen möglichen Befund erst erkennen muss, bevor reagiert werden kann. Wenn überhaupt, fallen stilistische Fehler bei der Qualitätssicherung auf. Um den Schreibstil zu verbessern und die Qualität der Anforderungen schon bei der Erstellung zu erhöhen, wäre daher unmittelbares Feedback wünschenswert. Ideal sind motivierte Nutzer, die bei der Verbesserung der heuristischen Kritiken aktiv mitwirken. Eher unerfahrene Nutzer können aber bereits von den initialen heuristischen Kritiken (nach dem Seeding) profitieren, ohne dass sie beim evolutionären Wachstum mitwirken.

Das Kritiksystem für Use-Cases dient dazu, die Effekte möglichst direktem Feedbacks beim Requirements Engineering genauer zu erforschen. Die heuristischen Kritiken dienen in diesem Fall dazu, den Nutzer schon beim Schreiben von Use-Cases auf mögliche Probleme aufmerksam zu machen und Lösungen vorzuschlagen. Die erste Version dieses Kritiksystems ist in der Masterarbeit von Crisp entstanden [45]. Die dahinter liegenden Konzepte wurden auf deutsch und englisch veröffentlich [91, 103] (vergleiche Abschnitt 3.3).

Use-Cases sind mittlerweile ein in der Praxis weit verbreitetes Mittel zur Dokumentation von Anforderungen [172]. Nicht nur bei Anfängern gibt es jedoch

Tabelle 7.4: HeRA.Kritik, Steckbrief eines Kritiksystems für Anforderungen und Use-Cases.

HeRA.Kritik, Steckbrief		
Name:	HeRA.Kritik	
Typ:	heuristisches RE-Werkzeug	
Beschreibung:	Direktes Feedback beim Dokumentieren von Anforderungen und Use-Cases	
Nutzerprofil:	Analyst dokumentiert Anforderungen, sollte offen für Feedback sein, idealerweise bereit, die Erfahrungsbasis zu verbessern	
Einsatzgebiet		
Reife:	Konzept bis Rohfassung	
Formalität:	identifizierbar bis attributierbar	
Beschreibung:	Das Kritiksystem analysiert die Eingabe und untersucht sie auf allgemeine Befunde (z.B. Passiv, Weakwords), auf Konsistenz (Bsp. Hauptakteur kommt im Szenario vor) sowie auf gute Struktur (Bsp. Use-Cases auf unterem Abstraktionsniveau werden in Use-Cases auf nächst höherer Ebene referenziert).	
Eigenschaften		
Autorität:	mittel (3)	●●●○○
Pro-Aktivität:	hoch (5), aktuelles Feld wird bei Änderung analysiert	●●●●●
Interpretierend:	mittel bis hoch (3-4), je nach Regel	●●●◐○
Lernfähig:	mittel (3), auf Kommentare, explizites Ignorieren oder Änderungen an den heuristischen Regeln angewiesen.	●●●○○
Perspektivwechselnd:	gering (1), lediglich eine Übersicht der Warnungen und Fehler erlaubt die Perspektive des Qualitätsbeauftragten.	●○○○○

häufig Unklarheit darüber, wie die Felder am Besten auszufüllen sind. Zudem wird die Analyse und Dokumentation von funktionalen Anforderungen in vielen Projekten im Vergleich zur Programmierung eher vernachlässigt. Das Resultat sind dann Use-Cases von schlechter Qualität, die nur einen zweifelhaften Nutzen für ein Projekt haben.

Verschiedene Fachbücher enthalten das notwendige Wissen, um gute Use-Cases zu erstellen (siehe beispielsweise [42, 5]). In der Praxis nimmt sich in dieser vernachlässigten Aktivität des Software Engineerings aber wohl selten jemand die Zeit, ein Problem dort nachzuschlagen.

Ein Kritiksystem kann in dieser Situation helfen, indem Informationen aus Fachbüchern oder aus den Erfahrungen vergangener Pojekte genau dann angezeigt werden, wenn sie benötigt werden. In den Abschlussarbeiten von Crisp und Kitzmann [45, 89] wurden daher oben genannte Fachbücher nach konkreten Empfehlungen und Handlungsanweisungen durchsucht, die sich als heuristische Kritik umsetzen lassen. Bei der Arbeit mit dem Kritiksystem sind noch zusätzliche heuristische Kritiken entstanden. Die Menge der verwendeten heuristischen Kritiken lässt sich auf verschiedene Art und Weise klassifizieren.

Klassifikation nach Inhalt der kritischen Meldung.

Heuristische Kritiken unterscheiden sich deutlich nach ihrer Art und Zielsetzung.

- *Allgemeiner Hinweis.* Viele Hinweise aus der Fachliteratur beziehen sich nicht auf ein spezielles Feld des Use-Cases, sondern geben eher allgemeine Hinweise, zum Beispiel zum Prozess der Erstellung von Use-Cases. Das Kritiksystem zeigt diese Hinweise zufällig an, wenn gerade kein spezieller Kontext gewählt ist. Beispiel: Use-Cases dienen dazu funktionale Anforderungen zu erfassen.
- *Allgemeiner Hinweis zur Eingabe.* Bestimmte mehrdeutige Formulierungen sollten in Anforderungsdokumenten allgemein nicht vorkommen. Dies gilt (mit wenigen Ausnahmen) auch für die Felder eines Use-Cases. Heuristische Regeln müssen also überprüfen, ob eine Indikation für ein allgemeines Problem in der Eingabe vorliegt. Beispiel: Der Inhalt in einem Feld sollte nicht zu lang sein, um die Lesbarkeit zu erhöhen.
- *Kontextspezifischer Hinweis.* Wenn der Nutzer in der Eingabemaske ein spezielles Feld anwählt, zeigt das Kritiksystem die Erläuterung zu diesem Feld nach Cockburn an [42]. Gerade bei den weniger gebräuchlichen Feldern wird dies auch von erfahrenen Nutzern als sehr hilfreich empfunden. Beispiel: Die Systemgrenzen dienen dazu, das zu erstellende System abzugrenzen. Sie soll-

ten so gewählt werden, dass der Hauptakteur des Use-Cases gerade nicht mehr zum System gehört.

- *Kontextspezifischer Hinweis zur Eingabe.* Einige heuristische Kritiken gelten speziell für ein Feld eines Use-Cases. Das Kritiksystem wertet diese dann aus, sobald der Nutzer ein Feld ausfüllt. Beispiel: Enthält ein Szenario-Schritt eine Bedingung („Wenn..., dann..."), so beschreibt er eher eine Erweiterung, die in einem speziellen Feld dokumentiert werden sollte.
- *Kontextübergreifender Hinweis zur Eingabe.* Ein Use-Case sollte in sich und zu den anderen Use-Cases eines Systems konsistent sein. Diese Klasse von heuristischen Kritiken überprüft, ob eine Eingabe zu den bisher dokumentierten Use-Cases passt. Beispiel: Die Titel zweier Use-Cases müssen unterschiedlich sein. Der Hauptakteur eines Use-Cases muss auch im Szenario vorkommen.

Klassifikation nach Art der heuristischen Regel.

Kitzmann hat die heuristischen Kritiken in seiner Bachelorarbeit nach der Art der Implementierung ihrer heuristischen Regeln klassifiziert [89]. Er unterscheidet dabei die folgenden Typen von Regeln:

- *Zufällige Auswahl.* Allgemeine Hinweise werden nur dann angezeigt, wenn kein spezieller Kontext vorliegt. Aus den verfügbaren allgemeinen Hinweisen werden drei zufällig ausgewählt.
- *Auswahl nach Kontext.* Kontextspezifische Hinweise werden immer dann angezeigt, wenn der Nutzer im entsprechenden Kontext arbeitet.
- *Suche nach Schlüsselwörtern.* Die meisten Kritiken kommen damit aus, das Vorkommen von Schlüsselwörtern im Text zu überprüfen.
- *Länge.* Die Länge von Attributen eines Use-Cases wird ebenfalls in vielen heuristischen Regeln verwendet. So gelten drei bis neun Schritte als optimale Länge eines Szenarios (siehe [42]) und die Beschreibung eines Use-Cases muss länger als der Titel sein (da sonst keine genauere Beschreibung zu erwarten ist).
- *Zählen.* Etwas seltener zählen heuristische Regeln das Vorkommen bestimmter Eigenschaften. Beispiel: Um zu komplizierte Formulierungen zu suchen, liefert bereits das Zählen der Kommata ein brauchbares Ergebnis für eine heuristische Regel.

7.3 KonPAss (Konsistenz-Prüf-Assistent)

Große Softwareprojekte werden in der Regel in kleinere Teilprojekte unterteilt. Anforderungen können dann für jedes Teilprojekt separat erhoben und dokumentiert werden. Dies erleichtert es Analysten die interne Konsistenz der Anforderungen für ein Dokument in einem Teilprojekt sicherzustellen. Dafür entsteht die neue Herausforderung, globale Konsistenz über mehrere voneinander abhängige Dokumente unterschiedlicher Teilprojekte herzustellen.

Um dies zu erreichen, definieren Organisationen zunächst mehr oder weniger explizite Konventionen und Regeln. Dadurch wird festgelegt, wie die verschiedenen Dokumente miteinander zusammenhängen. An verschiedenen Zeitpunkten in einem Softwareprojekt müssen diese Konventionen und Regeln überprüft werden. Diese Überprüfung ist sehr aufwendig und verschlingt in vielen Projekten einen beträchtlichen Teil der Ressourcen.

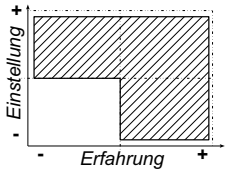
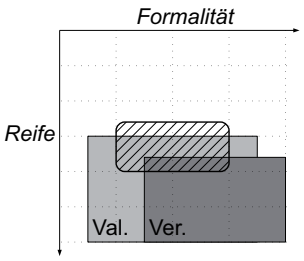
In diesem Szenario bietet sich die Verwendung von heuristischen Kritiken an: Die Reife der Anforderungsdokumente nimmt während des Projektverlaufs ständig zu, die Anforderungen werden in der Regel textuell dokumentiert und Anforderungen sind attributiert oder zumindest einzeln identifizierbar. Die Regeln und Konventionen lassen sich zum Teil mit wenig Aufwand formalisieren.

In Zusammenarbeit mit der Daimler AG wurde der Konsistenz-Prüf-Assistent (KonPAss) erstellt (siehe Steckbrief in Tabelle 7.5). KonPAss ist ein heuristisches Werkzeug, das die Konsistenz von Anforderungsdokumenten überprüft. KonPAss unterstützt die Definition und Überprüfung von Konsistenz zwischen verschiedenen Dokumenten in einer verteilten Anforderungsdatenbank. Durch eine hohe Konfigurierbarkeit lässt sich KonPAss an verschiedene Projektsituationen anpassen. Auf Basis der durch die Nutzer definierten Regeln erlaubt es KonPAss, kostengünstig viele Befunde zu Konsistenzproblemen zu finden. Diese werden nach der computergestützten Konsistenzprüfung in einer fokussierten Review-Sitzung analysiert und dann entweder zur Verbesserung der Dokumente oder zur Verbesserung der Konsistenzregeln genutzt.

7.3.1 Use-Cases für KonPAss

Abbildung 7.1 zeigt die Rollen und ihre Ziele im Kontext von KonPAss als UML Use-Case-Diagramm. Das primäre Ziel der Analysten ist es, gute Anforderungsdokumente zu erstellen. Daher überprüfen Analysten ihre Anforderungsdokumente regelmäßig. Bei dieser Aufgabe der Qualitätssicherung stehen ihnen Berater zur Verfügung. Eine spezielle Art der Überprüfung von Anforderungsdokumenten ist die Konsistenzprüfung. Bei dieser täglichen Aufgabe kann KonPAss verwendet werden.

Tabelle 7.5: KonPAss, Steckbrief eines heuristschen Werkzeugs zur Konsistenzsicherung in Anforderungsdokumenten.

KonPAss (Konsistenz-Prüf-Assistent), Steckbrief		
Name:	KonPAss	
Typ:	heuristisches RE-Werkzeug	
Beschreibung:	Prüft Konsistenz zwischen und in verschiedenen Anforderungsdokumenten.	
Nutzerprofil:	Qualitätsmanager beseitigt formale Schwächen vor (teurem) manuellem, inhaltlichen Review, ist in der Lage, Feedback zu bewerten.	
Einsatzgebiet		
Reife:	später Entwurf, Rohfassung	
Formalität:	textuell, identifizierbar bis attributierbar	
Beschreibung:	KonPAss überprüft Konsistenzregeln zwischen natürlichsprachlichen Anforderungen. Diese müssen identifizierbar sein, das Dokument sollte nicht mehr bewusst inkonsistent sein. Je komplexer / umfangreicher das Dokument, um so eher lohnt sich KonPAss.	
Eigenschaften		
Autorität:	stark (5), KonPAss soll Reviews entlasten	●●●●●
Pro-Aktivität:	keine (0), Aufruf ist explizite Aktivität im Qualitätssicherungsprozess	○○○○○
Interpretierend:	mittel bis stark (3-4), je nach Regel	●●●◐○
Lernfähig:	gering (1), eine API existiert	●○○○○
Perspektivwechselnd:	mittel (3), KonPAss integriert verschiedene Dokumente und erlaubt so weitere Perspektiven	●●●○○

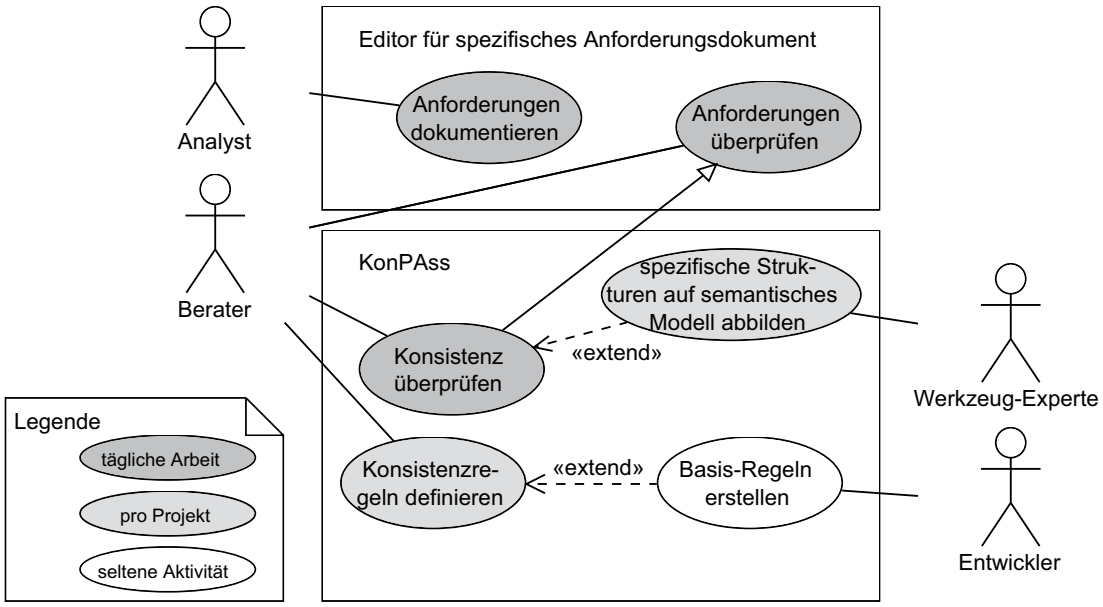


Abbildung 7.1: Das UML Use-Case zu KonPAss zeigt, welche Rollen an KonPAss beteiligt sind und welche Ziele sie dabei verfolgen.

Zu bestimmten Zeitpunkten (z.B. zu Beginn eines Projekts) definieren die Berater Konsistenzregeln. Diese Regeln werden mit der Zeit weiterentwickelt und so gelegentlich verändert. Einige der Regeln sind spezifisch für die Struktur der verwendeten Anforderungsdokumente. Die meisten Konsistenzregeln sind spezifisch für die Domäne des Projekts und projektinterne Konventionen. Daher ist es wichtig, dass Dokument-Struktur und Konsistenzregeln unabhängig von einander weiterentwickelt werden können. Um dies zu erreichen, werden Regeln mit Hilfe eines abstrakten, semantischen Modells der Dokument-Struktur definiert. Ein spezieller Berater, der Werkzeug-Experte, bildet die spezifische Struktur der verwendeten Anforderungswerkzeuge auf dieses abstrakte Modell ab.

Analysten denken normalerweise in Begriffen des abstrakten Modells und reden von *Schnittstellenbeschreibungen* an Stelle von *erste Spalte in der Tabelle unter einer Überschrift der Ebene 2 mit dem Text Parameter*. Die Abbildung zwischen der spezifischen Dokumentstruktur und dem abstrakten Modell erlaubt es, KonPAss mitzuteilen, an welcher Stelle die Gegenstände der Konsistenzprüfung zu finden sind.

Die Konsistenzregeln bauen auf Basis-Regeln auf. In seltenen Fällen müssen zusätzliche Basisregeln erstellt werden, um neue Konsistenzregeln zu erlauben. Da dies sehr selten vorkommt, ist dies Aufgabe der Entwickler von KonPAss.

Das Use-Case-Diagramm in Abbildung 7.1 grenzt KonPAss von der konstruktiven Aktivität der Erstellung von Anforderungsdokumenten ab. Im Gegensatz zu HeRA zielt KonPAss einzig auf die analytische Qualitätssicherung.

7.3.2 Heuristische Kritiken in KonPAss

Um die Konsistenz eines Dokuments mit KonPAss zu überprüfen, müssen Berater zunächst Konsistenzregeln definieren, die auf das Dokument angewendet werden sollen. Dieser Abschnitt liefert Beispiele solcher Regeln und zeigt, dass die meisten der relevanten Konsistenz-Aspekte automatisch geprüft werden können.

KonPAss führt eine abstrakte Infrastruktur ein, um auf Daten aus verschiedenen Typen von Anforderungsdokumenten zugreifen zu können. Dies reduziert den Aufwand, heuristische Konsistenzregeln in KonPAss zu formulieren.

1. Wenn ein neuer Typ oder ein neues Format für Anforderungsdokumente eingeführt wird, muss KonPAss konfiguriert werden, um die abstrakten Konzepte (z.B. Anforderungen, Glossar-Begriffe) extrahieren zu können. Dies geschieht vergleichsweise selten, die meisten Organisationen verwenden das selbe Format für Anforderungsdokumente in mehr als einem Projekt.
2. Wird eine neue Konsistenzregel in KonPAss formuliert, so kann dabei auf die abstrakten Konzepte zurückgegriffen werden. Dies erleichtert die Definition von Konsistenzregeln erheblich.

In KonPAss sind prinzipiell zwei Arten von heuristischen Konsistenzregeln zu unterscheiden:

- *Allgemeine Regeln* prüfen typische Qualitätseigenschaften von Anforderungen (z.B. keine Passiv-Formulierungen, keine Weakwords). Die Regeln helfen, Dokumente hinsichtlich allgemeiner Qualitätsregeln konsistent zu halten.
- *Projektspezifische Regeln* prüfen projektspezifische Konsistenzregeln (z.B. die Einhaltung einer Vorlage oder spezieller Namenskonventionen).

Gerade die Regeln der zweiten Gruppe sind interessant: Diese Regeln sind spezifisch für die Vorlagen und den RE-Prozess im jeweiligen Projekt. Sie decken damit das spezifische Wissen über eine sinnvolle Anforderungsspezifikation im Projektkontext ab. Die meisten dieser Regeln sind mit geringen Anpassungen in ähnlichen Projekten einer Organisation wiederverwendbar. Beispiele finden sich zum Einen in der Bachelorarbeit von Förmer, der KonPAss für die Softwareprojekte in der universitären Lehre der Leibniz Universität Hannover angepasst hat [69]. Zum Anderen werden Beispiele aus dem Kontext der Daimler AG in der Beschreibung des Evaluationsszenarios in Abschnitt 8.3 gegeben.

7.3.3 Technische Beschreibung

Dieser Abschnitt beschreibt die Bedienkonzepte und das konzeptionelle Dokumentmodell von KonPAss. KonPAss zielt darauf ab, Berater bei der Durchführung von Konsistenz-Reviews zu unterstützen.

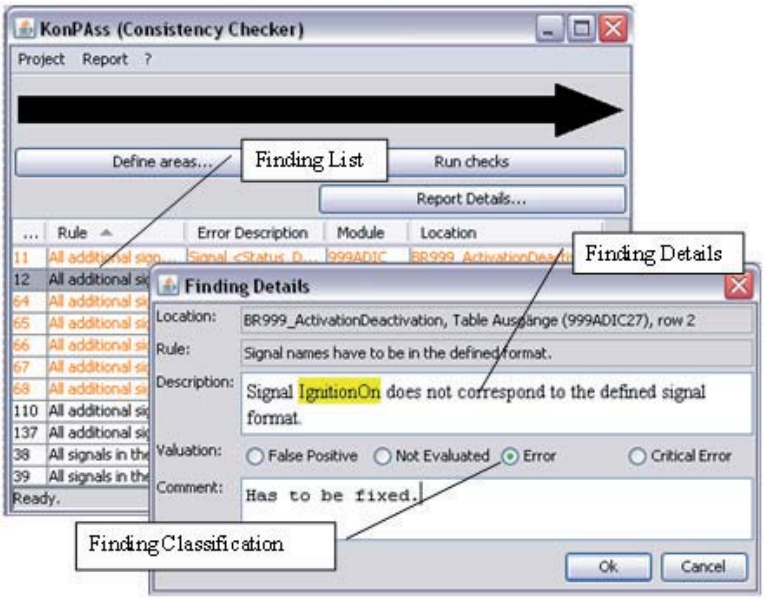


Abbildung 7.2: Hauptfenster von KonPAss und Details zu einem Befund.

Der Hauptteil von KonPAss in Abbildung 7.2 wird von einer sortierbaren Sicht auf die Befunde dominiert, die in einem Dokument gefunden wurden. Zudem zeigt das Hauptfenster Bedienelemente für die Einstellungen und die Durchführung einer Prüfung.

Spezielle Schnittstellen erlauben es KonPAss, auf verschiedene Datenquellen zurückzugreifen. Zur Zeit existieren Schnittstellen zum Einlesen von Word-, Excel-, OpenOffice- und LaTeX-Dokumenten sowie verschiedene Schnittstellen zu XML-Dateien, z.B. um den XML-Export einer DOORS-Anforderungs-Datenbank einzulesen und in das interne Dokumentmodell umzuwandeln.

Dazu müssen Berater definieren, wie relevante Bereiche in einem Dokument gefunden werden können, z.B. unter welcher Überschrift spezielle technische Details beschrieben sind. Abbildung 7.3 zeigt den Dialog, mit dem diese Bereichs-Definitionen erstellt werden können. Die Bereiche dienen dazu, die Lücke zwischen der physischen Struktur der verwendeten Dokumente und der semantischen Bedeutung für die Konsistenz zu überbrücken. Bereiche weisen ansonsten unstrukturierten Daten eine semantische Bedeutung zu. Ein Bereich kann ein bestimmter

Textabschnitt, der komplette Text oder eine spezielle Tabelle in einer Spezifikation sein.

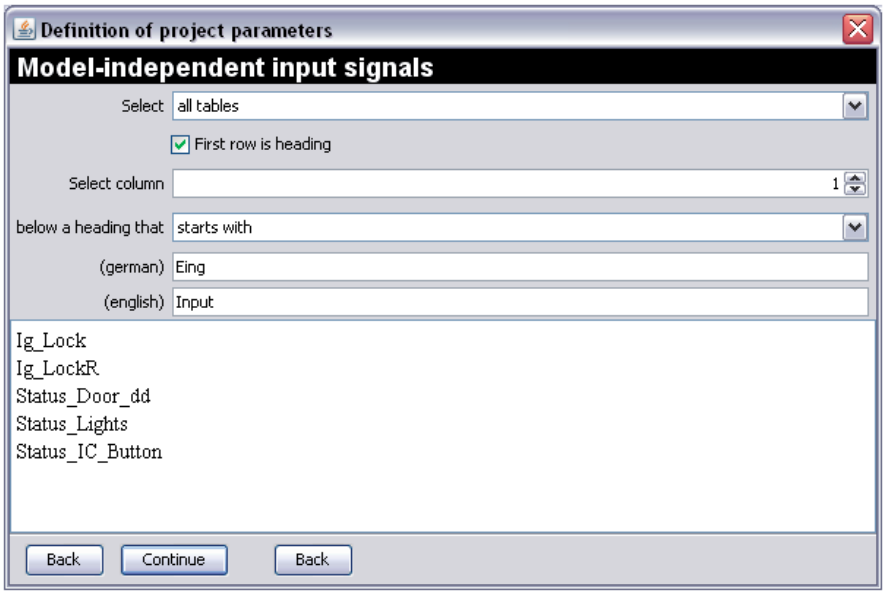


Abbildung 7.3: Definition von Bereichen in KonPAss.

Wichtig ist, dass die verwendeten Dokumente eine gewisse Struktur besitzen, die zur Definition der Bereiche genutzt werden kann. Abbildung 7.4 zeigt die abstrakte interne Struktur, in die Dokumente auf Grundlage der Bereichsdefinitionen in KonPAss überführt werden. Die aus Bereichen extrahierten Daten werden den Konsistenz-Regeln übergeben. Konsistenz-Regeln definieren, auf welchen semantischen Inhalten sie aufbauen. Baut beispielsweise eine Regel auf Tabellen mit Schnittstellenbeschreibungen auf, so werden diese Tabellen als Bereich definiert. Die Bereichsdefinition beschreibt dann genau, wie diese Tabellen im Dokument gefunden werden können und aus welchen Spalten oder Zeilen die Werte für den Bereich entnommen werden sollen (vergleiche Abbildung 7.3). Bei der Ausführung übergibt KonPAss die Daten dieser Tabellen an die Regel, ohne das diese Kenntnis darüber besitzt, wie die Tabellen in dem speziellen Dokument (z.B. Doors-XML-Export) gefunden werden können.

Konsistenz-Regeln sind gewöhnliche Java-Klassen, die bei KonPAss registriert werden. Das einfache Interface für Regeln schreibt Methoden vor, um die benötigten Bereiche zu definieren sowie um die Regel auszuführen.

Zu Beginn der Konsistenzprüfung werden die Daten für alle benötigten Bereiche aus den verschiedenen Dokumenten extrahiert. Daten werden auf der semantischen Ebene als Element-Objekte gespeichert. Danach werden die Elemente der Bereiche an alle aktivierten Regeln weitergeleitet. Schließlich werden die akti-

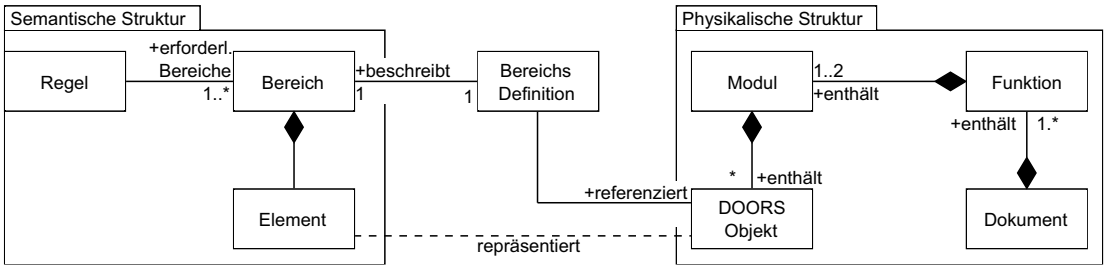


Abbildung 7.4: Semantisches Dokumentmodell in KonPAss.

vierten Regeln ausgeführt und die Ergebnisse in einen Report geschrieben. Jede Regel kann Befunde an den Report anhängen. Ein Befund besteht aus einer Beschreibung, dem Fundort in einem der Dokumente und der angewandten Regel. Er repräsentiert einen möglichen Fehler, der durch KonPAss entdeckt wurde. Der Berater kann Befunde beurteilen (falsch-positiv, Fehler, kritischer Fehler) und weitere Kommentare hinzufügen, die bei der Verbesserung der Dokumente oder der heuristischen Kritiken in KonPAss helfen können. Ein Report kann für die zukünftige Verwendung gespeichert werden und als HTML exportiert werden. Auf diese Weise können die Befunde einfach den Analysten zugänglich gemacht werden.

Berater haben zudem die Möglichkeit, die Anzahl der Befunde zu reduzieren, indem sie die zu überprüfenden Regeln in KonPAss gezielt auswählen. So kann das Dokument Regel für Regel überprüft werden.

7.4 Überblick der Beispielimplementierungen

Die Verbesserung von Anforderungsdokumentation auf Basis von Erfahrungen und Heuristiken geschieht durch die Verwendung von heuristischen und erfahrungsbasierten Werkzeugen. Für die Evaluation sind daher Beispielimplementierungen solcher Werkzeuge notwendig. Mit dem Steckbrief steht eine kompakte Darstellung zur Charakterisierung erfahrungsbasierter RE-Werkzeuge zur Verfügung, die es erlaubt, geeignete Beispiele für die Evaluation auszuwählen. Zudem wird diese Charakterisierung in Kapitel 9 verwendet, um verwandte Arbeiten mit dieser Arbeit zu vergleichen.

Neben dem Steckbrief wurden die in der Evaluation verwendeten Beispielimplementierung soweit beschrieben, wie dies für das Verständnis erforderlich ist. Tabelle 7.6 zeigt noch einmal übersichtlich die Eigenschaften der verwendeten Beispielimplementierungen erfahrungsbasierter RE-Werkzeuge. Für die Evaluation

Tabelle 7.6: Vergleich der Beispielimplementierungen für die Evaluation.

Name	Autorität	Pro-Akt.	Interpret.	Lernfähig	Kollab.
HeRA.EPK	●●●○○	●○○○○	●○○○○	○○○○○	●●●●○
HeRA.Glossar	●●○○○	●●●●○	●●●○○	●●●●●	●○○○○
HeRA.Kritik	●●●○○	●●●●●	●●●◐○	●●●○○	●○○○○
KonPAss	●●●●●	○○○○○	●●●◐○	●○○○○	●●●○○

ist wichtig, dass zu jeder Eigenschaft mindestens ein Werkzeug mit geringer und mindestens ein Werkzeug mit starker Ausprägung existiert.

8 Evaluation: heuristische Anforderungsdokumentation

Diese Arbeit beschäftigt sich mit einem Ausschnitt des Requirements Engineering: Dem Verbesserungspotential von Anforderungsdokumentation durch Erfahrungen und Heuristiken. Dieses Kapitel beinhaltet die Evaluation der dabei vorgestellten Zusammenhänge. In Abschnitt 6.2 wurden die folgenden Forschungsfragen gestellt:

Frage 1	Sind erfahrungsbasierte RE-Werkzeuge in der Lage, organisatorisches Lernen zu unterstützen?
Frage 2	Sind erfahrungsbasierte RE-Werkzeuge in der Lage, individuelles Lernen zu unterstützen?
Frage 3	Helfen erfahrungsbasierte RE-Werkzeuge, bessere Anforderungsdokumentation zu erstellen?

Um diese Fragen zu evaluieren, muss der Einsatz von konkreten erfahrungsbasierten RE-Werkzeugen in konkreten Situationen beobachtet werden.

Definition 33: *Evaluationsszenario*

Ein Evaluationsszenario legt fest, in welchem Kontext (Rollen, Ziele, Hintergrund) ein Evaluationsziel erreicht werden sollen.

Die Eigenschaften erfahrungsbasierter RE-Werkzeuge lassen sich nicht in einem einzigen Werkzeug vereinen, auch die Anwendungsszenarien unterscheiden sich recht deutlich. Daher ist es notwendig, geeignete Testumgebungen aufzubauen.

Definition 34: *Testumgebung*

Eine Testumgebung legt fest, wie in einem *Evaluationsszenario* eine *Beispielimplementierung* eines erfahrungsbasierten RE-Werkzeugs eingesetzt wird, um (ein) bestimmte(s) Evaluationsziel(e) zu erreichen.

Zu jeder Testumgebung wird frei nach der Vorlage für empirische Untersuchungen von Kitchenham [88] folgendes angegeben:

1. *Hintergrund:* Zunächst wird das *Evaluationsszenario* erläutert, um den Kontext der Evaluation zu beschreiben. Dabei wird insbesondere beschrieben, wie die jeweilige *Beispielimplementierung* sich in die *Testumgebung* einfügt.
2. *Ziel:* Aus den abstrakten Forschungsfragen und Evaluationszielen werden für jede Testumgebung *konkrete Evaluationsziele* abgeleitet.
3. *Methode:* Die *Evaluationsmethode* wird beschrieben, mit der diese konkreten Ziele evaluiert werden. Diese unterscheidet sich je nach *Testumgebung*.
4. *Ergebnisse:* Schließlich werden die *Evaluationsergebnisse* präsentiert, diskutiert und in den Zusammmenhang der Arbeit gesetzt.

Abschnitt 8.1 beschreibt, wie die Beispielimplementierungen aus Kapitel 7 in verschiedenen Testumgebungen eingesetzt werden, um die Forschungsfragen aus Abschnitt 6.2 zu adressieren. Eine Übersicht gibt an, welches Evaluationsziel in welcher Testumgebung evaluiert wird. In den darauf folgenden Abschnitten wird die Evaluation jeder Testumgebung detailliert beschrieben. Schließlich werden die Ergebnisse in Abschnitt 8.4 zusammengefasst.

8.1 Evaluationsstrategie

Dieser Abschnitt beschreibt die Strategie, mit der die Konzepte dieser Arbeit empirisch belegt werden. Ein solcher empirischer Beleg ist normalerweise schwer herzustellen, weil es eine Spannung zwischen den folgenden beiden Extremen gibt:

- *Reale Situationen* sind nur schwer und unter großen Kosten reproduzierbar.
- *Einfache, begrenzte Effekte* sind einfacher zu beobachten und zu evaluieren, haben aber nur wenig Bedeutung für praktische Anwendungen.

Bei empirischen Untersuchungen im Software Engineering sind Ergebnisqualität und Realisierbarkeit daher bereits bei der Planung zu berücksichtigen. Houdek beschreibt einen systematischen Ansatz, der in den folgenden Überlegungen berücksichtigt ist [78]. Entscheidend ist dabei, dass das Verhältnis zwischen Ergebnisqualität und Realisierbarkeit für die jeweilige Fragestellung angemessen ist.

Während Houdek besonderen Wert auf die Unterscheidung zwischen externen und internen Untersuchungen einer Organisation und der Übertragbarkeit von Ergebnissen studentischer Projekte in die Industrie legt, definieren Zelkovitz et al. [176] verschiedene Ebenen empirischer Validation zunächst unabhängig vom Kontext der Untersuchung. Anekdotische Evidenz ist einfach zu erheben, aber unzureichend, um übertragbare Ergebnisse zu erhalten. Streng formale Experimente ermöglichen es, statistische Methoden anzuwenden, aber es ist in nicht trivialen Umgebungen beinahe unmöglich alle Bedrohungen der Validität zu kontrollieren.

Um glaubwürdige Aussagen über relevante und nicht zu stark begrenzte Effekte treffen zu können, ist daher eine gute Balance notwendig. So wird in [16] argumentiert, dass kontrollierte Experimente auf Grund eines zu eingeschränkten Beobachtungsgegenstands irrelevante Ergebnisse liefern können. Weniger formale Evaluationsmethoden liefern tendenziell interessantere Ergebnisse, aber mit erheblich geringerer Signifikanz.

Im Rahmen dieser Arbeit wurde daher ein Mittelweg gesucht. Die GQM-Methode [14, 167] bietet einen Rahmen, um systematisch Metriken für die Prozessverbesserung herzuleiten. Das Ziel-orientierte Vorgehen erlaubt es, geeignete Metriken auszuwählen. Durch das disziplinierte und formale Vorgehen geht die Glaubwürdigkeit einer Evaluation nach der GQM-Methode deutlich über anekdotische Evidenz hinaus, auch wenn eine statistische Signifikanz oder Verallgemeinerbarkeit der Ergebnisse teilweise nicht nachgewiesen wird. Im Rahmen dieser Arbeit dient die GQM-Methode zur Definition und Strukturierung der zu evaluierenden Aussagen. Darüberhinaus wird diese Methode für Fallstudien eingesetzt, um systematisch Metriken herzuleiten, mit denen diese Aussagen evaluiert werden können.

8.1.1 Evaluationsmethode

Im empirischen Software Engineering sind neben anderen Methoden besonders *kontrollierte Experimente* und *Fallstudien* verbreitet.

Definition 35: *Kontrolliertes Experiment*

Ein kontrolliertes Experiment ist eine Untersuchung einer prüfbaren Hypothese, bei der eine oder mehr unabhängige Variablen manipuliert werden, um ihren Effekt auf abhängige Variablen zu messen.

(vergleiche Wohlin et al. [173] und Easterbrook et al. [55])

Ein solches kontrolliertes Experiment erlaubt es, den genauen Zusammenhang zwischen den Variablen zu bestimmen und festzustellen, ob dieser kausal ist.

Definition 36: *Fallstudie*

Eine Fallstudie ist eine empirische Untersuchung, mit der ein aktuelles Phänomen in seinem natürlichen Kontext untersucht wird, insbesondere, wenn die Grenzen zwischen Phänomen und Kontext nicht klar ersichtlich sind.

(vergleiche Yin [174])

Dabei wird prinzipiell zwischen explorativen Fallstudien, bei denen Phänomene untersucht werden, um neue Hypothesen abzuleiten oder Theorien aufzubauen

und bestätigende Fallstudien, bei denen existierende Hypothesen überprüft werden. In diesem Abschnitt kommen bestätigende Fallstudien zum Einsatz, um die in den vorherigen Kapiteln aufgebaute Theorie und die damit verbundenen Hypothesen zu evaluieren. In diesem Sinne ist die empirische Untersuchung in Abschnitt 4.3 eine bestätigende Fallstudie. Der Zusammenhang der Qualität von Anforderungsdokumentation und Projekterfolg wurde in seiner natürlichen Umgebung untersucht (Softwareprojekte), um eine Hypothese zu bestätigen oder zu widerlegen.

Um die zentralen Aussagen dieser Arbeit zu evaluieren, wurden je nach Kontext unterschiedliche Ansätze der Evaluation gewählt:

1. Ausgewählte Fragestellungen wurden durch *kontrollierte Experimente* evaluiert [173].
2. Wo dies auf Grund der Art der Aussage und der verfügbaren Ressourcen nicht möglich war, wurden *Fallstudien mit Hilfe der GQM-Methode* durchgeführt [14].
3. Für einige wenige Aussagen waren relevante Metriken mit den verfügbaren Mitteln nicht zu erheben. An diesen Stellen wird auf *anekdotische Evidenz* zurückgegriffen.

Die Evaluationsansätze werden in *Evaluationsszenarien* (siehe Definition 33) angewandt. In der Regel bedeutet dies, dass Testpersonen dabei beobachtet werden, wie sie die zu evaluierenden Tätigkeiten durchführen. Im Rahmen dieser Arbeit wurden die folgenden Arten von Evaluationsszenarien eingesetzt:

- *Studentische Projekte.* Softwareprojekte in der universitären Lehre eignen sich immer besonders für die Evaluation, weil die Bedingungen (Vorwissen, eingesetzte Werkzeuge, teilweise: Arbeitszeit) bekannt sind.
- *Industrieprojekte.* Softwareprojekte in der Industrie gelten in der Regel als wertvoller als studentische Projekte, da die Ergebnisse besser in die Praxis übertragbar sind. Die Evaluation wird jedoch durch wirtschaftliche Zwänge erschwert und die Rahmenbedingungen sind in der Regel schwerer zu kontrollieren.
- *Laborversuche.* Echte Daten (aus Industrieprojekten) werden nachträglich analysiert oder anderweitig außerhalb des Projekts, in dem sie entstanden sind, zur Evaluation verwendet. Dadurch wird die gute Übertragbarkeit der Ergebnisse von Industrieprojekten mit der guten Kontrolle über die Bedingungen von studentischen Projekten kombiniert. Laborversuche eignen sich immer dann, wenn die konkrete Projektsituation nicht im Vordergrund steht.

Zu beachten ist, dass kontrollierte Experimente sich am ehesten an Hand von Laborversuchen durchführen lassen, da sich nur in dieser definierten Umgebung die

verschiedenen Variablen weitgehend kontrollieren lassen. Fallstudien und anekdotische Evidenz kommen bei studentischen oder industriellen Softwareprojekten zum Einsatz. Nach [55] würde der anekdotische Ansatz dieser Arbeit vermutlich als informelle *Survey Research* klassifizieren lassen, da die Teilnehmer eines Softwareprojekts zu bestimmten Phänomenen befragt wurden.

8.1.2 Evaluationsziele

Das oberste Evaluationsziel, das mit diesen Testumgebungen zu untersuchen ist, leitet sich direkt aus dem Titel dieser Arbeit ab:

> Verbesserung der Dokumentation von Anforderungen auf Basis von Erfahrungen und Heuristiken.

Kapitel 6 hat gezeigt, wie sich dieses abstraktes Ziel in konkreten Architekturentscheidungen widerspiegelt. Die Teilziele lassen sich, wie in Abbildung 6.4 gezeigt, den Informationsflüssen zuordnen. In diesem Kapitel werden die abgebildeten Informationskreisläufe erneut aus anderer Perspektive betrachtet: das Ziel ist hier, Evaluationskriterien zu identifizieren.

In Kapitel 6 wurden für die Teilziele 1 bis 3 bereits Abstraction Sheets hergeleitet. Die Eigenschaften der erfahrungsbasierten RE-Werkzeuge finden sich dort in den Einflussfaktoren wieder. Tabelle 8.1 zeigt, welche Evaluationsziele durch die Beispielimplementierungen abgedeckt werden, welche Evaluationszenarien dabei zum Einsatz kommen und wie formal die Evaluation erfolgt. Die Tabelle zeigt, dass alle GQM-Fragen aus Abschnitt 6.2 abgedeckt sind. Für das Teilziel 1 *Evaluiere den Einfluss von erfahrungsbasierten RE-Werkzeugen auf das organisatorische Lernen* sind alle Fragen sogar durch industrielle Fallstudien und Experimente abgedeckt:

Q 1.1. *Akzeptieren die Nutzer das erfahrungsbasierte Werkzeug und schätzen sie es als nützlich ein?* Akzeptanz des Werkzeugs ist eine Grundvoraussetzung für den erfolgreichen Einsatz. Explizit evaluiert wurde dies in der KonPAss-Testumgebung, einem Laborversuch. Die Frage, ob KonPAss auch im Ernstfall eingesetzt würde, wurde deutlich mit *ja* beantwortet.

Q 1.2. *Sind die Nutzer in der Lage, Erfahrungen (heur. Kritiken) zu bewerten?* Die Bewertung von Erfahrungen ist eine wichtige Grundvoraussetzung für organisatorisches Lernen, da die Bewertungen in die Verbesserung der Erfahrungsbasis einfließen. HeRAs Argumentationskomponente für Glossare bezieht die Bewertung aus der Beobachtung, ob ein Nutzer den Vorschlag annimmt oder zurückweist. HeRAs Kritiksystem für Anforderungen und KonPAss erlauben dem Nutzer, Bewertungen durch Feedback-Formulare abzugeben. Diese Bewertungsmöglichkeiten sind in KonPAss die Grundlage für die Evaluation.

Tabelle 8.1: Beispielimplementierungen und Evaluationsziele. Die Tabelle zeigt die Vollständigkeit der Evaluation, indem den GQM-Fragen aus Kapitel 6 Evaluationsszenarien zugeordnet werden.

		HeRA		**KonPAss**
Ziel / Frage	EPK	Glossar	Kritik	
Teilziel 1 – Verbesserung des organisatorischen Lernens im Requirements Engineering				
Q 1.1 Akzeptanz durch Nutzer?				🏭, • •
Q 1.2 Sind Bewertungen heur. Krit. möglich?		🏭, • •	•	🏭, • • •
Q 1.3 Können Nutzer heur. Krit. ändern?			• • •	
Q 1.4 Erfahrung aus Nutzer-Beobachtung?		🏭, • •		
Q 1.5 Lernt LSO mehr Erfahrungen?		🏭, • •	•	🏭, • • •
Q 1.6 Verwendet LSO mehr Erfahrungen?	•	🏭, • •	• •	🏭, • • •
Teilziel 2 – Verbesserung des individuellen Lernens im Requirements Engineering				
Q 2.1 Lernt Analyst durch Pro-Aktivität?			• •	
Q 2.2 Lernt Analyst durch Interpret.?			• •	
Q 2.3 Profitiert Analyst von Persp.-wechsel?	•			
Q 2.4 Führt Autorität zu Akzeptanz?		🏭, •		🏭, • •
Teilziel 3 – Verbesserung des Erstellungsprozesses von Anforderungsdokumenten				
Q 3.1 Zeigen Indikatoren inhalt. Qualität?		*— Siehe Abschnitt 4.3 und 8.1.3 —*		
Q 3.2 Reduziert Autorität Fehlerzahl?				🏭, • • •
Q 3.3 Reduziert Pro-Aktivität Fehlerzahl?		🏭, • •	• •	
Q 3.4 Erhöhen heur. Werkz. Erst. Kosten?			• •	
Q 3.5 Senken heur. Werkz. Qual. Kosten?	•		• •	🏭, • • •
🏭 – Evaluation in industriellem Kontext		•	Anekdotische Evidenz	
		• •	GQM-basierte Fallstudie	
		• • •	kontrolliertes Experiment	

Q 1.3. *Sind die Nutzer in der Lage, Erfahrungen (heur. Kritiken) zu ändern oder neu anzulegen?* Nicht alle erfahrungsbasierte RE-Werkzeuge erlauben es ihren Nutzern, heuristische Kritiken und insbesondere die heuristischen Regeln zu ändern. In HeRAs Kritiksystem für Anforderungen ist dies möglich. In einem Experiment mit Studenten wurde gezeigt, dass diese in der Lage sind, auch komplexe Kritiken an ihre Bedürfnisse anzupassen oder neu zu erstellen.

Q 1.4. *Kann das erfahrungsbasierte RE-Werkzeug aus der Beobachtung von Nutzerverhalten Erfahrung ableiten?* Die Beobachtung von Nutzerverhalten ist ein entscheidendes Konzept bei HeRA.Glossar. Die Evaluation zeigt, dass

tatsächlich relevante Erfahrungen erfasst werden und die Qualität der Vorschläge steigt.

Q 1.5. *Lernt die Organisation durch den Einsatz erfahrungsbasierte Werkzeuge mehr neue Erfahrungen als ohne?* Diese Frage komplementiert die Fragen Q 1.2 und Q 1.3: Nutzer können nicht nur Erfahrung bewerten und ändern, sondern tun dies auch. Für HeRAs Argumentationskomponente für Glossare und Anforderungen sowie für KonPAss wird im Rahmen dieser Arbeit gezeigt, dass Organisationen tatsächlich mehr Erfahrung lernen.

Q 1.6. *Werden durch den Einsatz erfahrungsbasierter RE-Werkzeuge mehr Erfahrungen aus der Erfahrungsbasis der Organisation angewandt als ohne?* In allen Testumgebungen wurde untersucht, ob die Nutzer die präsentierten Erfahrungen auch verwenden. Die Ableitung von EPKs aus Use-Cases in HeRA hat tatsächlich dazu geführt, dass Lücken frühzeitig erkannt wurden. Auf Basis von Erfahrungen vorgeschlagene Glossarbegriffe wurden von Experten berücksichtigt. Das Kritiksystem für Anforderungen führte in der Untersuchung zu besseren Use-Case-Beschreibungen. KonPAss war in der Lage, kritische Fehler frühzeitig aufzudecken, die in der Realität sofort behoben würden.

Die Evaluation von Teilziel 2 *Evaluiere den Einfluss erfahrungsbasierter RE-Werkzeuge auf das individuelle Lernen* ist besonders schwierig, da der Wissensgewinn von Nutzern der Beispielimplementierungen nicht direkt messbar ist. Dieses Evaluationsziel ist daher größtenteils nur anekdotisch evaluiert.

Q 2.1. *Wird die Menge der neuen Erfahrungen durch den Einsatz eines proaktiven erfahrungsbasierten RE-Werkzeuges beim Analysten erhöht?* Durch die Beispielimplementierung HeRA.Kritik wird pro-aktiv direktes Feedback gegeben. Studenten berichten, dass sie dieses Feedback für bestimmte Qualitätseigenschaften sensibilisiert.

Q 2.2. *Wird die Menge der neuen Erfahrungen durch den Einsatz eines interpretierenden erfahrungsbasierten RE-Werkzeugs erhöht?* Die Evaluation der Beispielimplementierung HeRA.Kritik hat zudem ergeben, dass die Interpretation durch das Werkzeug wichtig ist.

Q 2.3. *Erlaubt es die Fähigkeit eines erfahrungsbasierten RE-Werkzeugs zum Perspektivwechsel dem Nutzer mehr Erfahrung anzuwenden?* Die Beispielimplementierung HeRA.EPK erlaubt es, von der funktionalen Benutzerziel-Perspektive in eine globale Geschäftsprozess-Perspektive zu wechseln. Nutzer berichten, dass sie nach der Verwendung die Konsistenz von Use-Cases besser einschätzen können.

Q 2.4. *Nimmt der Nutzer je nach Autorität des erfahrungsbasierten Werkzeugs, seiner eigenen Erfahrung und seines Typs kritisches Feedback des Werk-*

zeugs überhaupt an? Die Evaluation zeigt hier ein sehr heterogenes Bild. Die Beispielimplementierung HeRA.Glossar hat eine sehr niedrige Autorität, die meisten Glossarvorschläge sind offensichtlich irrelevant (sogenannte Stopwords). Nutzer berichten, dass sie durch die Verwendung dennoch ein besseres Gespür für wichtige Begriffe entwickelt haben. Bei KonPAss war die hohe Autorität hingegen Voraussetzung für die Akzeptanz der Nutzer. Neben Erfahrung und Typ des Nutzers und Autorität des Werkzeugs spielt vor allem auch der Stellenwert im Prozess eine Rolle.

Trotz der schwachen Evaluation der individuellen Lerneffekte darf angenommen werden, dass auch der persönliche Erfahrungsschatz der Nutzer erfahrungsbasierter Werkzeuge profitiert. Als Indikator dafür darf auch gewertet werden, dass Anforderungsdokumente bei Verwendung der Beispielimplementierungen bessere Qualität erreichen als ohne erfahrungsbasierte Unterstützung – schließlich sind immer noch die Nutzer für das finale Dokument verantwortlich. Für das Teilziel 3 sind die folgenden GQM-Fragen evaluiert worden.

Q 3.1. *Gibt es einen Zusammenhang zwischen messbaren, formalen Qualitätseigenschaften und inhaltlichen Problemen?* Dieser Zusammenhang ist bereits in Abschnitt 4.3 gezeigt worden und wird in Abschnitt 8.1.3 in Bezug auf diese Frage gesetzt: Formale Kriterien hängen mit inhaltlichen Problemen zusammen, Projekte mit vielen formalen Fehlern scheitern öfter.

Q 3.2. *Führt der Einsatz erfahrungsbasierter RE-Werkzeuge mit hoher Autorität dazu, dass Anforderungsdokumente weniger Fehler enthalten?* Durch die Beispielimplementierung KonPAss wurden kritische Fehler gefunden. Da KonPAss eine hohe Autorität besitzt, ist davon auszugehen, dass diese Fehler auch behoben würden.

Q 3.3. *Führt der Einsatz erfahrungsbasierter RE-Werkzeuge mit hoher Pro-Aktivität dazu, dass Anforderungsdokumente weniger Fehler enthalten?* Direktes, pro-aktives Feedback erlaubt es, Zusammenhänge schon bei der Erstellung eines Anforderungsdokuments besser zu verstehen und so Details der Dokumentation besser zu lösen, als dies mit nachträglicher Analyse möglich ist. So werden Glossare, die mit HeRA.Glossar bei der Dokumentation von Anforderungen erstellt werden, besser, als nachträglich erstellte Glossare. Zudem ist auch die Verwendung der Begriffe in den Dokumenten konsistenter. Der überwiegende Teil der Beispielimplementierungen versucht, Feedback so früh wie möglich zu geben - direkt oder zumindest in sehr kurzen Feedback-Schleifen (siehe Kapitel 3). Dies zahlt sich laut der Evaluation hier aus.

Q 3.4. *Erhöhen sich durch den Einsatz erfahrungsbasierter RE-Werkzeuge die Erstellungskosten von Anforderungsdokumentation?* Diese Frage wurde bei HeRAs Kritiksystem für Anforderungen evaluiert. Nutzer gaben an, dass sie

nicht oder nur geringfügig mehr Zeit für die Erstellung der Anforderungsdokumente benötigten.

Q 3.5. *Sinken die Kosten für Qualitätssicherung durch den Einsatz erfahrungsbasierter RE-Werkzeuge?* Die Beispielimplementierung KonPAss erreicht im Laborversuch bei der Konsistenzprüfung von Anforderungsdokumenten bessere Resultate als menschliche Reviewer. Durch die geringeren Reviewkosten ist damit im Kontext der Daimler AG eine Qualitätssteigerung realisierbar.

Die folgenden Abschnitte fassen jeweils ein Evaluationsszenario und eine Beispielimplementierung in eine Testumgebung zusammen und geben detaillierte Antworten zu den GQM-Fragen. Zuvor wird dieses Vorgehen an Hand von Frage Q 3.1 demonstriert. Dabei werden die Ergebnisse aus Abschnitt 4.3 in die Ziele dieser Arbeit (vergleiche Abschnitt 6.2) eingeordnet.

8.1.3 Beispiel: Evaluation von Frage Q 3.1

Der Zusammenhang zwischen formalen Indikatoren und formalen und inhaltlichen Problemen (Frage Q 3.1) wurde bereits in Kapitel 4 diskutiert. Dies ist eine wichtige Motivation und Voraussetzung für diese Arbeit. Abbildung 8.1 zeigt, wie sich die Ergebnisse aus Kapitel 4 in die Ziele der Arbeit einfügen. Zu beachten ist, dass in Kapitel 4 ebenfalls nach der GQM-Methode vorgegangen worden ist. Hier wird jedoch auf abstrakterer Ebene beschrieben, wie die Ergebnisse die Ziele der Arbeit unterstützen.

In Abbildung 8.1 schließt sich an die Frage 3.1 ein neuer GQM-Baum an. Dieser entspricht weitgehend dem GQM-Baum für das Experiment in Abschnitt 4.3. Die Fragen führen zu den drei untersuchten Hypothesen:

Hypothese 1: Projekte mit hoher gemessener Qualität haben mit größerer Wahrscheinlichkeit Erfolg (Kategorie ++).

Hypothese 2: Projekte mit sehr niedriger gemessener Qualität in der Anforderungsspezifikation haben eine größere Chance auf einen Projektfehlschlag (Kategorie +, -, oder --).

Hypothese 3: Es besteht ein statistisch relevanter Zusammenhang (Korrelation) zwischen der Qualität der Anforderungsspezifikation und dem Projekterfolg.

Die Metriken M X.1-3 in Abbildung 8.1 stehen stellvertretend für die in Abschnitt 4.3 verwendeten Metriken.

Zu beachten ist, dass in Kapitel 4 die inhaltliche und formale Qualität der Anforderungsdokumentation untersucht und in Relation mit dem Projekterfolg gesetzt wurde. Die Beobachtung, dass der überwiegende Teil der Metriken formale Anforderungsqualität betrifft und automatisch messbar ist, ist erst in der Untersuchung festgestellt worden.

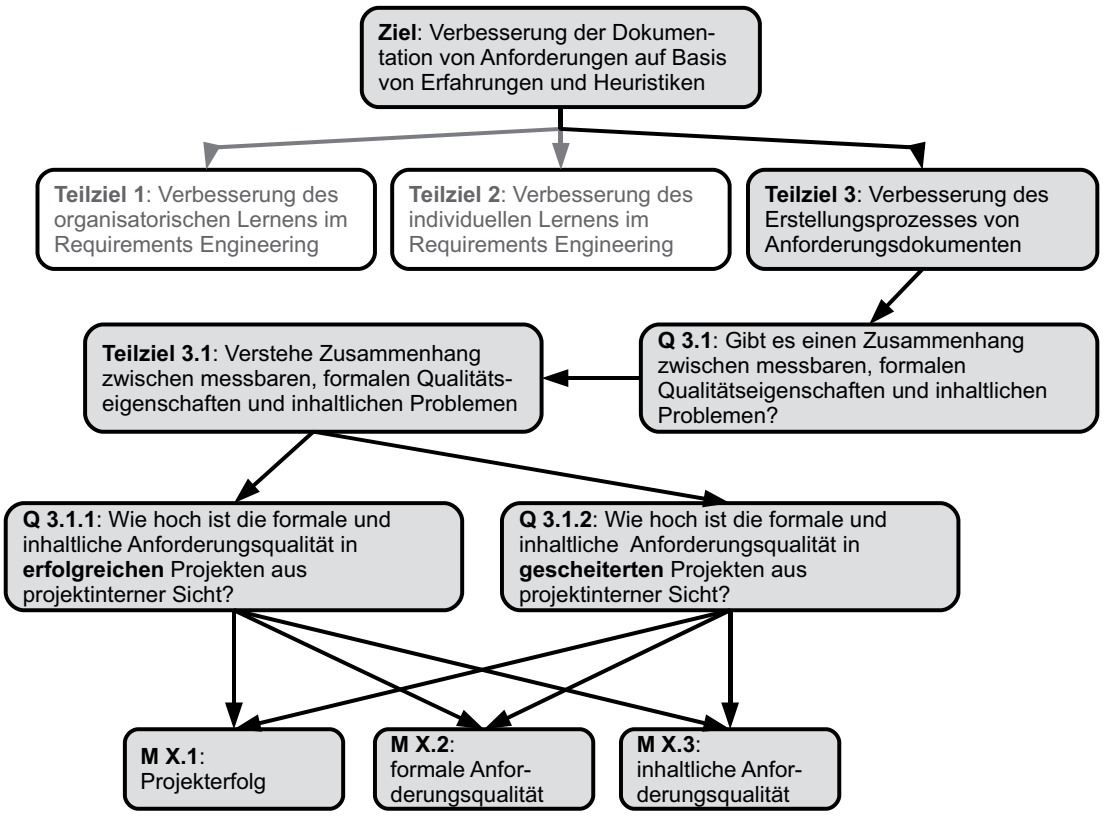


Abbildung 8.1: GQM-Baum für Frage 3.1 (vergleiche Abschnitt 4.3).

Nach dieser Beobachtung lässt sich Frage 3.1 mit *ja* beantworten. Die weiteren Fragen werden analog in den folgenden Abschnitten beantwortet.

8.2 Konstruktives heuristisches Feedback

8.2.1 Feedback durch Perspektivwechsel (HeRA.EPK)

Abbildung 8.2 zeigt die konkreten Forschungsfragen und die Metriken, die zur Evaluierung der EPK-Erweiterung von HeRA dienen. Die Evaluation der EPK-Erweiterung geschieht nur anekdotisch. Eine formalere Evaluierung wäre auf Basis des vorliegenden GQM-Baums möglich. Dazu wären zwei Gruppen nötig:

1. Gruppe 1 arbeitet mit Use-Cases *ohne* EPK-Erweiterung und Geschäftsprozessmodellen. Ziel ist es, die funktionalen Anforderungen als Use-Cases zu

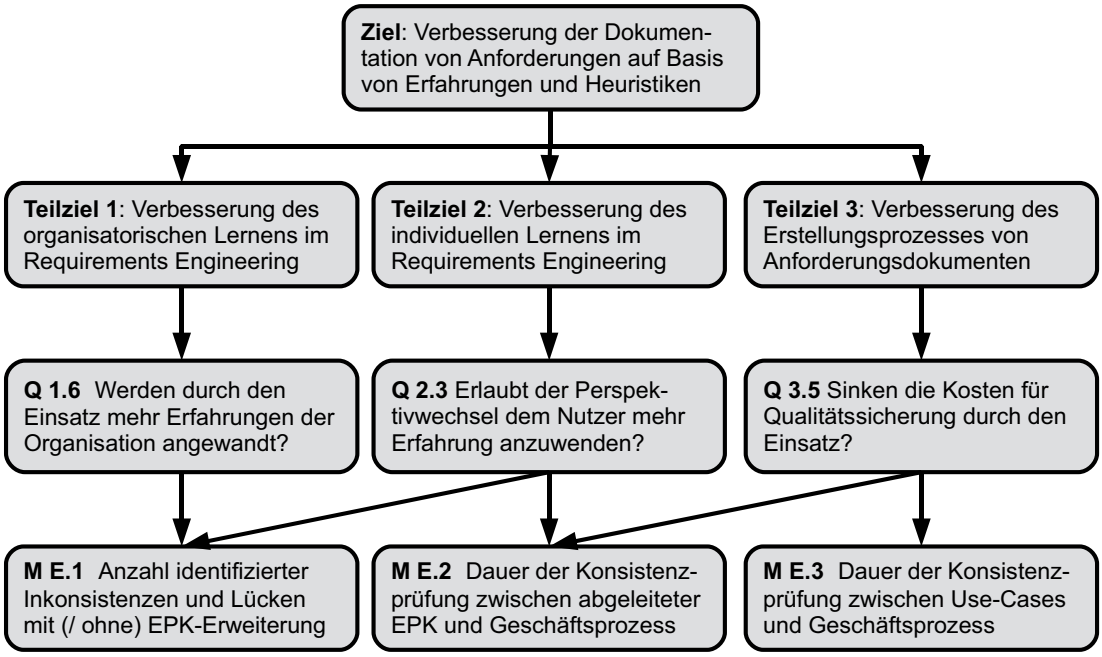


Abbildung 8.2: Der GQM-Baum zur Evaluation der EPK-Erweiterung von HeRA.

beschreiben und diese konsistent zum gewünschten Geschäftsprozess zu halten.

2. Gruppe 2 arbeitet mit Use-Cases *mit* EPK-Erweiterung und weiteren Geschäftsprozessmodellen. Ziel ist es, die funktionalen Anforderungen als Use-Cases zu beschreiben und diese konsistent zum gewünschten Geschäftsprozess zu halten.

Nach Abbildung 8.2 sind dann für beide Gruppen die Anzahl identifizierter Inkonsistenzen und Lücken in beiden Modellen und die Dauer der Konsistenzprüfung zwischen beiden Modellen zu erfassen.

Ohne diese Formalität zu erreichen, wurde HeRA mit der EPK Erweiterung in zwei Szenarien erfolgreich eingesetzt. Zum Einen wurden Use-Cases aus studentischen Softwareprojekten mit Hilfe der EPK Erweiterung analysiert. Die Erfahrung dabei zeigt, dass es erheblich einfacher wird, Inkonsistenzen und Lücken zu identifizieren. Auch der Aufwand, um die in den Use-Cases modellierte Funktionalität mit dem vom Kunden gewünschten Ablauf zu vergleichen, wird erheblich reduziert. Die Beispiele in Abschnitt 3.3.4 sind auf Basis dieser Erfahrungen entstanden.

Zum Anderen wurde die EPK-Erweiterung eingesetzt, um Anforderungen für ein Informationssystem im universitären Bereich zu erheben. Dabei wurde von Sachbearbeitern die lokale Sicht auf die gewünschte Funktionalität als Use-Cases dokumentiert. Die Möglichkeit, aus diesen lokalen Sichten einen globalen Prozess abzuleiten, hat das Projekt entscheidend vereinfacht. Inkonsistenzen zwischen den Sichtweisen der verschiedenen Sachbearbeiter wurden deutlich und konnten geklärt werden. Das abgeleitete Geschäftsprozessmodell wurde weiterverarbeitet und schließlich als explizites und zusätzliches Anforderungsartefakt gepflegt. Bei Änderungen konnte die EPK-Erweiterung genutzt werden, um schnell eine Konsistenzprüfung zwischen den Use-Cases und dem weiterverarbeiteten Geschäftsprozessmodell durchzuführen. Für den Zielbaum ergibt sich damit folgendes Resultat:

- Inkonsistenzen und Lücken lassen sich in Use-Cases durch ein abgeleitetes EPK-Modell leichter finden (Metrik M E.1). Dies deutet darauf hin, dass Erfahrungen der Organisation zu dem Zusammenhang zwischen Use-Cases und Geschäftsprozess leichter angewandt werden können (Frage Q 1.6 und Teilziel 1).
- Die Konsistenzprüfung zwischen einer Menge von Use-Cases und einem Geschäftsprozessmodell wird durch das abgeleitete EPK-Modell erheblich vereinfacht, da dieses die relevanten Eigenschaften für die Prüfung besser darstellt (Metrik M E.2 und Metrik M E.3).

In den Beispielen konnten Analysten mehr Inkonsistenzen und Lücken (Metrik M E.1) in kürzerer Zeit (Metrik M E.2) aufdecken. Dies deutet darauf hin, dass Nutzer tatsächlich mehr Erfahrungen anwenden können, wenn sie durch ein Werkzeug für den Perspektivwechsel unterstützt werden (Frage Q 2.3 und Teilziel 2).

Die Dauer für die Konsistenzsicherung zwischen den beiden konkurrierenden Anforderungsmodellen wurde in den Beispielen gesenkt (Metrik M E.2 und Metrik M E.3). Dies kann durchaus dazu führen, dass die Qualitätskosten sinken, zum Beispiel, wenn die Konsistenz nicht in einem expliziten Qualitätssicherungsprozess überprüft wird. Dann würde jede frühzeitig entdeckte Inkonsistenz oder Lücke die Qualitätskosten um die vermiedenen Fehlerkosten reduzieren. Dies unterstützt Frage Q 3.5 und Teilziel 3.

Der abgeleitete Geschäftsprozess visualisiert den globalen Kontrollfluß. Die Erfahrung in den beiden Szenarien deckt sich mit der Literatur zu abgeleiteten Kontrollflussmodellen. So beschreiben Svetinovic et al. die folgenden Vorteile für abgeleitete Kontrollflussmodelle [163]:

- Identifikation von Inkonsistenzen in der Abstraktheit der Ebenen sowie redundante Schritte in den Use-Cases.
- Identifikation falscher Reihenfolgen der Schritte in Use-Cases.
- Identifikation benötigter Funktionalität, die in keinem Use-Case enthalten ist.

- Vereinfachung der Beschreibung von Use-Cases.
- Hilfestellung für die Umstrukturierung von Use-Cases.
- Identifikation von Möglichkeiten gleichzeitiger (paralleler) Durchführung von Use-Cases.

8.2.2 Ableiten neuer Erfahrung (HeRA.Glossar)

Wie in Abschnitt 7 beschrieben wird, basiert die Glossar-Erweiterung von HeRA auf zwei zentralen Annahmen (vergleiche [120, 104]):

- **Annahme 1:** *Häufigkeit ist wichtig.* Wenn ein Begriff oft verwendet wird, ist er wahrscheinlich relevant für das Glossar, weil er oft falsch verwendet werden kann.
- **Annahme 2:** *Erfahrung ist wichtig.* Wenn ein Begriff in einem alten Projekt dem Glossar hinzugefügt wurde, sollte dieser vermutlich im aktuellen Glossar auch definiert werden.

Diese beiden Annahmen sind in der Glossar-Erweiterung als heuristische Regeln implementiert und dazu, Vorschläge für Glossar-Begriffe zu generieren.

Evaluationsszenario

Für die Evaluation wurde das Kritiksystem für Glossare auf sechs Projekte angewandt. Diese Projekte hatten zum Ziel, Software in Forschungsprojekten zu erstellen. Tabelle 8.2 zeigt, dass die Team-Größe der Projekte zwischen 3 und 8 Entwicklern und die Projektdauer zwischen 2 Wochen und 24 Monaten liegt.

Tabelle 8.2: Details der Projekte zur Evaluation von HeRA.Glossar.

	Dauer	Aufwand	Team-größe	Finan-zierung	Glossar-größe	Beschreibung
P1	24 M.	30 PM.	bis zu 8	intern	7	Desktop-App. (Editor)
P2	4 M.	1,5 PM.	3	intern	19	Web-App.
P3	20 M.	10 PM.	3	Festpreis	37	Desktop-App. (Data-Mining)
P4	2 W.	4 PM.	8	intern	7	Desktop-App. (Ress.-Planung)
P5	2 W.	4 PM.	8	intern	10	Desktop-App. (Risiko-Mgmt)
P6	3 M.	3 PM.	5	Festpreis	13	Web-App.

M. – Monat, W. – Woche, PM. – Personenmonat

Konkrete Evaluationsziele

Die Glossar-Erweiterung basiert auf zwei Techniken: Zum Einen wird eine auf die dokumentierten Anforderungen basierende Vorschlagsliste erstellt. Zum Anderen werden dem Nutzer die Top-10 Begriffe von dieser Vorschlagsliste angezeigt. Um diesen Ansatz zu evaluieren, sind die folgenden konkreten Evaluationsziele wichtig (siehe auch Abbildung 8.3):

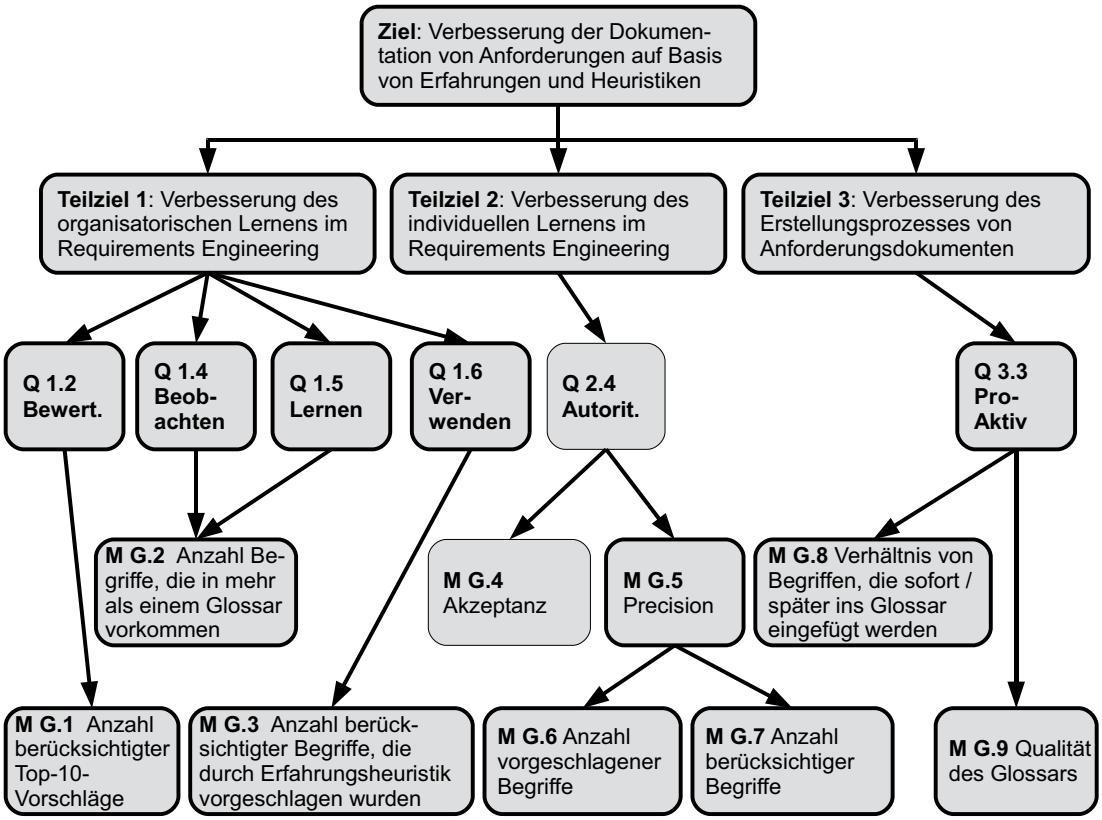


Abbildung 8.3: GQM-Baum für das Kritiksystem für Glossare.

Teilziel 1: Zeige, dass die Glossar-Erweiterung organisatorisches Lernen im Requirements Engineering unterstützt:

1. Kann eine Organisation bei Verwendung der Glossar-Erweiterung mehr neue Erfahrungen kodieren (lernen), als ohne? Diese Frage adressiert die Evaluationsfragen Q 1.2 (Bewerten von Erfahrungen), Q 1.4 (Lernen durch Beobachtung des Nutzers) und Q 1.5 (Lernen neuer Erfahrung).

M G.1 Wie oft wird ein Begriff aus den Top-10 dem Glossar hinzugefügt? Mit dieser Frage wird die Annahme 1 evaluiert: *Häufigkeit ist wichtig.*

M G.2 Wieviele Begriffe werden in mehr als einem Projekt definiert? Werden Begriffe öfter zu einem Glossar hinzugefügt, dann sind die erfahrungsbasierten Vorschläge relevant und wertvoll für die Vorschlagsliste. Mit dieser Frage wird die Annahme 2 evaluiert: *Erfahrung ist wichtig.*

2. Ist die Vorschlagsliste hilfreich, werden also durch den Einsatz der Glossar-Erweiterung mehr Erfahrungen angewandt als ohne (Frage Q 1.6)?

M G.3 Wie oft wird ein Begriff hinzugefügt, der durch die Erfahrungsheuristik vorgeschlagen wurde?

Teilziel 2: Zeige, dass die Glossar-Erweiterung individuelles Lernen im Requirements Engineering unterstützt:

1. Nehmen Nutzer der Glossar-Erweiterung Vorschläge überhaupt an, obwohl die Autorität des Werkzeugs niedrig ist (Frage Q 2.4)? Die Berücksichtigung der Vorschläge war in den untersuchten Projekten nicht vorgeschrieben und mit einer hohen Precision ist nicht zu rechnen.

M G.4 Akzeptieren die Nutzer Vorschläge und empfinden sie die Glossar-Erweiterung als hilfreich?

M G.5 Wie ist das Verhältnis der vorgeschlagenen und berücksichtigten Begriffe zu allen vorgeschlagenen Begriffen (Precision)?

Teilziel 3: Zeige, dass die Glossar-Erweiterung den Erstellungsprozess von Anforderungsdokumentation verbessert:

1. Führt die hohe Pro-Aktivität der Glossar-Erweiterung zu besseren Glossaren (Frage Q 3.3)?

M G.8 Wann fügen Analysten Begriffe zum Glossar hinzu? Ziehen sie es vor, einen Begriff während der Dokumentation von Anforderungen über die Vorschlagsliste hinzuzufügen, oder definieren sie die Begriffe später in einem separaten Arbeitsschritt manuell?

M G.9 Wie ist die Qualität des Glossars? Gehören alle enthaltenen Begriffe in ein Glossar? Sind die Begriffe gut definiert?

Evaluationsmethode

Um die Metriken M G.1, M G.3, M G.5 und M G.8 zu erfassen, wurden die Analysten in den Projekten während der Dokumentation der Anforderungen beobachtet. Immer wenn ein Begriff zum Glossar hinzugefügt wurde, wurde notiert, ob der Begriff über die Top-10-Liste oder explizit als neuer Glossar-Begriff hinzugefügt wurde. Das Ergebnis dieser Evaluation ist in Tabelle 8.3 dargestellt und wird im nächsten Abschnitt diskutiert.

Tabelle 8.3: Ergebnisse für die Metriken M G.1 und M G.8 : Der überwiegende Teil der Begriffe wird sofort (über die Top-10-Liste) eingefügt.

Projekt	*über Top-10-Liste*	*auf andere Weise*
P1	6	1
P2	17	2
P3	36	1
P4	6	1
P5	10	0
P6	13	0
Summe:	88	5

Für die Metrik M G.2 wurden die Glossare der Projekte miteinander verglichen, um Bergriffe zu finden, die in mehr als einem Projekt definiert wurden.

Die Metrik M G.4 wurde mit Hilfe einer Befragung der beteiligten Analysten erhoben.

Für die Metrik M G.9 wurden die Glossare von Experten gereviewt. Auf diese Weise wurde sichergestellt, dass keine unsinnigen Begriffe oder Erläuterungen im Glossar standen. Es zeigte sich, dass alle Begriffe gut definiert und berechtigterweise Bestandteil des Glossars waren.

Evaluationsergebnisse

Wie Tabelle 8.3 zeigt, wurden 88 von 93 Begriffen über die Top-10-Vorschlagsliste hinzugefügt. Daraus lässt sich schließen, dass die heuristischen Vorschläge gut genug sind, um nützlich zu sein.

Im ersten Projekt, das durch das Kritiksystem für Glossare unterstützt wurde, verwendeten alle Autoren die Ignorierfunktion, um Begriffe, die oft verwendet werden, aber nicht in ein Glossar gehören, auf die rote Liste zu setzen (z.B. *und, oder, ...*). Um beispielsweise die 36 Begriffe in Projekt P3 über die Top-10-Liste hinzuzufügen, mussten die Analysten zuvor 187 Begriffe auf die rote Liste setzen. Selbst dieses eher schlechte Verhältnis wurde von den Analysten nicht als Nachteil empfunden: Die irrelevanten Begriffe können leicht mit einem Klick ignoriert werden. Auf diese Weise wird der zusätzliche Aufwand schnell durch den Vorteil aufgewogen, mögliche Mehrdeutigkeiten und Missverständnisse zu entdecken. Zudem ist die rote Liste projektunabhängig, so dass zu ignorierende Wörter nur einmal auf die Liste gesetzt werden müssen. In nachfolgenden Projekten enthät die Vorschlagsliste dann viel weniger irrelevante Wörter.

Tabelle 8.4: Zusammengefasste Ergebnisse für alle Metriken aus Abbildung 8.3.

Metrik	*Beschreibung*	*Messergebnis*
M G.1	Anzahl berücksichtigter Top-10-Vorschläge	88 (von 93)
M G.2	Anzahl der Begriffe, die in mehr als einem Glossar vorkommen	6
M G.3	Anzahl berücksichtigter Begriffe, die durch die Erfahrungsheuristik vorgeschlagen wurden	3
M G.4	Akzeptanz durch die Nutzer	Die Glossar-Erweiterung wurde einstimmig als nützlich bezeichnet
M G.5	Precision der Vorschläge	Precision $= \frac{MG.7}{MG.6} = 0,19$ in P3
M G.6	Anzahl vorgeschlagener Begriffe	187 Begriffe in P3
M G.7	Anzahl berücksichtigter Begriffe	36 Begriffe in P3
M G.8	Verhältnis von Begriffen, die sofort / später ins Glossar gefügt werden	$\frac{88}{93} = 0,95$
M G.9	Qualität des Glossars	Die Glossare enthalten keine unnötigen Begriffe, alle Begriffe sind gut definiert

Drei der fünf nicht über die Top-10-Liste hinzugefügte Begriffe sind technische Begriffe, die aus mehr als einem Wort bestehen. Diese zusammengesetzten Begriffe können durch die Häufigkeitsheuristik nicht erkannt werden, werden aber in nachfolgenden Projekten von der Erfahrungsheuristik abgedeckt, nachdem sie einmal dem Glossar hinzugefügt wurden. Tauchen diese Begriffe in den Anforderungsspezifikationen späterer Projekte erneut auf, werden sie mit hoher Priorität vorgeschlagen. Durch diesen Effekt ist zu erwarten, dass das Kritiksystem in zukünftigen Projekten sogar noch nützlicher sein wird. Auf Grund der teilweise parallelen Beobachung in den Projekten konnte dies jedoch nur 3 mal beobachtet werden.

Die Analyse der entstandenen Glossare zeigt, dass diese Begriffe enthalten, die in mehreren Projekten verwendet wurden. Beispielsweise wurde der Begriff *Projekt* in fünf der sechs Projekte verwendet, fünf weitere Begriffe kamen in mindestens zwei Projekten vor. Die Tatsache, dass wiederkehrende Begriffe gefunden wurden, zeigt, dass die Erfahrungsheuristik zurecht eine höhere Priorität als die Häufigkeitsheuristik hat, da es relevante Begriffe gibt, die immer wieder verwendet werden.

Zusätzlich zur Analyse wurden die Analysten befragt, wie nützlich sie das Kritiksystem für Glossare einschätzen. Die Analysten gaben an, dass sie einige Begriffe nicht dem Glossar zugefügt hätten, wenn diese nicht vorgeschlagen worden

wären. Sie waren sich zudem einig, dass alle Begriffe im jeweiligen Glossar die Klarheit der zugehörigen Spezifikation verbesserten. In der Regel wurden die Begriffe mit einem Klick während der Dokumentation von Anforderungen in das Glossar übernommen, danach kehrten die Analysten zur Bearbeitung der Anforderungen zurück. So entstand eine Liste von im Glossar zu definierenden Begriffen als Nebenprodukt der eigentlichen Tätigkeit (Anforderungen zu dokumentieren). In einem späteren Arbeitsschritt wurden die Begriffe dann definiert. Tabelle 8.4 fasst die Ergebnisse für alle Metriken zusammen.

Diskussion der Validität

Bevor die Auswirkungen der Evaluationsergebnisse auf die abstrakten Evaluationsziele dieser Arbeit diskutiert werden, wird die Validität und Übertragbarkeit in Anlehnung an Wohlin et al. [173] diskutiert. Die meisten Bedrohungen der Validität aus dieser Quelle betreffen formalere Experimente als die vorliegende Fallstudie. Die anderen typischen Bedenken an empirischen Untersuchungen hinsichtlich ihrer Gültigkeit werden hier diskutiert:

Niedrige statistische Aussagekraft. Mit nur sechs untersuchten Projekten können mit der vorliegenden Fragestellung keine statistisch signifikanten Ergebnisse erzeugt werden. Daher konzentriert sich die Untersuchung der Fallstudien auf ein systematisches Vorgehen nach der GQM-Methode [167] für die zentralen Aspekte. In den beobachteten Fallstudien waren die Ergebnisse sehr gut und lassen darauf schließen, dass eine größere Testmenge die Ergebnisse bestätigen würde.

Eingeengte Perspektive. Die evaluierten Fragen zielten darauf ab nachzuweisen, ob der Ansatz funktioniert oder nicht. Die beobachteten Analysten standen dem Ansatz mit positiver Neugier gegenüber und könnten daher mögliche Nachteile des Ansatzes übersehen haben. Immerhin wurde der Ansatz an echten Projekten ausprobiert, die ernstzunehmende finanzielle Auswirkungen auf die ausführende Organisation hatten. Im Falle von unzureichenden Ergebnissen wäre die Evaluation zu Gunsten des Projekts gestoppt worden.

Externe Gültigkeit. Die beobachteten Projekte fanden in einer speziellen Domäne statt: wissenschaftliche Projekte im Bereich Software Engineering. Verglichen mit Industrieprojekten sind sie zudem eher klein. Basierend auf den Ergebnissen kann keine Aussage darüber getroffen werden, ob sich in Projekten anderer Domänen genauso viele domänenspezifische (und daher wiederverwendbare) Fachbegriffe finden wie in den evaluierten Projekten. Darüber hinaus kann keine Aussage über die Skalierbarkeit des Ansatzes getroffen werden.

Zusammengefasst konnte gezeigt werden, dass das Kritiksystem gute Ergebnisse in kleinen Projekten liefert. Vorschläge, die auf Erfahrung alter Projekte basieren, haben sich als geeignet erwiesen – vor allem wenn alle Projekte in einer ähnlichen Domäne durchgeführt werden. Der Ansatz hat zu guten Glossaren geführt, zum Teil wären die Glossare ohne Werkzeugunterstützung unvollständiger ausgefallen.

Diese Evaluation zeigt also, dass die Glossar-Erweiterung organisatorisches Lernen im Requirements Engineering verbessert (Teilziel 1): Die heuristischen Kritiken der Erweiterung wurden als sehr nützlich bewertet (vergleiche Metrik M G.1). Durch das Hinzufügen eines Begriffs wird der Vorschlag nicht nur als nützlich bewertet, sondern auch automatisch als wichtiger Begriff für die Erfahrungsheuristik *gelernt*. Da in verschiedenen Glossaren die gleichen Begriffe auftauchen (Metrik M G.2), sind diese Erfahrungen relevant und wertvoll. Die Erfahrungsheuristik ist in der Lage, diese Erfahrungen wieder der Verwendung zuzuführen (Metrik M G.3).

Das individuelle Lernen (Teilziel 2) stand nicht im Fokus der Evaluation. Dennoch gibt es interessante Ergebnisse: Obwohl die Precision mit 0,19 (Metrik M G.5) teilweise sehr schlecht war und es auch keine Vorschrift zur Verwendung des Werkzeugs gab, wurde die heuristische Kritik von allen beteiligten Analysten angenommen (Metrik M G.1) und geschätzt (Metrik M G.4).

Zusammengenommen verbessert die Glossar-Erweiterung den Erstellungsprozess von Anforderungsdokumentation (Teilziel 3): Analysten schätzen die Möglichkeit, schon begleitend zu der Dokumentation von Anforderungen Begriffe in das Glossar einfügen zu können (Metrik M G.8). Dies erleichtert ihre Arbeit, ein konsistentes Anforderungsdokument zu erstellen. Die dabei entstehenden Glossare sind qualitativ gut, enthalten also keine unwichtige Begriffe (M G.9).

8.2.3 Pro-aktives Feedback (HeRA.Kritik)

Das Kritiksystem für Use-Cases wurde bereits in [98] zum Teil evaluiert. Diese Evaluation wird hier in Bezug auf die Ziele der Arbeit gesetzt und um ein Experiment erweitert, welches zeigt, dass Nutzer in der Lage sind, neue Kritiken anzulegen.

Evaluationsszenario

Um das Kritiksystem für Use-Cases zu evaluieren wurden in einem studentischen Softwareprojekt 3 von 8 Gruppen mit dem Kritiksystem ausgestattet. Die Softwareprojekte dauerten ein Semester, davon dienen die ersten drei Wochen zur Anforderungsanalyse. Jede Gruppe bestand aus fünf bis sechs Informatikstudenten, die im 5. Semester waren und die grundlegenden Vorlesungen bereits gehört hatten.

Einige Studenten hatten auch schon an weiterführenden Vorlesungen teilgenommen. Für die Fallstudie wurden die Studenten so auf die verschiedenen Gruppen eingeteilt, dass die Gruppenstärken vergleichbar waren. Die Post-Mortem-Analyse der Projekte zeigte, dass dies erfolgeich war.

Der besondere Aufbau der Lehrveranstaltung hat den Vorteil, dass alle Gruppen gleichmäßig mit Ressourcen wie Betreuung und Zeit für Kundenbefragungen ausgestattet sind. Nachteilig für die empirische Evaluation war jedoch, dass alle Gruppen unterschiedliche Aufgaben hatten. Es gab ein Projekt für klassische administrative Software, ein Projekt mit algorithmischen Schwerpunkt, drei Projekte für eingebettete Systeme und drei Projekte, die sich mit SOA-Technologie beschäftigten.

Nach der Analysephase von drei Wochen durchlaufen die Projekte ein Quality Gate. Im Rahmen der Evaluation macht es Sinn, die Qualität zu messen, bevor die Use-Cases basierend auf dem Feedback aus dem Quality Gate verbessert wurden, da sonst nicht die Effektivität des Kritiksystems sondern die Effektivität des Quality Gates gemessen würde. Damit bleibt die Frage offen, ob das Kritiksystem überflüssig ist, wenn das Projekt zusätzlich mit einem guten Quality Gate ausgestattet ist. Daher wird auf Basis der Erfahrungen des Qualitätsmanagement-Teams untersucht, welche Aspekte sich leichter konstruktiv mit Hilfe des Kritiksystems abdecken lassen und welche Aspekte durch das Quality Gate besser adressiert werden.

Konkrete Evaluationsziele

Das Kritiksystem HeRA.Kritik war das erste heuristische Werkzeug, das im Rahmen dieser Arbeit entstanden ist. Begleitend zur Arbeit wurde es eingesetzt, um die Konzepte zu entwickeln und zu evaluieren. Dies hat dazu geführt, dass sich mit dem Kritiksystem sehr viele Forschungsfragen dieser Arbeit adressieren lassen (vergleiche Tabelle 8.1). Zur besseren Übersichtlichkeit wird die Evaluation hier in zwei Teile aufgeteilt: An dieser Stelle werden alle Fragen evaluiert, die mit der Wiederverwendung von Erfahrungen einer Organisation einhergehen. Die Frage, ob Analysten in der Lage sind, ihre Erfahrungen zu kodieren und so der Organisation neue Erfahrungen zur Verfügung zu stellen, wird in Abschnitt 8.2.4 untersucht.

Abbildung 8.4 zeigt den GQM-Baum für die Wiederverwendung von Erfahrungen in HeRA.Kritik. Die zentrale Metrik ist die Qualitätssteigerung durch den Einsatz des Kritiksystems (Metrik M C.5): Werden in Projekten, in denen HeRA.Kritik eingesetzt wird, weniger Fehler in der Dokumentation von Use-Cases gefunden, deutet dies darauf hin, dass die Erfahrungen in HeRA genutzt wurden. Dies würde außerdem zeigen, dass pro-aktive heuristische Werkzeuge helfen kön-

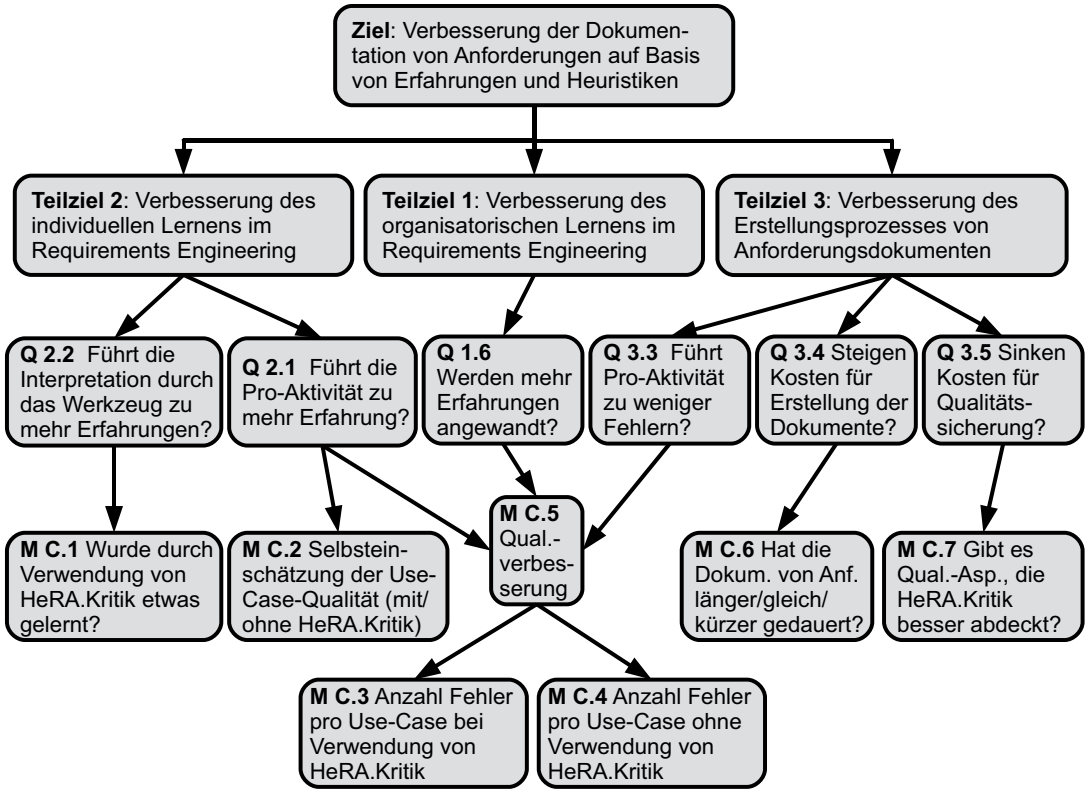


Abbildung 8.4: GQM-Baum für die Wiederverwendung von Erfahrungen in HeRA.Kritik.

nen, Fehler in Anforderungsdokumenten zu vermeiden. Indirekt würde dies auch darauf hindeuten, dass die Analysten durch die Verwendung des Kritiksystems gelernt haben, bessere Anforderungsdokumente zu erstellen, HeRA nur Feedback gibt und letztendlich die Analysten für die Behebung der Fehler verantwortlich sind.

Evaluationsmethode

Tabelle 8.5 zeigt den Messplan für die Evaluation. Die Metriken werden für die verschiedenen Teilziele dieser Arbeit wie folgt erhoben:

Teilziel 1: Organisatorisches Lernen. Im Rahmen dieser Untersuchung wird evaluiert, ob es durch den Einsatz von HeRA.Kritik gelingt, mehr Erfahrungen wiederzuverwenden, als ohne. Da in diesem Evaluationsszenario eine direkte Be-

Tabelle 8.5: Messplan für die Evaluation von HeRA.Kritik.

Metrik	*Beschreibung*	*Art der Messung*
M C.1	Wurde durch Verwendung von HeRA.Kritik etwas gelernt?	Fragebogen nach der Analysephase an Nutzer des Kritiksystems
M C.2	Selbsteinschätzung der Use-Case-Qualität (mit / ohne HeRA.Kritik)	Fragebogen nach der Analysephase an alle Teilnehmer der Studie
M C.3	Anzahl Fehler pro Use-Case bei Verwendung von HeRA.Kritik	Experten-Review der Abgabe für das Quality Gate
M C.4	Anzahl Fehler pro Use-Case ohne Verwendung von HeRA.Kritik	Experten-Review der Abgabe für das Quality Gate
M C.5	Qualitätsverbesserung durch HeRA.Kritik	Verhältnis von M C.3 zu M C.4
M C.6	Hat die Anforderungsdokum. länger / gleich/ kürzer gedauert?	Fragebogen nach der Analysephase an Nutzer des Kritiksystems
M C.7	Gibt es Qualitäts-Aspekte, die HeRA.Kritik besser abdeckt?	Experten-Analyse nach Ende des Projekts

obachtung der Analysten bei der Arbeit nicht möglich ist, wird dies indirekt über die Qualitätssteigerung durch die Verwendung des Kritiksystems gemessen (Metrik M C.5).

Teilziel 2: Individuelles Lernen. In Kapitel 5.2.1 wird beschrieben, wie heuristisches Feedback beim individuellem Lernen helfen kann. Zu evaluieren ist also, ob diese Effekte in der Praxis tatsächlich sichtbar werden. Zu diesem Zweck wird die Qualitätssteigerung durch die Verwendung des Kritiksystems gemessen (Metrik M C.5). Kommt dabei eine Verbesserung der Qualität heraus, kann daraus abgeleitet werden, dass die Studenten ihre Use-Cases basierend auf dem Feedback selbstständig verbessert haben. Sie haben also gelernt, bessere Use-Cases zu schreiben. Darüberhinaus wurden die Teilnehmer nach dem Projekt interviewt. Dabei wurden sie danach gefragt, ob sie durch die Verwendung des Kritiksystems etwas gelernt haben (Metrik M C.1) und wie groß ihr Vertrauen in die Qualität der erstellten Use-Cases ist (Metrik M C.2).

Teilziel 3: Verbesserung der Dokumentation. Ziel des Kritiksystems ist es, auch weniger erfahrene Nutzer bei der Erstellung von Use-Cases zu unterstützen. Zentrale Metrik ist auch hier die Qualitätssteigerung durch die Verwendung des Kritiksystems (Metrik M C.5). Da für die Erstellung von Use-Cases typischer-

weise nicht viel Zeit veranschlagt wird, soll die Verwendung des Kritiksystems die Arbeitsgeschwindigkeit nicht reduzieren. Dies wird mit Hilfe eines Fragebogens überprüft (Metrik M C.6). Konkret ist im Rahmen der Evaluation also zu zeigen, dass Nutzer mit Hilfe des Kritiksystems bessere Use-Cases erstellen und dabei nicht (deutlich) mehr Zeit benötigen, als ohne Kritiksystem. Schließlich soll gezeigt werden, ob das konstruktive Feedback durch heuristische Kritiken dabei hilft, Qualitätskosten zu senken. Zu diesem Zweck werden die untersuchten Projekte Post-Mortem durch das Qualitätsmanagement analysiert. Dabei werden gängige Qualitäts-Aspekte für Anforderungsdokumente untersucht, in wie weit sie durch ein konstruktives Kritiksystem überprüft werden können, um das Quality Gate zu entlasten (Metrik M C.7).

Diskussion der Validität

Die Diskussion der Validität ist ein wesentlicher Bestandteil einer empirischen Studie. Die folgenden Validitäts-Aspekte nach Wohlin et al. und Easterbrook et al. wurden im Rahmen dieser Studie betrachtet [173, 55]:

Interne Validität. Die interne Validität betrifft die Frage, ob die Ergebnisse wirklich aus den Daten folgen, oder ob es Scheinkorrelationen durch unkontrollierte Variablen geben kann [55]. Die interne Validität ist bei allen empirischen Untersuchungen besonders wichtig, weil sie die zu untersuchenden Zusammenhänge betrifft [173].

Bei den Experimentgruppen, die mit HeRA.Kritik gearbeitet haben, sind hier vor allem Probleme mit HeRA zu nennen. Die eingesetzte Version von HeRA war nicht ohne Bugs und auch die Integration in den Entwicklungsprozess hätte besser sein können. Die Notwendigkeit, Use-Cases in HeRA zu bearbeiten und für die Abgabe in Textdokumente einzufügen, hat die Experimentiergruppen benachteiligt.

Ein Problem ist auch, dass sich Experimentgruppen und Kontrollgruppen zum Teil ausgetauscht haben. Auf diese Weise haben zwei Kontrollgruppen indirekt von den Erfahrungen in HeRA profitiert.

Beide Effekte sollten sich nachteilig auf die Ergebnisse der Experimentgruppen auswirken. Das hier dennoch bessere Use-Cases zu beobachten sind, ist ein Indikator dafür, dass die Schlussfolgerungen valide sind. Zudem wurde ja gerade das direkte Feedback durch HeRA.Kritik evaluiert, so dass die indirekte Beeinflussung der Testgruppe nicht so schwer wiegt.

Validität des Aufbaus. Die Validität des Aufbaus beschreibt, ob die theoretischen Konstrukte korrekt abgebildet und gemessen sind [55].

Ein Problem in diesem Zusammenhang kann sein, dass in die Messung der Qualität der Use-Cases das gleiche Verständnis von Anforderungsqualität eingeflossen ist, wie in die Erstellung der heuristischen Kritiken. Dies ist zunächst eine starke Bedrohung der Validität der Untersuchung, die durch zwei Aspekte abgeschwächt wird: Zum Einen wurde die in der Untersuchung gemessene Qualität von einem weiteren Reviewer mitgetragen. Zum Anderen steckt hinter den zu untersuchenden Konstrukten auch die Frage, ob es möglich ist, Erfahrungen zur Anforderungsqualität so zu kodieren, dass sie im Projekt nützlich sind.

Validität der Schlussfolgerung. Easterbrook et al. umschreiben diesen Validitätsbegriff auch mit Glaubwürdigkeit der Untersuchung. Im Kern geht es darum, ob die Untersuchung die selben Ergebnisse liefern wird, wenn sie von anderen Forschern wiederholt wird [55]. Laut Wohlin et al. ist die Validität der Schlussfolgerung beim Testen von Theorien nicht so wichtig wie die interne Validität und die Validität des Aufbaus [173]. Demnach wird eine Studie durch statistisch signifikante Ergebnisse nur dann gestärkt, wenn der Aufbau prinzipiell in Ordnung ist.

Die Hauptprobleme für valide Schlussfolgerungen sind hier die statistische Aussagekraft, die bei der Anzahl der untersuchten Projekte nicht gegeben ist, und der Fakt, dass die Untersuchung in der gleichen Forschergruppe durchgeführt wurde, in der HeRA.Kritik auch entwickelt worden ist. Diese Einschränkungen der Validität wurden zu Gunsten eines guten Aufbaus und interner Validität in Kauf genommen.

Externe Validität. Externe Validität betrifft die Frage, ob die Ergebnisse einer Studie verallgemeinert werden können und ist laut Wohlin et al. bei der Überprüfung von Theorien der unwichtigste Validitätsbegriff [173]. Im Rahmen dieser Untersuchung betrifft das vor allem die Frage, ob von studentischen Projekten auf Effekte in industriellen Projekten geschlossen werden kann. Im Kontext der weiteren Studien in diesem Kapitel besteht durchaus Grund zu der Annahme, dass direktes und pro-aktives Feedback auch in der Industrie zu guten Effekten bei der Dokumentation von Anforderungen führt. Es ist anzunehmen, dass zumindest solche Analysten von HeRA profitieren, die genau wie die Studenten in dieser Studie im Umgang mit Use-Cases unerfahren sind.

Evaluationsergebnisse

Tabelle 8.6 zeigt die Ergebnisse der Evaluation, die in diesem Abschnitt diskutiert werden.

Die Verwendung des Kritiksystems war für die drei Projektgruppen freiwillig. In einer Gruppe wurden die heuristischen Kritiken als sehr störend empfunden, so

Tabelle 8.6: Ergebnisse der Metriken für HeRA.Kritik.

Metrik	*Beschreibung*	*Ergebnis der Messung*
M C.1	Wurde durch Verwendung von HeRA.Kritik etwas gelernt?	ja
M C.2	Selbsteinschätzung der Use-Case-Qualität (mit / ohne HeRA.Kritik)	Nutzer des Kritiksystems haben mehr Vertrauen in ihre Use-Cases
M C.3	Anzahl Fehler pro Use-Case bei Verwendung von HeRA.Kritik	siehe Abbildung 8.5
M C.4	Anzahl Fehler pro Use-Case ohne Verwendung von HeRA.Kritik	siehe Abbildung 8.5
M C.5	Qualitätsverbesserung durch HeRA.Kritik	Verbesserung bei Verwendung des Kritiksystems (vgl. Abb. 8.5)
M C.6	Hat die Anforderungsdokum. länger/gleich/kürzer gedauert?	leichte Vorteile bei der Schreibgeschwindigkeit bei Verwendung des Kritiksystems
M C.7	Gibt es Qualitäts-Aspekte, die HeRA.Kritik besser abdeckt?	Kritiksystem ist eine gute Ergänzung analytischer Methoden des Qualitätsmanagements

dass die Gruppe die Verwendung des Kritiksystems einstellte. Dies liegt auch daran, dass die Gruppe ein Projekt für eingebettete Software hatte und das Kritiksystem für diesen speziellen Fokus nicht so gut ausgelegt war. Rückblickend hat sich gezeigt, dass sich die Use-Cases aller drei Projekte mit eingebetteter Software deutlich von denen der anderen Projekte abhoben. Dieser Effekt wird in [93] genauer beschrieben. Aus diesem Grund wurden alle drei Projekte mit diesem Hintergrund aus der Wertung genommen.

Eine zweite Gruppe (im Folgenden als Projekt 2 bezeichnet) hat das Kritiksystem vor allem am Anfang verwendet, aber die Verwendung noch vor dem Ende der Anforderungsanalyse eingestellt. Das lag daran, dass die Gruppe sich entschlossen hatte, Telelogic Doors als Anforderungsmanagementwerkzeug zu verwenden, nachdem alle Use-Cases geschrieben worden waren. Die Gruppe hatte zunehmend Schwierigkeiten die Anforderungen in beiden Werkzeugen synchron zu halten und wechselte daher die Werkzeuge. Die Verwendung von Doors war nicht erwartet worden, deshalb wurde keine ausreichende Interoperatibilität zwischen HeRA und Doors bereitgestellt. Die Daten dieser Gruppe waren aber für die Evaluation durchaus brauchbar.

Die dritte Gruppe (Projekt 1) schließlich hat sehr positives Feedback gegeben. Die Gruppe hat HeRA mit dem Kritiksystem wie vorgesehen verwendet. Die Use-Cases wurden zunächst mit HeRA geschrieben und dann in ein Schreibprogramm

exportiert und in die restliche Spezifikation integriert. Waren Änderungen an den Use-Cases erforderlich, wurden diese in HeRA mit Hilfe des Kritiksystems durchgeführt und dann erneut exportiert, wodurch der vorherige Export überschrieben wurde. Unglücklicherweise arbeitete diese Gruppe sehr eng mit zwei anderen Gruppen zusammen (Projekt 4 und 5). Einige Anforderungen waren auch für die anderen Gruppen relevant. Durch den intensiven Austausch profitierten nach eigenen Angaben auch die anderen beiden Gruppen von den Erfahrungen in dem Kritiksystem.

Trotz dieser Schwierigkeiten können einige qualitative Aussagen darüber getroffen werden, ob das Kritiksystem zu besseren Use-Cases führt. Nach der Fertigstellung der Anforderungsspezifikationen wurden die Use-Cases der Gruppen nach folgenden Gesichtspunkten analysiert (betrifft Metrik M C.3 und M C.4):

1. Korrekte Verwendung der Felder des Use-Case-Vorlage (zum Beispiel wurden Erweiterungen im Hauptszenario oder als technische Variation beschrieben, statt der Abstraktionsebene des Use-Cases wurden die Systemgrenzen angegeben).
2. Bei jeder Aktion im Use-Case muss klar sein, welcher Akteur dafür verantwortlich ist.
3. Es muss definiert sein, wie die Use-Cases zusammen hängen.
4. Jedes Feld muss lesbar und verständlich sein (teilweise durch schlechte Grammatik oder fehlende Information nicht gegeben).

Aus Abbildung 8.5 lassen sich die Ergebnisse für die Metriken M C.3, M C.4 und M C.5 wie folgt ableiten: Die Projekte 1 und 2 haben von dem Kritiksystem profitiert. Die Use-Cases der drei Projekte mit eingebetteter Software wurden nicht gewertet, weil sie in allen drei Projekten sehr schlechte Qualität hatten und keine der Gruppen das Kitiksystem verwendet hatte. Abbildung 8.5 zeigt einen positiven Effekt bei den Projekten, die das Kritiksystem verwendet haben bezüglich der Befunde pro Use-Case. Der Umstand, dass immer noch Befunde enthalten sind, zeigt, dass einige der Kritiken ignoriert oder ausgetrickst wurden. Daraus lässt sich folgern, dass das Kritiksystem keine analytische Qualitätssicherung ersetzen kann.

Die Befragung der Studenten ergab, dass Nutzer des Kritiksystems ein größeres Vertrauen in die Qualität ihrer Use-Cases hatten (Metrik M C.2). Abgesehen von dem Aufwand, Anforderungen in verschiedenen Werkzeugen synchron zu halten, wurde die Verwendung des Kritiksystems nicht als bremsend empfunden (Metrik M C.6). Im Gegenteil, einige Studenten berichteten sogar, dass die Erstellung von Use-Cases mit dem Kritiksystem schneller geht, da keine Zeit für die Suche nach Informationen, wie man einen Use-Case schreibt, verloren geht.

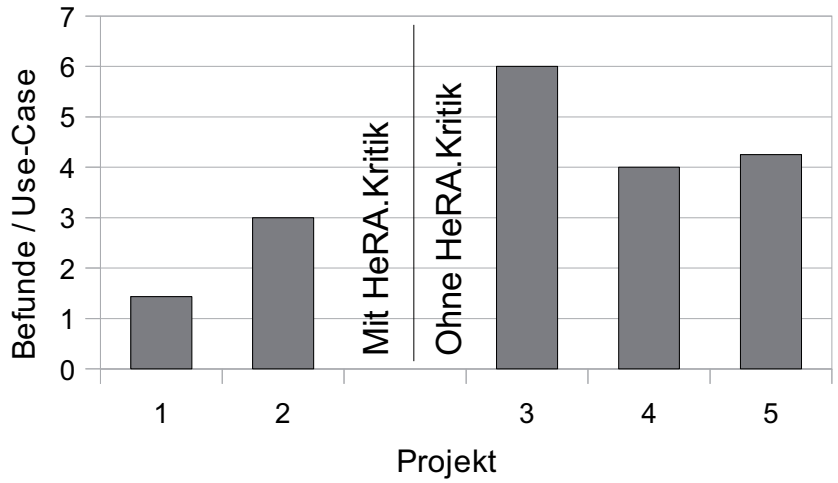


Abbildung 8.5: Befunde (Anzahl Fehler) pro Use-Case bei einem Review nach Abschluss der Anforderungsanalyse. Für Metrik M C.5 lässt sich eine Verbesserung der Qualität bei Verwendung des Kritiksystems feststellen.

Sowohl eine Befragung der teilnehmenden Studenten, als auch die Kommentare während der Post-Mortem-Analyse haben ergeben, dass die Studenten durch die heuristischen Kritiken dazu gelernt haben. Die Tatsache, dass Studenten mit Hilfe des Kritiksystems bessere Use-Cases erstellt haben (Metrik M C.5) ist ein weiterer Indikator dafür. Interessant ist darüber hinaus, dass sich die Projektgruppen 1, 4 und 5 über die durch die heuristischen Kritiken abgedeckten Erfahrungen ausgetauscht haben – ohne dass das Kritiksystem weiter verteilt wurde. Offensichtlich haben die Studenten aus Projekt 1 wichtige Lehren aus der Verwendung des Kritiksystems gezogen, die sie dann an ihre Kommilitonen weiter gegeben haben. Dies sind gute Argumente dafür, dass direktes heuristisches Feedback individuelles Lernen unterstützt.

Abgrenzung zu analytischen Verfahren der Qualitätssicherung. Für die Frage, wie sich konstruktives Feedback mit dem Kritiksystem von analytischen Verfahren der Qualitätssicherung abhebt, wurden die überprüften Qualitätsaspekte nach Robertson und Robertson verglichen [139]. Tabelle 8.7 zeigt, wie die Qualitätsmanager der untersuchten Projekte das Kritiksystem im Vergleich zu dem ebenfalls durchgeführten Quality Gate beurteilen. Die ersten beiden Spalten geben den Qualitätsaspekt aus [139] und eine kurze Erläuterung an. Die dritte Spalte gibt an, wie gut die Qualitätsmanager diesen Aspekt durch das Kritiksystem konstruktiv abgedeckt sehen. Die vierte Spalte gibt an, wie geeignet die Quali-

Tabelle 8.7: Abdeckung von Qualitätsaspekten für Anforderungen des Kritiksystems im Vergleich zu einem Quality Gate.

Qualitätsaspekt	*Beschreibung*	*Kritiksystem*	*Quality Gate*
Vollständigkeit	Sind alle Teile der Anforderung beschrieben?	••	
Verständlichkeit	Kann die Anforderung von einem gegebenen Stakeholder verstanden werden?	•	••
Verfolgbarkeit	Kann die Anforderung zu ihrem Ursprung zurückverfolgt werden?	•	•
Konsistente Begriffe	Verwendet die Spezifikation Namenskonventionen und ein definiertes Vokabular?	•	•
Relevanz	Ist die Anforderung relevant für den Zweck des Produkts?	•	••
Mehrdeutigkeit	Ist es von einem objektiven Standpunkt her möglich festzustellen, ob eine Anforderung erfüllt oder nicht erfüllt ist?	•	•
Umsetzbar	Ist die Anforderung unter Berücksichtigung der Rahmenbedingungen und Einschränkungen des Projekts umsetzbar?		••
Lösungsgebunden	Ist die Anforderung frei von möglichen Lösungen?	•	•
Kundennutzen	Enthält die Anforderung eine Bewertung ihres Wertes für den Kunden?	•	•
Gold-Plating	Gibt es Anforderungen, die nicht zum Projektziel beitragen?	•	••
Requirements-Creeping	Existieren Mechanismen, um Anforderungen systematisch hinzuzufügen oder zu ändern, nachdem die Anforderungsanalyse abgeschlossen ist?		••
Requirements-Leakage	Existieren Mechanismen, die systematisch verhindern, dass Anforderungen ohne einen verfolgbaren Ursprung hinzugefügt werden, nachdem die Anforderungsanalyse abgeschlossen ist?	•	••
••	Wird hier geprüft		
•	Wird teilweise hier geprüft		
	Wird hier nicht geprüft		

tätsmanager das Quality Gate einstufen, diesen Qualitätsaspekt zu adressieren. Tabelle 8.8 gibt Beispiele, wie das Kritiksystem die gegebenen Qualitätsaspekte unterstützt.

Tabelle 8.8: Strategien zur Abdeckung von Qualitätsaspekten für Anforderungen durch heuristische Kritiken.

Qualitätsaspekt	*Heuristisches Feedback*
Vollständigkeit	Warnung, wenn Teile der Vorlage leer sind oder die Beschreibung zu kurz ist.
Verständlichkeit	Warnung, wenn ein Satz zu lang ist, eine Aufzählung zu viele Elemente hat oder bestimmte Schlüsselwörter auftreten (z.B. oft, ungefähr).
Verfolgbarkeit	Warnung, wenn ein Use-Case a) sich nicht auf der höchsten Abstraktionsebene befindet und b) nicht mit einem Use-Case auf höherer Abstraktionsebene verlinkt ist.
Konsistente Begriffe	Unterstützung durch HeRA.Glossar
Relevanz	siehe Verfolgbarkeit
Mehrdeutigkeit	Warnung, wenn bestimmte Schlüsselwörter auftreten (z.B. ungefähr, oft, immer, alle).
Umsetzbar	HeRA. Warnung, wenn Use-Case-Punkte eine bestimmte Grenze überschreiten.
Lösungsungebunden	Warnung, wenn bestimmte Schlüsselwörter auftreten (z.B. button, linux, WLAN, Bluetooth).
Kundennutzen	Warnung, wenn keine Priorität gesetzt wurde oder die Priorität niedriger ist, als die von referenzierten Anforderungen.
Gold-Plating	siehe Verfolgbarkeit
Requirements-Creeping	
Requirements-Leakage	Anzeige der Entwicklung der Use-Case-Punkte über die Zeit.

Insgesamt zeigt sich, dass das Kritiksystem in beinahe allen Punkten das Quality Gate ergänzen kann und so wertvolle Unterstützung besonders in den Punkten liefert, in denen die Qualitätssicherung im Quality Gate schwerfällt. Somit ist das Kritiksystem aus Sicht der Qualitätsmanager als gute Ergänzung zu sehen. Die Tabelle 8.7 liefert somit die Abgrenzung direkter heuristischer Kritiksysteme zu analytischen Verfahren der Qualitätssicherung.

8.2.4 Kodierung von Erfahrung (HeRA.Kritik)

Nachdem im letzten Abschnitt 8.2.3 beschrieben worden ist, wie die Erfahrungen einer Organisation zur besseren Dokumentation von Anforderungen eingesetzt werden können, wird in diesem Abschnitt untersucht, wie neue Erfahrungen kodiert werden können.

Evaluationsszenario

HeRA wird zur Zeit überwiegend von Studenten am Ende ihres Bachelor- oder während ihres Masterstudiums in Projekten der universitären Lehre eingesetzt. Es liegt Nahe, diese Zielgruppe für die Evaluation der Kodierung heuristischer Kritiken heranzuziehen, auch, weil sich aus dieser Gruppe die Analysten von Morgen rekrutieren.

In dieser Testumgebung soll untersucht werden, ob eine repräsentative Auswahl von Studenten der Zielgruppe in der Lage ist, unter kontrollierten Bedingungen in einem Laborversuch bestimmte Aufgaben zu lösen. Daher wurde in einschlägigen Lehrveranstaltungen nach Freiwilligen für ein Experiment gesucht:

1. Grundlagen der Softwaretechnik: Laut Regelstudienverlauf wird diese Lehrveranstaltung im 3. Semester des Bachelorstudiums besucht.
2. Softwareprojekt: Dieses studentische Projekt wurde im Rahmen dieser Arbeit bereits mehrfach beschrieben und findet laut Regelstudienverlauf im 5. Semester des Bachelorstudiums statt.
3. Labore, Seminare und fortgeschrittene Vorlesungen aus dem Wahlbereich des Masterstudiums.

Dem Aufruf sind 7 Studenten gefolgt, nur zwei der Freiwilligen befanden sich zum Zeitpunkt der Untersuchung noch in ihrem Bachelorstudium (5. und 9. Semester).

Konkrete Evaluationsziele

Die Frage, ob eine Organisation durch den Einsatz des Kritiksystems mehr Erfahrungen für ihre Erfahrungsbasis generieren kann, hat zwei Facetten: Zum Einen die Fähigkeit und Möglichkeit der Nutzer, neue Erfahrungen einzugeben, zum Anderen die Motivation der Nutzer, diese Fähigkeiten und Möglichkeiten zu nutzen. Dadurch ergeben sich für diese Evaluation die folgenden drei Fragen:

1. Haben die Nutzer von HeRA die Möglichkeit, ihre Bewertung zu einzelnen heuristischen Kritiken abzugeben (Frage Q 1.2)?

2. Können die Nutzer heuristische Kritiken ändern oder neu anlegen und so ihre Erfahrungen geeignet kodieren (Frage Q 1.3)?
3. Sind Nutzer bereit, Erfahrungen (Bewertungen von heuristischen Kritiken, Änderungen an heuristischen Kritiken, neue heuristische Kritiken) in das System einzugeben (Frage Q 1.5)?

Abbildung 8.6 zeigt den entsprechenden Ausschnitt des GQM-Baums und ordnet den Fragen Metriken zu.

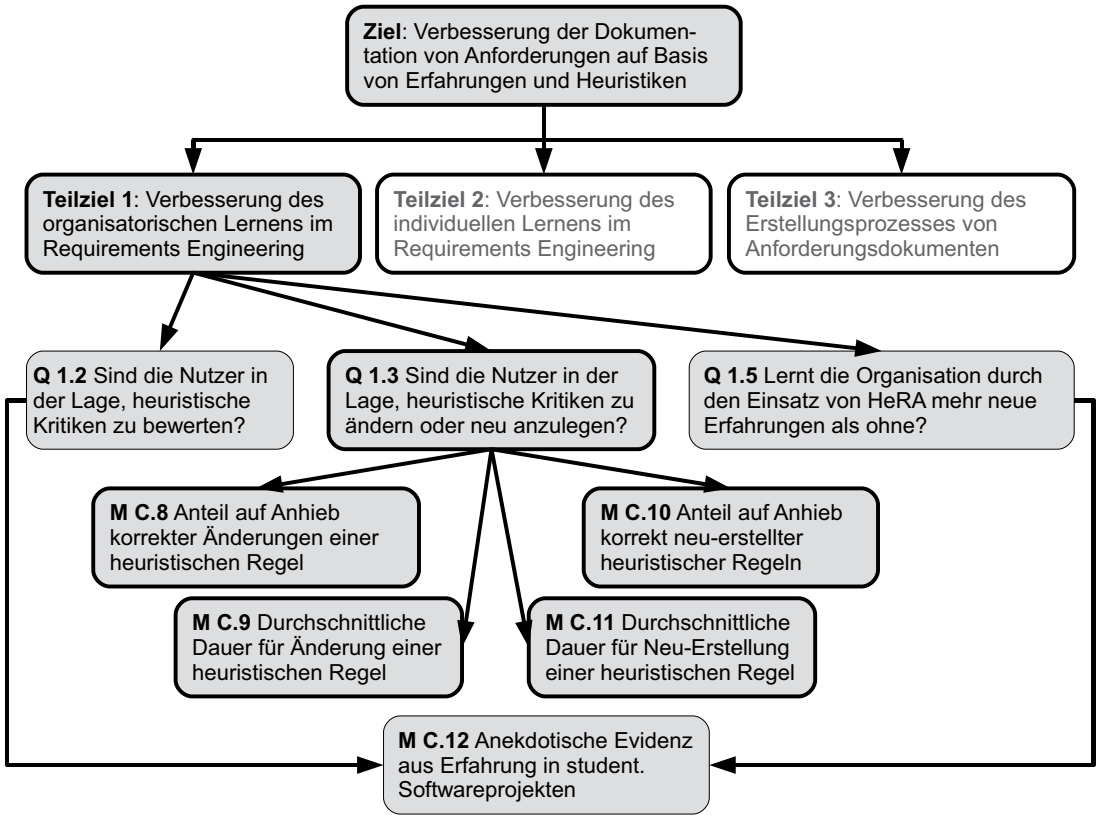


Abbildung 8.6: GQM-Baum für die Kodierung neuer Erfahrungen in HeRA.Kritik.

Dieser Abschnitt beschäftigt sich daher hauptsächlich mit Frage Q 1.3: Ist die Zielgruppe von HeRA in der Lage, heuristische Kritiken zu ändern oder neu anzulegen. Als Ausgangshypothese sollten mindestens 75 % der Studenten in der Lage sein, einfache und mittelschwere Aufgaben korrekt zu lösen und mehr als 50 % der Studenten sollten auch die schwierigen Aufgaben lösen können. Zudem sollte das Erstellen einer heuristischen Regel nicht länger als 15 Minuten und das Ändern einer heuristischen Regel nicht länger als 10 Minuten dauern.

Tabelle 8.9: Aufgaben an die Probanden für die Evaluation von Frage Q 1.3.

Aufgabe	*Beschreibung*
Aufgabe 1.a.1	Es soll beschrieben werden, was die dargestellte heuristische Regel tut (Lösung: feuert, wenn der Titel eines Use-Cases mehr Zeichen beinhaltet, als dessen Beschreibung).
Aufgabe 1.a.2	Es soll beschrieben werden, was die dargestellte heuristische Regel tut (Lösung: feuert, wenn die Bedingung einer Erweiterung in einem Use-Case leer ist).
Aufgabe 1.b	Die heuristische Regel aus 1.a.2 soll so angepasst werden, dass sie feuert, wenn der Name des zu erweiternden Schritts leer ist.
Aufgabe 2.1	Es soll eine heuristische Regel erstellt werden, die feuert, wenn der Titel eines Use-Cases leer ist.
Aufgabe 2.2	Es soll eine heuristische Regel erstellt werden, die feuert, wenn das Hauptszenario weniger als drei oder mehr als 9 Schritte hat.
Aufgabe 2.3	Es soll eine heuristische Regel erstellt werden, die feuert, wenn zwei Use-Cases den gleichen Titel haben.

Evaluationsmethode

Die Freiwilligen wurden gebeten, einen Aufgabenzettel (vegleiche Tabelle 8.9) zu bearbeiten. Für die Bearbeitung unter Aufsicht wurden 45 Minuten veranschlagt. Die Aufgaben waren in zwei Teilen angeordnet. Zum Einen sollten die Probanden im Verständnis-Teil (Aufgabe 1.a) zeigen, ob sie die heuristischen Regeln (eine mittelschwere und eine komplexe) verstehen. Zum Anderen sollten sie im Mach-Teil eine der Regeln abändern (Aufgabe 1.b) und neue heuristische Regeln erstellen (Aufgabe 2). Dabei sollten drei Kritiken umgesetzt werden: Eine einfache Regel, eine Regel mittlerer Komplexität und eine schwierig umzusetzende Regel. Tabelle 8.9 gibt einen Überblick über die Aufgaben.

Das Ziel war es, dass sich die Studenten in Aufgabe 1.a zunächst in die Thematik einarbeiten. Die konkreten heuristischen Regeln dienen zudem als Beispiele für die folgenden Aufgaben. Deshalb geht die Aufgabe 1.a auch nicht in den GQM-Baum mit ein. Da die heuristischen Regeln in Javascript kodiert sind und die Studenten gute Vorkenntnisse in Java besitzen, sind im Mach-Teil (Aufgabe 1.b und Aufgabe 2) gute Ergebnisse zu erwarten.

Als Hilfsmittel konnten alle Teilnehmer auf eine Sprachbeschreibung zurückgreifen, die die zentralen Kommandos und das Datenmodell eines Use-Cases in HeRA zeigt.

Die Studenten sollten bei jeder Aufgabe die genaue Zeit vermerken, zu der sie mit der Aufgabe begonnen haben. Auf diese Weise konnte im Anschluss festgestellt

Tabelle 8.10: Messplan für die Evaluation der Kodierung von neuen Erfahrungen.

Metrik	*Beschreibung*	*Messhinweise*	*Ausgangshypothese*
M C.8	Anteil auf Anhieb korrekter Änderungen einer heur. Regel	Zähle* Anzahl korrekter Antworten zu Aufgabe 1.b	Anteil korrekter Antworten $\geq 75\%$
M C.9	Durchschnittliche Dauer für Änderung einer heur. Regel	Messe durchschn. Dauer der Bearbeitung von Aufgabe 1.b	Dauer ist ≤ 10 Min
M C.10	Anteil auf Anhieb korrekt neu-erstellter heur. Regeln	Zähle* Anzahl korrekter Antworten zu Aufgabe 2.1 bis 2.3	Anteil ist $\geq 75\%$ bei Aufgabe 2.1 und 2.2, $> 50\%$ bei Aufgabe 2.3
M C.11	Durchschnittl. Dauer für Neu-Erstellung einer heur. Regel	Messe durchschn. Dauer der Bearbeitung der Aufgaben 2.1 – 2.3	Dauer ist bei jeder der drei Aufgaben ≤ 15 Min
M C.12	Anekdotische Evidenz aus Erfahrung in student. Softwareproj.	hier nicht gemessen	Studenten können heuristische Kritiken bewerten und sind bereit dazu

* Antworten mit kleinen Fehlern werden mit 0,5 gezählt.

werden, wie lange die Bearbeitung der Aufgabe gedauert hat. Tabelle 8.10 zeigt, wie diese Aufgaben mit den Metriken aus Abbildung 8.6 zusammenhängen.

Diskussion der Validität

Interne Validität. Der wichtigste interne Aspekt betrifft die Lerneffekte. Während die ersten Berührungen mit dem neuen Aufgabentyp noch recht zäh sind, ist davon auszugehen, dass die Studenten schnell dazu lernen und im weiteren Verlauf auch komplexere Aufgaben des selben Typs schnell lösen können. Dem wurde im Experiment-Design Rechnung getragen, indem die Aufgabe 1.a ausschließlich zur Einarbeitung dient.

Zu berücksichtigen ist auch, dass das Experiment am späten Nachmittag stattfand. Viele Teilnehmer hatten direkt vorher eine praktische Übung und könnten bereits etwas müde gewesen sein.

Zudem enthielt das Aufgabenblatt und die Sprachbeschreibung selbst Fehler, die glücklicherweise bei der Auswertung leicht berücksichtigt werden konnten.

Validität des Aufbaus. Im Rahmen dieser Studie ist vor allem zu klären, ob die Ausgangshypothesen valide sind. Darf man aus 75 % (50 %) richtigen Antworten unter Prüfungskonditionen schließen, dass Nutzer einfache (komplexe) heuristische Kritiken korrekt erstellen können? Sind 10 Minuten für das Ändern und 15 Minuten für das Erstellen von heuristischen Kritiken kurz genug, um dies Nutzern in ihrem Arbeitsalltag zu erlauben? Gemessen an der strengen Bewertung der Abgaben wären diese Ergebnisse im Vergleich zu Prüfungen zur Programmierung durchaus gut. Weniger als 15 Minuten zur Lösung teilweise auch komplexer Aufgaben könnte es erlauben, diese Tätigkeit in andere Aufgaben eines Analysten zu integrieren. Immerhin würden Analysten bei dieser Aufgabe durch einen Compiler unterstützt und könnten direkt die Effekte ihrer Kritik in HeRA sehen.

Validität der Schlussfolgerung. Auf Grund der niedrigen statistischen Aussagekraft wären bei einer Replikation des Experiments kleine bis mittlere Abweichungen zu erwarten. Der Zeitpunkt des Experiments am Nachmittag nach einer fortgeschrittenen praktischen Übung könnte die Performanz der Experimentgruppe im Vergleich zu einer Replikation des Experiments nachteilig beeinflusst haben.

Externe Validität. Im Rahmen dieser Untersuchung wird die Verallgemeinerbarkeit der Ergebnisse vor allem durch die Frage betroffen, ob von studentischen Projekten auf Effekte in industriellen Projekten geschlossen werden kann. Auch bei geringem Aufwand müsste sich ein Analyst im industriellen Kontext erst überwinden, neben seinen eigentlichen Aufgaben eine heuristische Kritik zu bearbeiten. Dazu müsste der Analyst seinen mentalen Kontext von der aktuellen Aufgabe auf die Programmierung von heuristischen Regeln wechseln. Diese Hürde könnte am niedrigsten sein, wenn sich der Analyst gerade mit der Qualitätssicherung von Anforderungsdokumenten beschäftigt. Da in diesem Experiment alle Einflüsse auf die Geschwindigkeit und Korrektheit der Bearbeitung konstant gehalten wurden, kann zu den kausalen Zusammenhängen keine Auskunft gegeben werden.

Immerhin ist die externe Validität laut Wohlin et al. bei der Überprüfung von Theorien der unwichtigste Validitätsbegriff [173].

Ergebnisse

Abbildung 8.7 zeigt die Ergebnisse des Experiments. Die Abbildung zeigt, wie lange die Teilnehmer des Experiments für die Lösungen mindestens, durchschnittlich und maximal gebraucht haben sowie die Anzahl der richtigen Lösungen (Aufga-

be 1.a: verstehen, Aufgabe 1.b: ändern, Aufgabe 2: neu erstellen heuristischer Kritiken).

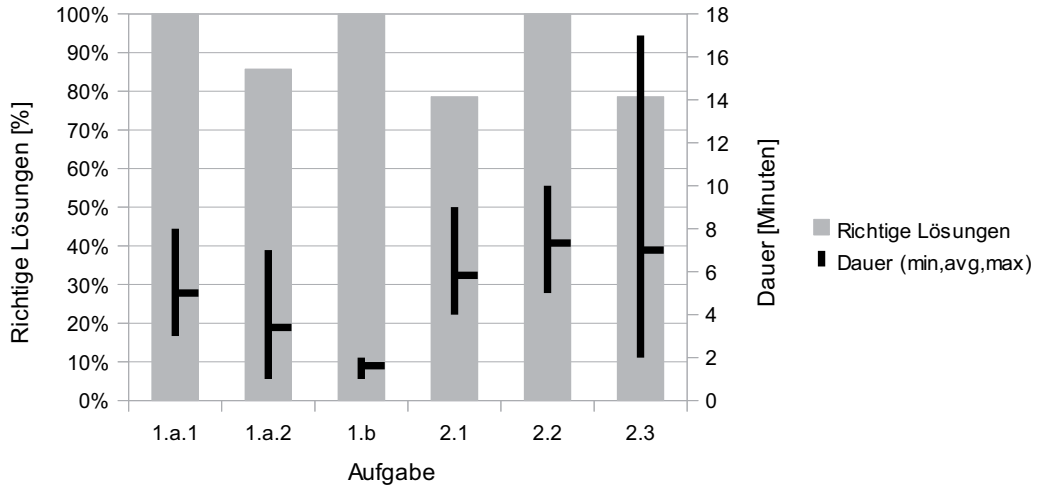


Abbildung 8.7: Ergebnisse des Experiments. Die Abbildung gibt zum Einen die Bearbeitungszeit (schwarz) und zum Anderen das Verhältnis richtiger Antworten (grau) wieder.

Die Bearbeitungszeit ist bei Aufgabe 1.a.2 bereits deutlich niedriger als bei Aufgabe 1.a.1. Dies liegt vermutlich daran, dass sich die Probanden schnell in die Thematik einarbeiten. Eine einmal verstandene Aufgabe zu ändern dauert nicht lange (1–2 Minuten). Die zunehmende Schwierigkeit der Aufgaben 2.1 – 2.3 zeigt sich vor allem an der maximalen Bearbeitungszeit. Die durchschnittliche und minimale Bearbeitungszeit sinkt von Aufgabe 2.2 zu Aufgabe 2.3. Dies deutet darauf hin, dass einige Studenten zu diesem Zeitpunkt gut geübt waren und mit der höheren Komplexität gut zurecht kamen.

Das Verhältnis der richtigen Antworten zu allen Antworten zeigt ein recht positives Bild. Aufgabe 1.a.1 wurde von allen Studenten richtig gelöst, bei Aufgabe 1.a.2 wurde in einem Fall ein falscher Datentyp für die Bedingung einer Erweiterung (Array statt String) angenommen. Dieser Fehler wäre in der Praxis durch eine Fehlermeldung der Javascript Engine leicht zu beheben gewesen.

Aufgabe 2.1 wurde von einem Studenten auf Grund eines Missverständnisses in der falschen Programmiersprache kodiert, bei einem weiteren lagen syntaktische Fehler vor, so dass hier 5,5 (von 7) richtige Antworten gewertet wurden. Bei Aufgabe 2.3 lag wieder bei einem Studenten ein syntaktischer Fehler und bei

einem weiteren ein logischer Fehler vor, so dass auch diese Aufgabe mit 5,5 von 7 richtigen Antworten gewertet wurde.

Zusammengefasst sind 89 % der Aufgaben richtig bearbeitet worden (Änderungen (M C.8) = 100 %, Neu-Erstellungen (M C.10) = 86 %). Leichte Änderungen an heuristischen Regeln dauern weniger als 2 Minuten, das Anlegen von neuen heuristischen Regeln dauert im Durchschnitt weniger als 7 Minuten. Es ist anzunehmen, dass sich diese Werte noch verbessern, wenn nicht mit Zettel und Stift, sondern mit angemessener Werkzeugunterstützung programmiert wird. Dieses Evaluationsszenario ist also ein starker Indikator dafür, dass Nutzer von heuristischen RE-Werkzeugen in der Lage sind, heuristische Regeln und damit auch heuristische Kritiken zu ändern oder neu anzulegen (Frage Q 1.3).

Die Erfahrung aus den studentischen Projekten hat zudem gezeigt, dass heuristische Kritiken bewertet werden (M C.12): Die Möglichkeit zur Bewertung ist in HeRA gegeben, zum Beispiel durch das Ignorieren oder Kommentieren von heuristischen Kritiken (vergleiche Kapitel 3). Die Bachelorarbeit von Rumann beschreibt, wie solche Änderungen zentral verwaltet werden können [141]. Die Evaluation von HeRA fand zeitlich vor Abgabe der Arbeit von Rumman im Softwareprojekt statt. Daher liegt nur anekdotische Evidenz dazu vor, dass Studenten die Bewertungsfunktionen genutzt haben: Studenten haben nach eigener Auskunft Kommentare angelegt. Die Ignorierfunktion wurde jedoch in der Regel nicht verwendet, da die Studenten Sorge hatten, dass einmal ignorierte Kritiken nicht mehr wieder hergestellt werden können, wenn sie später doch wichtig sein sollten. Es kann daher davon ausgegangen werden, dass Nutzer heuristische Kritiken in HeRA bewerten können (Frage Q 1.2) und auch unter bestimmten Umständen bereit sind, bei der Verbesserung der Erfahrungsbasis mitzuwirken (Frage G 1.5). Dies kann als anekdotische Evidenz für die Fragen Q 1.2 und Q 1.5 gewertet werden.

8.3 Analytisches heuristisches Feedback (KonPAss)

Das Ziel von KonPAss ist es, typische Fehler in Anforderungsdokumenten kostengünstig zu finden. Dadurch wird der Arbeitsaufwand von Qualitätsbeauftragten gesenkt und manuelle Reviews können sich auf Aspekte konzentrieren, die sich nicht automatisch überprüfen lassen. Der Kostenfaktor nimmt also eine zentrale Stelle bei der Evaluation von KonPAss ein (Ergebnisse in Abschnitt 8.3.4):

1. Es wird verglichen, wie aufwendig Konsistenzprüfungen mit KonPAss im Vergleich zu manuellen Prüfungen sind. Die Kosten für manuelle Konsistenzprüfung sind zu hoch für eine regelmäßige Durchführung. Dieses Experiment zeigt, dass hingegen mit Hilfe von KonPAss auch regelmäßige Prüfungen möglich sind.

2. Es wird zudem gezeigt, dass KonPAss relevante Qualitätsaspekte effektiv überprüfen kann. Der Recall von KonPAss ist bei vielen relevanten Aspekten mindestens genauso hoch wie bei manuellen Prüfungen.
3. Schließlich wird gezeigt, dass der Aufwand durch falsch-positive Befunde von KonPAss nicht ins Gewicht fällt. Das liegt zum Einen an der hohen Präzision der verwendeten Regeln, zum Anderen lassen sich falsche Befunde sehr schnell aus der Liste aussortieren. Schließlich produzieren auch manuelle Reviews falsche Befunde.

KonPAss ist in Kooperation mit der Daimler AG entstanden und dort mit authentischen Anforderungsdokumenten evaluiert worden. Zum besseren Verständnis wird hier ein kurzer Überblick über den RE-Prozess bei der Daimler AG gegeben. Ein moderner Mercedes-Benz enthält mehr als 50 Steuergeräte, die jeweils durch Software gesteuert werden. Die verschiedenen Features eines Autos erfordern oft das Zusammenspiel mehrerer Steuergeräte. Um das ordnungsgemäße Zusammenspiel der Steuergeräte sicherzustellen, ist es nötig, die Konsistenz verschiedener Dokumente herzustellen. KonPAss bietet hier wertvolle Unterstützung.

8.3.1 Evaluationsszenario: RE-Prozess und Dokumentmodell

Der RE-Prozess bei der Daimler AG dient der Bearbeitung großer Mengen von Anforderungen. Viele dieser Anforderungen ähneln dabei Anforderungen vorheriger Projekte oder sind sogar gleich. Daher werden Anforderungen systematisch wiederverwendet [77]: Die Grundlage dazu bilden stabile Anforderungen, die wiederverwendet werden können. Zusätze für ein neues Modell oder eine neue Variante enthalten nur spezifische Anforderungen, die sich vom wiederverwendeten Kern unterscheiden. Durch diese Trennung wird der Aufwand, eine neue Spezifikation zu erstellen, deutlich reduziert. Dafür wird jedoch eine neue Quelle möglicher Inkonsistenzen eingeführt. Referenzen auf den allgemeinen Teil müssen überprüft werden. Zudem muss sichergestellt werden, dass alle Lücken des allgemeinen Teils im spezifischen Teil geschlossen wurden. Dieses Problem manifestiert sich insbesondere bei Referenzen auf Signale. Signale werden verwendet, um zu beschreiben, wie die Steuergeräte eines Autos miteinander kommunizieren.

Beispiel 27: *Wiederverwendung von Anforderungen*

Abbildung 8.8 zeigt ein Beispiel dieser Wiederverwendung von Anforderungen (vergleiche Heumesser und Houdek [77]): Die allgemeine, modellunabhängige Anforderungsspezifikation (MIRS = model-independant requirements specification) definiert, welche Signale als Eingabe, Ausgabe und Parameter einer Funktion dienen. Die modellspezifische Anforderungsspezifikation (MDRS = model-dependant requirements specification) muss diese logischen Signale auf konkrete physikali-

sche Signale des Netzwerks abbilden, das die Steuergeräte miteinander verbindet (z.B. CAN-Busse [153]). Eine MDRS kann außerdem zusätzliche spezifische Signale einführen. Die sogenannte C-Matrix (Communication-Matrix) gibt einen Überblick über die Sender und Empfänger von Signalen. Sie ist die Referenz für alle Signale, die über die Kommunikationsbusse eines Automobils gesendet werden. Wegen der großen Menge von Signalen (über 3000) ist die C-Matrix eines Autos häufig als Datenbank implementiert.

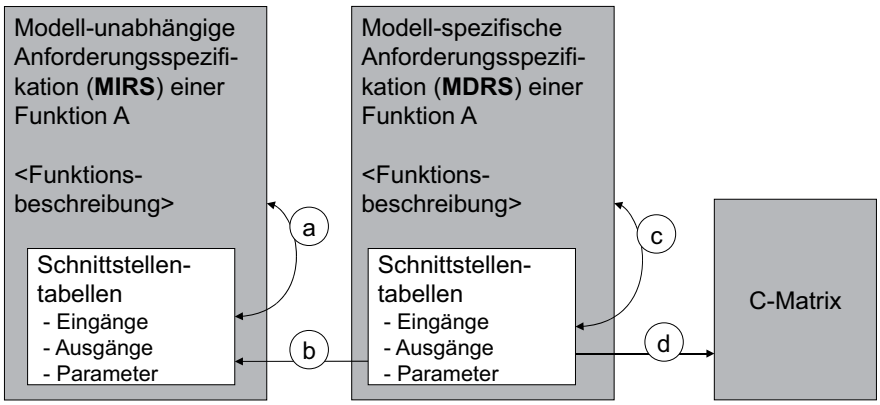


Abbildung 8.8: Beispiel einer Dokumentstruktur. Die Pfeile zeigen Teile, die zueinander konsistent sein sollen. Tabelle 8.11 gibt Beispiele für Konsistenzregeln.

Die in Abbildung 8.8 gezeigte Struktur von MIRS, MDRS und C-Matrix wird zur Zeit in einigen Projekten der Daimler AG verwendet. Daneben werden auch andere Strukturen verwendet. Daher muss KonPAss auf die projektspezifische Struktur anpassbar sein. Im Rahmen der Evaluation gilt die Struktur aus Abbildung 8.8.

Zu einem definierten Zeitpunkt während der Dokumentation von Anforderungen wird ein Review durchgeführt, um die konsistente Verwendung von Signal-Namen in MIRS, MDRS und C-Matrix zu garantieren. Dies ist unangenehme und teure Arbeit, weil Reviewer parallel drei umfangreiche Dokumente lesen und vergleichen müssen. Zudem sehen sich die Signal-Namen sehr ähnlich – Fehler sind häufig das Ergebnis verdrehter Buchstaben in kryptischen Namen (z.B. `IG_Lock_R` anstatt `Ig_Lck_R`). Wegen engen Zeitplänen wird diese Aufgabe daher oft vernachlässigt.

Das Ziel von KonPAss ist es, die konsistente Verwendung von Signal-Namen in beiden Teilen der Spezifikation automatisch zu überprüfen und mit der C-Matrix abzugleichen. Abbildung 8.9 gibt einen Überblick über den Prozess dieser Überprüfung mit und ohne Verwendung von KonPAss. Im Fall ohne KonPAss sind zwei Reviews angesetzt: ein Vor-Review, bei dem formale Aspekte wie Konsistenz

und korrekte Verwendung von Vorlagen geprüft werden, sowie das Hauptreview, bei dem technische und inhaltliche Aspekte geprüft werden. Der graue Bereich auf der linken Seite von Abbildung 8.9 hebt die zeitaufwendigen Aktivitäten des optionalen Vor-Reviews hervor. Vom Standpunkt eines Qualitätsmanagers ist ein gutes Vor-Review sehr nützlich, weil sich die Reviewer dadurch im Hauptreview auf die wichtigen Dinge konzentrieren können: die Funktionalität. In der Praxis fehlt leider häufig die Zeit für dieses Vor-Review.

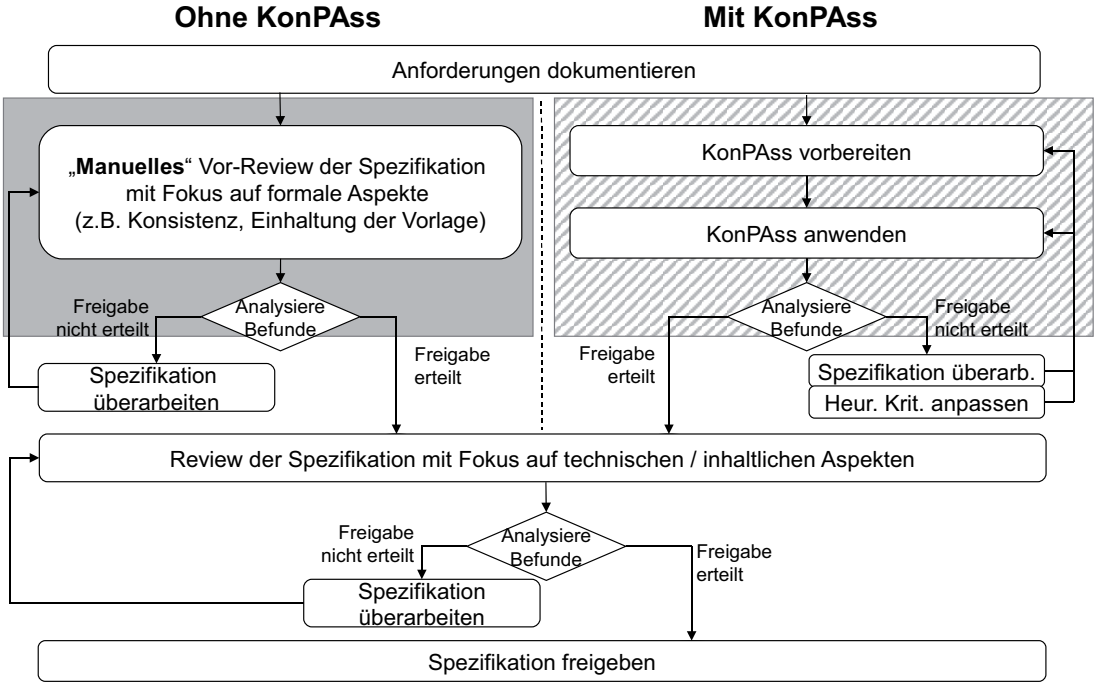


Abbildung 8.9: Qualitätsmanagement-Prozess mit und ohne KonPAss.

Die schräg schraffierte Fläche auf der rechten Seite von Abbildung 8.9 zeigt die Rolle von KonPAss in diesem Zusammenhang. KonPAss ersetzt das Vor-Review. Konsistenzfehler können effizient behoben werden und eine verbesserte Version der Anforderungsdokumente mit weniger Inkonsistenzen kann manuell inhaltlich geprüft werden. Dies erlaubt es den Reviewern sich auf inhaltliche Aspekte zu konzentrieren, beispielsweise um Anforderungen zu finden, die fälschlicherweise als Zusatzinformation gekennzeichnet sind, die im falschen Teil der Spezifikation spezifiziert wurden, oder die inkonsistent zu Werten in technischen Zeichnungen der Spezifikation sind. Durch die Aktivität *adapt check-rules* wurde ein expliziter Erfahrungskreislauf eingeführt. Die heuristischen Kritiken können so während der Verwendung verbessert werden.

In den Prozess in Abbildung 8.9 sind drei wichtige Rollen eingebunden: C-Matrix Autoren sind für die C-Matrix verantwortlich, die sich unabhängig von den Anforderungsspezifikationen entwickelt. Änderungen an der C-Matrix sind selten und müssen formal beantragt werden. Im Beispiel von Abbildung 8.8 sind Analysten verantwortlich dafür, die MIRS und MDRS für konkrete Funktionen zu erstellen und zu pflegen. Wenn nötig, können Analysten auf die Unterstützung von Beratern zurückgreifen, die neben anderen Dingen für die Überprüfung der Konsistenz und Verbesserungsvorschläge zuständig sind. KonPAss zielt darauf ab, diese Berater in ihrer Aufgabe zu unterstützen. Wenn sich die Struktur der Dokumente in einem Projekt ändert, kann der Berater KonPAss entsprechend anpassen.

Beispiele für heuristische Kritiken

Die Infrastruktur von KonPAss bietet unter anderem eine Abstraktionsschicht über der physikalischen Repräsentation der Anforderungen. Diese erlaubt es, Konsistenzregeln auf angemessener Abstraktionsebene kompakt zu formulieren:

Beispiel 28: *Konsistenzregel im Evaluationsszenario*

In einer MDRS sollen alle in einer MIRS definierten Signale korrekt in Bezug auf die C-Matrix gesetzt werden.

Die Hürde, Signale in MIRS, MDRS und C-Matrix zu finden, ist durch die Infrastruktur von KonPAss bereits genommen. Diese werden direkt in das semantische Dokumentmodell (siehe Abbildung 7.4) eingelesen, so dass Regeln ohne Kenntnis der Syntax der Dateiformate formuliert werden können.

Basierend auf dieser Infrastruktur wurden im ersten Anlauf 12 Regeln implementiert, die in einer Brainstorming-Sitzung bei der Daimler AG identifiziert wurden. Darunter sind allgemeine Konsistenzregeln (die Regeln 6, 7, 10 und 11 in Tabelle 8.15) sowie projektspezifische Regeln (die Regeln 1-5, 8, 9, und 12 in Tabelle 8.15). Tabelle 8.11 gibt weitere Beispiele für projektspezifische Konsistenzregeln im Kontext des Evaluationsszenarios.

Die technische Umsetzung dieser Regeln war meistens relativ einfach. Bei der Entwicklung von KonPAss hat es weit mehr Zeit gekostet, alle Spezialfälle und Ausnahmen der Regeln zu ermitteln, damit diese so wie erwartet funktionieren. Durch die Kodierung des Domänenwissens als Regeln wurden sich die Berater erst der Feinheiten der Konsistenzregeln bewusst, die sie vorher implizit angewandt hatten.

Tabelle 8.11: Beispiele für heuristische Kritiken in KonPAss.

Regel-Nr.	Beschreibung
1	Signale, die in den Tabellen der Schnittstellenbeschreibungen der MIRS vorkommen, müssen auch in den entsprechenden Tabellen der Schnittstellenbeschreibungen (Eingänge, Ausgänge oder Parameter) der MDRS vorkommen (b in Abbildung 8.8).
2	Signale, die in Textpassagen der funktionalen Beschreibung einer Spezifikation vorkommen, müssen auch in deren Tabellen der Schnittstellenbeschreibung vorkommen (a und c in Abbildung 8.8). Zu beachten ist, dass es viel schwieriger ist, Signal-Namen in Fließtext zu identifizieren, als in Tabellenzellen.
4	Signale, die in den Tabellen der Schnittstellenbeschreibung der MIRS vorkommen, müssen auch in deren Textpassagen der funktionalen Beschreibung verwendet werden (a in Abbildung 8.8).
12	Signale, die in den Tabellen der Schnittstellenbeschreibung der MDRS vorkommen, müssen dort in Bezug auf die physikalischen Signale der C-Matrix gesetzt werden. Zudem müssen Signale in der Tabelle der Eingänge vorkommen, wenn sie in der C-Matrix als Empfänger gekennzeichnet sind, oder in der Tabelle der Ausgänge, wenn sie in der C-Matrix als Sender gekennzeichnet sind (d in Abbildung 8.8).

8.3.2 Konkrete Evaluationsziele

Typische Spezifikationen bei der Daimler AG haben zwischen 100 und 2000 Seiten (vergleiche Leuser [111]). Eine interne Untersuchung bei der Daimler AG hat ergeben, dass die Überprüfung eines einzelnen Konsistenz-Aspekts in einer Spezifikation von 400 Seiten 1,5 Personentage dauert. Soviel Zeit ist oft nicht vorhanden. Daher ist es nicht immer möglich, eine vollständige Spezifikation zu überprüfen, wodurch sich Konsistenzfehler in der Spezifikation ergeben. Diese können zu teuren Folgefehlern führen, wenn sie nicht in den folgenden Phasen mühsam beseitigt werden.

Computerbasierte Konsistenzprüfungen mit heuristischen Kritiken können ein Schlüssel zu diesem Problem sein, wenn sie ähnlich effektiv wie manuelle Konsistenzprüfungen sind und dabei eine höhere Effizienz erreichen. Um hier zu sinnvollen Evaluationsergebnissen zu kommen, müssen Effektivität und Effizienz manueller und computerbasierter Konsistenzprüfungen miteinander verglichen werden. Tabelle 8.12 fasst das Ziel nach der Facetten-Beschreibung von Zielen nach Wohlin et al. zusammen [173].

Um die Effektivität computerbasierter Konsistenzprüfungen im Vergleich zu manuellen Prüfungen nachzuweisen, muss gezeigt werden, dass computerbasierte Konsistenzprüfungen:

Tabelle 8.12: Zielfacetten für KonPAss.

Zweck	Qualitäts-aspekt	Betrachtungs-gegenstand	Perspek-tive	Kontext
Untersuche	Effektivität und Effizienz	manuelle und computerbasierte Konsistenzprüfungen von Anforderungsdokumenten	Qualitätsmanager	jeweils durchgeführt von Anforderungsexperten bei der Daimler AG auf einer preparierten Anforderungsspezifikation in einem abge- schlossenen Versuchsaufbau

1. die gleichen relevanten Defekt-Typen überprüfen können,
2. dabei mindestens genauso viele Fehler finden und
3. dabei höchstens genauso viele Falsch-Positive erzeugen.

Um die Effizienz zu evaluieren, muss zusätzlich gezeigt werden, dass computerbasierte Konsistenzprüfungen nicht mehr Zeit benötigen, als manuelle Konsistenzprüfungen.

Abbildung 8.10 zeigt, wie sich diese Evaluationsziele in den GQM-Plan der Arbeit einfügt. Die Abbildung zeigt auch, durch welche Metriken die Fragen evaluiert werden. Tabelle 8.13 erläutert die verwendeten Metriken. Mit Hilfe der Metriken und basierend auf den Ergebnissen bei Tests während der Entwicklung von KonPAss können die folgenden vier Hypothesen aufgestellt werden (die verwendete mathematische Notation wird in Tabelle 8.14 erläutert).

Tabelle 8.13: Metriken für die Evaluation von KonPAss.

	Name	Beschreibung	Definition
$b_{\oplus/\ominus}$	M K.1	Anzahl der als relevant ($\oplus$) oder irrelevant ($\ominus$) bewerteten Befunde	
f	M K.8	Anzahl der gemeldeten Befunde	
rf	M K.3	Anzahl der korrekten Befunde	$rf = f - b_{\ominus} = b_{\oplus}$
fp	Falsch-Positive	Anzahl der falsch-positiven Befunde	$fp = f - b_{\oplus} = b_{\ominus}$
r	M K.9	Anzahl der vorhandenen Befunde	
rec	M K.6	siehe Definition 26 (Recall)	$rec = \frac{rf}{r}$
pre	M K.5	siehe Definition 27 (Precision)	$pre = \frac{rf}{f}$
fval	F-Maß	vergleiche Definition 29 (Autorität)	$fval = \frac{2 \cdot rec \cdot pre}{rec + pre}$
t	M K.7	Zeit, um einen Befund zu finden	

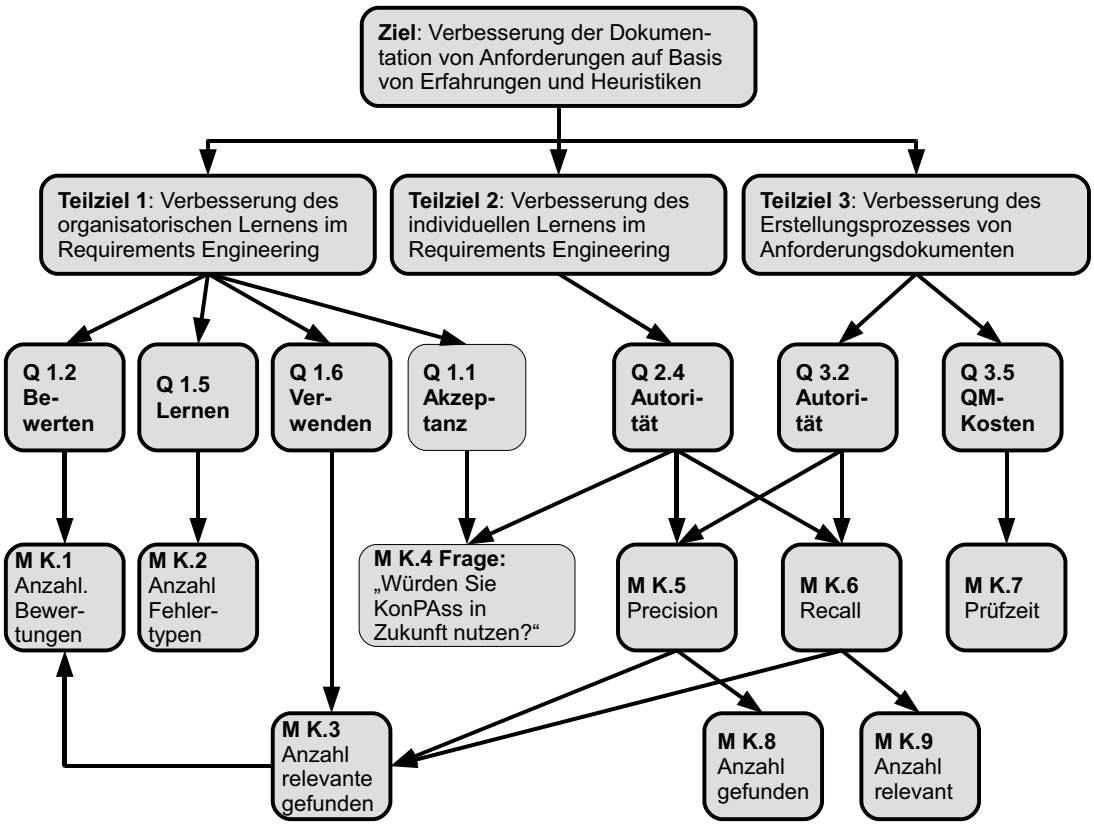


Abbildung 8.10: GQM-Baum für die Evaluation von KonPAss. Metriken, die das Ziel der Evaluation von KonPAss Effektivität und Effizienz unterstützen, sind fett umrandet.

H_1: In einer computerbasierten Konsistenzprüfung können mehr als 90 % der Fehlertypen, die durch manuelle Reviews gefunden werden können, als heuristische Kritiken kodiert werden:

$$H_{0,1} : \pi_c \leq 0.9 \; und \; H_{a,1} : \pi_c > 0.9$$

H_2: Eine computerbasierte Konsistenzprüfung findet mindestens so viele Befunde wie eine manuelle Konsistenzprüfung:

$$H_{0,2} : P_c^f < P_m^f \; und \; H_{a,2} : P_c^f \geq P_m^f$$

H_3: Eine computerbasierte Konsistenzprüfung findet höchstens so viele falschpositive Befunde wie eine manuelle Konsistenzprüfung:

$$H_{0,3} : P_c^{fp} > P_m^{fp} \; und \; H_{a,3} : P_c^{fp} \leq P_m^{fp}$$

Tabelle 8.14: Mathematische Konventionen für dieses Experiment.

$H_{0,x}$	Null-Hypothese, x gibt die zugehörige Hypothese H_1 bis H_4 an.
$H_{a,x}$	Alternative Hypothese, x gibt die zugehörige Hypothese H_1 bis H_4 an.
c	computerbasierte Konsistenzprüfung
m	manuelle Konsistenzprüfung
π	Wahrscheinlichkeit, dass ein Defekt-Typ durch eine Regel von KonPAss entdeckt werden kann
P	Anzahl, wie oft die entsprechende Prüfung bessere Ergebnisse liefert

H_4: Eine computerbasierte Konsistenzprüfung findet mehr Befunde als eine manuelle Konsistenzprüfung in der gleichen Zeitspanne:

$$H_{0,4} : P_c^t < P_m^t \; und \; H_{a,4} : P_c^t P_m^t$$

Hypothese H_1 unterstützt damit Teilziel 1: Ist H_1 erfüllt, so zeigt sich, dass relevante Erfahrungen und technisches Wissen mit heuristischen Kritiken kodiert werden können.

Hypothese H_4 unterstützt Teilziel 3: Ist H_4 erfüllt, können regelmäßig Konsistenzprüfungen durchgeführt werden, die sonst nicht möglich sind. Ohne die Unterstützung würden einige Fehler also nie oder erst in späteren Phasen gefunden werden. Konsistenzprüfung auf Basis von heuristischen Kritiken würde in diesem Fall also zu einer verbesserten Dokumentation führen.

8.3.3 Evaluationsmethode

Experimentaufbau

Im Rahmen der Evaluation wurde eine kleine, aber repräsentative Anforderungsspezifikation in einem *offline-setup* unter Laborbedingungen untersucht (vergleiche Wohlin et al. [173]). Diese Spezifikation wurde in der Vorbereitungsphase des Experiments mit künstlichen Fehlern versehen. Dabei wurde darauf geachtet, dass weder manuelle noch automatische Konsistenzprüfung durch die Art der künstlichen Fehler benachteiligt werden.

Der Experiment-Aufbau sieht einen Faktor und zwei davon abhängige Variablen vor. Der einzige Faktor (unabhängige Variable) ist die Methode zur Konsistenzprüfung, die zwei Werte annehmen kann: manuelle oder computerbasierte Kon-

sistenzprüfung. Die abhängigen Variablen sind die Effektivität und die Effizienz der Prüfung.

Abbildung 8.11 zeigt den Evaluationsprozess. Die Spezifikation wird parallel von zwei Teams überprüft. Team A prüft die Konsistenz der Spezifikation manuell (linker Pfad in Abbildung 8.11). Drei Reviewer mit unterschiedlichem Wissen über die Struktur des Dokuments wurden während dem Review-Kickoff instruiert. Sie erhielten eine Review-Checkliste die auf den während der Anforderungserhebung für KonPAss erfassten Regeln basiert. Team B verwendet das KonPAss-Werkzeug (rechter Pfad in Abbildung 8.11).

Die grauen Bereiche in Abbildung 8.11 zeigen die verschiedenen Review-Phasen. Während jeder dieser Phasen wurde die Zeit gemessen. Die Zeit wurde dabei für jeden Pfad einzeln gestoppt. Zum Beispiel wurde in Phase 3 gemessen, wie lange Team A benötigte, die Befunde zussamenzuführen und auf die tatsächlichen Defekte zu beziehen. Außerdem wurde gemessen, wie lang Team B brauchte, um die Befunde von KonPAss zu klassifizieren und auf die tatsächlichen Defekte zu beziehen.

Diskussion der Validität

Nach Wohlin et al. [173] gibt es verschiedene Typen von Gefährdungen der Validität. Dieser Abschnitt behandelt Stärken und Gefährdungen der Validität des Experiment-Aufbaus.

Validität der Schlussfolgerung. Die Größe der Stichprobe in diesem Experiment reicht nicht aus, Schlussfolgerungen mit statistischer Konfidenz zu ziehen. Die statistischen Tests erreichen eine minimale Teststärke (engl: statistical power) von 62 % (siehe Ergebnisse in Abschnitt 8.3.4). Um statistisch signifikante Schlüsse ziehen zu können, wird eine Teststärke von mindestens 80% benötigt.

Einige Messungen sind zudem durch subjektive Beurteilungen in ihrer Glaubwürdigkeit gefährdet. So hat beispielsweise die Frage, ob ein bestimmtes Wort ein Weakword ist, zu einigen Diskussionen geführt. Im speziellen Kontext des Experiments fällt diese Bedrohung der Glaubwürdigkeit jedoch nicht so stark ins Gewicht, weil die Diskussionen von ausgewählten Anforderungsexperten der Daimler AG geführt wurden, die zum Teil auch für die KonPAss zugrunde liegende Weakwordliste verantwortlich sind.

Interne Validität. Der unterschiedliche Wissensstand über die Struktur der Anforderungsspezifikationen (MIRS, MDRS und C-Matrix) hätte zu großen Unterschieden in der Qualität und Quantität der Befunde führen können. Im Expe-

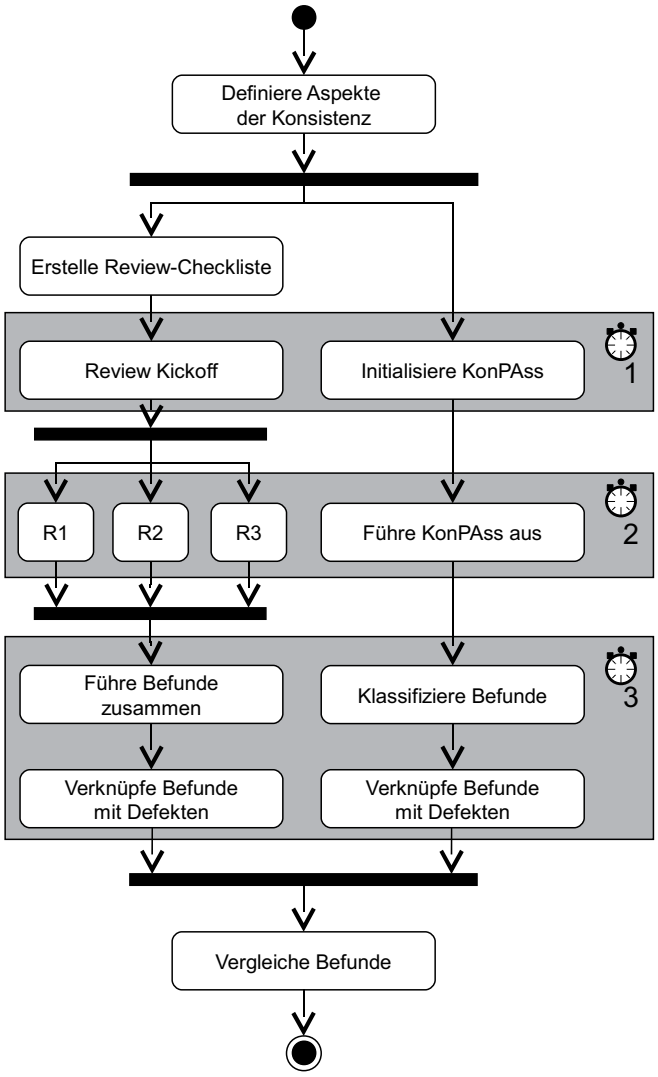


Abbildung 8.11: Der Evaluationsprozess für die Beispielimplementierung KonPAss im Labor-Evaluationsszenario bei der Daimler AG. Die grauen Bereiche mit Stoppuhr zeigen, wo die Zeit gemessen wurde. Links sind die Aktivitäten für die manuelle Prüfung, rechts die Aktivitäten für die computerbasierte Prüfung dargestellt.

rimentverlauf konnten jedoch nur sehr kleine Unterschiede beobachtet werden, vermutlich wegen des formalen Charakters der Konsistenzregeln und abgeleiteten Review-Checklisten.

Eine kleinere Verunreinigung des Experiments könnte auch in der Instrumentalisierung liegen. Die verwendete Spezifikation ist zwar authentisch, durch die zusätzlich eingefügten Defekte enthält sie jedoch mehr Fehler als üblich. Dadurch wurde es für die Reviewer schwieriger, alle Befunde zu entdecken. Zudem waren einige Defekttypen auch nur sehr schwer für die Reviewer zu finden. Um dieses Problem abzuschwächen, wurden die Befunde von drei Reviewern zusammengeführt, um eine vergleichbare Abdeckung der Defekte zu erreichen. Anders herum konnten jedoch einige Defekte auch unmöglich von KonPAss gefunden werden, was in manchen Regeln zu einer hohen Anzahl von falsch-positiven Befunden führte (vergleiche Regel 7 in Tabelle 8.15).

Validität des Aufbaus. Ein Problem ist, dass im Rahmen des Experiments nur ein Teil des theoretischen Potentials von Reviews ausgenutzt wurde. Es können nur Schlüsse aus den formalen Konsistenzchecks gezogen werden, jedoch nicht aus inhaltlichen (semantischen) Checks der Anforderungsspezifikation. Gemäß der Forschungsfrage ist dieser Effekt erwünscht.

Ein gößeres Problem stellt hingegen die Ermittlung der genauen Anzahl der gefundenen Befunde dar. Reviewer tendieren dazu, einen Defekt in einem Befund zu dokumentieren. KonPAss hingegen dokumentiert einen Befund für jede verletzte Konsistenzregel. Das kann zu mehreren Befunden führen, die von einem einzelnen Defekt verursacht wurden. Dies verursacht einen großen Unterschied in der Anzahl der Befunde. Dieses Problem wird nachfolgend noch behandelt.

Externe Validität. Nach der Durchführung des Experiments wurde klar, dass die Zeitmessung in Phase 3 des Reviews ausschließlich das Zusammenführen und die Klassifikation der Befunde hätte umschließen sollen, weil das in Bezug setzen der Befunde zu den Defekten ausschließlich der Messung diente. Dies hat jedoch keinen Einfluss auf das Ergebnis, weil auch die Befunde von KonPAss in Bezug auf die Defekte gesetzt wurden. Bei der Übertragung der Ergebnisse in die industrielle Praxis muss diese Zeit jedoch wieder herausgerechnet werden.

Insgesamt liegt die größte Stärke der Untersuchung in der externen Validität. Das Experiment konnte unter Mitwirkung professioneller Reviewer im Kontext einer Firma durchgeführt werden. Darüberhinaus konnte eine repräsentative Spezifikation verwendet werden.

8.3.4 Evaluationsergebnisse

In der ursprünglichen Anforderungserhebung für KonPAss wurden im Kontext der Daimler AG 15 Konsistenzregeln aufgestellt. Mit Hilfe der KonPAss Plattform wurden davon 12 Regeln umgesetzt. Die verbliebenen drei Regeln waren vor der Evaluation nicht umgesetzt worden, weil sie eine komplexe Analyse natürlicher Sprache oder komplizierte Algorithmen erfordern. Tabelle 8.15 zeigt die Ergebnisse pro Regel. Dargestellt sind die Anzahl der generierten Befunde (M K.8) sowie die Anzahl der generierten Befunde, die tatsächlich auf einen Fehler hindeuten (M K.3). Zusammen mit der Anzahl der tatsächlich vorhandenen Befunde (M K.9) lassen sich Recall, Precision und F-Maß berechnen. Aus Platzgründen sind diese Werte nur für die computerbasierte Konsistenzprüfung gegeben. Im Rahmen der Evaluation wurde Regel 6 nicht betrachtet, weil sie von den Reviewern nicht verwendet wurde.

In 8 von 11 Fällen liefert der computerbasierte Konsistenzcheck mindestens die gleiche Anzahl von Befunden, wie der manuelle Check. Der Vergleich der Falsch-Positiven liefert sogar ein noch besseres Ergebnis: In nur einem Fall lieferte KonPAss mehr Falsch-Positive als die Reviewer.

Tabelle 8.16 fasst die Prüfzeiten nach den Reviewphasen zusammen. In der ersten Phase mussten alle Regeln erklärt werden. Dadurch dauerte die Vorbereitung beider Teams länger. Ein Team von drei Reviewern einzuweisen dauerte zusätzlich Zeit. In der Phase drei war KonPAss im Vorteil, weil nur eine Review-Liste betrachtet wurde. Wäre nur ein Reviewer gegen KonPAss angetreten, wäre die durchschnittliche Zeit um einen Fehler zu finden, geringer gewesen. Keiner der Reviewer alleine konnte jedoch genügend Befunde entdecken, um sich mit KonPAss in den anderen Hypothesen zu messen.

Evaluation der Hypothesen

Für die Evaluation der Hypothesen wurde das Signifikanzlevel auf $\alpha = 0,05$ gesetzt. Die Wahrscheinlichkeit eines Typ-1-Fehlers (die Null-Hypothese wird zurückgewiesen, obwohl sie wahr ist) liegt daher bei 0,05.

Um die Hypothesen H_2 und H_3 zu evaluieren wurde ein *sign test* verwendet, weil nur untersucht werden soll, in welchen Fällen eine Regel bessere Ergebnisse liefert. Da die Testmenge 11 Regeln umfasst, ist die Wahrscheinlichkeit eines Typ-II-Fehlers beim *sign test* ungefähr $\beta \approx 0,38$. Dadurch ergibt sich eine statistische Kraft von ungefähr 62%.

Für die Hypothese H_2 ergibt sich eine Wahrscheinlichkeit $p_2 = 0,113$ und für Hypothese H_3 eine Wahrscheinlichkeit $p_3 = 0,006$. Dementsprechend kann die Null-Hypothese $H_{0,2}$ nicht zurückgewiesen werden, weil $p_2 > 0,05$ ist. Es gibt

Tabelle 8.15: Evaluation: Befunde pro Regel. Die Tabelle zeigt die Anzahl der Befunde (f) und die Anzahl der relevanten Befunde (rf) für die manuelle und computerbasierte Konsistenzprüfung. Zudem wird der Recall (rec), die Precision (pre) und das F-Maß (fval) der computerbasierte Konsistenzprüfung jeweils auf eine Nachkommastelle gerundet gegeben. Die Zahl der tatsächlich vorhandenen Befunde gibt die Spalte (relev.) an.

Regel	**computerbasiert**					**manuell**		**relev.**	**Bemerkung**
#	f	rf	rec	pre	fval	f	rf		
1	11	11	1,0	1,0	1,0	9	8	11	
2	6	6	1,0	1,0	1,0	5	4	6	
3	3	3	0,4	1,0	0,6	6	6	8	Setzt fortgeschr. Verarbeitung natürl. Sprache voraus.
4	18	18	1,0	1,0	1,0	11	8	18	
5	2	2	1,0	1,0	1,0	2	2	2	
6	9	9	1,0	1,0	1,0	0	0	9	Bei manuellem Review nicht berücksichtigt.
7	22	12	0,4	0,6	0,5	7	6	29	KonPAss konnte bestimmte Falsch-Positive systematisch nicht ausfiltern.
8	7	7	0,9	1,0	0,9	8	7	8	
9	9	9	1,0	1,0	1,0	3	3	9	
10	11	9	0,7	1,0	0,8	15	13	17	Reviewer sind in der Lage, neue Weakwords zu finden, während KonPAss eine statische Liste von Wörtern verwendet.
11	8	8	0,5	1,0	0,6	11	10	17	KonPAss konnte diese Regel auf den englischen Teilen der Dokumente nicht anwenden.
12	20	13	0,7	0,7	0,7	18	11	20	
Summe	124	107	0,7	0,9	0,8	95	78	154	

also keine statistische Aussage darüber, ob es einen Unterschied zwischen der Anzahl der Befunde bei manuellen oder computerbasierten Konsistenzprüfungen gibt. Für H_3 kann mit statistischer Konfidenz ausgesagt werden, dass computerbasierte Konsistenzprüfung weniger Falsch-Positive liefert. Dies ist ein durchaus überraschendes Ergebnis, da von den Teilnehmern des Experiments das Gegenteil erwartet worden war.

Für die Hypothese H_1 wurde ein binomial Test eingesetzt. Weil die Testmenge für H_1 mit 15 Regeln größer ist, ist die Wahrscheinlichkeit für einen Typ-II-Fehler hier mit $\beta \approx 0,22$ geringer. Daraus folgt eine statistische Kraft von ungefähr 78%. Insgesamt ergibt sich eine Wahrscheinlichkeit $p_1 = 0,184$. Dementsprechend kann auch für H_1 (mindestens 90% der Konsistenzregeln können mit KonPAss realisiert

Tabelle 8.16: Evaluation der Prüfungszeit mit und ohne KonPAss.

Review Phase (nach Abbildung 8.11)	**Zeit für computerbasierte Konsistenzprüfung**	**Zeit für manuelle Konsistenzprüfung**
1	$<$ 30 min	30 min
2	$<$ 1 min	3 * 70 min = 210 min
3	$<$ 120 min	165 min
Summe	$\approx$ 150 min	405 min
Durchschnittliche Zeit, um einen Befund zu entdecken	$\approx$ 0,83 Befunde / Minute	$\approx$ 0,2 Befunde / Minute

werden) keine statistische Aussage getroffen werden.

Für Hypothese H_4 wurde nur die Gesamtprüfzeit erfasst und in Bezug zur Gesamtzahl der Befunde gesetzt. Daher ist die Testmenge (eins) nicht groß genug für statistische Betrachtungen. Deskriptive Statistik zeigt, dass computerbasierte Konsistenzprüfung viel weniger Zeit pro Befund benötigt, als manuelle Konsistenzprüfung. KonPAss entdeckt im Schnitt vier mal so viele Befunde pro Zeiteinheit.

Nach diesen Ergebnissen kann also die meiste Zeit in der zweiten Review-Phase gespart werden, wenn die Dokumente überprüft werden. In der letzten Phase kann viel weniger Zeit gespart werden, weil die Befunde in dieser Phase diskutiert werden. KonPAss bietet dazu Unterstützung, weil die Befunde nach verschiedenen Kriterien sortiert und so komfortabel bearbeitet werden können.

Weitere Beobachtungen

Während der Konsistenzprüfung wurden häufig mehrere Befunde für einen einzelnen Defekt geliefert. Zum Beispiel führt ein einfacher Tippfehler bei einem Signalnamen in den Tabellen einer MDRS dazu, dass mehrere Konsistenzregeln verletzt werden:

1. Das eigentliche Signal, welches in der MIRS spezifiziert wurde, kann in der MDRS nicht gefunden werden.
2. Das falsch geschriebene Signal kann im Text der MDRS nicht gefunden werden.
3. Das eigentliche Signal im Text der MDRS kann in den Tabellen nicht gefunden werden.

Tabelle 8.17: Evaluation: Effektive Anzahl der Defekte.

computerbasierte Konsistenzprüfung	**manuelle Konsistenzprüfung**	**Fehler gesamt**
114	54	154

In diesem Fall lieferte KonPAss drei Befunde, zwei der Reviewer lieferten je zwei Befunde und ein Reviewer bemängelte einen Schreibfehler.

Aus diesem Grund erscheint auf den ersten Blick angemessen, die Effektivität auf Basis der Defekte anstatt der Befunde zu messen. Dabei ergeben sich aber mehrere Probleme: Es ist einfach, Befunde auf die bekannten, künstlich eingefügten Defekte abzubilden. Im Gegensatz dazu erfordert es eine beträchtliche Interpretationsleistung, zusätzliche Befunde auf bislang unbekannte Fehler abzubilden. Zudem ist es aus pragmatischer Sicht eines Anforderungsautoren irrelevant, ob beim Beheben eines Defekts ein oder mehr Befunde gelöst werden. Schließlich zeigt Tabelle 8.17, dass die Evaluation nach Befunden keinen Vorteil für KonPAss darstellt. Dort wird die effektive Anzahl der Defekte, die von KonPAss oder mindestens 2 Reviewern gefunden wurden, gegeben. Dadurch, dass Defekte mindestens durch Befunde zweier Reviewer gefunden werden müssen, fallen zufällig entdeckte Befunde etwas weniger stark ins Gewicht. Zu beachten ist, dass in der Praxis höchstens ein Reviewer ein Dokument überprüft. Ein Fehler, der bei zwei von drei Reviewern im Experiment nicht auffällt, würde mit hoher Wahrscheinlichkeit übersehen. *Fehler gesamt* gibt dabei die Vereinigung der Menge der Fehler die a) von KonPAss gefunden wurden, b) von den Reviewern gefunden wurden und c) bei der Vorbereitung des Experiments künstlich angelegt wurden. Nach Tabelle 8.17 wurden im manuellen Review nur 47% der Fehler gefunden, die durch KonPAss aufgedeckt wurden. Zum Vergleich, in Tabelle 8.15 ergab das manuelle Review 77 % der Anzahl der Befunde, die von KonPAss berichtet wurden.

Die verschiedenen Regeln sind unterschiedlich schwer zu überprüfen. Überraschenderweise gaben die Reviewer an, dass Regeln, die Passiv oder nicht-atomare Anforderungen betreffen, schwerer zu evaluieren sind, als Regeln, die die korrekte Verwendung von Signalen betreffen. Die am Experiment beteiligten Personen hatten vorher vermutet, dass die Reviewer KonPAss in diesen sprachlichen Aspekten überlegen wären.

Die Reviewer gaben außerdem an, dass es einfach ist, zu überprüfen, ob ein Signal in einer Tabelle auch in den funktionalen Beschreibungen der selben Spezifikation vorkommt (Regel 4). Die Ergebnisse der Untersuchung zeigen jedoch, dass diese Annahme falsch ist: Selbst in der kombinierten Befundliste aller drei Reviewer wurden nur 11 von 17 Problemen gefunden.

Am Anfang der Entwicklung von KonPAss war eine der Hauptsorgen, dass eine automatische Konsistenzprüfung zu viele falsch-positive Befunde generiert, um nützlich zu sein. Daher ist es überraschend, dass KonPAss sogar weniger falsch-positive Befunde liefert, als die manuelle Konsistenzprüfung. Die Reviewer berichteten mehr falsch-positive Befunde als erwartet, interessanterweise wurden einige falsch-positive Befunde übereinstimmend von allen drei Reviewern gemeldet. Die Reviewer ermüdeten mit der Zeit, da es schwierig ist, sich lange auf die monotone Arbeit mit den eher formalen Konsistenzregeln zu konzentrieren. Dies ist vermutlich der Grund dafür, dass alle drei Reviewer um eine automatische Lösung für die Konsistenzprüfung gebeten haben.

Im Gegensatz zu KonPAss waren die Reviewer in der Lage, zusätzliche Kommentare zum Inhalt der Spezifikation zu machen. Dies geht über die Konsistenzprüfung hinaus und gehört eher in das Review der technischen Aspekte, dass der Konsistenzprüfung gewöhnlich folgt (vergleiche Abbildung 8.9). Ein Ziel bei der Entwicklung war es schließlich auch, es Reviewern zu erlauben, sich auf diese inhaltlichen Aspekte zu konzentrieren.

Evaluation des Nutzen

Nach Abschnitt 8.3.3 dauert die Konsistenzprüfung einer einzigen Regel in einer Spezifikation von 400 Seiten 1,5 Arbeitstage. Für die Evaluation des Nutzens sei angenommen, dass a) der Prüfaufwand linear zur Anzahl der Seiten steigt und b) die Anzahl der parallel überprüften Konsistenzregeln den Prüfaufwand nicht erhöht. Beide Annahmen sollten zu einer zu niedrigen Schätzung führen. Durch die Ermüdung der Reviewer würden beide Faktoren die Chance, Fehler zu finden, zudem dramatisch reduzieren.

Tabelle 8.16 zeigt, dass KonPAss hauptsächlich die Zeit für die Review-Phase 2 senkt. Zusammen mit den Annahmen a) und b) ergibt sich, dass KonPAss den Aufwand für Konsistenzprüfungen um die Zeit für das eigentliche Review senkt. Sei p die Anzahl der Seiten und r die Anzahl der eingesetzten Reviewer. Dann ergibt sich nach den Annahmen der gesparte Aufwand c wie folgt:

$$c = r * \left(\frac{p}{400\ Seiten} * 1,5\ Arbeitstage\right)$$

Die hier präsentierten Ergebnissen legen Nahe, dass mindestens drei Reviewer benötigt werden, um gute Review-Ergebnisse zu erreichen. Zudem haben die Reviewer im Experiment etwa doppelt so lange gebraucht (3,6 min/Seite statt 1,8 min/Seite). In einem typischen Einsatzszenario ergibt sich als Einsparung mit KonPAss also:

$$c = 3 * \left(\frac{2000\ Seiten}{400\ Seiten} * 1,5\ Arbeitstage\right) > 21\ Arbeitstage$$

Bezug zu den Forschungsfragen

Es hat sich gezeigt, dass sich das implizite Wissen der Berater über die Konsistenz von Anforderungsdokumenten durch das erfahrungsbasierte RE-Werkzeug *KonPAss* externalisieren lässt. Eine große Rolle hat dabei gespielt, dass sich für die Berater ein direkter Nutzen dargestellt hat.

Q 1.1 *Akzeptieren die Nutzer das erfahrungsbasierte Werkzeug und schätzen sie es als nützlich ein?* Alle Reviewer waren sich einig, dass die Prüfung der Konsistenz am Besten computerbasiert erfolgen sollte. KonPAss kam bei seinen Nutzern gut an, die Einführung in den Projektbetrieb ist geplant.

Q 1.2 *Sind die Nutzer in der Lage, Erfahrungen (heur. Kritiken) zu bewerten?* Im Rahmen des Experiments wurden 124 Befunde in unter 2 Stunden bewertet. Die Nutzer gaben an, dass die Arbeit mit einer langen Liste von Befunden überraschend einfach ist. Die Bewertungen sind nützlich, um die heuristischen Kritiken nach ihrer Verlässlichkeit zu ordnen und gegebenenfalls zu überarbeiten.

Q 1.5. *Lernt die Organisation durch den Einsatz erfahrungsbasierter Werkzeuge mehr neue Erfahrungen als ohne?* H_1 hat gezeigt: KonPAss unterstützt relevante Fehlertypen. Damit konnte das Wissen über die Konsistenz von Anforderungswerkzeugen externalisiert werden. Verglichen mit der anfänglichen Spezifikation waren einige Regeländerungen nötig. Dabei wurden im Testeinsatz bereits wichtige Details zu den Konsistenzregeln gelernt. Die Anzahl der falsch-positiven Befunde unter den Reviewern zeigt, dass diese neue Erfahrung wichtig ist und sich im Projektalltag auszahlen kann.

Q 1.6 *Werden durch den Einsatz erfahrungsbasierter RE-Werkzeuge mehr Erfahrungen aus der Erfahrungsbasis der Organisation angewandt als ohne?* KonPAss hat mehr tatsächliche Befunde entdeckt, als manuelle Reviews. Die Bewertung der Befunde ist auch im tatsächlichen Einsatz vorgeschrieben. Es kann angenommen werden, dass diese Befunde zu einem ähnlichen Anteil berücksichtigt werden wie Befunde eines manuellen Reviews. Da KonPAss mehr Befunde findet und die Konsistenzprüfung mit KonPAss erheblich günstiger ist, findet durch KonPAss tatsächlich mehr Konsistenzerfahrung Anwendung.

Die Frage, ob Individuen durch KonPAss lernen, bessere Anforderungsdokumente zu erstellen (Teilziel 2) wurde in diesem Evaluationsszenario nur beiläufig untersucht.

Q 2.4 *Nimmt der Nutzer je nach Autorität des erfahrungsbasierten Werkzeugs, seiner eigenen Erfahrung und seines Typs kritisches Feedbacks des Werkzeugs überhaupt an?* Die Nutzer von KonPAss haben das Feedback klar angenommen und auch auf gelegentliche falsch-positive Befunde gelassen reagiert. Starke Autorität eines heuristischen Werkzeugs erfordert nach Definition 29 (Autorität) ein F-Maß von über 85%. Dieses wird von 5 der evaluierten Regeln nicht erreicht. Immerhin erreichen 6 der Regeln ein F-Maß von 100%. Insgesamt ergibt sich für KonPAss in dieser Evaluation ein mittlerer Recall ($rec = \frac{107}{154} = 0,69$), eine hohe Precision ($prec = \frac{107}{124} = 0,86$) und ein mittleres F-Maß ($fval = 0,77$).

KonPAss kann Konsistenzregeln genauso gut wie menschliche Reviewer überprüfen, erzeugt dabei jedoch deutlich weniger Aufwand. Dadurch stellt dieses Werkzeug eine große Hilfe zur Erstellung besserer Anforderungsdokumente dar. Ohne KonPAss ist im Review erheblich mehr Aufwand nötig, um alle Inkonsistenzen zu beseitigen. Das Teilziel 3 wird durch diese Evaluation also ebenfalls deutlich unterstützt.

Q 3.2 *Führt der Einsatz erfahrungsbasierter RE-Werkzeuge mit hoher Autorität dazu, dass Anforderungsdokumente weniger formale Fehler enthalten?* Besonders durch die hohe Präzision (wenige Falsch-Positive) erreicht KonPAss eine hohe Akzeptanz. Dadurch lässt sich im Projektbetrieb fordern, dass die Befunde (einiger Regeln) berücksichtigt werden sollen oder müssen. So können Fehler schon früh behoben werden, die sonst später erhebliche Kosten verursachen würden.

Q 3.5 *Sinken die Kosten für Qualitätssicherung durch den Einsatz erfahrungsbasierter RE-Werkzeuge?* Die Frage wurde hier eindeutig mit ja beantwortet: In einem realistischen Szenario lassen sich bis zu 21 Arbeitstage für die Prüfung einer Spezifikation sparen.

8.4 Zusammenfassung der Ergebnisse

Dieses Kapitel beschreibt die Evaluation der in dieser Arbeit vorgestellten Konzepte. Dazu wurden in Testumgebungen die Beispielimplementierungen aus Kapitel 7 in Evaluationsszenarien eingesetzt. Die hier vorgestellten Szenarien umfassen Laborversuche sowie studentische und industrielle Projekte. Auch wenn dies die Signifikanz der Ergebnisse nicht beeinflusst, so ist es doch als Pluspunkt zu bewerten, dass die in dieser Arbeit vorgestellten Konzepte Anforderungsdokumentation in realen Projekten verbessert haben. Mit diesen Testumgebungen (Einsatz einer Beispielimplementierung in einem) wurde folgendes evaluiert:

Tabelle 8.18: Übersicht der Evaluationsergebnisse zu Teilziel 1: Verbesserung des organisatorischen Lernens im Requirements Engineering.

Frage	*Beispielimpl.*	*Ergebnis*	*Formalität*
Q 1.1 Akzeptanz durch Nutzer?	KonPAss	Alle Reviewer waren sich einig, dass die Prüfung der Konsistenz am besten computerbasiert erfolgen sollte. KonPAss kam bei seinen Nutzern gut an, die Einführung in den Projektbetrieb ist geplant.	🏭, • •
Q 1.2 Sind Bewertungen heur. Krit. möglich?	HeRA.Glossar	Nutzer bewerten Vorschläge positiv, indem sie diese in das Glossar aufnehmen. Dies wurde bei 88 (von 93) Begriffen getan.	🏭, • •
	HeRA.Kritik	Die Möglichkeit zur Bewertung heuristischer Kritiken besteht [141], Studenten nutzen diese.	•
	KonPAss	Im Experiment wurden 124 Befunde in unter 2 Stunden bewertet.	🏭, • • •
Q 1.3 Können Nutzer heur. Krit. ändern?	HeRA.Kritik	Studenten sind in der Lage, heuristische Kritiken auf Anhieb richtig zu kodieren (86 % richtige Abgaben) und brauchen dazu je nach Schwierigkeitsgrad durchschnittlich 2-7 Minuten.	• • •
Q 1.4 Lernen durch Nutzer-Beobachtung?	HeRA.Glossar	Immerhin 6 von 93 Begriffen kommen in mehr als einem Glossar vor. Die Erfahrungsheuristik lernt damit relevante Erfahrung durch Beobachtung.	🏭, • •
Q 1.5 Lernt LSO mehr Erfahrungen?	HeRA.Glossar	Die wiederkehrenden Begriffe wären von der beobachteten Organisation ohne den Einsatz von HeRA.Glossar nicht erlernt worden.	🏭, • •
	HeRA.Kritik	Die Möglichkeit zur Eingabe neuer Erfahrungen besteht ([141] und Q 1.3), Studenten nutzen diese.	•
	KonPAss	KonPAss unterstützt relevante Fehlertypen, das Wissen über Konsistenz von Anforderungsdokumenten lässt sich also in KonPAss externalisieren. Durch die Arbeit mit heuristischen Kritiken werden Details der Konsistenzregeln sukzessive verbessert.	🏭, • • •
Q 1.6 Verwendet LSO mehr Erfahrungen?	HeRA.Kritik	Durch den Einsatz des Kritiksystems verbessert sich die Qualität der Use-Cases. Die beobachtete Organisation kann also ihre Erfahrung besser anwenden.	• •
	HeRA.EPK	Die EPK-Perspektive hilft, Inkonsistenzen und Lücken zu entdecken.	•
	KonPAss	KonPAss erlaubt es, mehr Konsistenzverstösse in weniger Zeit zu finden, als manuelle Reviews. Dadurch kann eine Organisation mit KonPAss mehr Erfahrungswissen zur Konsistenz von Anforderungsdokumenten anwenden, als ohne.	🏭, • • •

Teilziel 1: Heuristische und erfahrungsbasierte Werkzeuge unterstützen organisatorisches Lernen. Tabelle 8.18 zeigt die Evaluationsergebnisse zu Teilziel 1. Es ist mit den Beispielimplementierungen durchaus gelungen, das organisatorische Lernen zu unterstützen. Heuristische Werkzeuge haben dazu geführt, dass mehr Erfahrungen der beobachteten Organisationen angewandt wurden.

Auch die Aufnahme neuer Erfahrungen wurde verbessert. Der Einsatz heuristischer Kritiken bei der Dokumentation von Anforderungen führt dazu, dass diese kontinuierlich verbessert werden können. Um so wichtiger war es, festzustellen, dass Studenten mit relativ wenig Vorbereitung in der Lage sind, heuristische Kritiken anzupassen oder neu zu erstellen. Dies ist also durchaus eine Aufgabe, die man den Nutzern heuristischer Werkzeuge zumuten kann. Viele Nutzer sind überdies bereit, Feedback zu geben. Für ein vollständiges Bild sind Mechanismen möglich, mit denen die Nützlichkeit von heuristischem Feedback automatisch erfasst wird (zum Beispiel die automatische Protokollierung von akzeptierten Vorschlägen in HeRA.Glossar).

In KonPAss ist die Bewertung der heuristischen Kritiken sogar Teil der Methode: Bei der Anwendung von KonPAss ist ein Prozessschritt zur Analyse der Befunde vorgesehen. Dabei wird jeder Befund in kritisch, wichtig, unwichtig oder falschpositiv eingeordnet. KonPAss kann aus diesen Informationen direkt Statistiken zur Leistungsfähigkeit der einzelnen Regeln ableiten.

Teilziel 2: Heuristische und erfahrungsbasierte Werkzeuge unterstützen individuelles Lernen. Tabelle 8.19 zeigt die Evaluationsergebnisse zu Teilziel 2. Interessant ist, dass heuristische Werkzeuge ihren Nutzern mehr Sicherheit geben. Darüber hinaus gaben Nutzer bei Befragung an, dass sie durch die Verwendung der Werkzeuge gelernt haben. Dabei war vor allem das pro-aktive und interpretierende Werkzeug hilfreich. Eine der Überraschungen bei der Evaluation war der Zusammenhang zwischen Akzeptanz und Autorität. Nutzer sind durchaus bereit, Feedback von einem heuristischen Werkzeug anzunehmen, dessen Autorität schon auf Grund der schlechten Precision niedrig ist. Der Recall lässt sich bei der Glossar-Erweiterung kaum bestimmen, da die Anzahl der für ein Glossar relevanten Begriffe nicht allgemein ermittelt werden kann.

Teilziel 3: Heuristische und erfahrungsbasierte Werkzeuge führen zu qualitativ besseren Anforderungsdokumenten. Tabelle 8.20 zeigt die Evaluationsergebnisse zu Teilziel 3. Die Beispielimplementierungen haben zu qualitativ besseren Anforderungsdokumenten geführt. Ein wichtiges Resultat dieser Arbeit ist, dass dies kein Selbstzweck ist: Es existiert ein Zusammenhang zwischen guten Anforderungsdokumenten und dem Erfolg von Projekten.

Zudem reicht es nicht, Qualitätsmängel zu melden. Nutzer müssen diese Meldungen auch berücksichtigen. Bei besonders genauen heuristischen RE-Werkzeugen

Tabelle 8.19: Übersicht der Evaluationsergebnisse zu Teilziel 2: Verbesserung des individuellen Lernens im Requirements Engineering.

Frage	*Beispielimpl.*	*Ergebnis*	*Formalität*
Q 2.1 Lernen durch Pro-Aktivität	HeRA.Kritik	Studenten hatten mehr Vertrauen in die Qualität ihrer Use-Cases als die Vergleichsgruppe. Die Qualität der Use-Cases ist zudem wirklich besser.	• •
Q 2.2 Lernen durch Interpr.	HeRA.Kritik	Studenten gaben an, dass sie durch das wertende Feedback von HeRA.Kritik etwas gelernt haben.	• •
Q 2.3 Perspektiv-wechsel	HeRA.EPK	Die EPK-Perspektive hilft dem Nutzer, Inkonsistenzen und Lücken zu entdecken und senkt die dafür benötigte Zeit.	•
Q 2.4 Autorität und Akzeptanz	HeRA.Glossar	Nutzer akzeptieren Vorschläge von HeRA.Glossar (einstimmig) trotz sehr niedriger Precision (0,19)	🏭, •
	KonPAss	Nutzer akzeptieren KonPAss bei mittleren Recall (0,69), hoher Precision (0,86) und mittlerem F-Maß (0,77). Die Bearbeitung von durch KonPAss generierten Befunden soll Prozessvorgabe werden.	🏭, • •

kann man dies von den Nutzern fordern. In diesem Fall reicht es, zu zeigen, dass relevantes Feedback gegeben wird, da sichergestellt wird, dass auf dieses Feedback auch reagiert wird. Bei HeRA.Glossar und HeRA.Kritik musste zudem noch ermittelt werden, wie gut die Qualität der erstellten Anforderungsdokumente tatsächlich ist.

Insbesondere bei direktem, pro-aktiven Feedback stellt sich die Frage, ob durch die ständigen Unterbrechungen die genannten Qualitätsverbesserungen nicht durch eine erheblich höhere Bearbeitungszeit erkauft werden. Dieser Effekt konnte im Rahmen der hier vorgestellten Testumgebungen nicht beobachtet werden. Vielmehr gaben Nutzer in Selbstauskunftsbögen an, dass sich die Bearbeitungszeit eher gesenkt hat.

Eigenschaften erfahrungsbasierter Werkzeuge. Die zentrale Tabelle dieses Kapitels, Tabelle 8.1, ordnet den Evaluationszielen Testumgebungen zu. Die Matrix in Tabelle 8.1 gibt an, wie formal die GQM-Fragen zu den Teilzielen untersucht wurden.

Der einzige Faktor, der damit noch nicht ausreichend abgedeckt ist, ist der Einfluss des Nutzerprofils. In der jeweiligen Testumgebung wurde beschrieben, mit

Tabelle 8.20: Übersicht der Evaluationsergebnisse zu Teilziel 3: Verbesserung der Qualität von Anforderungsdokumenten.

Frage	*Beispielimpl.*	*Ergebnis*	*Formalität*
Q 3.1 Indikatoren und inhalt. Qualität		Es besteht ein statistisch relevanter Zusammenhang zwischen der Qualität der Anforderungsspezifikation und dem Projekterfolg. Die dazu notwendigen Qualitätsmetriken lassen sich weitgehend automatisch messen.	• •
Q 3.2 Reduziert Autorität Fehlerzahl?	KonPAss	Durch die hohe Akzeptanz lässt sich im Projektbetrieb fordern, dass KonPAss Befunde berücksichtigt werden sollen. So können Fehler schon früh behoben werden, die sonst erhebliche Kosten verursachen könnten.	🏭, • • •
Q 3.3 Reduziert Pro-Aktivität Fehlerzahl?	HeRA.Glossar	95 % der Begriffe in den untersuchten Glossaren wurden auf Basis der heuristischen Vorschläge aufgenommen. Die untersuchten Glossare enthielten darüberhinaus keine überflüssigen oder schlecht definierten Begriffe.	🏭, • •
	HeRA.Kritik	Durch den Einsatz des pro-aktiven Kritiksystems verbessert sich die Qualität der Use-Cases.	• •
Q 3.4 Erhöhen heur. Werkz. Erst. Kosten?	HeRA.Kritik	Nutzer berichten, dass sie mit Hilfe des Kritiksystems gleich schnell oder schneller Use-Cases schreiben, als ohne.	• •
Q 3.5 Senken heur. Werkz. Qual. Kosten?	HeRA.EPK	Nutzer berichten, dass sich die Konsistenzsicherung zwischen funktionalen Anforderungen der Nutzer und dem Geschäftsprozess durch die abgeleitete EPK schneller durchführen lässt.	•
	HeRA.Kritik	Das pro-aktive Kritiksystem ist eine gute Ergänzung analytischer Methoden des Qualitätsmanagements.	• •
	KonPAss	In einem realistischen Szenario lassen sich durch den Einsatz von KonPAss in der Qualitätssicherung bis zu 21 Arbeitstage sparen.	🏭, • • •

welchem Nutzertyp die Ergebnisse erzielt wurden. Daraus lässt sich aber nicht ableiten, welcher Personentyp am meisten (bzw. am wenigsten) von der Verwendung erfahrungsbasierter RE-Werkzeuge profitiert. Um diesen Einflussfaktor erschöpfend zu untersuchen, ist ein großflächiges Experiment mit vielen Teilnehmern und Beispielimplementierungen nötig.

So lange der Einfluss des Nutzertyps nicht geklärt ist, ist es schwer, einen bestimmten Typ von Werkzeug zu empfehlen. Dass alle hier vorgestellten Beispielimplementierungen positive Ergebnisse liefern liegt auch daran, dass sie weitestgehend von ihren Nutzern akzeptiert wurden. Aus Diskussionen ist jedoch ersichtlich geworden, dass das HeRA.Kritik in dem Kontext, in dem KonPAss bei der Daimler AG evaluiert worden ist, wohl nicht akzeptiert worden wäre. Umgekehrt konnten die Studenten in den Softwareprojekten der Leibniz Universität Hannover nicht viel mit der Konsistenzprüfung anfangen. Selbst der geringe Aufwand dafür stand in keinem Verhältnis zu dem noch geringeren Nutzen, die sehr kurzen Spezifikationen auf Konsistenz zu prüfen.

9 Vergleich und Abgrenzung zu relevanten Arbeiten

Verwandte Arbeiten gibt es zu drei Bereichen dieser Arbeit:

1. Verwandte Arbeiten zu automatisierter Analyse, Qualitätssicherung und dem Ableiten von Modellen aus Anforderungsdokumenten werden in Abschnitt 9.1 diskutiert.
2. Verwandte Arbeiten zur Unterstützung einer LSO durch Lernmodelle und erfahrungsbasierte Werkzeuge werden in Abschnitt 9.2 diskutiert.
3. Verwandte Arbeiten zur Klassifikation und Modellierung erfahrungsbasierter Werkzeuge werden in Abschnitt 9.3 diskutiert.

Dieses Kapitel stellt damit verwandte Arbeiten in den genannten Bereichen vor und erläutert Bezüge zu den hier vorgestellten Ergebnissen.

9.1 Heuristische Überprüfung von Anforderungsdokumenten

In der Literatur finden sich viele Hinweise darauf, wie computerbasiertes Feedback zu Anforderungsdokumenten die Qualität von Anforderungsdokumenten verbessert. Die meisten dieser Ansätze beschäftigen sich mit der automatischen Evaluation der Qualität von Anforderungsspezifikationen [59, 170, 171, 118].

Typischerweise wird in diesen Ansätzen zunächst ein Qualitätsmodell für Anforderungsspezifikationen aufgestellt. Dann werden Indikatoren für die Qualitätsaspekte definiert, die computerbasiert ausgewertet werden können, wie zum Beispiel in dem ARM Werkzeug, das von Wilson et. al. vorgestellt wurde [170, 171]. Diese Indikatoren können zu möglichen Befunden in einer Anforderungsspezifikation führen, und entsprechen damit den Heuristiken in dieser Arbeit. Indikatoren werden meistens durch Schlüsselwortlisten ausgedrückt, können aber in seltenen Fällen auch ausgereifte Analyse der natürlichen Sprache beinhalten (zum Beispiel die Suche nach *Unterspezifikation* im QuARS Werkzeug [59, 58]).

Fabrini et al. berichten, dass die Kriterien des QuARS Werkzeugs eine Anforderungsspezifikation effektiv beurteilen können [58]. Sie behaupten außerdem, dass das QuARS Werkzeug nicht nur dabei hilft, die Anforderungsqualität zu messen, sondern diese auch verbessert. Zu dieser Behauptung kann allerdings keine quantitative Evaluation gefunden werden. Fantechi et al. haben das ARM und das QuARS Werkzeug insbesondere auch auf Use-Cases angewandt [60]. Demnach

eignen sich Use-Cases auf Grund ihrer Struktur besonders gut für die automatische Analyse. Melchisedech geht mit seinem Ansatz noch einen Schritt weiter: ADMIRE [118] basiert auf spezifischen Prozess- und Informationsmodellen, die es erlauben, die Zusammenhänge zwischen Anforderungen näher zu untersuchen. Dies setzt allerdings voraus, die entsprechenden Modelle als Grundlage für das Vorgehen und die Struktur des Anforderungsdokuments zu wählen. Im Gegensatz zu dieser Arbeit sehen ARM, QuARS und ADMIRE keine Erweiterbarkeit des zu Grunde liegenden Qualitätsmodells durch den Nutzer vor und Zielen auf die analytische Konsistenzsicherung ab.

Breaux und Antón arbeiten an der Extraktion von Anforderungen aus Gesetzestexten (z.B. in [32]). Wie bei den heuristischen Kritiken dieser Arbeit kommen dabei Techniken der Analyse natürlicher Sprache zum Einsatz. Das halbautomatische Verfahren unterstützt Nutzer einer LSO auch bei diesem Ansatz nicht dabei, neue Erfahrungen zu kodieren. Von den hier vorgestellten Beispielimplementierungen hat KonPAss die meisten Ähnlichkeiten. Es wäre vorstellbar, Breauxs und Antóns Verfahren in KonPAss zu integrieren, um so prüfen zu können, ob eine Spezifikation konsistent zu einem Gesetzestext ist.

Somé beschreibt eine Methode und ein Werkzeug, mit dem sich Use-Case-Modelle systematisch erstellen lassen [159]. Der Methode liegt ein striktes Metamodell für die Struktur der Use-Cases und ihrer Zusammenhänge sowie eine Grammatik für die natürlichsprachlichen Beschreibungen zu Grunde. Durch die Kontrolle der natürlichen Sprache können einige Ungenauigkeiten und Inkonsistenzen vermieden werden. Die formale Struktur erlaubt es, die in den Use-Cases beschriebenen Abläufe schon früh zu simulieren. Dies wird durch das Werkzeug UCed unterstützt. Im Gegensatz zu den Ansätzen in dieser Arbeit werden erheblich höhere Anforderungen an die Formalität der Dokumente gestellt. Der Nutzer hat viel geringere Freiheitsgrade bei der Formulierung von Anforderungen. Eine Erweiterbarkeit der Regeln für die Analyse der natürlichen Sprache ist zudem nicht vorgesehen.

Chantree arbeitet an der Identifikation von Mehrdeutigkeiten in natürlichsprachlichen Texten [38], insbesondere bei Konstruktionen mit *und / oder* [39]. Er schlägt ein Werkzeug vor, mit dem Nutzer interaktiv die Mehrdeutigkeiten in natürlichsprachlichen Dokumenten analysieren kann. An diesem Ansatz sind zwei Aspekte besonders interessant: Zum Einen gibt Chantree ein Maß dafür, wie gefährlich eine Mehrdeutigkeit ist. Zum Anderen kann das vorgeschlagene Werkzeug dieses Maß durch Analyse der Reaktion des Nutzers anpassen. Die interaktive Entfernung von Mehrdeutigkeiten ist jedoch ein eigener Prozessschritt und nicht konstruktiv in die Erstellung der Anforderungsdokumente eingebunden, wie dies beispielsweise in HeRA.Kritik der Fall ist.

Kiyavitskaya et al. beschreiben Anforderungen an (heuristische) Werkzeuge zur Identifikation von Mehrdeutigkeiten [90]. Ausgehend von einer Menge allgemeiner Mehrdeutigkeitsmaße untersuchen sie, welche Mehrdeutigkeiten werkzeuggestützt

identifiziert werden können und vergleichen die Resultate mit manuellen Prüfungen. Die Resultate deuten auf eine geringe Autorität hin – aufgedeckte Mehrdeutigkeiten scheinen nicht schlimm zu sein, schlimme Mehrdeutigkeiten werden zum Teil nicht entdeckt. Kiyavitskaya et al. leiten daraus ein prinzipielles Problem für solche Werkzeuge ab und fordern 100% Recall. Auf Basis der Ergebnisse in dieser Arbeit scheint es vielversprechender zu sein, das computerbasierte Feedback geschickt in den Prozess einzubinden. Eine hohe Pro-Aktivität solcher Werkzeuge kann einen geringen Recall kompensieren. Der Analyst wird nicht seiner Verantwortung enthoben, das vollständige Dokument zu überprüfen, das Werkzeug hilft ihm gelegentlich dabei. Die von Kiyavitskaya et al. verwendeten Mehrdeutigkeitsmaße basieren größtenteils auf Sprachkorpi. Eine projektspezifische Anpassbarkeit wird nicht direkt erwähnt und die Lernfähigkeit ist laut Kiyavitskaya et al. keine Anforderung an Werkzeuge [90] .

Svetinovic et al. beschreiben die Vorteile, wenn globale Kontrollflussmodelle aus Use-Cases abgeleitet werden [163] (siehe Abschnitt 8.2.1). Das Ableiten solcher Modelle ist eine wertvolle Fähigkeit, die Übersicht schafft und im Sinne von Fischers Vorschlag einer Simulationskomponente einer DODE eine *was-wenn*-Analyse erlaubt [64]. Das Ableiten von Modellen ist zudem einfacher, als Anforderunsmodelle zu simulieren. Seybold et al. beschreiben, wie Simulation bei der Dokumentation von Anforderungen eingesetzt werden kann, um Anforderungsmodelle zu validieren [155]. Simulation hilft demnach Anforderungen besser zu entwickeln. Modelle im Requirements Engineering sind aber erst dann für Simulation ausreichend vollständig und formal, wenn die Anforderungsanalyse abgeschlossen ist. Seybold et al. schlagen vor, Ideen aus dem Testen von Software zu übernehmen (zum Beispiel sogenannte Driver, Stubs sowie Regression), um die Simulation von halbformalen und unvollständigen Modellen zu erlauben. Sie präsentieren eine auf der Modellierungssprache ADORA basierende Umsetzung dieser Idee. Diese Arbeit hat eine große Ähnlichkeit zu HeRA.EPK und unterstreicht den Bedarf an solchen Mechanismen.

Jang schlägt eine formale Anforderungsspezifikationssprache vor, die auf Mechanismen des Wissensmanagement basiert [84]. Durch die an Prolog erinnernde Syntax der Anforderungssprache wird es einfach, Spezifikationen automatisch zu analysieren. So kann beispielsweise untersucht werden, ob die spezifizierten Aktivitäten erreicht werden können, ob alle spezifizierten Bedingungen notwendig sind oder ob es bestimmte Arten von Inkonsistenzen gibt. Einen ähnlichen Ansatz verfolgen Hunter und Nusseibeh [80]. Auch hier werden Anforderungen auf Basis von logischen Ausdrücken dokumentiert und überprüft. Um die kontinuierliche Weiterentwicklung von Anforderungsbeschreibungen zu unterstützen, erweitern sie diese *klassische Logik*, um automatisches Reasoning auf inkonsistenten Anforderungsbeschreibungen zu erlauben. Es ist anzunehmen, dass auf Basis formaler Spezifikationssprachen erheblich mächtigere computerbasierte Analyse möglich ist (vergleiche auch Diskussion in Abschnitt 4.2). Im Vergleich ist der Beitrag

dieser Arbeit in einem Konzept zu finden, mit dem sich schwächere Formen solcher Analysearten auf weniger formalen Anforderungsbeschreibungen anwenden lassen.

Interessante Erweiterungen des logischen Ansatzes erlauben es, natürlichsprachliche Anforderungsbeschreibungen bei Bedarf in eine formale logische Spezifikation zu überführen und zu verifizieren, wie zum Beispiel bei Gervasi et al. [71, 72]. Dieser Ansatz weist große Ähnlichkeiten zu dem Vorgehen bei KonPAss auf: Zunächst wird Stil, Struktur und Sprache der Anforderungsspezifikation definiert, um das Parsen der natürlichsprachlichen Anforderungsbeschreibungen zu erlauben. Regeln und Modelle für die zu überprüfenden Aspekte müssen ebenfalls eingerichtet werden, können aber genau wie bei KonPAss über die vorliegende Anforderungsspezifikation hinaus wiederverwendet werden. Das Einlesen der Dokumente erfolgt im Gegensatz zu KonPAss halbautomatisch, dafür enthält das Zielmodell jedoch wesentlich mehr semantische Information. Da Anforderungen in englischer Sprache vorliegen, können Gervasi et al. zudem auf mächtigere Analysemöglichkeiten zurückgreifen. Insgesamt ist Gervasis Ansatz damit mit größerem Freiheitsgrad und höherer Mächtigkeit sowie niedrigerem Automatisierungsgrad als KonPAss gekennzeichnet. Der Ansatz in dieser Arbeit, Prüfung von Anforderungsdokumenten direkt an ein Werkzeug zu binden, erlaubt im Vergleich dazu eine bessere Unterstützung des Nutzers und die Integration erfahrungsbasierter Konzepte zur kontinuierlichen Verbesserung. Zudem verhindert der höhere manuelle Aufwand des Ansatzes von Gervasi et al. direktes, konstruktives Feedback an den Analysten.

Tjong et al. beschreiben *Natural Language Requirements Patterns*, eine Menge von Mustern in natürlichsprachlichen Anforderungsspezifikationen, die es zu vermeiden gilt [165]. Aus diesen Mustern leiten sie eine Menge von Richtlinien ab, die es einzuhalten gilt, um gute Anforderungsdokumente zu erstellen. Diese Richtlinien sind eine hervorragende Quelle für das *Seeding* der Erfahrungsbasen der hier vorgestellten Ansätze. Auch Tjong et al. deuten die Möglichkeit, die Einhaltung der Richtlinien automatisch zu prüfen. Eine systematische und kontinuierliche Verbesserung der Richtlinien ist jedoch nicht vorgesehen.

Berenbach schlägt einen Ansatz zur automatischen Prüfung von UML-Modellen vor [19, 20]. Dieser Ansatz verwendet Heuristiken, um ein Analysemodell für UML-Modelle zu erstellen. Mit diesem Analysemodell kann überprüft werden, ob UML-Anforderungsmodelle eine ausreichende Qualität haben, um zum Beispiel Verifikationstechniken anzuwenden oder Anforderungen automatisch zu extrahieren. Laut Berenbach können diese Heuristiken entweder interaktiv durch den Modellierer oder analytisch durch die Qualitätssicherung eingesetzt werden. Souza et al. schlagen vor, Kritiksysteme zu verwenden, um Dokumente des Software Engineering auf Inkonsistenzen zu prüfen [50]. Im Vergleich zu abstrakten Zustandsautomaten, wissensbasierten Systemen und prädikatenlogischen Ansätzen heben

sich Kritiksysteme demnach ab, weil sich durch die relativ kleinen Kritiken effiziente und skalierbare Konsistenzprüfung realisieren lässt. Sie demonstrieren diese Mechanismen mit DAISY, einer kritikbasierten Entwicklungsumgebung, die über ein früheres Kritiksystem (ABCDE-Critic, [160]) hinausgeht. Sowohl DAISY als auch ABCDE-Critic beziehen sich auf die Erstellung von UML-Diagrammen. Zumindest ABCDE-Critic erlaubt es seinen Nutzern, Kritiken selbst einzugeben oder anzupassen. Die Kritiken beziehen sich in der Hauptsache darauf, ob ein UML-Diagramm wohl geformt ist und ob es zu bereits existierenden UML-Diagrammen passt. Im Vergleich zu den Arbeiten von Berenbach und Souza et al. liegt der Fokus dieser Dissertation darauf, ähnliche Mechanismen für textuelle Anforderungsbeschreibungen zur Verfügung zu stellen (vergleiche Abschnitt 4.2).

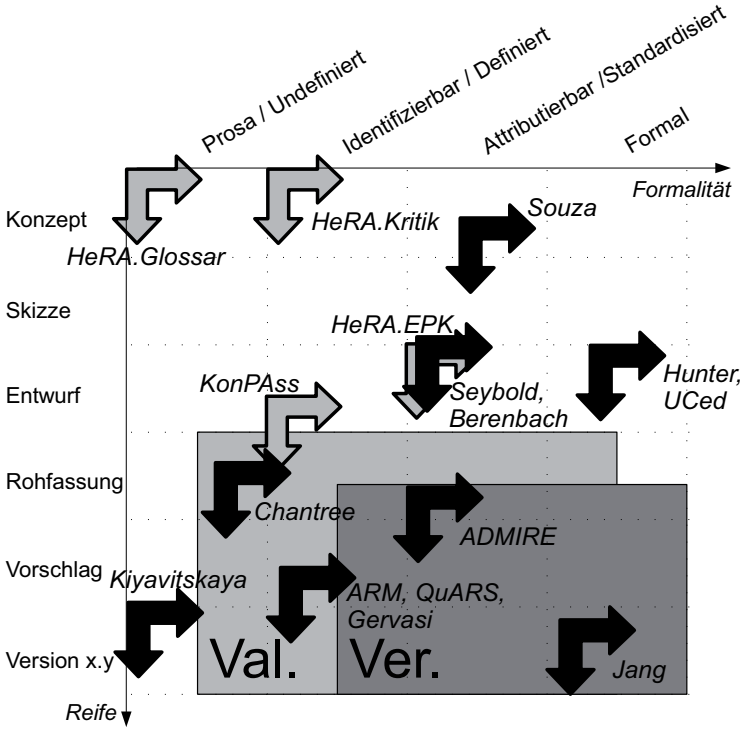


Abbildung 9.1: Notwendige Reife und Formalität von Anforderungsdokumenten für den Einsatz der vorgestellten Arbeiten.

Abbildung 9.1 ordnet die in diesem Abschnitt vorgestellten Arbeiten in das Modell der Reife und Formalität von Anforderungsdokumenten ein. Die Pfeile geben an, ab welcher Formalität und Reife ein Einsatz sinnvoll ist und markieren die linke obere Ecke der entsprechenden Flächen im Steckbrief der Beispielimplementierungen (vergleiche Kapitel 7). Die Abbildung zeigt, dass die meisten verwandten Arbeiten eher analytisch als konstruktiv gemeint sind. Die größte Ähnlichkeit haben hier Arbeiten von Berenbach, Hunter, Seybold et al. sowie Souza et al., da sie

Tabelle 9.1: Vergleich der Beispielimplementierungen mit verwandten Arbeiten.

Name	Autorität	Pro-Akt.	Interpret.	Lernfähig	Perspekt.
HeRA.EPK	●●●○○	●○○○○	●○○○○	○○○○○	●●●●○
HeRA.Glossar	●●○○○	●●●●○	●●●○○	●●●●●	●○○○○
HeRA.Kritik	●●●○○	●●●●●	●●●◐○	●●●○○	●○○○○
KonPAss	●●●●●	○○○○○	●●●◐○	●○○○○	●●●○○
ARM [170, 171]	●●●●○	○○○○○	●●●○○	●●○○○	○○○○○
QuARS [59, 58]	●●●●○	○○○○○	●●●○○	○○○○○	○○○○○
ADMIRE [118]	●●●●○	○○○○○	●●●●○	○○○○○	●●●○○
Breaux et al. [32]	●●○○○	○○○○○	●●●○○	○○○○○	●●●○○
Chantree [38, 39]	●●○○○	●○○○○	●●●●●	●●●●○	○○○○○
Kiyavitskaya et al. [90]	●○○○○	○○○○○	●●●●●	○○○○○	○○○○○
Seybold et al. [155]	●●●○○	●○○○○	●○○○○	○○○○○	●●●●○
Jang [84]	●●●●●	○○○○○	●●○○○	○○○○○	○○○○○
Hunter [80]	●●●●●	○○○○○	●●○○○	○○○○○	○○○○○
Gervasi et al. [71, 72]	●●●●●	○○○○○	●●●○○	○○○○○	○○○○○
Berenbach [19, 20]	●●●○○	●●●○○	●●●○○	○○○○○	●●○○○
Souza et al. [160, 50]	●●●○○	●●●○○	●●○○○	●●○○○	●●○○○
UCed [159]	●●●●○	●○○○○	●●●●○	○○○○○	●●●●○

explizit den Einsatz auf inkonsistenten Dokumenten thematisieren. Da diese Ansätze meist das Ziel verfolgen, zumindest partielle Konsistenz herzustellen, wird auch ein gewisser Inhalt vorausgesetzt. Durch den Fokus auf grafische Notationen wird bei diesen Arbeiten eine eher hohe Formalität vorausgesetzt. Dies gilt auch für die Arbeit um den UCed von Somé [159]: Die große Ähnlichkeit in Zielsetzung und Aufbau dieses Use-Case-Editors zu HeRA endet bei der hohen angestrebten Formalität der durch eine Grammatik kontrollierten natürlichen Sprache, sowie an der bei Somé fehlenden Lernfähigkeit.

Tabelle 9.1 stellt den Vergleich zu den Beispielimplementierungen in Kapitel 8 auf Basis der relevanten Eigenschaften erfahrungsbasierter RE-Werkzeuge an. Das wichtigste Alleinstellungsmerkmal dieser Arbeit ist hier die Pro-Aktivität und Lernfähigkeit. Einordnung und Vergleich in Abbildung 9.1 und Tabelle 9.1 basieren auf den zitierten Veröffentlichungen und persönlicher Einschätzung.

9.2 Erfahrungen in einer LSO nutzen

9.2.1 Verwandte Lernmodelle

In Kapitel 5 wurde ein Lernmodell vorgestellt. Dieses Modell zeigt, wie Individuen und Organisationen mit Hilfe von erfahrungsbasierten Werkzeugen lernen können, die Dokumentation von Anforderungen zu verbessern. Dieses Modell wird nun

in Zusammenhang mit anderen Lernmodellen gestellt. Im Vergleich zu den hier vorgestellten Arbeiten grenzt sich diese Disseration vor allem durch den konkreten Bezug auf die Domäne des Requirements Engineering ab.

Earl hat ein Framework eingeführt, das Ansätze zu Wissens- und Erfahrungsmanagement auf Basis der Forschungszielsetzung klassifiziert [54]. Earl unterscheidet dabei verschiedene sogenannte *Schulen*:

- Technokratische Ansätze, die sich mit Systemen, Kartographie (Landkarten) oder mit der ingeniersmäßigen Verarbeitung von Wissen auseinander setzen.
- Die Ökonomische Schule, die sich mit dem kommerziellen Wert von Wissen beschäftigt.
- Verhaltensorientierte Ansätze, die sich mit organisatorischen, räumlichen und strategischen Aspekten von Wissen auseinander setzen.

Heuristische Werkzeuge lassen sich damit als technokratischer, systemorientierter Ansatz klassifizieren. Der Fokus ist die Technik (das Werkzeug), mit dem innerhalb einer Domäne das Ziel verfolgt wird, eine Wissensbasis aufzubauen, zu pflegen und zu nutzen, indem Erfahrungen geeignet kodiert werden.

Heuristische Werkzeuge erleichtern zudem auch einen ingeniersmäßigen Ansatz, bei dem der Fokus auf den Prozessen liegt. In dieser Schule soll die Reife von Aktivitäten durch Erfahrungsflüsse verbessert werden. Wie in dieser Arbeit gezeigt wird, können heuristische Werkzeuge genutzt werden, um Erfahrungsflüsse gezielt zu steuern.

Indem Perspektivwechsel erlaubt werden, können heuristische Werkzeuge auch in der verhaltensorientierten, organisatorischen Schule eingesetzt werden. Durch die verschiedenen Perspektiven können Gruppen mit unterschiedlichem Hintergrund ihr Wissen zusammenlegen und vernetzen. Zusammenarbeit wird so durch heuristische Werkzeuge indirekt unterstützt und ein positiver Effekt auf organisatorische Ansätze ist zu erwarten.

Argyris und Schön haben die Begriffe des Single- und Double-Loop-Learning geprägt [8, 9]. Im Single-Loop-Learning lernt eine Person oder eine Organisation, ein festes Ziel immer effektiver und effizienter zu erreichen, beispielsweise gute Anforderungen zu schreiben. Nun kann sich das Ziel aber verändern, zum Beispiel, wenn Kostendruck in einer Organisation wichtiger wird, oder sich die Definition „guter Anforderungen“ verändert. Ansätze, die auch eine Veränderung dieses übergeordneten Ziels (der sogenannten *steuernden Variable*, englisch: *governing variable*) erlauben, nennen sich nach Agyris und Schön Double-Loop-Learning.

Der in Abschnitt 5.2.3 vorgestellte Lebenszyklus von heuristischen Erfahrungen beinhaltet mehrere Schritte, in denen Erfahrungen angepasst werden können. Spä-

testens in der Konsolidierungsphase können veraltete Erfahrungen aussortiert werden. Prinzipiell ist damit Double-Loop-Learning möglich, dies hängt jedoch davon ab, ob der Prozess richtig durchgeführt wird und liegt außerhalb der Grenzen der Werkzeuge. Eine wichtige Unterstützung für Double-Loop-Learning bieten heuristische Werkzeuge, indem sie Feedback vom Nutzer erlauben. Kommentare bieten Nutzern die Möglichkeit, steuernde Variablen an einem konkreten Gegenstand zu diskutieren. Negativ kann sich hier jedoch auswirken, wenn ein erfahrungsbasiertes Werkzeug stark an einer steuernden Variabel ausgerichtet ist.

Beispiel 29: *Analyse der Ableitung einer EPK aus Sicht des Double-Loop-Learning.*

Die Inferenz einer EPK aus Use-Cases folgt der steuernden Variable, dass die Menge der Use-Cases einen implizierten globalen Prozess darstellen soll. Verliert dieses Ziel an Bedeutung, verliert auch dieses Werkzeug an Wert.

Ein weiteres bekanntes Lernmodell stammt von Kolb [107]. Auch Kolb modelliert den Prozess des Lernens als Kreislauf. Im Gegensatz zu anderen Modellen zeichnet Kolbs *Lernkreislauf* aus, dass er sehr detailliert ist und aktives Lernen betont. Ein *aktives Experiment* dient dazu, etwas Neues herauszufinden und führt zu einer *konkreten Erfahrung*, die der Beobachtung als Bestandteil einer Erfahrung nach Schneider entspricht (vergleiche [151]). Im nächsten Schritt entsteht nach Kolb eine *reflektive Beobachtung* und schließlich eine *abstrakte Konzeptionalisierung*. Diese beiden Schritte entsprechen der Schlussfolgerung oder Hypothese als Bestandteil einer Erfahrung nach Schneider (vergleiche [151]). Interessant ist hier bei Kolb insbesondere die Trennung dieser beiden Schritte, die noch einmal zur Schärfung des Lernmodells in Abschnitt 5.2.1 genutzt werden kann (siehe Tabelle 9.2). Tabelle 9.2 stellt an Hand Kolbs *Lernkreislauf* exemplarisch dar, was mit Hilfe von erfahrungsbasierten Werkzeugen gelernt werden kann. Dabei sind zwei Perspektiven interessant. Zum Einen lernt der Nutzer eines heuristischen Werkzeugs, zum Anderen wird auch die Erfahrungsbasis des heuristischen Werkzeugs verbessert. Die neue Erfahrung als Ergebnis eines Durchlaufs durch Kolbs *Lernkreislauf* fließt in den nächsten Durchlauf ein, in dem sie Grundlage für das nächste *aktive Experiment* sein kann.

Nonaka und Takeuchi haben ebenfalls ein Modell für den Lebenszyklus von Wissen in einer lernenden Software erstellenden Organisation aufgestellt [121]. Die Zustände von Wissen in diesem Modell beziehen sich zunächst darauf, ob das Wissen explizit vorliegt oder *tacit*, also nur stillschweigend und unbewusst verfügbar ist. Für den Lebenszyklus ist vor allem die Weitergabe von Wissen von Interesse. In Abbildung 9.2 sind die vier prinzipiellen Arten der Weitergabe dargestellt:

Tabelle 9.2: Lernen durch Verwendung eines heuristischen Werkzeugs am Beispiel von Kolbs *Lernkreislauf* [107].

Lernphase nach Kolb [107]	**Nutzer-Perspektive (individuelles Lernen)**	**Werkzeug-Perspektive (organisatorisches Lernen)**
Aktives Experiment	Use-Case dokumentieren.	Use-Case analysieren und Feedback geben.
Konkrete Erfahrung	Beobachtung einer Warnung auf Basis einer heuristischen Kritik.	Beobachtung der Reaktion auf Feedback (wird berücksichtigt oder ignoriert).
Reflexive Beobachtung	Erkennen des Fehlers, der zur Warnung geführt hat.	Erkennen, ob Feedback relevant oder irrelevant ist.
Abstrakte Konzeptionalisierung	Verbesserte Erkenntnis, was einen guten Use-Case ausmacht.	Verbesserte heuristische Kritiken.

Sozialisation. Wenn Wissen direkt von einer Person zu einer anderen transferiert wird, nennt man dies *Sozialisation*. Dies ist zum Beispiel der Fall, wenn Personen miteinander reden oder zusammenarbeiten. Auslöser hierfür im Sinne dieser Arbeit können automatisch abgeleitete Modelle sein, die eine andere Perspektive auf dokumentierte Anforderungen erlauben. Auch heuristische Kritiken können zu einer Diskussion führen, die einen regen Wissensaustausch erlaubt.

Externalisation. Wenn stillschweigendes (tacit) Wissen einer Person explizit dokumentiert wird, so nennt man dies *Externalisation*. Im Rahmen dieser Arbeit ist dies vor allem dann gegeben, wenn heuristische Kritiken kommentiert, bewertet, deaktiviert, verändert oder neu erstellt werden.

Kombination. Wenn explizites Wissen neu kombiniert wird und dadurch eine systematischere explizite Dokumentation von Wissen entsteht, so nennt man dies *Kombination*. Dies ist das Hauptziel der Konsolidierungsphase von Erfahrungen und Kritiken.

Internalisation. Wenn explizit dokumentiertes Wissen in implizites (und später womöglich stillschweigendes Wissen) umgewandelt wird, nennt man dies *Internalisation*. Im Rahmen dieser Arbeit wird dies durch werkzeuginduzierte Breakdowns erreicht, wenn heuristische Kritiken feuern und dadurch die konkrete Anwendbarkeit von Wissen signalisieren.

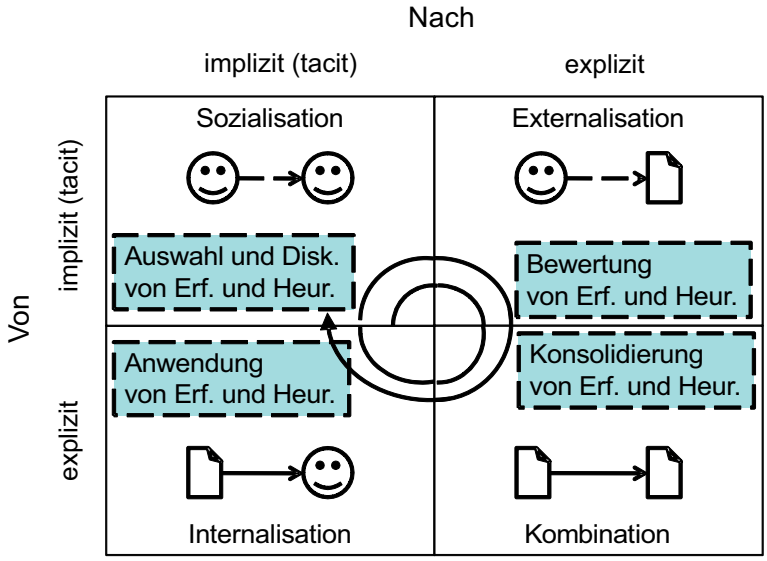


Abbildung 9.2: Weitergabe von Wissen nach Nonaka und Takeuchi [121].

9.2.2 Domain Oriented Design Environments

HeRA basiert auf Fischers Architektur für Domain Oriented Design Environments (DODE) [64], die in Abschnitt 2.2.4 detailliert beschrieben wird. Der grundlegende Aufbau einer DODE ist in Abbildung 2.3 dargestellt. Drei der vier Informationskreisläufe aus dem Referenzmodell in Abschnitt 6.2.4 werden von einer DODE unterstützt:

- *Erfahrungskreislauf:* Die Argumentationskomponente lernt aus der Argumentation mit dem Nutzer und kann so besser werden.
- *Direktes heuristisches Feedback:* Die Analyse des Entwurfs durch die Argumentationskomponente führt in einer DODE zu heuristischem Feedback nach der Definition in dieser Arbeit.
- *Feedback durch neue Perspektiven:* Dies entspricht der Aufgabe der Simulationskomponente in der DODE-Architektur.

Die DODE-Architektur erfüllt damit die Anforderung an ein erfahrungsbasiertes Werkzeug nach Definition 20, mindestens einen der folgenden Aspekte zu unterstützen:

- *Lernen:* Das Erzeugen von neuem Wissen und Erfahrungen ist durch die Argumentationskomponente erfüllt.
- *Evaluieren:* Das Feedback, das ein Nutzer zu den Kritiken gibt, kann als

Tabelle 9.3: Vergleich einer DODE mit HeRA. Zur besseren Vergleichbarkeit werden HeRA.Kritik, HeRA.Glossar und HeRA.EPK hier kombiniert, da HeRA erst so die wichtigsten Komponenten einer DODE besitzt.

Name	Autorität	Pro-Akt.	Interpret.	Lernfähig	Perspekt.
HeRA.EPK	●○○○○	●○○○○	●○○○○	○○○○○	●●●●○
HeRA.Glossar	●●○○○	●●●●○	●●●○○	●●●●●	●○○○○
HeRA.Kritik	●●●○○	●●●●●	●●●◐○	●●●○○	●○○○○
HeRA	●●●○○	●●●●●	●●●◐○	●●●●●	●●●●○
DODE	●●●○○	●●●●●	●●●●○	●●●●●	●●●●○

Evaluation vorhandenen Wissens gewertet werden. Ungeeignete oder gute Argumentationen werden so identifiziert.

- *Verwalten:* Verteilung, Aktualisierung und Überarbeitung von vorhandenem Wissen und vorhandenen Erfahrungen wird von einer DODE nicht direkt unterstützt. Fischer verweist bei diesem Aspekt auf den SER-Zyklus, der außerhalb der DODE betrieben werden muss.
- *Anwenden:* Die Hauptaufgabe einer DODE ist es, die Nutzung von Erfahrung und Wissen in einer speziellen Domäne voranzutreiben.

Auf dieser Grundlage lässt sich nach dem Steckbrief dieser Arbeit eine Charakterisierung für die DODE Architektur geben. Die Ausprägung der *Autorität* einer DODE liegt per Definition im Mittelfeld. Die Argumentationskomponente nimmt dem Nutzer keine Entscheidungen ab, sondern regt ihn zur Diskussion und Reflexion an. Eine DODE ist vor allem *pro-aktiv*, *interpretierend* und *lernend*. Die Möglichkeit des *Perspektivwechsels* ist durch die Simulationskomponente gegeben. Der *Formalitätsgrad der Artefakte*, die in der Konstruktionskomponente erzeugt werden, ist eher hoch, da auf Basis dieser Artefakte eine Simulation möglich sein soll.

Tabelle 9.3 zeigt die Charakterisierung von HeRA im Vergleich zu einer DODE. Die Eigenschaften von HeRA sind in ihrer Ausprägung weitgehend identisch zu einer DODE. Unterschiede gibt es bei der Interpretationsleistung sowie im unterstützten Formalitätsgrad der Artefakte.

HeRAs Kritiken sind bislang noch nicht so interpretierend, wie dies für eine DODE angemessen wäre. Das liegt zum Teil daran, dass sich viele heuristische Kritiken auf die deutsche Sprache beziehen, für deren Verarbeitung wichtige Infrastruktur fehlt (z.B. gute und günstige Part-of-Speech-Tagger).

An den Formalitätsgrad der Artefakte stellt HeRA weitaus geringere Anforderungen als eine typische DODE. Das ist wichtig, da Anforderungen nicht direkt formal dokumentiert werden können. Um dies zu erreichen, wurde in HeRA teilweise dar-

auf verzichtet, Simulationskomponenten im Sinne einer DODE zu erstellen. An deren Stelle treten abgeleitete Perpektiven wie UML Use-Case-Diagramme und die Berechnung der Use-Case-Points.

9.3 Eigenschaften und Klassifikation erfahrungsbasierter Werkzeuge

Die in dieser Arbeit vorgestellten Beispielimplementierungen lassen sich als Kritiksysteme klassifizieren. Kritiksysteme ähneln in Zielsetzung und Aufbau Empfehlungssystemen (englisch: Recommender Systems, vergleiche [156]) und können als Teilmenge dieser adaptiven Systeme gesehen werden. Ein Empfehlungssystem generiert automatisch Vorschläge oder Empfehlungen. Basierend auf Eingabedaten (in der Regel durch einen Nutzer) werden aus Hintergrunddaten möglichst gute Empfehlungen erzeugt (in der Regel für den entsprechenden Nutzer). Dies kann nach Huang et al. im Requirements Engineering beispielsweise genutzt werden, um die Negotiation bei sehr großen Systemen zu unterstützen: Basierend auf den Wünschen, die ein Stakeholder zum Beispiel in einem Forum geäußert hat, können weitere relevante Wünsche anderer Stakeholder identifiziert werden [41, 37]. Kritiksysteme zeichnen sich dadurch aus, dass diese Vorschläge sich direkt auf die Aktivität des Nutzers beziehen, meistens wertend sind und das Nutzerziel unterstützen sollen. Die Konzepte sind daher sehr ähnlich, der Fokus eines Empfehlungssystems ist jedoch schwächer abgegrenzt.

Kritiksysteme sind kein neues Konzept. Über die letzten Jahre haben sich zahlreiche Autoren mit diesem Thema beschäftigt [63, 65, 157, 138, 136, 50, 122]. Diese Autoren beschreiben bestimmte Eigenschaften, die ein Kritiksystem haben sollte. Norhayati Mohd Ali et al. geben einen Überblick über existierende Kritiksysteme [6] und ordnen sie in eine Taxonomie ein (siehe Tabelle 9.4). Dieser Überblick geht auf ähnliche Überblicke von Robbins [138, 136], Silvermann [157] und Oh [122] zurück und ergänzt diese.

Die Domäne der in dieser Arbeit vorgestellten Beispielimplementierungen ist das Requirements Engineering. Bei den Autorenwerkzeugen sind recht unterschiedliche Konzepte entstanden. HeRA.Kritik erlaubt es Nutzern in einem speziellen Editor mit Syntax-Highlighting Kritiken zu ändern. Zusätzlich stehen Assistenten zur Verfügung, um Standard-Kritiken zu erstellen. In KonPAss sind die Kritiken in Java implementiert, für Änderungen muss KonPAss neu kompiliert werden. Immerhin können Nutzer die Parameter der Regeln mit einem speziellen Assistenten anpassen. Dieser erlaubt es zum Beispiel, reguläre Ausdrücke über eine grafische Benutzerschnittstelle einzustellen. Bei der Umsetzung der Kritiken hat sich durch die spezielle Zielgruppe (hauptsächlich Studenten und Softwareentwickler mit guten Java-Kenntnissen) die Programmiersprache Java (beziehungsweise

Tabelle 9.4: Klassifizierungsschema für Kritiksysteme (nach Ali et al. [6]).

Aspekt	**Beschreibung**	**Werte**
Kritikdomäne	Die Domäne des Problems, auf das die Kritiken angewandt werden.	Textuelle Beschreibung
Autorenwerkzeug für Kritiken	Die Möglichkeit, wie Nutzer / Designer / Endbenutzer kritische Regeln hinzufügen, entfernen, ändern, aktivieren oder deaktivieren können.	Textuelle Beschreibung
Umsetzung von Kritiken	Mechanismus, mit dem die Kritiken realisiert sind.	[regelbasiert, OCL, Pattern-Matching, wissensbasiert, Programmiersprache]
Realisierung von Kritiken	Basieren die Kritiken auf einfachen Vergleichen oder auf komplexer Analyse?	[vergleichend, analytisch]
Dimension der Kritik	Nach Fischer [63]	[reaktiv, pro-aktiv, lokal, global]
Typ des Feedbacks		[Erläuterungen, Argumentationen, Vorschläge, Beispiele, Interpretationen, Simulationen]
Modus des Feedbacks	Präsentation der Kritiken	[textuell, graphisch, 3D Visualisierung]
Qualitätsaspekte		[Korrektheit, Vollständigkeit, Konsistenz, Optimierung, Alternativen, Skalierbarkeit, Präsentation]

Javascript) angeboten. Regelbasierte Sprachen und OCL waren in Tests von den Nutzern nicht vergleichbar gut zu bedienen. Nach Ali et al. ist zwischen Umsetzung und Realisierung zu unterscheiden:

Beispiel 30: *Umsetzung und Realisierung von Kritiken*

ArgoUML setzt Kritiken beispielsweise als Java-Klassen um [137], ATTENDING verwendet einen vergleichenden Ansatz, um Kritiken zu realisieren.

In einem vergleichenden Ansatz vergleicht das System die Arbeitsergebnisse (bzw. den Entwurf) des Nutzers mit einer vom System generierten Referenz. Ein solches System muss umfassendes (vollständiges) Wissen über die Domäne enthalten. Demnach ist das Kritiksystem für Anforderungen analytisch, die Argumentationskomponente für Glossare hingegen vergleichend: HeRA vergleicht, ob das vom Nutzer erstellte Glossar alle Begriffe enthält, die bereits früher in Glossaren enthalten waren.

Als Dimension der Kritik hat sich in allen Beispielen eine globale Herangehensweise angeboten. KonPAss und HeRA.EPK sind dabei reaktiv, HeRA.Kritik und HeRA.Glossar pro-aktiv. Als Typ des Feedbacks werden in der Regel Argumentationen, Erläuterungen und Vorschläge geboten. HeRA.EPK bietet zudem Simulation. Der Modus ist textuell oder visuell. Die adressierten Qualitätsaspekte sind Korrektheit, Vollständigkeit, Konsistenz und Optimierung.

Das Klassifizierungsschema nach Ali et al. ist allgemeiner und weniger stark auf die Domäne Requirements Engineering bezogen als der Steckbrief in Kapitel 7. Die Aspekte lassen sich aber in beiden Klassifikationen wiederfinden:

1. *Kritikdomäne:* Die Domäne ist im Rahmen dieser Arbeit die Dokumentation von Anforderungen. KonPAss konzentriert sich auf die Qualitätssicherung, die anderen Beispielimplementierungen beziehen sich auf die Erstellung von Dokumenten.
2. *Autorenwerkzeug:* Im Rahmen dieser Arbeit ist hier eine genauere Betrachtung notwendig, dies entspricht der Eigenschaft *Lernfähigkeit*.
3. *Umsetzung der Kritiken:* In dieser Arbeit sind Kritiken in der Regel in einer Programmiersprache umgesetzt (KonPAss: Java, HeRA: Javascript/Java)
4. *Realisierung der Kritiken:* HeRA.Glossar ist vergleichend, die anderen Beispielimplementierungen sind analytisch. Dieser Aspekt entspricht der Eigenschaft *Interpretierend*.
5. *Dimension der Kritik:* Alle Dimensionen sind durch Beispielimplementierungen abgedeckt. HeRAs Kritiksysteme sind pro-aktiv, KonPAss ist reaktiv. Sowohl lokale als auch globale Kritiken wurden vorgestellt.

6. *Typ des Feedbacks:* HeRAs Kritiksystem für Anforderungen und KonPAss geben Argumentationen, die Glossar-Erweiterung macht Vorschläge.
7. *Modus des Feedbacks:* Die EPK Erweiterung von HeRA gibt grafisches Feedback (dieser Modus entspricht der Eigenschaft *Perspektivwechselnd*), ansonsten wird textuelles Feedback gegeben.
8. *Qualitätsaspekte:* Die in dieser Arbeit vorgestellten Beispielimplementierungen konzentrieren sich auf die Qualitätsaspekte Korrektheit, Vollständigkeit, Konsistenz und Optimierung von Anforderungsdokumentation.

Zettel hat die Akzeptanz beim Nutzer abhängig von der Form des Feedbacks von Konsistenzregeln empirisch untersucht [177]. Die Untersuchung wurde mit Hilfe von SPEARMINT, eines CASE-Werkzeugs für Prozessmodellierung untersucht. Die Domäne unterscheidet sich demnach deutlich von dieser Arbeit. Dennoch sind die Ergebnisse interessant. Feedback von Konsistenzregeln kann in SPEARMINT verschiedene Formen annehmen:

- Keine Konsistenzsicherung: Die Konsistenzregel ist deaktiviert.
- Passive Konsistenzsicherung: Die Konsistenzregel wird nur durch eine explizite Aktion des Nutzers überprüft, das Ergebnis wird in einem Dialog angezeigt.
- Tolerierende Konsistenzsicherung: Ein Verstoß gegen die Konsistenzregel wird toleriert, der Nutzer wird je nach Unterart des tolerierenden Feedbacks durch einen Dialog in der Arbeit unterbrochen, oder erhält einen dezenten Hinweis im Werkzeug auf Konsistenzverstöße.
- Korrigierende Konsistenzsicherung: Ein Konsistenzverstoß wird direkt korrigiert. Je nach Ausprägung wird der Nutzer auf die Korrektur hingewiesen oder nicht.
- Präventive Konsistenzsicherung: Aktionen, durch die die Konsistenz verletzt wird, werden unterbunden.

Getestet wurden drei Varianten: Keine Konsistenzsicherung, (automatisch) korrigierende Konsistenzsicherung ohne Feedback und tolerierende Konsistenzsicherung mit Unterbrechung.

Aktive Konsistenzsicherung hat unabhängig von der Feedbackform nach dem Test einen positiven Einfluss auf die Akzeptanz. Die Nutzer haben in die automatische Korrektur mehr Vertrauen, als in die unterbrechende Meldung. Die unterbrechende Meldung führt zu besseren Ergebnissen, Nutzer können die Ergebnisse auch besser darstellen. Die Sachdienlichkeit ist jedoch von den Nutzern bei der automatischen Korrektur besser bewertet worden. Die automatische Korrektur wird als nützlicher wahrgenommen, die Nutzer empfinden sie aber als schlechter bedienbar.

Im Rahmen dieser Arbeit wurde ausschließlich mit passiven Mechanismen und tolerierenden Regeln, die den Nutzer nicht unterbrechen, gearbeitet. Automatische Korrektur ist schon auf Grund des heuristischen Charakters der Kritiken schwer vorstellbar. Die Beobachtung, dass methodisches Feedback vom Werkzeug einen positiven Einfluss auf die Akzeptanz und die durch den Nutzer wahrgenommene Nützlichkeit des Werkzeugs hat, lässt sich aus den Erfahrungen im Rahmen dieser Arbeit bestätigen: Nutzer von HeRA fühlten sich im Vergleich zur Kontrollgruppe ebenfalls sicherer. Zettels Schlussfolgerung, dass die Art des Feedbacks durch den Nutzer konfigurierbar sein sollte, um optimale Ergebnisse zu erreichen, kann auf Basis der Erfahrungen in dieser Arbeit ebenfalls unterstützt werden. Die Art der Aufgabe und der Typ des Nutzers hat entscheidenden Einfluss auf die Akzeptanz computerbasierten Feedbacks.

10 Zusammenfassung und Ausblick

10.1 Zusammenfassung der Ergebnisse

Heuristische Kritiken eignen sich, um Erfahrungen bereits sehr früh in die Erstellung von Anforderungsdokumenten einfließen zu lassen. Um dies zu zeigen, wurden Reife und Formalität als wichtige Eigenschaften des Modells von Anforderungsdokumenten eingeführt. Ein Anforderungsdokument durchläuft verschiedene Phasen, vom Konzept über den Enwurf, bis eine endgültige, vollständige und konsistente Fassung des Dokuments entsteht. Model-Checking und Simulation haben sich in der Sofwareentwicklung zur Verifikation und Validierung bewährt. Bei der Dokumentation von Anforderungen sind sie jedoch erst einsetzbar, wenn eine gewisse Vollständigkeit und Konsistenz vorliegt. Zudem muss die Semantik der Anforderungsbeschreibung definiert sein, damit Computer-Programme die dokumentierten Anforderungen prüfen können. Damit entsteht eine Lücke bei der Überprüfung weniger reifer und formaler Anforderungsdokumente.

Als Lösungsvorschlag führt diese Dissertation heuristische Kritiken ein. Heuristische Kritiken analysieren Anforderungsdokumente und geben Feedback, wenn sie die Indikation für ein Problem entdecken. Von heuristischen Kritiken wird nicht gefordert, dass sie immer optimale Lösungen produzieren. Fehler der 1. Art (heuristische Regel feuert, aber die Erfahrung ist in diesem Kontext nicht anwendbar) und Fehler der 2. Art (heuristische Regel feuert nicht, aber die Erfahrung ist in diesem Kontext anwendbar) werden explizit erlaubt, so lange die heuristische Kritik dennoch nützlich ist. Dies erleichtert es, heuristische Kritiken zu entwickeln und erlaubt es, auch inkonsistente und unvollständige Dokumente zu überprüfen.

Fehlerhaftes Feedback durch heuristische Kritiken ist kein Problem sondern eine Chance. Ist die Ursache ein unvollständiges oder inkonsistentes Dokument, so kann die Kritik direkt oder später zur Verbesserung des Dokuments genutzt werden. Andernfalls kann ein falscher Befund zum Anlass genommen werden, um die heuristische Kritik zu verbessern. Diese Arbeit definiert um diesen Mechanismus ein Lernmodell, das Organisationen beim Erfahrungsmanagement in der Anforderungsanalyse hilft.

Damit heuristische Kritiken trotz ihrer relativen Unzuverlässigkeit bei der Dokumentation von Anforderungen als nützlich empfunden werden, müssen eine Reihe von Herausforderungen bewältigt werden. So dürfen heuristische Kritiken ihre Nutzer nicht zu sehr stören, selbst wenn

- Befunde zu einem Fehler gemeldet werden, der bekannt ist und in der aktuellen Reifestufe des Dokuments ignoriert wird.
- Befunde gemeldet werden, obwohl kein Fehler vorliegt.
- Befunde nicht gemeldet werden, obwohl ein Fehler vorliegt.

Dies kann auf verschiedene Arten erreicht werden:

1. Heuristische Kritiken können auf eine zurückhaltende Art in erfahrungsbasierte Werkzeuge für das Requirements Engineering eingebunden werden.
2. Heuristische Kritiken können an einer Stelle in den Requirements Engineering Prozess eingebunden werden, an der sie nicht die letzte Instanz sind und ihr Feedback trotz der relativen Unzuverlässigkeit hilfreich ist.
3. Die Zuverlässigkeit von heuristischen Kritiken kann so weit verbessert werden, dass Fehler vernachlässigbar selten vorkommen.

10.1.1 Beitrag der Arbeit

Der wichtigste Beitrag dieser Arbeit ist die Modellierung des Zusammenhangs der computerbasierten Analyse von Anforderungsdokumentation und dem Erfahrungsmanagement einer Organisation (vergleiche Kapitel 5). Die Einbettung von heuristischen Kritiken in das Erfahrungsmanagement einer lernenden Software erstellenden Organisation erlaubt dieser die kontinuierliche Verbesserung der Anforderungsdokumentation.

Entscheidend für den Erfolg dieser kontinuierlichen Verbesserungen sind die Eigenschaften entsprechender erfahrungsbasierter Werkzeuge für das Requirements Engineering. Mit dem Steckbrief steht ein wichtiges Instrument zur Verfügung, um erfahrungsbasierte Werkzeuge nach diesen Eigenschaften und dem angestrebten Einsatzszenario zu charakterisieren (vergleiche Abschnitt 6.1). Im Vergleich zu ähnlichen Arbeiten aus dem Requirements Engineering fallen die Voraussetzungen für den Einsatz der Konzepte dieser Arbeit geringer aus. Die hier vorgestellten Beispielimplementierungen adressieren natürlichsprachliche Anforderungsdokumente bereits in sehr frühen und inkonsistenten Zuständen. Dabei wird nur eine geringe Formalität vorausgesetzt. Zudem hat der Nutzer sehr hohe Freiheitsgrade. Die hier vorgestellten Ansätze erzwingen keine bestimmte Struktur oder Repräsentation der Anforderungen. Der Nutzer wird lediglich darauf hingewiesen, dass ein Anforderungsdokument auf Basis von Erfahrungen verbessert werden könnte. Der Nutzer kann dieses Feedback in vielen Fällen ignorieren und dennoch von dem Ansatz profitieren, wenn er es in seltenen Fällen nutzt.

Im Detail ergeben sich durch die spezielle Einbindung der erfahrungsbasierten Konzepte noch weitere Alleinstellungsmerkmale zu verwandten Arbeiten:

- Entgegen der Meinung in der Literatur können heuristische RE-Werkzeuge mit geringem Recall nützlich sein, wenn ihr Feedback während der Erstellung von Dokumenten den Autoren pro-aktiv unterstützt[1]. Ein guter Hinweis hilft direkt, unzureichende können ignoriert werden und fehlende Hinweise fallen nicht in das Gewicht – ohne das heuristische Werkzeug hätte der Analyst ebenfalls darauf verzichten müssen.
- Selbst heuristische RE-Werkzeuge, die für die Qualitätsanalyse von Anforderungsdokumenten herangezogen werden, müssen nicht unbedingt einen Recall von 100% haben. Um nützlich zu sein, müssen sie nur gleich gut oder besser als menschliche Reviewer bei niedrigeren Kosten sein.
- Heuristische Kritiken erlauben es, große Mengen von Information effizient zu analysieren und ihrem Nutzer relevante Aspekte zur Anforderungsdokumentation übersichtlich aufzubereiten. So enthebt das Ableiten übersichtlicher Diagramme, Modelle und Kritiken aus Anforderungsdokumentation die Nutzer der unangenehmen Pflicht, große Mengen Text zu analysieren und unterstützt sie bei ihrer zentralen Aufgabe: Gute Entscheidungen für die bestmögliche Dokumentation von Anforderungen zu treffen.

Der Zusammenhang zwischen computerbasierter Anforderungsanalyse und dem Erfahrungsmanagement spiegelt sich auch in dem zweiten wichtigen Beitrag dieser Arbeit wieder: Der Evaluationsstrategie (vergleiche Abschnitt 6.2). Diese berücksichtigt neben der Qualität und den Kosten von Anforderungsdokumenten auch die Effekte auf das organisatorische und individuelle Lernen. Das Lernmodell aus Kapitel 5 wird auf diese Weise konkretisiert und in den Evaluationsszenarien in Kapitel 8 operationalisiert.

Durch die Evaluation sind darüberhinaus erste Datenpunkte zu den zu erwartenden Effekten gegeben. Die Evaluation hat ergeben, dass erfahrungsbasierte RE-Werkzeuge ein wertvolles Mittel für das Erfahrungsmanagement im Requirements Engineering Software erstellender lernender Organisationen sind. Erfahrungen der Organisation werden öfter angewendet und der Rückfluss neuer Erfahrungen wird stimuliert.

Individuen werden ebenfalls durch den Einsatz erfahrungsbasierter Werkzeuge beim Lernen neuer Erfahrungen unterstützt. Die konstruktive Unterbrechung durch heuristische Kritiken kann eine Reflexion anstoßen und zu neuen Erfahrungen führen.

Schließlich konnte auch gezeigt werden, dass sich mit dem Ansatz dieser Arbeit bessere Anforderungsdokumente erstellen lassen.

[1] Dazu müssen Autoren wissen, dass der Recall niedrig ist, oder es muss auf andere Art vermieden werden, dass sich Benutzer in falscher Sicherheit wiegen.

10.1.2 Grenzen der Arbeit

Diese Arbeit konzentriert sich auf textuelle Anforderungsbeschreibungen, und in diesem Bereich auf natürlichsprachliche Ansätze. In kontrollierten oder formalen Sprachen wären die hier gezeigten Konzepte zwar ebenfalls einsetzbar, jedoch lassen sich erheblich mächtigere Mechanismen zur Qualitätskontrolle einsetzen, wie Kapitel 9 gezeigt hat. Da diese zum Teil in der Lage sind, ein eindeutiges Ergebnis zu liefern, sind erfahrungsbasierte Ansätze zur Verbesserung nicht so wichtig. Zudem sind sie sehr schwer einzuführen, da die formalen Mechanismen in der Regel zu schwierig zu ändern sind, als dass ein Nutzer dies neben seiner eigentlichen Arbeit durchführen könnte.

Auf grafische Modelle von Anforderungen ließe sich der Ansatz durchaus anwenden. Diese Modelle sind gerade am Anfang eines Projekts noch nicht so formal, dass sich Model-Checker einsetzen ließen. Ansätze von Berenbach und Souza et al. gehen auch in eine ähnliche Richtung. Eine genauere Untersuchung war jedoch nicht Fokus dieser Arbeit.

In dieser Arbeit wurden erfahrungsbasierte RE-Werkzeuge in bestehende Werkzeuge und Dokumentstrukturen integriert, um so eine kontinuierliche Verbesserung der Dokumentation von Anforderungen zu erreichen. Abbildung 10.1 zeigt Grenzen einer solchen erfahrungsbasierten Verbesserung.

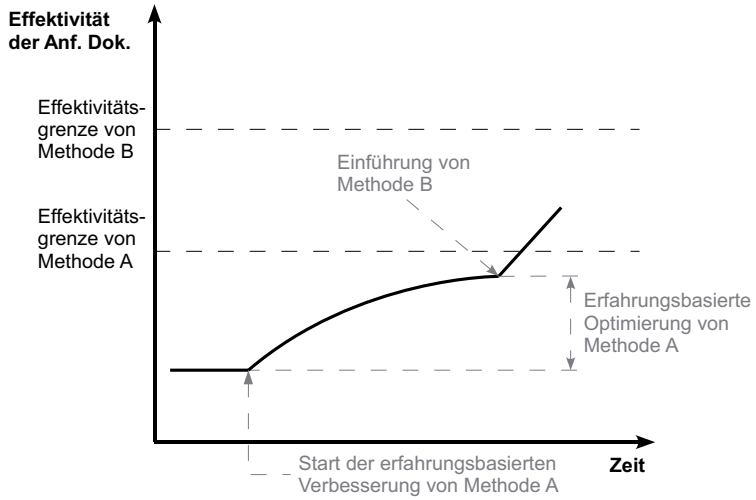


Abbildung 10.1: Kontinuierliche und strukturelle Verbesserung der Anforderungsdokumentation einer Organisation.

Mit Hilfe heuristischer Kritiken kann die Organisation in Abbildung 10.1 die Anforderungsdokumentation nach einer Methode A kontinuierlich verbessern. Die Effektivität der Organisation kommt immer näher an die theoretische Grenze der

Methode. Weitere Verbesserungen werden immer schwerer, da die Analysten schon beinahe optimal arbeiten. Eine signifikante Verbesserung lässt sich nun nur noch durch eine Änderung der Methode herbeiführen. Es empfiehlt sich, diese neue Methode ebenfalls erfahrungsbasiert zu verbessern, da die Effektivität nach ihrer Einführung vermutlich weit unter ihrer Effektivitätsgrenze liegt.

10.2 Ausblick

Die Ansätze in dieser Arbeit unterstützen Analysten, gute natürlichsprachliche Anforderungsdokumente zu erstellen. Die erforderliche Qualität von Anforderungsdokumenten ist aber von Projekt zu Projekt unterschiedlich. Ein kleines Projekt, in dem sich Analysten und Entwickler kennen und täglich treffen, kann mit qualitativ schwächerer Anforderungsdokumentation auskommen, als beispielsweise Projekte mit einem Zulieferer-Szenario. Heuristische Kritiken können die Kosten für Qualitätssicherung zwar senken, die Kosten für Anforderungsdokumentation sind jedoch abhängig von der angestrebten Qualität. Nicht alle Projekte können es sich leisten, die bestmögliche Anforderungsdokumentation anzustreben. Die Kosten müssen für den jeweiligen Kontext angemessen sein. Dabei muss zum Teil auch erheblicher Zeitdruck berücksichtigt werden.

Durch das evolutionäre Wachstum ist es möglich, Erfahrungen auf den Kontext eines Projekts abzustimmen. In diesem Zusammenhang wäre jedoch noch zu untersuchen, ob mit heuristischen Kritiken über das Ziel hinaus, möglichst gute Anforderungsdokumentation zu erstellen, nicht auch ein alternatives Ziel erreicht werden kann: Im Rahmen der verfügbaren Ressourcen genau die für das Projekt nützlichsten Anforderungen in ausreichender Qualität zu dokumentieren. Grundlage dafür wäre eine Priorisierung aller möglichen Aktivitäten des Requirements Engineering einer Organisation nach dem Nutzen für das konkrete Projekt. Vorstellbar wäre hier ein Empfehlungssystem, das Methoden und Techniken des Requirements Engineering auf Basis des Projektkontexts empfiehlt. Erste Schritte in diese Richtung sind durch die Diplomarbeit von Liza Sarkisyan im Rahmen dieser Arbeit unternommen worden, indem die Aktivitäten nach den Risiken geordnet werden, die sie adressieren [145].

Eine weitere interessante Richtung, um auf der vorliegenden Arbeit aufzubauen, betrifft das Lernmodell. Nach den Erfahrungen während der Evaluation in dieser Arbeit ist anzunehmen, dass sich direktes heuristisches Feedback in der Ausbildung von Analysten bewährt. Positive Lerneffekte erfahrungsbasierter RE-Werkzeuge bei Studenten wären dafür noch nachzuweisen.

Das gilt auch für langfristige Lerneffekte einer LSO. Solche langfristigen Lerneffekte empirisch zu beobachten birgt einige Herausforderungen. So ist über einen längeren Beobachtungszeitraum damit zu rechnen, dass die Organisation einen

Methoden- oder zumindest Werkzeug- oder Vorlagenwechsel (wie zum Beispiel in Abbildung 10.1) vornimmt. Für die in dieser Arbeit vorgestellten Konzepte wäre ein solcher Bruch ein Problem, da Erfahrungen dann neu aufgebaut werden müssten und die erfahrungsbasierten Werkzeuge neu in der Organisationskultur etabliert werden müssten. Einen Ausweg könnten Methoden der modellgetriebenen Softwareentwicklung bieten. In aktuellen Arbeiten wird aufbauend auf dieser Dissertation untersucht, wie Erfahrungen an ein variables Metamodell von Anforderungsdokumenten gekoppelt werden können. So könnten heuristische Kritiken unabhängig vom Datenmodell eines konkreten heuristischen Werkzeugs formuliert werden. Die heuristischen Regeln könnten dann in einer speziellen Regelsprache beschrieben und in die Zielsprache eines konkreten Werkzeugs transformiert werden. Auf diese Weise könnten Erfahrungskreisläufe plattformunabhängig gestaltet werden. Bei einem Methodenwechsel könnten dann womöglich einige konzeptionell passende Kritiken übernommen werden.

Ein interessanter Unterschied zwischen KonPAss und HeRA.Kritik auf der einen Seite und HeRA.Glossar auf der anderen Seite ist, dass neben der Kontrolle und Verbesserung vorhandener Dokumente auch die Erstellung zusätzlicher relevanter Artefakte (zum Beispiel eines Glossars) unterstützt werden kann. Berücksichtigt man die zentrale Stellung von Anforderungsdokumentation in Softwareprojekten, wäre es interessant, heuristische Kritiken zu nutzen, um mit den Anforderungen verwandte Themen frühzeitig zu verfolgen. Heuristische Kritiken könnten Anforderungen identifizieren, die durch Spezialisten, beispielsweise für Sicherheit, Bedienbarkeit oder rechtliche Konformität, weiter verfeinert werden sollten. Mit SecReq nutzt eine Methode für das Requirements Engineering in sicherheitskritischen Systemen bereits Konzepte aus dieser Arbeit [79]: SecReq entlastet die wenigen Sicherheitsexperten einer Organisation, in dem diese nicht die gesamte Anforderungsspezifikation auf sicherheitsrelevante Aspekte überprüfen müssen. Falls in einem Projekt kein Sicherheitsexperte zur Verfügung steht, kann die SecReq-Komponente von HeRA neben der Identifikation von sicherheitsrelevanten Aspekten auch einen Assistenten für deren Verfeinerung zur Verfügung stellen [126]. Ausgehend von schlichten Schlüsselwortlisten wurden HeRAs heuristische Regeln mittlerweile um Bayes'sche Filter erweitert, mit denen sich bei gutem Training hohe Trefferquoten erzielen lassen.

Einen besonders wichtigen Beitrag können heuristische und erfahrungsbasierte Werkzeuge zudem bei der verteilten Softwareentwicklung leisten. Wird ein Anforderungsdokument von verschiedenen Analysten geschrieben, die nicht am gleichen Standort arbeiten, so kann heuristische Konsistenzprüfung schon früh im Projekt Mehraufwände vermeiden. Ein interessantes Konzept in diese Richtung wurde im Rahmen dieser Arbeit mit der Audi AG erarbeitet [161]. In der Vor-Entwicklung erstellen kleine Projektteams Prototypen zukunftsweisender Technologien. Sind diese Vor-Projekte erfolgreich, werden die Technologien in die Serienproduktion transferiert. Erst jetzt entstehen umfangreiche Anforderungsspezifikationen, zu

einem Zeitpunkt, an dem die ursprünglichen Projektteams nicht mehr existieren. Vorher ist umfangreiche Dokumentation nicht sinnvoll, weil diese im Falle eines Fehlschlags umsonst erstellt worden wäre. In diesem Umfeld wurde das SmartWiki eingeführt, ein heuristisches RE-Werkzeug, das besonders Kollaboration unterstützt [96, 95]. Während der Vor-Entwicklung werden in dem Wiki Notizen gesammelt. Sieht der Prototyp vielversprechend aus, werden die heuristischen Kritiken eingeschalten, um Lücken in der Dokumentation zu finden und bis zum Abschluss des Projekts zu schließen. Auf Basis der Erfahrungen in dem SmartWiki Projekt scheinen heuristische Kritiken ein großes Potential bei der Unterstützung verteilter und kollaborativer Anforderungsanalyse zu haben. Hier ist besonders die Möglichkeit interessant, Analysten gezielt mit den notwendigen Informationen zum Stand der Anforderungsdokumentation zu versorgen. Mit der Analyse des Awareness-Bedarfs bei kollaborativer Anforderungsdokumentation wurde ein erster Schritt auch in diese Richtung schon unternommen [100].

Liste der Definitionen

Literaturverzeichnis

[1] IEEE Standard Glossary of Software Engineering Terminology, 1990.

[2] CHAOS Chronicles v3.0. Technical report, The Standish Group, 2003.

[3] IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications, 1998.

[4] Alain Abran, James W. Moore, Pierre Bourque, and Robert Dupuis, editors. *SWEBOK: Guide to the Software Engineering Body of Knowledge: 2004 Version*. IEEE, 2005.

[5] Steve Adolph, Paul Bramble, Alistair Cockburn, and Andy Pols. *Patterns for Effective Use Cases*. Addison Wesley, 2003.

[6] Norhayati Mohd Ali, John Hosking, and John Grundy. A Taxonomy of Computer-supported Critics. In *Proceedings of International Symposium on Information Technology (ITSim)*, Kuala Lumpur, Malaysia, 2010.

[7] V. Ambriola and V. Gervasi. Processing natural language requirements. In *International Conferenceon Automated Software Engineering*, pages 36–45, Los Alamitos, CA, USA, 1997. IEEE Computer Society.

[8] Chris Argyris and Donald A. Schön. *Organizational Learning: A Theory of Action Perspective*. Organization Development. Addison-Wesley, Reading, MA, 1978.

[9] Chris Argyris and Donald A. Schön. *Organizational Learning II : Theory, Method and Practice*. Addison-Wesley, Reading, MA, 1996.

[10] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, Addison Wesley, 1999.

[11] V. Basili, G. Caldiera, and D.H. Rombach. *The Experience Factory*. John Wiley and Sons, 1994.

[12] Victor R. Basili. The Experience Factory and its Relationship to Other Improvement Paradigms. In *ESEC '93: Proceedings of the 4th European Software Engineering Conference on Software Engineering*, pages 68–83, London, UK, 1993. Springer-Verlag.

[13] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The Experience Factory. In *Proceedings of the 14th annual Software Engineering Workshop*, 1989.

[14] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*, pages 646–661. Wiley, 1994.

[15] Victor R. Basili, Scott Green, Oliver Laitenberger, Filippo Lanubile, Forrest Shull, Sivert Sørumgård, and Marvin V. Zelkowitz. The Empirical Investigation of Perspective-Based Reading. *Int. Journal of Empirical Software Engineering*, 1(2):133–164, Januar 1996.

[16] Victor R. Basili, Dieter Rombach, Kurt Schneider, Barbara Kitchenham, Dietmar Pfahl, and Richard W. Selby. *Empirical Software Engineering Issues – Critical Assessment and Future Directions*, volume LNCS 4336. Springer, 2007.

[17] Kent Beck. *Extreme Programming Explained*. Addison-Wesley, 2000.

[18] Brian Berenbach. Evaluating the Quality of a UML Business Model. In *RE '03: Proceedings of the 11th IEEE International Conference on Requirements Engineering*, page 280, Monterey Bay, CA, 2003. IEEE Computer Society.

[19] Brian Berenbach. The Evaluation of Large, Complex UML Analysis and Design Models. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 232–241, Edinburgh, Scotland, UK, 2004. IEEE Computer Society.

[20] Brian Berenbach and Gail Borotto. Metrics for model driven requirements development. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 445–451, Shanghai, China, 2006. ACM.

[21] Brian Berenbach and Timo Wolf. A unified requirements model; integrating features, use cases, requirements, requirements analysis and hazard analysis. In *ICGSE '07: Proceedings of the International Conference on Global Software Engineering*, pages 197–203, Munich, Germany, 2007. IEEE Computer Society.

[22] Brian A. Berenbach. Comparison of UML and text based requirements engineering. In *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 247–252, Vancouver, CA, 2004. ACM.

[23] Ralph Bergmann. *Experience Management: Foundations, Development Methodology, and Internet Based Applications*. LNAI 2432. Springer, 2002.

[24] Daniel M. Berry. The importance of ignorance in requirements engineering: An earlier sighting and a revisitation. *The Journal of Systems and Software*, 60(1):83–85, 2002.

[25] Daniel M. Berry and Erik Kamsties. The Syntactically Dangerous All and Plural in Specifications. *IEEE Software*, 22(1):55–57, 2005.

[26] D.M. Berry and E. Kamsties. *Perspectives on Requirements Engineering*, chapter 2. Ambiguity in Requirements Specification, pages 7–44. Kluwer, 2004.

[27] D.M. Berry, E. Kamsties, and M.M. Krieger. From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity. Technical report, Computer Science Dept., Univ. of Waterloo, 2003.

[28] Andreas Birk. Erfahrungen mit Anwendungsfallspezifikation bei der Entwicklung großer IT-Systeme. *GI Softwaretechnik-Trends*, 27(1), Feb 2007.

[29] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, first edition, 1981.

[30] Barry W. Boehm. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, 21(5):61 – 72, Mai 1988.

[31] Christian El Boustani. Quantitative und qualitative Messung von Software-Anforderungen. Bachelorarbeit, Leibniz Universität Hannover, Fachgebiet Software Engineering, January 2008.

[32] Travis D. Breaux and Annie I. Antón. Analyzing regulatory rules for privacy and security requirements. *IEEE Transactions on Software Engineering, Special Issue on Software Engineering for Secure Systems*, 34(1):5–20, 2008.

[33] Lionel C. Briand, Christiane M. Differding, and H. Dieter Rombach. Practical Guidelines for Measurement-Based Process Improvement. *Software Process: Improvement and Practice*, 2(4):253–280, 1996.

[34] Olesia Brill. Integration eines Wikis in einen Requirements Engineering Prozess. Studienarbeit, Leibniz Universität Hannover, Fachgebiet Software Engineering, 2007.

[35] Eckard Bringmann and Andreas Krämer. Model-based Testing of Automotive Systems. In *Proceedings of First International Conference on Software Testing, Verification, and Validation (ICST 2008)*, pages 485–493, Lillehammer, Norway, 2008. IEEE Computer Society.

[36] Fred P. Brooks. No Silver Bullet — Essence and Accident in Software Engineering. *IEEE Computer*, 20(4):10–19, 1987.

[37] Carlos Castro-Herrera, Chuan Duan, Jane Cleland-Huang, and Bamshad Mobasher. Using Data Mining and Recommender Systems to Facilitate Large-Scale, Open, and Inclusive Requirements Elicitation Processes. In

RE '08: Proceedings of the 2008 16th IEEE International Requirements Engineering Conference, pages 165–168, Washington, DC, USA, 2008. IEEE Computer Society.

[38] Francis Chantree. Ambiguity management in natural language generation. In *Proceedings of 7th annual CLUK resarch colloquium*, 2004.

[39] Francis Chantree, Bashar Nuseibeh, Anne de Roeck, and Alistair Willis. Identifying Nocuous Ambiguities in Natural Language Requirements. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, pages 56–65, Minneapolis, USA, 2006. IEEE Computer Society.

[40] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 2000.

[41] Jane Cleland-Huang and Bamshad Mobasher. Using data mining and recommender systems to scale up the requirements process. In *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems*, pages 3–6, 2008.

[42] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Professional, January 2000.

[43] J. Cohen. *Statistical power analysis for the behavioral sciences*. Lawrence Erlbaum, New Jersey, 2nd edition, 1988.

[44] Rita J. Costello and Dar-Biau Liu. Metrics for requirements engineering. In *Selected papers of the sixth annual Oregon workshop on Software metrics*, pages 39–63, New York, NY, USA, 1995. Elsevier Science Inc.

[45] Christian Crisp. Konzept und Werkzeug zur erfahrungsbasierten Erstellung von Use Cases. Masterarbeit, Leibniz Universität Hannover, Fachgebiet Software Engineering, 2006.

[46] DaimlerChrysler. Requirements Engineering Referenzmodell. Dagstuhl Seminar "Requirements Engineering", 1998.

[47] Daniela Damian, Luis Izquierdo, Janice Singer, and Irwin Kwan. Awareness in the Wild: Why Communication Breakdowns Occur. In *Proceedings of Second Intl Conference on Global Software Engineering*, pages 81–90, Munich, Germany, 2007.

[48] Alan M. Davis. *Just Enough Requirements Management: Where Software Development meets Marketing*. Dorset House Publishing, New York, 2005.

[49] M. Davis and Didar Zowghi. Good requirements practices are neither necessary nor sufficient. *Requir. Eng.*, 11(1):1–3, 2005.

[50] Cleidson R. B. de Souza, Hamilton L. R. Oliveira, Cleber R. P. da Rocha, Kléder Miranda Gonçalves, and David F. Redmiles. Using Critiquing Systems for Inconsistency Detection in Software Engineering Models. In *SEKE*, pages 196–203, 2003.

[51] Tom deMarco. *The Deadline: A Novel about Project Management*. Dorset House Publishing, New York, USA, 1997.

[52] W. Edwards Deming. *Out of the Crisis*. MIT Press, 1986.

[53] Jörg Dörr, Tom Koenig, Thomas Olsson, and Sebastia Adam. Das ReqMan Prozessrahmenwerk. IESE-Report 141.06/D, Fraunhofer IESE, Oktober 2006.

[54] Michael Earl. Knowledge Management Strategies: Toward a Taxonomy. *J. Manage. Inf. Syst.*, 18(1):215–233, 2001.

[55] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. *Guide to Advanced Empirical Software Engineering*, chapter 11, Selecting Empirical Methods for Software Engineering Research, pages 285–311. Springer, London, 2008.

[56] Michael Eisenbarth and Jörg Dörr. Facilitating Project Management by capturing Requirements Quality and Volatility Information. In *Proceedings of Workshop on Measuring Requirements for Project and Product Success*, Palma de Mallorca, Spain, November 2007. in conjunction with the IWSM-Mensura Conference.

[57] Albert Endres and Dieter Rombach. *A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories*. Addison Wesley, 2003.

[58] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. An Automatic Quality Evaluation for Natural Language Requirements. In *Proceedings of the Seventh International Workshop on RE: Foundation for Software Quality (REFSQ 2001)*, pages 150–164, Interlaken, Switzerland, 2001.

[59] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. The Linguistic Approach to the Natural Language Requirements Quality: Benefit of the use of an Automatic Tool. In *SEW '01: Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, pages 97–105, Washington, DC, USA, 2001. IEEE Computer Society.

[60] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari. Application of linguistic techniques for Use Case analysis. In *Proceedings of IEEE Joint International Conference on Requirements Engineering*, pages 157 – 164, Essen, Germany, 2002.

[61] G. Fischer. Social Creativity, Symmetry of Ignorance and Meta-Design. *Knowledge-Based Systems Journal*, 2000.

[62] G. Fischer, K. Nakakoji, J. Ostwald, G. Stahl, and T. Sumner. Embedding Computer-Based Critics in the Contexts of Design. In *Proc. of INTERCHI-93*, pages 157–164, Amsterdam, The Netherlands, 1993.

[63] Gerhard Fischer. Human-Computer Interaction Software: Lessons Learned, Challenges Ahead. *IEEE Softw.*, 6(1):44–52, 1989.

[64] Gerhard Fischer. Domain-Oriented Design Environments. *Automated Software Engineering*, 1:177–203, 1994.

[65] Gerhard Fischer, Andreas C. Lemke, Thomas W. Mastaglio, and Anders I. Morch. Critics: An Emerging Approach to Knowledge-Based Human-Computer Interaction. *International Journal of Man-Machine Studies*, 35(5):695–721, 1991.

[66] Thomas Flohr. *Ein Framework als Grundlage der Ausgestaltung von Quality-Gate-Referenzprozessen für die Software-Entwicklung.* PhD thesis, Leibniz Universität Hannover, Welfengarten 1, 30167 Hannover, 2008.

[67] K. Forsberg and H. Mooz. The Relationship of Systems Engineering to the Project Cycle. In *First Annual Symposium of the National Council On Systems Engineering (NCOSE)*, Oct. 1991.

[68] K. Forsberg and H. Mooz. System Engineering Overview. In R. H. Thayer, M. Dorfman, and A. M. Davis, editors, *Software Requirements Engineering*, pages 44–72, Los Alamitos CA, 1997. IEEE Computer Society Press.

[69] Philipp Förmer. Heuristische Konsistenzprüfung in Anforderungsdokumenten. Bachelorarbeit, Leibniz Universität Hannover, Fachgebiet Software Engineering, 2009.

[70] Donald C. Gause and Gerald M. Weinberg. *Exploring Requirements: Quality Before Design.* Dorset House Publishing Company, Incorporated, 1989.

[71] Vincenzo Gervasi and Bashar Nuseibeh. Lightweight Validation of Natural Language Requirements: A Case Study. In *In Proceedings of the 4th IEEE International Conference on Requirements Engineering*, pages 140–148, Schaumburg, Illinois, USA, 2000. IEEE Computer Society.

[72] Vincenzo Gervasi and Didar Zowghi. Reasoning About Inconsistencies in Natural Language Requirements. *ACM Trans. Softw. Eng. Methodol.*, 14(3):277–330, 2005.

[73] Martin Glinz. A Lightweight Approach to Consistency of Scenarios and Class Models. In *In Proceedings of the 4th IEEE International Conference on Requirements Engineering*, pages 49–58, Schaumburg, Illinois, USA, 2000. IEEE Computer Society.

[74] Martin Glinz. Improving the Quality of Requirements with Scenarios. In *Proceedings of the Second World Congress for Software Quality (2WCSQ)*, pages 55–60, Yokohama, September 2000.

[75] Martin Glinz. Problems and Deficiencies of UML as a Requirements Specification Language. In *Proceedings of the 10th International Workshop on Software Specification and Design (IWSSD-10)*, pages 11–22, San Diego, 2000.

[76] Tony Gorschek and Claes Wohlin. Requirements Abstraction Model. *Requir. Eng.*, 11(1):79–101, 2005.

[77] Nadine Heumesser and Frank Houdek. Towards Systematic Recycling of Systems Requirements. In *Proceedings of 25th International Conference on Software Engineering (ICSE)*, pages 512–519, Portland, Oregon, USA, 2003.

[78] Frank Houdek. *Empirisch basierte Qualitätsverbesserung. Systematischer Einsatz externer Experimente im Software Engineering.* PhD thesis, Universtät Ulm, 1999.

[79] Siv Hilde Houmb, Shareeful Islam, Eric Knauss, Jan Jürjens, and Kurt Schneider. Eliciting Security Requirements and Tracing them to Design: An Integration of Common Criteria, Heuristics, and UMLsec. *Requir. Eng.*, 15(1):63–93, March 2010.

[80] Anthony Hunter and Bashar Nuseibeh. Managing inconsistent specifications: reasoning, analysis, and action. *ACM Trans. Softw. Eng. Methodol.*, 7(4):335–367, 1998.

[81] IABG. V-Modell XT (Version 1.2). Technical report, Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung, 2006.

[82] Michael Jackson. *Problem Frames: Analysing and Structuring Software Development Problems.* Addison-Wesley Professional, 2001.

[83] Jonathan Jacky. *The Way of Z: Practical Programming with Formal Methods.* Cambridge University Press, 1997.

[84] Hung-Chin Jang. A knowledge-based analyzer for requirements specification analysis. In *Proceedings of Sixth International Conference on Tools with Artificial Intelligence*, pages 276–282, Nov 1994.

[85] Mayumi Itakura Kamata and Tetsuo Tamai. How Does Requirements Quality Relate to Project Success or Failure? In *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 69–78, Oct. 2007.

[86] Gustav Karner. Resource Estimation for Objectory Projects. *Objectory Systems*, 1993.

[87] Stefan Keil. Entwicklung eines erfahrungsgestützten Editors für Qualitätsmodelle. Studienarbeit, Leibniz Universität Hannover, Fachgebiet Software Engineering, 2007.

[88] Barbara Kitchenham, Hiyam Al-Khilidar, Muhammad Ali Babar, Mike Berry, Karl Cox, Jacky Keung, Felicia Kurniawati, Mark Staples, He Zhang, and Liming Zhu. Evaluating guidelines for empirical software engineering studies. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pages 38–47, New York, NY, USA, 2006. ACM.

[89] Ingo Kitzmann. Kritiksystem für ein erfahrungsbasiertes Anforderungswerkzeug. Bachelorarbeit, Leibniz Universität Hannover, Fachgebiet Software Engineering, 2007.

[90] Nadzeya Kiyavitskaya, Nicola Zeni, Luisa Mich, and Daniel M. Berry. Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. *Requirements Engineering Journal*, 13(3):207–239, September 2008.

[91] Eric Knauss. Einsatz computergestützter Kritiken für Anforderungen. *GI Softwaretechnik-Trends*, 27(1):27–28, Februar 2007.

[92] Eric Knauss. Projektmanagement auf Basis von Story Cards: Ein Erfahrungsbericht. In *Proceedings of Workshop Anforderungsbasiertes Projektmanagement, Fulda*, 2007.

[93] Eric Knauss. Quantitativer Vergleich von Anforderungsspezifikationen. In *Proceedings of GI-Fachgruppen-Treffen Requirements Engineering*, Berlin, November 2007.

[94] Eric Knauss, Christian El Boustani, and Thomas Flohr. Investigating the Impact of Software Requirements Specification Quality on Project Success. In Bomarius, Oivo, Päivi Jaring, and Pekka Abrahamsson, editors, *Proceedings of 10th International Conference on Product Focused Software Process Improvement (PROFES 2009)*, LNBIB, pages 28 – 42, Oulu, Finland, June 2009. Springer.

[95] Eric Knauss and Olesia Brill. SmartWiki: erfahrungsbasierte und leichtgewichtige Unterstützung beim Requirements Engineering für innovative Projekte. *GI Softwaretechnik-Trends*, 29(1):19–20, 11 2009.

[96] Eric Knauss, Olesia Brill, Ingo Kitzmann, and Thomas Flohr. SmartWiki: Support for High-Quality Requirements Engineering in a Collaborative Setting. In *Proceedings of 4th Workshop on Wikis for Software Engineering at ICSE 2009*, Vancouver, Canada, 2009.

[97] Eric Knauss and Christian El Boustani. Assessing the Quality of Software Requirements Specifications. In *Proceedings of 16th International Requirements Engineering Conference*, pages 341–342, Barcelona, Spain, Sept. 2008. IEEE Computer Society.

[98] Eric Knauss and Thomas Flohr. Managing Requirement Engineering Processes by Adapted Quality Gateways and critique-based RE-Tools. In *Proceedings of Workshop on Measuring Requirements for Project and Product Success*, Palma de Mallorca, Spain, November 2007. in conjunction with the IWSM-Mensura Conference.

[99] Eric Knauss, Thomas Flohr, Denis Beßen, Daniel Lübke, Andreas Röpke, Kurt Schneider, Beate Strüber, Joachim Trütken, Christian Trump, and Martin Willmann. Nicht perfekt, aber richtig - Erfahrungen mit Software-Anforderungen. *GI Softwaretechnik-Trends*, 27(1):65–70, 2007.

[100] Eric Knauss, Marina Koffler, and Olesia Brill. Analyse von Awareness-Bedarf bei verteiltem Requirements Engineering. *Softwaretechnik Trends*, 30(1):21–22, 2010. Beitrag zum Jahrestreffen 2009 der GI-Fachgruppe Requirements Engineering.

[101] Eric Knauss and Daniel Lübke. Using the Friction between Business Processes and Use Cases in SOA Requirements. In *Proceedings of the 2nd IEEE International Workshop on Requirements Engineering For Services*, 2008.

[102] Eric Knauss, Daniel Lübke, and Thomas Flohr. Learning to Tailor Documentation of Software Requirements. *Journal of Universal Knowledge Management*, 1(2):103–111, 2006.

[103] Eric Knauss, Daniel Lübke, and Sebastian Meyer. Feedback-Driven Requirements Engineering: The Heuristic Requirements Assistant. In *31st International Conference on Software Engineering (ICSE 2009)*, pages 587 – 590, Vancouver, Canada, 5 2009.

[104] Eric Knauss, Sebastian Meyer, and Kurt Schneider. Recommending Terms for Glossaries: A Computer-Based Approach. In *Proc. First International Workshop on Managing Requirements Knowledge MARK '08*, pages 25–31, September 8–8, 2008.

[105] Eric Knauss, Kurt Schneider, and Kai Stapel. A Game for Taking Requirements Engineering More Seriously. In *Proceedings of Third International Workshop on Multimedia and Enjoyable Requirements Engineering (MERE 08)*, Barcelona, Spain, 9 2008.

[106] Eric Knauss, Kurt Schneider, and Kai Stapel. Learning to Write Better Requirements through Heuristic Critiques. In *Proceedings of 17th IEEE Requirementes Engineering Conference (RE 2009)*, pages 387–388, Atlanta, USA, 2009. Poster.

[107] David A. Kolb. *Experiential Learning: Experience as a Source of Learning and Development*. Prentice Hall, Englewood Cliffs, N.J., 1984.

[108] Daniel Lübke. Transformation of Use Cases to EPC Models. In *Proceedings of the EPK 2006 Workshop, Vienna, Austria*, 2006.

[109] Daniel Lübke. *An Integrated Approach for Generation in Service-Oriented Architecture Projects*. Verlag Dr. Kovac, Hamburg, 2008.

[110] Daniel Lübke and Eric Knauss. Dealing with User Requirements and Feedback in SOA Projects. In *Proceedings of Workshop on Software Engineering Methods in Service Oriented Architecture*, Hannover, Germany, 2007.

[111] Jörg Leuser. Herausforderungen für halbautomatische Traceability-Erkennung. In *Systems Engineering Infrastructure Conference (SEIS-CONF)*, 11 2009.

[112] Kurt Lewin. Action Research and Minority Problems. *Journal of social issues*, 2:34–46, 1946.

[113] P. Liggesmeyer. *Software-Qualität. Testen, Analysieren und Verifizieren von Software.* Spektrum Akademischer Verlag, 2002.

[114] Stefanie N. Lindstaedt and Kurt Schneider. Bridging the Gap between Face-to-Face Communication and Long-term Collaboration. In *Proceedings of GROUP 97*, Phoenix, USA, Nov 1997. ACM.

[115] H. P. Luhn. The Automatic Creation of Literature Abstracts. *IBM Journal of Research and Development*, 2:159–165, 1958.

[116] Leszek A. Maciaszek. *Requirements Analysis and System Design*. Pearson Education Limited, 3rd edition, 2007.

[117] Ralf Melchisedech. Investigation of Requirements Documents Written in Natural Language. *Springer Requirements Engineering*, 3:91–97, 1998.

[118] Ralf Melchisedech. *Verwaltung und Prüfung natürlichsprachlicher Spezifikationen*. PhD thesis, Fakultät Informatik, Universität Stuttgart, Stuttgart, 2000.

[119] Jan Mendling and Markus Nüttgens. EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-Driven Process Chains (EPC). *Information Systems and e-Business Management (ISeB)*, 4(3):245–263, July 2005.

[120] Sebastian Meyer. Halbautomatische Generierung eines Glossars während der Dokumentation von Anforderungen. Masterarbeit, Leibniz Universität Hannover, Fachgebiet Software Engineering, 2007.

[121] Ikujiro Nonaka and Hirotaka Takeuchi. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, Oxford, 1995.

[122] Yeonjoo Oh, Mark D Gross, and Ellen Yi-Luen Do. Computer-Aided Critiquing Systems: Lessons Learned and New Research Directions. In *Proceedings of Computer Aided Architectural Design and Research in Asia (CAADRIA)*, pages 161–167, Chiang Mai, Thailand, 2008.

[123] Thomas Olsson, Richard Berntsson Svensson, and Björn Regnell. Nonfunctional requirements metrics in practice - an empirical document analysis. In *Proceedings of Workshop on Measuring Requirements for Project and Product Success*, Palma de Mallorca, Spain, November 2007. in conjunction with the IWSM-Mensura Conference.

[124] OMG. OMG Unified Modeling Language (OMG UML), Infrastructure: Version 2.2. OMG Specification 2.2, Object Management Group, 2009.

[125] OMG. OMG Unified Modeling Language (OMG UML), Superstructure: Version 2.2. OMG Specification 2.2, Object Management Group, 2009.

[126] Sinan Petrus Toma. Heuristische Unterstützung für die Formulierung von Sicherheitsanforderungen. Masterarbeit, Leibniz Universität Hannover, Fachgebiet Software Engineering, Hannover, 2009.

[127] Julia Pilarski. Konzept und Implementierung von Transformationen zwischen Use Case Diagrammen und tabellarischen Darstellungen. Bachelorarbeit, Leibniz Universität Hannover, Fachgebiet Software Engineering, 2007.

[128] Julia Pilarski and Eric Knauss. Transformationen zwischen UML-Use-Case-Diagrammen und tabellarischen Darstellungen. In *Proceedings des Workshop Domänenspezifische Modellierungssprachen im Rahmen der GI Modellierung*, Berlin, 2008.

[129] A. R. Plummer. Model-in-the-Loop Testing. In *Proceedings of the Institution of Mechanical Engineers*, volume 220 of *Part I: Journal of Systems and Control Engineering*, pages 183–199, 2006.

[130] Klaus Pohl. The three dimensions of requirements engineering: a framework and its applications. *Inf. Syst.*, 19(3):243–258, 1994.

[131] Klaus Pohl. *Requirements Engineering: Grundlagen, Prinzipien, Techniken*. dpunkt.verlag GmbH, Heidelberg, Germany, 2 edition, 2008.

[132] M. Polanyi. *The Tacit Dimension*. Doubleday, Garden City, NY, 1966.

[133] M. F. Porter. An algorithm for suffix stripping. *Readings in information retrieval*, 1:313–316, 1997.

[134] Oxford University Press. *Oxford Dictionary of English*. Oxford University Press, 2005.

[135] Andreas Rausch and Stephan Höppner. *V-Modell XT – eine Einführung*, volume 3 of *Software-Qualitätsmanagement: Theorie und Praxis*, chapter 1, pages 1–24. Logos-Verlag, Berlin, 2005.

[136] Jason E. Robbins. Design Critiquing Systems. Technical report, Department of Information and Computer Science, University of California, Irvine, 1998.

[137] Jason E. Robbins and David F. Redmiles. Software architecture critics in the Argo design environment. *Knowl.-Based Syst.*, 11(1):47–60, 1998.

[138] Jason Elliot Robbins, David M. Hilbert, and David F. Redmiles. Using Critics to Analyze Evolving Architectures. In *Proceedings of Second International Software Architecture Workshop*, 1996.

[139] S. Robertson and J. Robertson. *Mastering the Requirements Process,*. Addison-Wesley, 1999.

[140] Winston W. Royce. Managing the Development of Large Software Systems. In *Proceedings of IEEE WESCON*, pages 328 – 338, 1970.

[141] Thorsten Rumann. Erfahrungsmanagement in der heuristischen Softwareprüfung. Bachelorarbeit, Leibniz Universität Hannover, Fachgebiet Software Engineering, 2007.

[142] Chris Rupp. *Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxiss*. Hanser Fachbuchverlag, 2007. 4. Auflage.

[143] Stuart. Russell and Peter Norvig. *Artificial Intelligence: a Modern Approach*. Prentice Hall, New Jersey, 1995.

[144] V.F.A. Santander and J.F.B. Castro. Deriving use cases from organizational modeling. In *Proceedings of IEEE Joint International Conference on Requirements Engineering (RE)*, pages 32–39, Essen, Germany, 2002.

[145] Liza Sarkisyan. Herleitung und Validierung eines Modells zur Planung von Requirements Engineering Aktivitäten. Diplomarbeit, Leibniz Universität Hannover, Fachgebiet Software Engineering, 2009.

[146] D. A. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York, 1983.

[147] K. Schneider, J.-P. von Hunnius, and V.R. Basili. Experience in implementing a learning software organization. *IEEE Software*, 19(3):46–49, May/Jun 2002.

[148] Kurt Schneider. *Abenteuer Softwarequalität: Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement*. Dpunkt, 2007.

[149] Kurt Schneider. Generating Fast Feedback in Requirements Elicitation. In Pete Sawyer, Barbara Paech, and Patrick Heymans, editors, *Requirements Engineering: Foundation for Software Quality (REFSQ 2007)*, volume 4542 of *LNCS*, pages 160 – 174, Trondheim, Norway, 2007. Springer.

[150] Kurt Schneider. Improving Feedback on Requirements through Heuristics. In *Proceedings of 4th World Congress for Software Quality (4WCSQ)*, 2008.

[151] Kurt Schneider. *Experience and Knowledge Management in Software Engineering*. Springer-Verlag, 2009.

[152] Kurt Schneider, Kai Stapel, and Eric Knauss. Beyond Documents: Visualizing Informal Communication. In *Proceedings of Third International Workshop on Requirements Engineering Visualization (REV 08)*, Barcelona, Spain, 9 2008.

[153] J. Schäuffele and T. Zurawka. *Automotive Software Engineering*. Vieweg, 2006.

[154] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2002.

[155] Christian Seybold, Silvio Meier, and Martin Glinz. Evolution of Requirements Models by Simulation. In *Proceedings of the International Worksho on Principles of Software Evolution (IWPSE'04)*, pages 43–48, Kyoto, September 2004.

[156] Dongmin Shin, Jae won Lee, Jongheum Yeon, and Sang goo Lee. Context-Aware Recommendation by Aggregating User Context. *IEEE International Conference on E-Commerce Technology*, pages 423–430, 2009.

[157] Barry G. Silverman. Survey of expert critiquing systems: practical and theoretical frontiers. *Commun. ACM*, 35(4):106–127, 1992.

[158] Joel So and Daniel M. Berry. Experiences of Requirements Engineering for Two Consecutive Versions of a Product at VLSC. In *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, pages 216–221, Washington, DC, USA, 2006. IEEE Computer Society.

[159] Stéphane S. Somé. Supporting Use Case Based Requirements Engineering. *Information & Software Technology*, 48(1):43–58, 2006.

[160] Cleidson R. B. Souza, Jair S. Ferreira Jr., Kleder M. Gonçalves, and Jacques Wainer. A Group Critic System for Object-Oriented Analysis and Design. In *ASE '00: Proceedings of the 15th IEEE international conference on Automated software engineering*, page 313, Washington, DC, USA, 2000. IEEE Computer Society.

[161] Kai Stapel, Eric Knauss, and Christian Allmann. Lightweight Process Documentation: Just Enough Structure in Automotive Pre-Development. In Rory V. O'Connor, Nathan Baddoo, Kari Smolander, and Richard Messnarz, editors, *Proceedings of the 15th European Conference, EuroSPI*, Communications in Computer and Information Science, pages 142–151, Dublin, Ireland, 9 2008. Springer.

[162] Kai Stapel, Eric Knauss, Kurt Schneider, and Matthias Becker. Towards Understanding Communication Structure in Pair Programming. In Alberto Sillitti, editor, *XP 2010*, volume 48 of *LNBIP*, pages 117–131, Trondheim, Norway, 6 2010. Springer.

[163] Davor Svetinovic, Daniel M. Berry, Nancy A. Day, and Michael W. Godfrey. Unified use case statecharts: case studies. *Requir. Eng.*, 12(4):245–264, 2007.

[164] Christian Szongott. Werkzeuggestützte Aufwandsabschätzung bei der Erstellung von Use Cases. Bachelorarbeit, Leibniz Universität Hannover, Fachgebiet Software Engineering, 2007.

[165] Sri Fatimah Tjong, Nasreddine Hallam, and Michael Hartley. Improving the Quality of Natural Language Requirements Specifications through Natural Language Requirements Patterns. In *Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)*, pages 199–204, Seoul, Korea, 2006. IEEE Computer Society.

[166] Sylvie Trudel and Alain Abran. Improving the Qualitiy of Functional Requirements by Measuring Their Functional Size. *Proceeding of the International Conferences IWSM, MetriKon, and Mensura*, pages 287–301, 2008.

[167] Rini van Solingen and Egon Berghout. *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development.* McGraw-Hill Publishing Company, 1999.

[168] Rini van Solingen, Egon Berghout, Rob Kusters, and Jos Trienekens. No Improvement without Learning: Prerequisites for Learning the Relations between Process and Product Quality in Practice. In Frank Bomarius and Markku Oivo, editors, *Product Focused Software Process Improvement*, volume 1840/2000 of *Lecture Notes in Computer Science*, pages 59–76. Springer Berlin / Heidelberg, 2000.

[169] Matthias Weidlich, Alexander Grosskopf, Daniel Lübke, Kurt Schneider, Eric Knauss, and Leif Singer. Verzahnung von Requirements Engineering und Geschäftsprozessdesign. In *Workshop Proceedings of the SE 2009 - REBPM, Kaiserslautern, Germany*, 2009.

[170] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt. Automated quality analysis of Natural Language requirement specifications. In *Proceedings of PNSQC Conference*, 1996.

[171] William M. Wilson, Linda H. Rosenberg, and Lawrence E. Hyatt. Automated analysis of requirement specifications. In *Proceedings of the 19th International Conference on Software Engineering (ICSE '97)*, pages 161–171, New York, NY, USA, 1997. ACM.

[172] Stefan Winkler. Information Flow Between Requirement Artifacts. In *Proceedings of REFSQ 2007 International Working Conference on Requirements Engineering: Foundation for Software Quality*, volume 4542 of *Lecture Notes in Computer Science*, pages 232–246, Trondheim, Norway, 2007. Springer Berlin / Heidelberg.

[173] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation In Software Engineering: An Introduction.* Kluwer Academic Publishers, Boston / Dordrecht / London, 2000.

[174] R. K. Yin. *Case Study Research: Design and Methods.* Sage, Thousand Oaks, CA, 2002.

[175] Eric Yu. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In *Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97)*, pages 226–235, Washington D.C., USA, 1997.

[176] Marvin V. Zelkowitz and Dolores R. Wallace. Experimental validation in software engineering. *Information & Software Technology*, 39(11):735–743, 1997.

[177] Jörg Zettel. Methodology Support in CASE Tools and Its Impact on Individual Acceptance and Use: A Controlled Experiment. *Empirical Software Engineering*, 10(3):367–394, Juli 2005.

[178] Wolfgang Zuser, Thomas Grechening, and Monika Köhle. *Software Engineering mit UML und dem Unified Process.* Pearson Studium, 2004.

Lebenslauf

Geboren am	17. September 1977 in Hattingen
1984–1997	Grundschule und Gymnasium in Bochum
10/1998–03/2002	Studiengang Diplom Informatik an der Universität Dortmund
Abschluss	Vordiplom
04/2002–11/2003	Studiengang Angewandte Informatik (Bachelor of Science) an der Universität Hannover
Abschluss	Bachelor of Science, **Bachelorarbeit:** Fachgebiet Datenbanksysteme, *Erweiterung eines Visualisierers für räumliche Datenbanken um die Unterstützung von Updates*
11/2003–10/2005	Studiengang Angewandte Informatik (Master of Science) an der Universität Hannover. **Schwerpunkte:** Softwaretechnik, Datenbanksysteme, Algorithmen und Datenstrukturen **Anwendungsfächer:** Maschinenbau, Betriebswirtschaftslehre
Abschluss	Master of Science, **Masterarbeit:** Fachgebiet Datenbanksysteme, *Updates und Versionierung in räumlichen Datenbanken*
11/2005–10/2010	Studiengang Informatik (Promotion) an der Leibniz Universität Hannover, Institut für Praktische Informatik, Fachgebiet Software Engineering **Schwerpunkte:** Requirements Engineering, Agile Methoden, Projektmanagement
Abschluss	Doktor-Ingenieur, **Doktorarbeit:** *Verbesserung der Dokumentation von Anforderungen auf Basis von Erfahrungen und Heuristiken*
11/2005–03/2010	Leibniz Universität Hannover, Institut für Praktische Informatik, Fachgebiet Software Engineering, **Wissenschaftlicher Mitarbeiter**, Schwerpunkt: Requirements Engineering
seit 04/2010	Leibniz Universität Hannover, Institut für Praktische Informatik, Fachgebiet Software Engineering, **Wissenschaftlicher Mitarbeiter**, Schwerpunkt: Global Software Engineering
Mitgliedschaften	Gesellschaft für Informatik (seit 1999), Aktiv in Arbeitskreisen der Fachgruppe Requirements Engineering
	IEEE Computer Society
Ämter	Vertreter der Wissenschaftlichen Mitarbeiter in der Studienkomission Informatik
	Vertreter der Promovierenden der Fakultältät für Elektrotechnik und Informatik in der Graduiertenakademie