Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Diskussion von Agilen und Plan-basierten RE Praktiken und deren Kombination

Masterarbeit

im Studiengang Technische Informatik

von

Reza Faraij Jenaghard

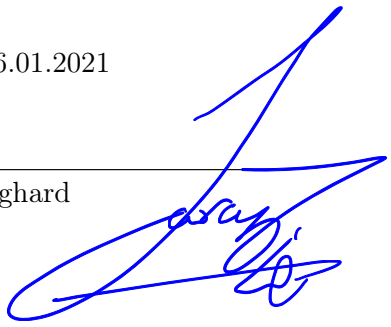
Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Dr. Jil Klünder
Betreuer: Nils Prenner

Hannover, 06.01.2021

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 06.01.2021



Reza Faraij Jenaghard

Kurzfassung

Einer der wichtigsten Schritte im Software Engineering ist das Requirements Engineering. Requirements Engineering wird auf strukturierte (traditionelle) und agile Weise praktiziert, um Anforderungen zu erheben, auszuhandeln, zu dokumentieren und zu verwalten. In dieser Masterarbeit werden die Herausforderungen des agilen Requirements Engineering und deren Lösungen in einem systematischen Literaturreview (SLR) Prozess untersucht. Das Ergebnis sind 13 Over-Challenges. Diese Over-Challenges fassen ähnliche Herausforderungen zusammen, die im Forschungsprozess identifiziert werden. Lösungen für diese Herausforderungen aus dem SLR-Prozess wurden ebenfalls identifiziert und zu den Herausforderung zugeordnet. Darüber hinaus werden die Herausforderungen, für die es im SLR-Prozess keine Lösung gibt, diskutiert und es werden Lösungen für sie aus anderen Arbeiten aus der agilen und traditionellen Forschung vorgestellt. Einige Vorschläge werden zudem in dieser Arbeit gegeben, die manche dieser Problemen beseitigen könnten. Allerdings sind einige Probleme noch ungelöst und erfordern weitere Forschung, wie z. B. das formale Umsetzen oder Verifizieren von Prozessen im agilen Requirements Engineering, die Re-Priorisierung und eine etablierte Technik zur Erfassung von Qualitätsanforderungen.

Abstract

Discussion of agile and plan-based RE practices and their combination

One of the most important steps in software engineering is requirements engineering. Requirements engineering is practiced in a structured (traditional) and agile manner to elicit, negotiate, document, and manage requirements. In this master thesis, the challenges of agile requirements engineering and their solutions are investigated in a systematic literature review (SLR) process. The result is 13 over-challenges. These over-challenges summarize similar challenges identified in the research process. Solutions to these challenges from the SLR process were also identified and mapped to each challenge. In addition, the challenges that do not have a solution in the SLR process are discussed and solutions for them are presented from other work in agile and traditional research. Some suggestions are also given in this work that could eliminate some of these problems. However, some problems are still unsolved and require further research, such as formal execution or verification of processes in agile requirements engineering, re-prioritization, and an overall acceptable technique for gathering quality requirements.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Einer der wichtigsten Schritte im Software Engineering (SE) ist das Requirements Engineering (RE), bei dem die Anforderungen im Entwicklungsprozess definiert, dokumentiert und fortlaufend angepasst werden. Das RE ist heutzutage für das SE unerlässlich und bildet eine Brücke zwischen den Stakeholdern und Entwicklern dar, indem die Anforderungen korrekt von den Stakeholdern an die Entwickler übertragen werden. Softwareprojekte scheitern oft aufgrund der Tatsache, dass nicht genau das entwickelt wird, was der Kunde tatsächlich wünscht oder braucht. Wenn der Requirements Engineer die Anforderungen nicht vollständig oder nicht eindeutig dokumentiert, kann dies problematisch werden und im schlimmsten Fall zum Scheitern des Projekts führen. Deswegen wird ein erfolgreiches Projekt oft durch den Requirements Engineering-Prozess vorherbestimmt [36]. Es gibt zwei Arten des Requirements Engineering zur Ermittlung von Anforderungen: das traditionelle Requirements Engineering (TRE) und das agile Requirements Engineering (ARE). Im TRE wird die Dokumentation für die Systementwicklung durch das Sammeln aller Systemanforderungen vom Kunden in einem sehr frühen Stadium erstellt [37]. Auf der einen Seite nimmt der TRE-Prozess viel Zeit in Anspruch, weil zunächst alle Systemanforderungen spezifiziert werden müssen, bevor die eigentliche Entwicklung beginnen kann. Auf der anderen Seite bietet TRE entscheidende Vorteile wie eine solide und konsistente Dokumentation der funktionalen Anforderungen sowie der Qualitätsanforderungen.Dies erleichtert während der anschließenden Phase der Implementierung die Aufgabenverteilung und die Zusammenarbeit in großen Projekten sowie die Traceability.Bei der agilen Entwicklung wird viel Wert auf kontinuierliche Zusammenarbeit und Interaktion mit den Kunden gelegt. Die Entwicklung in der Softwareindustrie erfordert einen flexiblen und schnellen Prozess, um auf Projektänderungen zu reagieren und ein Produkt möglichst schnell an den Kunden zu liefern [5,38]. Agilität ist ein Vorteil des ARE, die eine schnelle Anpassung an Marktveränderungen und schnelle und flexible Reaktionen auf Kundenwünsche ermöglicht.Außerdem hilft die Agilität dabei, sich an Änderungen ohne Qualitätsverlust produktiv und kosteneffizient anzupassen und diese umzusetzen, woraus ein kontinuierlicher Wettbewerbsvorteil entsteht [35]. Viele Unternehmen arbeiten heutzutage mit agilen Methoden wie die Agile Software-Entwicklung (ASD), um qualitativ hochwertige Software schnell entwickeln zu können.Dafür werden kurze, intensive Entwicklungsphasen verwendet, die zu wiederholten kleinen Releases führen. Die ASD stellt eine Abkehr von der bis dahin üblichen dokumentengesteuerten Methode für die Software-Entwicklung dar, jedoch ist agile Entwicklung ein Überbegriff für mehrere agile Methoden, deren Charakteristikum vor allem die Flexibilität ist [36].

Durch die Verwendung kurzer Entwicklungszyklen werden neue Anforderungen in der nächsten Iteration zeitnah umgesetzt und unnötige bzw. veraltete Anforderungen in der neuen Iteration entfernt. Dieser Prozess ist daher schneller als der TRE-Prozess, da zu Beginn weniger Aufwand zur Erfassung der Anforderungen erforderlich ist und die Entwicklung schnell gestartet werden kann.Zudem bietet das ARE einen iterativen, inkrementellen Rahmen, der den Teams hilft, den Schwerpunkt auf eine schnelle Lieferung der Produkte zu setzen. ARE-Methoden helfen dabei, den RE-Prozess in kleinere Abschnitte aufzuteilen und in jedem Schritt einen Teil des Systems bereitzustellen und es im nächsten Schritt durch Feedback und neue Anforderungen von Stakeholdern zu verbessern. ARE bietet den Vorteil einer konstanten Kommunikation zwischen Kunden und Entwicklern. Als Ergebnis können die Entwickler das System rechtzeitig liefern, welches die Erwartungen des Kunden erfüllt und somit den Geschäftswert steigert [5,39].

1.1 Problemstellung

Es gibt sowohl agile und traditionelle Praktiken für das Requirements Engineering, die von zahlreichen Wissenschaftlern erforscht und veröffentlicht wurden. Trotz der vielen Vorteile der agilen Softwareentwicklung und deren Methoden gibt es einige damit verbundene Schwierigkeiten wie die Kosten- und Zeitplanabschätzung, nicht-funktionale Anforderungen sowie Zugang und Beteiligung der Kunden [42]. In diesen Punkten bieten die Methoden des Traditionellen Requirements Engineering Vorteile gegenüber den agilen Methoden. Deshalb nutzen viele Firmen das traditionelle Requirements Engineering, um diese Probleme auszugleichen. Bei der die Kundenbeteiligung am Entwicklungsprozess führt die große Bedeutung der Dokumentation im TRE-Prozess zu Austausch zwischen Entwicklern und Kunden hauptsächlich am Beginn des Entwicklungsprozesses, wohingegen sich das ARE auf die direkte Kommunikation konzentriert, sodass der Kunde während des gesamten Lebenszyklus der Softwareentwicklung problemlos mit dem Team kommunizieren kann [5, 39]. Es gibt jedoch keine Übersicht darüber, wie die einzelnen Probleme im agilen durch traditionelle Methoden angegangen werden, um Unternehmen zu unterstützen.

1.2 Zielsetzung und Struktur der Arbeit

Diese Masterarbeit hat zwei Ziele. Zuerst steht die Durchführung einer systematischen Literaturrecherche (SLR) zu den Herausforderungen des Agilen Requirements Engineering im Zentrum, um Herausforderungen und auch mögliche Lösungen zu identifizieren, die in der Literatur erwähnt werden. Das zweite Ziel besteht in der Untersuchung und Lösungsentwicklung ausgehend von agilen oder traditionellen Methoden des Requirements Engineering für diejenigen Herausforderungen, für die keine Lösungen in der Literatur vorgeschlagen wurden. Diese Masterarbeit ist wie folgt strukturiert: Im zweiten Kapitel werden die Grundlagen für diese Arbeit behandelt. Im dritten Kapitel werden verwandte Arbeiten präsentiert. Im vierten Kapitel wird die Implementierung der SLR beschrieben. Im fünften Kapitel werden die Resultate aus der SLR vorgestellt und diskutiert. Im sechsten Kapitel werden Lösungen für die Herausforderungen des agilen Requirements Engineering diskutiert und abschließend im siebten Kapitel werden die wichtigsten Inhalte zusammengefasst und ein Ausblick auf die zukünftige Arbeit gegeben.

Kapitel 2

Grundlagen

Dieses Kapitel enthält die grundlegenden Inhalte dieser Masterarbeit in sechs Abschnitten. Zuerst wird das traditionelle Requirements Engineering (RE) beschrieben. Dann werden agile Softwareentwicklung (ASD), agiles Requirements Engineering und Kategorien möglicher Anforderungen beschrieben. Danach werden der Systematische Literaturreview (SLR) und das Snowballing-Verfahren erläutert.

2.1 Traditionelles Requirements Engineering (TRE)

Das Requirements Engineering beschäftigt sich mit der Identifizierung, Modellierung, Kommunikation und Dokumentation der Anforderungen an ein System sowie der Kontexte, in denen das System eingesetzt werden soll [39, 40]. Die Verwendung des Begriffs "Engineering" impliziert, dass systematische und wiederholbare Techniken verwendet werden, um sicherzustellen, dass die Systemanforderungen vollständig, konsistent und relevant sind [40]. Hierfür stehen zahlreiche Techniken zur Verfügung. Das Ziel des RE ist, vor Beginn der Systementwicklung zu wissen, was genau zu implementieren ist, um kostspielige Nacharbeiten zu vermeiden. Der RE-Prozess besteht aus zwei Teilen: Anforderungsanalyse und Anforderungsmanagement. Im ersten Teil werden die Anforderungen identifiziert, abgestimmt, dokumentiert und validiert. Hierbei wird ein Systemanforderungsdokument erstellt, validiert und gepflegt [40]. Die darin festgehaltenen Anforderungen beschreiben, was zu tun ist, aber nicht, wie die Anforderungen implementiert werden [39, 42]. Im zweiten Teil werden die Anforderungen verwaltet,um Nachvollziehbarkeit von etwaigen Änderungen der Anforderungen und Traceability sicherzustellen (siehe [43]). In der Beschreibung von [39, 42] werden diese beiden Teile in folgende fünf Hauptaktivitäten zusammengefasst: (1) Erhebung; (2) Analyse (Interpretation und Negotiation); (3) Dokumentation; (4) Validierung & Verifizierung; (5) Management (Änderung, Management & Tracing). In den folgenden Abschnitten wird jede dieser Aktivitäten ausführlich erläutert.

2.1.1 Requirements Erhebung

Die erste Phase ist das Erheben von Anforderungen eines Systems und deren Ausarbeitung [42], indem die Beteiligten, z. B. Kunden, Entwickler und Benutzer befragt werden [39]. Diese Anforderungen können sehr gut oder aber nur vage verstanden werden oder es kann sich um neuartige Probleme handeln [42]. Die Anwendungsdomäne, die Geschäftsbedürfnisse, die Systemgrenzen, die Stakeholder und das Problem selbst müssen verstanden werden, da sie für das Verständnis des zu entwickelnden Systems

unerlässlich sind [39]. Hierbei wurden die folgenden Methoden zur Anforderungserhebung hervorgehoben: „Interviews, use cases / scenarios, observation and social analysis, focus groups, brainstorming, prototyping" (vgl. [39], Seite 2).

2.1.2 Requirements Analyse (Interpretation & Negotiation)

Der Hauptzweck dieser zweiten Hauptaktivität besteht darin, jegliche Probleme (Konflikte, Überschneidungen, Auslassungen, und Inkonsistenzen) in den formulierten Anforderungen zu identifizieren und zu lösen und damit die zuvor ermittelten Anforderungen zu verfeinern. Die Ausgangsliste dieser Aufgabe wird dann verhandelt und priorisiert, indem sich die Stakeholder des Systems untereinander auf eine Reihenfolge der Anforderungen einigen [42]. Elshandidy und Mazen in [42] wiesen darauf hin, dass der größte Teil der Forschung in diesem Bereich auf die Entwicklung neuer Methoden und Techniken oder auf die Verbesserung bestehender Methoden und Techniken ausgerichtet ist, um die Qualität der produzierten Anforderungen möglichst effizient und genau zu bewerten (vgl. [42], Abschnitt 4.2). Eine der wichtigsten Techniken zur Anforderungsanalyse ist die „Joint Application Development (JAD), Requirements Priorization und Modeling" (vgl. [39], Seite 2). Dies ist eine modifizierte Gruppensitzung oder ein Workshop mit einem strukturierten Analyseansatz, bei der Entwickler und Kunden die gewünschten Produktmerkmale diskutieren (vgl. [39], Abschnitt 2.2).

2.1.3 Requirements Dokumentation

Der Zweck der Anforderungsdokumentation besteht in der Festlegung von Anforderungen zwischen Stakeholdern und Entwicklern. Das Anforderungsdokument ist die Grundlage für die Bewertung nachfolgender Produkte und Prozesse wie Design-, Test-, Verifikations- und Validierungsaktivitäten sowie für die Änderungskontrolle. Ein gutes Anforderungsdokument ist eindeutig, vollständig, korrekt, verständlich, konsistent, prägnant und durchführbar. Abhängig von der Kunden-Lieferanten-Beziehung kann die Spezifikation der Anforderung Teil des Vertrags sein [39]. Wichtige Techniken zur Dokumentation sind „data flow diagrams (DFDs), entity relationship diagrams (ERDs), statecharts, class diagrams, use cases, (viewpoint, object oriented, and formal requirements modeling methods), Unified Modelling Language (UML)" (vgl. [42]).

2.1.4 Requirements Validierung und Verifizierung

Die Validierung der Anforderungen ist der Prozess, der sicherstellt, dass das Entwicklungsteam das richtige Produkt implementiert. Dies wird durch eine Überprüfung der Anforderungsartefakte mit den Stakeholdern erreicht, um eine exakte Abbildung ihrer Bedürfnisse in den Modellen und in der Dokumentation zu gewährleisten [42]. Elshandidy et al. [42] berichten, dass die Hauptforschungen in diesem Bereich auf Simulationen, Animationen und der automatischen Generierung von Zustandsinvarianten aus Anforderungsspezifikationen basieren. Die Verifizierung der Anforderungen hingegen ist der Prozess, der sicherstellt, dass das Entwicklungsteam das Produkt richtig herstellt. Formale Anforderungen können durch Modell-Prüfung (Modell-Checking), Modell-Erfüllbarkeit (Model-Satisfiability) und Notationen verifiziert werden [42]. Typische Anforderungsvalidierungs- und Verifikationsansätze, die in der traditionellen Entwicklung verwendet werden, sind: Tracing-Ansätze, Prototyping, Testen, Schreiben von

Benutzeranleitungen (user manual writing), formale Validierungen sowie Reviews und Inspektionen, z. B. Walkthroughs, formale Inspektionen und Checklisten [42].

2.1.5 Requirements Management

Das Ziel des Requirements Management ist, alle Informationen zu erfassen, zu speichern, zu verbreiten und zu verwalten. Das Requirements Management beschäftigt sich mit der Kontrolle von Anforderungsänderungen, Versionierung, sowie das Requirements Tracing und das Requirements Status Tracing. Dabei verwaltet das Requirements Tracing Änderungen eines Systems, indem es Beziehungen zwischen Anforderungen, Design und Implementierung eines Systems herstellt [39].

2.2 Agile Software Entwicklung (ASD)

ASD beschreibt eine Gruppe von Software-Entwicklungsmethoden, die auf inkrementeller und iterativer Entwicklung basieren. Gemäß dem agilen Manifest wird die ASD in Form von Werten, Prinzipien und Praktiken definiert. Im Gegensatz zu den traditionellen Entwicklungsmethoden verlangt die agile Entwicklung keine detaillierte Vorausplanung des gesamten Projekts und fördert sowohl eine schnelle Lieferung als auch die Beteiligung der Stakeholder am gesamten Entwicklungsprozess [42]. Elshandidy et al. [42] fassen die Hauptunterschiede zwischen traditioneller und agiler Entwicklung wie in Tabelle 2.1 dargestellt zusammen. Die agile Entwicklung umfasst sechs Hauptmethoden, die alle die Kernwerte und -prinzipien sowie einige gemeinsame Praktiken teilen, wie z. B. die Entwicklung mit oder beim Kunden vor Ort (on-site customer), kurze Iterationen, häufige Releases, Priorisierung auf der Basis der zu liefernden Software-Funktionen (am wichtigsten werden die Funktionen eingestuft, den höchsten Geschäftswert für den Kunden darstellen), einfaches Design und Peer-Reviews [42]. Intuitiv definiert jede Methode ihre eigenen Praktiken auf der Grundlage der Vision dieser Methode. Beispiele für agile Methoden sind: Scrum, Extreme Programming (XP; XP2), Feature-Driven Development (FDD), Dynamic Systems Development Method (DSDM), Crystal Methodologies und Adaptive Software Development (vgl. [42]).

2.3 Agiles Requirements Engineering

Sowohl vage Formulierungen als auch häufige Änderungen der Anforderungen sind unvermeidlich. Daraus entsteht die Herausforderung, den traditionellen RE-Praktiken zu folgen und klare, konsistente und vollständige Anforderungen zu entwickeln, was vor Beginn des Designs und der Implementierung nur schwer möglich ist [42]. Elshandidy et al. [42] weisen darauf hin, dass ein Vorteil der Agilen Software-Entwicklung darin besteht, den ständigen Änderungen der Anforderungen zu begegnen, mit der wachsenden technologischen Entwicklung umzugehen und funktionierende Software frühzeitig auf den Markt zu bringen, was die ASE zu einem sehr attraktiven Software-Entwicklungsansatz gemacht hat [42]. In den nächsten Unterabschnitten werden die agilen RE-Praktiken erläutert.

2.3.1 Requirements Elicitation in agiler Software Entwicklung

Die Techniken, die in dieser Phase der Entwicklung (Elicitation) verwendet werden, sind der traditionellen Entwicklung sehr ähnlich. Bei der agilen Methode wird der Elicitation-Prozess allerdings iterativ und kontinuierlich vor jeder Entwicklungsiteration durchgeführt, wobei ein Schwerpunkt auf der direkten Kommunikation der erarbeiteten Anforderungen mit dem Kunden liegt. „Face-to-Face Communication, User stories and Prototyping“ sind die häufigsten Techniken, die bei der agilen Anforderungserhebung verwendet werden (vgl. [42]). Allerdings können diese Techniken bei agilen und traditionellen Ansätzen unterschiedlich eingesetzt und auch zur Verwirklichung unterschiedlicher Ziele verwendet werden. Elshandidy et al. [42] erklären, dass beispielsweise Prototypen in der agilen Methode im Gegensatz zur traditionellen Methode verwendet werden, um komplizierte Anforderungen zu verstehen und die Lücke zwischen den verschiedenen Perspektiven verschiedener Stakeholder zu überwinden.

Tabelle 2.1: Hauptunterschiede zwischen traditioneller und agiler Entwicklung [42]

	Traditionell	Agile
Planung Voraussagend (Predictive)	Anpassungsfähig	(Adaptive)
Fokus	Prozessorientiert	Menschenorientiert
Dokumentation	Umfassend und konstant	Minimal und evolutiv
Rollen der einzelnen Mitglieder des Entwicklungsteams	Strikte Trennung der Rollen durch Spezialisierung	Selbstorganisierendes und funktionsübergreifendes (cross-functional) Team
Rolle des Kunden	Wichtig	Kritisch
Entwicklungsmodell	Lebenszyklusmodell (waterfall, spiral oder andere Varianten)	Iterativ und inkrementell
Kommunikation	Formal	Informell
Management-Stil	Befehl und Kontrolle	Führung und Zusammenarbeit
Qualitätskontrolle	Starke Planung und strenge Kontrolle mit späten intensiven Tests	Kontinuierliche Kontrolle von Anforderungen und Entwicklung mit kontinuierlichen Tests

2.3.2 Requirements Analyse in agiler Software Entwicklung

In der Interpretations- und Verhandlungsphase werden in der agilen Entwicklung, ähnlich wie beim TRE, die Anforderungen auf Vollständigkeit, Konsistenz, Wesentlichkeit und Durchführbarkeit geprüft. Dies wird beispielsweise mittels JADs für alle beteiligten Stakeholder erreicht, um die bereits erstellten User Stories zu priorisieren und mögliche Konflikte oder Versäumnisse in den Anforderungen, falls vorhanden, zu klären. JADs, User Stories und Priorisierung werden in [42] beschrieben.

2.3.3 Requirements Dokumentation in agiler Software Entwicklung

Auch bei der Software-Anforderungsspezifikation und Dokumentation unterscheidet sich die agile Methode vom traditionellen RE-Ansatz. In TRE müssen alle Anforderungen im Detail dokumentiert werden, hingegen ist beim Agilen Ansatz die Dokumentation nicht

verpflichtend, sondern wird nur bei Bedarf, durch das Entwicklungsteam erstellt. Im Agile RE werden in der Regel User Stories anstelle einer Dokumentation verwendet, um einen gemeinsamen Referenzpunkt unter den Stakeholdern zu erzeugen [16].

2.3.4 Requirements Validierung und Verifikation in ASD

Agile Methoden verwenden häufige Review-Meetings und verschiedene Tests zur Validierung und Verifizierung von Anforderungen. Die Verifikation erfolgt in agilen Ansätzen durch Feedback im Rahmen von Review-Meetings und die Validierung geschieht eher informell. Obwohl die agile Software Entwicklung (ASD) die gleichen Techniken wie traditionelle Ansätze verwenden, gibt es dennoch entscheidende Unterschiede. Anders als bei der traditionellen Entwicklung wird bei der agilen Validierung von Anforderungen ein bereits funktionierendes Stück der entwickelten Software validiert und nicht ein riesiges Spezifikationsdokument. Am Ende eines Entwicklungszyklus wird ein Review-Meeting abgehalten, um die Bedenken und Probleme, die während des Zyklus aufgetreten sind, zu kommunizieren und zu lösen. Das Testen ist eine weitere Methode zur Validierung des Ergebnisses eines Entwicklungszyklus. Die gängigsten Testmethoden sind der Akzeptanztest, die test- und die akzeptanztest-getriebene Entwicklung (vgl. [42]).

2.3.5 Requirements Management in agiler Software Entwicklung

In der traditionellen Entwicklung besteht die Überprüfung der Einhaltung von Anforderungen im Abgleich mit der Dokumentation, da hierin alle Details jeder Anforderung von Beginn der Erhebung an bis zum Abschluss ihrer Implementierung erfasst, gespeichert und verfolgt werden. Obwohl das Einhalten aller dieser Informationen Beziehungen zwischen den Anforderungen, dem Design und der Implementierung des Systems herstellt, ist das Schreiben und Verwalten einer so umfangreichen Dokumentation eine komplizierte und zeitaufwändige Aufgabe. Außerdem besteht ein hohes Risiko, dass bei der Suche nach einer Information in dieser riesigen Dokumentation den Überblick verloren geht [42]. Die agile Entwicklung hingegen ist der Ansicht, dass Änderungen der Systemanforderungen unvermeidlich sind. Daher sind das Begrüßen und Anpassen von Änderungen in jeder Phase des Projekts das Kernstück agiler Methoden. Der agile Ansatz macht es einfacher und kostengünstiger im Vergleich zur traditionellen Entwicklung, Änderungen zu implementieren. Das Management im ARE erfolgt in der Regel durch direkte Kommunikation und wird durch drei Hauptpraktiken erreicht: iterative requirements engineering, frequent short releases, and immediate customer's feedback (vgl. [42]).

2.4 Arten von Anforderungen

Die Anforderungen werden in zwei Gruppen unterteilt: Auf der einen Seite gibt es funktionale Anforderungen (FR), auf der anderen Seite nicht-funktionale Anforderungen (NFR). Funktionale Anforderungen enthalten alle Funktionen, die ein System erfüllt oder erfüllen sollte sowie Vorgaben für das Aussehendes Systems. Sie werden aus Geschäftsprozessen abgeleitet, in denen das System angewendet werden soll. FR umfassen in der Regel funktionale, verhaltensbezogene und strukturelle Anforderungen. Hingegen definieren NFR das Verhalten des Systems und begrenzen das Systemverhalten (vgl. [43]). NFR werden auch Qualitätsanforderungen genannt, da sie die Eigenschaften des Systems

wie z. B. Leistungsanforderungen, Anforderungen an die Wartbarkeit, Zuverlässigkeit usw. beschreiben [1, 2, 3, 4].

2.5 Systematische Literaturrecherche (SLR)

Kitchen et al. [44] beschreiben eine Richtlinie zur Durchführung einer systematischen Literaturrecherche im Software Engineering. Diese Richtlinie wird im nächsten Abschnitt beschrieben. Eine systematische Literaturreview (SLR) stellt ein Mittel zur Bewertung und Interpretation aller verfügbaren Forschungsarbeiten dar, die für eine bestimmte Forschungsfrage, ein bestimmtes Themengebiet oder ein bestimmtes Phänomen von Interesse relevant sind [45]. Es gibt viele Gründe für die Durchführung einer systematischen Literaturreview. Die bekanntesten Gründe sind gemäß [44, 45]:
Die vorhandene Evidenz zu einer Behandlung oder Technologie zusammenzufassen, z. B. um die empirische Evidenz für den Vorteil und die Grenzen einer bestimmten agilen Methode zusammenzufassen; Lücken in der aktuellen Forschung zu identifizieren (um Gebiete für weitere Forschungen vorschlagen zu können); Ein Framework bzw. Hintergrund zur Verfügung stellen, um neue Forschungsaktivitäten angemessen einzuordnen. Der SLR-Prozess besteht aus drei Phasen: Planung des Reviews, Durchführung des Reviews und Berichterstattung über den Review [44, 45]. In der Planungsphase ist die Identifizierung des Bedarfs für einen Review der erste Schritt, danach müssen die Forschungsfragen (RQ) festgelegt werden, da diese die Basis für die Forschung und daher den wichtigsten Teil darstellen. Parallel werden relevante Schlüsselwörter identifiziert, da sie für die Definition passender Suchbegriffe erforderlich sind, wohingegen falsche oder schlechte Schlüsselwörter die Gültigkeit des SLR-Ergebnisses negativ beeinflussen. Als nächstes werden Einschluss- und Ausschlusskriterien aufgestellt zur Entscheidung darüber, welche Arbeiten für die Forschung qualifiziert sind und welche nicht. Am Ende wird ein Review-Protokoll entwickelt und ausgewertet (vgl. [44, 45]). In der Durchführungsphase ist die Identifizierung der Forschung die erste Aufgabe. Danach müssen die primären Forschungen ausgewählt und deren Studienqualität bewertet werden. Im letzten Schritt folgen die Datenextraktion, das Monitoring und die Datensynthese. Die Identifizierung der Forschungen erfolgt durch die Definition eines Suchbegriffs unter Verwendung des Booleschen Operators (“AND” und “OR”) in Kombination mit Schlüsselwörtern aus dem vorherigen Schritt (vgl. [44, 45]). In der Berichterstattungsphase wird der Hauptbericht erstellt und schließlich ausgewertet (vgl. [44, 45]).

2.6 Snowballingverfahren in SLR

Wohlin et al. [46] präsentieren einen Ansatz zur Optimierung des SLR-Prozesses namens Snowballing, wobei der Suchprozess von zentralem Interesse ist. Die Basisplanung und die Motivation für eine systematische Literaturstudie sei unabhängig vom Suchprozess [46]. Daher sind die grundlegenden Schritte zur Planung einer Literaturstudie, wie sie in [44] vorgestellt werden, auch dann noch relevant, wenn ein anderer Ansatz für die Suche verwendet wird. Das Snowballing-Verfahren ist in Abbildung 2.1 dargestellt und wird in den folgenden Unterabschnitten beschrieben. Nachdem das Start-Set erstellt ist, wird das Snowballing-Verfahren in drei Iterationen in jeweils drei Schritten durchgeführt (Rückwärts-, Vorwärts-Snowballing und Inklusion & Exklusion von Artikeln). Das Start-

Set enthält Arbeiten, die im SLR-Prozess gesammelt wurden. Nach drei Iterationen wird eine Datenextraktion des endgültigen Satzes von Artikeln durchgeführt [46].

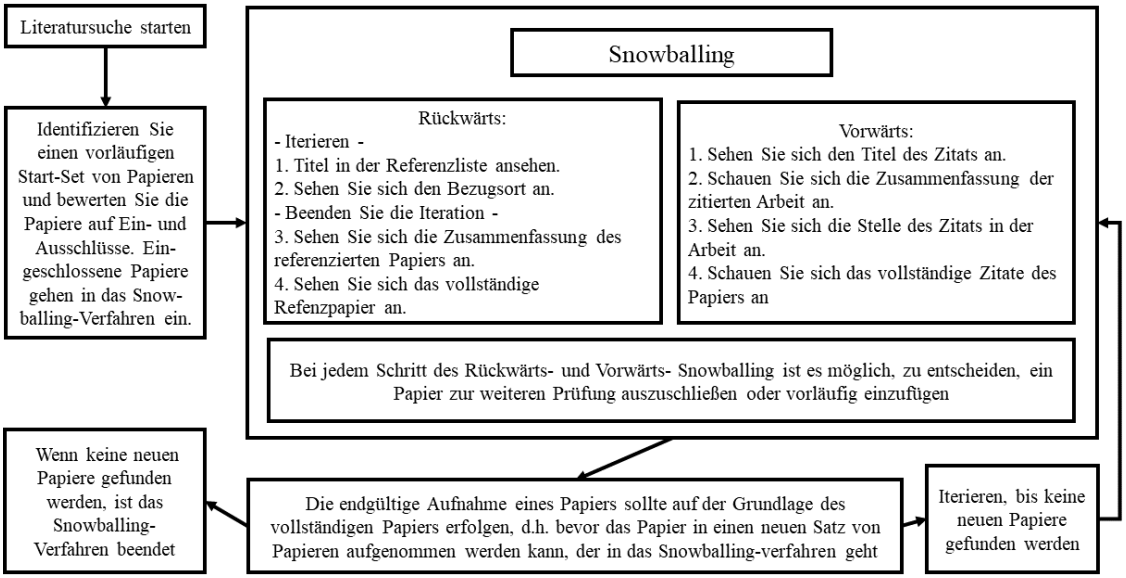


Abbildung 2.1: Snowballing-verfahren [46].

2.6.1 Start-Set

Bei der Datenbankrecherche werden im ersten Schritt Schlüsselwörter identifiziert und Suchbegriffe formuliert. Die erste Herausforderung bei der Anwendung des Snowballing-Verfahrens besteht darin, einen Start-Set von Arbeiten zu identifizieren, die für das Snowballing verwendet werden sollen. Das Start-Set ist in Abbildung 2.1 dargestellt. Bei jeder Suche nach Arbeiten, die in das Start-Set aufgenommen werden sollen, wird ein Start-Set vorläufig erstellt und anhand von Ein- und Ausschlusskriterien bewertet. Darunter werden die qualifizierten Arbeiten als tatsächliches Start-Set bezeichnet, welche schließlich in die systematische Literaturstudie aufgenommen werden (vgl. [46]).

2.6.2 Iterationen

Sobald das Start-Set festgelegt ist, das nur Artikel enthält, die in die endgültige Analyse aufgenommen werden, ist wird die erste Iteration durchgeführt, die aus dem Vorwärts- und Rückwärts-Snowballingverfahren besteht. Wenn Artikel eingeschlossen werden, die auf einem Artikel basieren, der später ausgeschlossen wird, ist ein Rollback erforderlich. Daher ist es wichtig, sicherzustellen, dass der Artikel tatsächlich einbezogen wird, bevor er überhaupt zum Snowballingverfahren verwendet wird (vgl. [46]).

Rückwärts Snowballing

Rückwärts-Snowballing bedeutet, die Referenzliste zu benutzen, um weitere Artikel zu identifizieren, die aufgenommen werden sollen. Dazu wird zunächst die Referenzliste durchgegangen und Artikel, die die grundlegenden Kriterien wie z. B. Sprache, Publikationsjahr und Art der Publikation, z. B. wenn nur peer-reviewte Artikel berücksichtigt werden, nicht erfüllen, werden ausgeschlossen. Als nächstes werden Artikel aus der Liste gestrichen, die bereits geprüft wurden, weil sie in dieser oder einer früheren Iteration

entweder durch Rückwärts- oder Vorwärts-Snowballing gefunden wurden. Sobald diese entfernt sind, sind die verbleibenden Artikel echte Kandidaten für die Aufnahme (vgl. [46]).

Vorwärts Snowballing

Vorwärts-Snowballing bezeichnet die Identifizierung neuer Artikel auf Grundlage der Artikel, die den untersuchten Artikel zitieren und damit neuer als dieser sind (siehe Abbildung 2.1 rechts). Die Zitate zu dem untersuchten Artikel werden mit Google Scholar untersucht. Jeder Kandidat, der die Arbeit zitiert, wird untersucht. Die erste Prüfung erfolgt auf der Grundlage der in Google Scholar bereitgestellten Informationen. Wenn diese Informationen für eine Entscheidung nicht ausreichen, wird die zitierte Arbeit detaillierter untersucht. Zuerst wird die Zusammenfassung betrachtet und wenn diese nicht ausreicht, wird die Stelle untersucht, die die bereits enthaltene Arbeit zitiert. Wenn diese Informationen nicht ausreichen, wird der Volltext untersucht, um eine Entscheidung bezüglich des neuen Artikels zu treffen. Die Herangehensweise beim Durcharbeiten der Artikel ist ähnlich wie bei Arbeiten, die durch Rückwärts-Snowballing identifiziert werden (vgl. [46]).

Inklusion und Exklusion

Wie in Abbildung 2.1 dargestellt, ist es wichtig, vor der Verwendung eines neuen Artikels über Einschluss oder Ausschluss zu entscheiden. Nur Arbeiten, die anhand von Inklusionsartikeln gefunden wurden, dürfen für die Analyse verwendet werden. Nach einem Vorwärts- und Rückwärts-Snowballing gehen die in der Iteration identifizierten neuen Artikel in die nächste Iteration ein. Es ist wichtig, eine Iteration zur gleichen Zeit durchzuführen, um Traceability zu erhalten (vgl. [46]).

2.6.3 Datenextraktion

Alle identifizierten Artikel gehen in die Datenextraktion ein, die in Übereinstimmung mit den in der SLR gestellten Forschungsfragen durchgeführt werden sollte. Da die vollständigen Artikel untersucht werden müssen, bevor sie in das Snowballing-Verfahren eingehen, kann die Datenextraktion auch gleichzeitig mit der Entscheidung, ob der Artikel einbezogen werden soll oder nicht, durchgeführt werden (vgl. [46]).

Kapitel 3

Verwandte Arbeiten

Es gibt einige verwandte Arbeiten über Hybridmethoden für RE und SE in wissenschaftlichen Datenbanken, die in den nächsten Absätzen beschrieben werden: Zakari et al. [41] schlagen eine hybride, dreiphasige Anforderungserhebungstechnik vor, die drei bekannte Techniken der Verwendung von Fragebogen, Interviews und Prototyping in einem einheitlichen Rahmen kombiniert. Außerdem wurde diese Technik bei der Umsetzung eines Online-Bildungssystems untersucht. Die Ergebnisse zeigten eine hohe Positivität bei der Validität der Anforderungen (aus den drei Phasen) und, dass ein hybrider Ansatz zur Anforderungserhebung einen sinnvollen Lösungsansatz für die Anforderungserhebung bietet. Außerdem können die gesammelten Anforderungen von der ersten bis zur letzten Phase wesentlich präziser aufgestellt werden. Diese Arbeit wurde im Zusammenhang mit den traditionellen RE-Erhebungsaktivitäten durchgeführt, aber es besteht die Möglichkeit, daraus einen innovativen agilen Ansatz zu entwickeln.
Ochodek et al. [47] führen eine Fallstudie über einen hybriden Ansatz durch und werten diese Fallstudie aus, um den Reifegrad der Requirements Engineering-Praktiken im agilen Projekt (REMMA) zu bewerten. Zentrale Fragen der Studie betreffen die Nützlichkeit, Benutzerfreundlichkeit und Kosteneffizienz von REMMA. REMMA umfasst zwei Hauptkomponenten: ein Reifegradmodell für agile RE und eine Beurteilungsmethode, zur Bewertung der Umsetzung der agilen RE-Methoden. Die Autoren gehen davon aus, dass diese Methode einfach zu verwenden und preiswert sein soll. Außerdem wird REMMA als ein nützliches Instrument im Verbesserungsprozess der RE-Praktiken in agilen Projekten eingeschätzt. Die Fallstudie wurde in einem der größten Softwarehäuser Mitteleuropas durchgeführt.
Kumar et al. [48] stellen einen hybriden Ansatz für das Requirements Engineering in der agilen Softwareentwicklung vor. Dieser Ansatz basiert auf JAD sowie auf der Priorisierung der Anforderungen im agilen RE und wird zusätzlich vom Viewpoint unterstützt. Ein Viewpoint ist eine Technik zur Organisation der Anforderungen an ein Software-System und basiert auf einer bestimmten Perspektive, z. B. der eines Endverbrauchers oder eines Managers. Die Autoren behaupten, dass dieser Ansatz ein effektiver Weg sein kann, um Probleme bei der Anforderungserhebung in der agilen Entwicklung zu bewältigen. Außerdem wird die Methode als Mittel zur Reduktion des Overheads des Scrum Managers dargestellt, sodass sich die Entwickler auf die perfekte inkrementelle Lieferung und qualitativ hochwertige Software konzentrieren können. Haad et al. [49] präsentieren einen hybriden Ansatz zur Anforderungspriorisierung, indem sie wichtige und dringende Faktoren (Eisen Hower Matrix) mit MoSCoW kombinieren. Sie

behaupteten, dass ihre vorgeschlagene Methode dazu beitragen kann, das Problem der Zuweisung von Prioritäten für große Anforderungssätze zu lösen. Das vorgeschlagene Modell bietet den Vorteil, die Anforderungen in großen Fragmenten zu priorisieren. Die Validierung des vorgeschlagenen Modells wurde mittels Detailsimulationen mit der Stella Architekt Software durchgeführt. Shahmoradpour et al. [50] stellen einen neuen Ansatz im aspektorientierten Requirement Engineering für die agile Softwareentwicklung vor, bestehend aus einem Agilitätsbewertungsmodell zur Bewertung der Agilität von Softwareunternehmen auf der Grundlage universeller Metriken im aspektorientierten RE. Der hybride Ansatz trage dazu bei, herausfordernde Ziele bei der Systementwicklung zu erreichen. Hierbei wurden anstelle von agilen Methoden agile Praktiken als Grundlage für das Bewertungsmodell verwendet. Die Autoren behaupten, dass das vorgeschlagene Modell in allen Unternehmen anwendbar ist, auch in solchen, die sich teilweise an eine bestimmte agile Methode angepasst haben. Die Stärke ihres Ansatzes beschreiben die Autoren besonders bei einer dynamischen Veränderung der Beziehungen im Laufe der Zeit, da hierbei der Agent in der Lage sei, sich in kurzer Zeit an neue Bedingungen anzupassen und neue Beziehungen zu erlernen.
Prenner et al. [51] untersuchen in einem systematischen Literaturreview mit dem Ziel eines besseren Verständnisses für die Organisation von hybriden Ansätzen, wie hybride Entwicklungsprozesse im Software Engineering organisiert werden. Sie identifizieren drei allgemeine Muster für die Organisation hybrider Ansätze und stellen fest, dass alle untersuchten Muster auf dem traditionellen Wasserfallmodell von Royce basieren und die Standard-SE-Aktivitäten verwenden. Elshandidy et al. [42] vergleichen in einer Survey-Studie agiles und traditionelles Requirement Engineering und helfen den Lesern dabei: 1) einen schnellen und dennoch umfassenden Überblick über RE im traditionellen und agilen SE zu erhalten; 2) die Synergien/Gemeinsamkeiten zwischen den beiden Ansätzen im Umgang mit RE zu verstehen; 3) die Herausforderungen erkennen, die mit der Adoption von ASD verbunden sind; 4) die derzeit prominenten Forschungsgebiete des agilen Requirements Engineering zu identifizieren. Paetsch et al. [39] vergleichen TRE-Ansätze und Agile Software-Entwicklung, analysieren Gemeinsamkeiten und Unterschiede beider Ansätze und ermitteln mögliche Wege, wie agile Softwareentwicklung von RE-Methoden profitieren kann. Sie stellten fest, dass die Erhebungs-, Analyse- und Validierungsphasen des RE-Prozesses in allen agilen Prozessen vorhanden sind. Allerdings sind die angewandten Techniken je nach Ansatz unterschiedlich und die Phasen sind auch nicht so klar voneinander abgegrenzt wie im TRE-Prozess, sondern die Phasen verschmelzen teilweise.

Kapitel 4

Forschungsdesign

Wie bereits in Abschnitt 1.2 erwähnt wurde, bereitet das agile Requirements Engineering auch einige Schwierigkeiten in Projekten. Die Firmen nutzen deshalb oft das traditionelle Requirements Engineering, um diese Probleme auszugleichen. Es gibt jedoch keine Übersicht darüber, wie die einzelnen Probleme bei den agilen durch traditionelle Methoden angegangen werden, um Unternehmen zu unterstützen. Aus dieser Motivation ergeben sich für diese Masterarbeit zwei Ziele. Das erste Ziel ist die Durchführung einer Literaturrecherche zu den Herausforderungen des Agilen Requirements Engineering, indem Herausforderungen, die in der Literatur erwähnt werden und auch mögliche Lösungen zu identifizieren und einen Überblick über die bestehenden Probleme und Lösungen zu erhalten. Das zweite Ziel ist, mögliche Herausforderungen zu untersuchen, für die keine Lösungen vorgeschlagen wurden und für diese Lösungen aus dem agilen oder traditionellen Requirements Engineering vorzuschlagen. Zum Erreichen dieser Ziele wird eine Struktur definiert, um den Forschungsprozess in verschiedene Schritte aufzuteilen und die Arbeit so zu planen, dass ein gutes Ergebnis erzielt wird. In diesem Kapitel werden zunächst die Forschungsfragen beschrieben. In Abschnitt 4.2 wird die Forschungsmethode vorgestellt. In Abschnitt 4.3 wird die Durchführung des SLR beschrieben. In Abschnitt 4.4 wird die Datenanalyse kurz diskutiert und in Abschnitt 4.5 wird die Gültigkeit der Forschungsmethode besprochen.

4.1 Forschungsfragen (RQs)

Ziel dieser Masterarbeit ist es, folgende Forschungsfragen in Tabelle 4.1. zu beantworten.

Tabelle 4.1: Forschungsfragen

RQ1	Was sind die typischen Probleme des ARE und wie wirken sie sich auf den gesamten Prozess des RE aus?
	Mit dieser ersten Forschungsfrage werden die Probleme der ARE-Ansätze untersucht und identifiziert.
RQ2	Welche Lösungen sind in der Literatur bereits für die Probleme des ARE vorgeschlagen worden?
	Hier werden zunächst die Lösungsvorschläge zu den Problemen des ARE aus der verschiedenen Literatur herausgefiltert, um festzustellen, welche Probleme noch offen sind, und danach wird geprüft, ob passende Lösungen für diese offenen Probleme zu finden sind.

4.2 Forschungsmethode

In dieser Forschung wird das von Kitchen et al. [44] vorgeschlagene Verfahren des Systematische Literaturreviews (SLR) verwendet. Dieser Prozess gliedert sich in die im Folgenden erläuterten Schritte der Definition des Suchbegriffs und der Ein- und Ausschlusskriterien sowie der Auswahl der Datenbank.

4.2.1 Definition des Suchbegriffs (Suchstring)

Abbildung 4.1. zeigt die Struktur des Suchstrings. In der vorliegende SLR werden Arbeiten gesucht, die sich mit Problemen agiler Requirements Engineering-Methoden befassen. In der Literatur wird die Forschung im Bereich des Requirements Engineering mit verschiedenen Begriffen wie "Requirements Engineering, Requirements, RE" bezeichnet (vgl. [6, 7, 12, 17], usw.). Daher wurden diese Begriffe in den Suchstring aufgenommen. Darüber hinaus werden Probleme in der Literatur mit verschiedenen Wörtern wie „issues, problems, challenges" bezeichnet. Daher wurden diese Begriffe auch in den Suchstring aufgenommen. Wie bereits erwähnt, ist es in erster Linie wichtig, die Probleme des agilen Requirements Engineering herauszufiltern, daher werden Wörter wie „agile" oder „agility" ebenfalls berücksichtigt. Dies trägt dazu bei, den Suchprozess zu präzisieren und die Suchergebnisse auf eine erfassbare Anzahl an Artikeln zu beschränken und außerdem mehr relevante Arbeiten über das ARE zu erhalten.

("requirements engineering" OR requirements OR RE) AND (agile OR agility) AND (issues OR Problems OR challenges)

Abbildung 4.1. Suchbegriff (Suchstring).

Tabelle 4.2: Ein- und Ausschlusskriterien

	Kriterien	Beschreibung
Einschluss	EK_1	Der Artikel beschreibt Vor- und Nachteile bzw. Probleme im ARE.
	EK_2	Die Arbeit präsentiert Lösungen zu RE Challenges.
	EK_3	Der Artikel präsentiert eine Fallstudie oder einen Erfahrungsbericht über ARE.
	EK_4	Die Arbeit ist ein von Gutachtern geprüfter Beitrag zu einer Konferenz oder Zeitschrift.
Ausschluss	AK_1	Der Artikel beschreibt nicht die Probleme des ARE.
	AK_2	Die Arbeit ist weder auf Deutsch noch auf Englisch verfasst.

4.2.2 Definition der Ein- und Ausschlusskriterien

Es wurden auch Einschluss- und Ausschlusskriterien definiert. Diese gewährleisten eine höhere Qualität und Relevanz der Forschung im SLR-Prozess und sind eine Richtlinie für die Forschungsauswahl. Um sicherzustellen, dass relevante Artikel für die erste Forschungsfrage ausgewählt werden, wurde das erste Einschlusskriterium definiert. Ebenso wurde das zweite Einschlusskriterium für die Relevanz der Forschung zur zweiten Forschungsfrage aufgestellt. Fallstudien und Erfahrungsberichte wurden mittels des dritten Einschlusskriteriums einbezogen und durch das vierte Einschlusskriterium wurden ausschließlich überprüfte Inhalte in die Forschung einbezogen und nicht graue

Literaturen mit fragwürdigen Inhalten. Es wurden zwei Ausschlusskriterien definiert, um alle irrelevanten Arbeiten auszuschließen. Das erste Ausschlusskriterium ist definiert, um Arbeiten auszusortieren, die sich nicht mit den Problemen des agilen Requirements Engineering beschäftigen und das zweite Ausschlusskriterium sortiert Arbeiten aus, die in anderen Sprachen als Englisch und Deutsch verfasst wurden. Die konkreten Kriterien zu Einschluss und Ausschluss (vier Einschluss- und zwei Ausschlusskriterien) der Artikel sind in Tabelle 4.2 zusammengefasst.

4.2.3 Auswahl der Datenbank

Für die Literaturrecherche werden fünf Datenbanken ausgewählt: IEEE, ACM, Science Direct, SpringerLink, Google Scholar. Google Scholar wurde ausgewählt, weil es eine Voraussetzung für den Snoballing-Prozess in [46] ist und außerdem Ergebnisse aus verschiedenen Datenbanken in einem Suchergebnis zusammenführt. IEEE, ACM, Science Direct und SpringerLink sind weltweit bekannte Datenbanken und veröffentlichen jedes Jahr eine große Anzahl von Publikationen. Damit eignen sie sich als Basisdatenbank für diese Forschung [3, 30, 51]. Am Ende wurde eine gründliche Suche in all diesen Datenbanken durchgeführt.

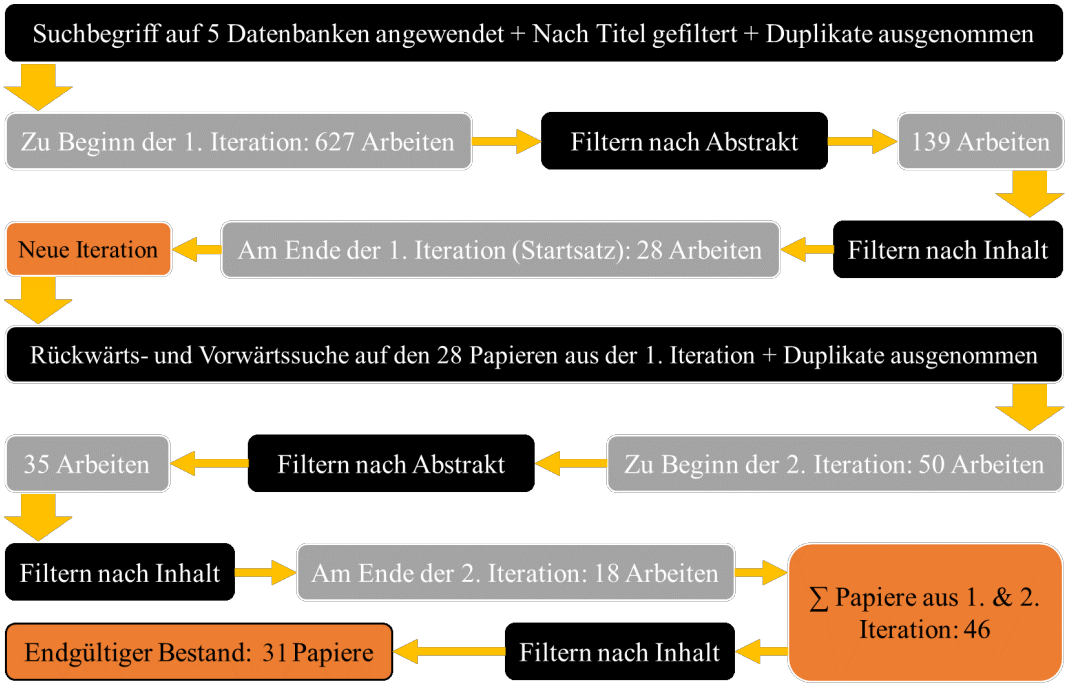


Abbildung 4.2: Darstellung des Such- und Filterprozesses.

4.3 Durchführung

Abbildung 4.2. stellt den Prozess der Literatursuche und die Anzahl der in jedem Schritt und jeder Iteration gefundenen Artikel dar. Es wurde eine automatisierte Suche in den ausgewählten Datenbanken anhand des Suchbegriffs durchgeführt. Zunächst wurden alle SLR-Ergebnisse anhand ihres Titels und unter Anwendung von Ein- und Ausschlusskriterien überprüft und irrelevante Artikel gefiltert. Dann wurden Duplikate entfernt, sodass das Ergebnis aus 627 Artikeln bestand, die im nächsten Schritt anhand ihrer Kurzzusammenfassungen und Schlüsselwörter gefiltert wurden, wodurch 139 Artikel

übrigblieben. Im letzten Schritt wurden die Artikel gründlich gelesen und ihr Inhalt analysiert. Dies führte zu einem Ausschluss von 111 Arbeiten, sodass als Start-Set 28 Arbeiten übrig blieben. In der ersten Iteration wurde für jede Arbeit im Start-Set eine Vorwärts- und Rückwärtssuche durchgeführt, wobei auch Duplikate gefiltert wurden. Nach der Snowballing Methode [46] wurden die Referenzen jedes Artikels und alle Arbeiten, in denen der jeweilige Artikel zitiert wurden, berücksichtigt. Dieser Schritt führte zu 50 weiteren Arbeiten. Wie zuvor wurden zunächst Ein- und Ausschlusskriterien auf die Schlüsselwörter und Kurzzusammenfassungen der Artikel angewandt und 15 Arbeiten ausgeschlossen. Danach wurden die übrigen Artikel gründlich gelesen und 17 Artikel ausgeschlossen. Am Ende wurden noch 12 Arbeiten nach einer inhaltlichen Prüfung zu Forschungsfragen ausgeschlossen, sodass zuletzt 31 Arbeiten übrig sind.

4.4 Datenanalyse

Um das Ziel dieser Arbeit zu erreichen, die Probleme des ARE-Prozesses und ihre Lösungen aus dem TRE zu finden, wurden die gefundenen Artikel eingehend analysiert. Diskutierte Probleme aus jedem Artikel wurden extrahiert und zusammengefasst. In diesem Prozess wurden auch Lösungen für einige Probleme gefunden und zusammengefasst. Tabelle 4.3 und Abbildung 4.3 geben einen Überblick über die analysierten Arbeiten (Thema, Autor, Jahr). In den nächsten Schritten wurden Probleme aus derselben Kategorie zusammengefasst und unter 13 Over-Challenges gruppiert.

4.5 Bedrohungen der Validität

Die Ergebnisse dieser Arbeit sind mit einigen Risiken für die Validität verbunden. Nach Wohlin et al. [52] zur Klassifizierung der Validität, werden hier Konstrukt-, interne, Schlussfolgerung- und externe Validität diskutiert. Diese Begriffe werden in den nächsten Unterabschnitten erläutert.

4.5.1 Konstrukt-Validität

Nach [42] sind die Hauptgefahren beim Entwurf einer SLR die Erstellung des Suchstrings und die Wahl der Datenbanken. Für den Suchstring besteht die Hauptbedrohung in der Verwendung der falschen Wörter für die Suche. Um diese Bedrohung zu mindern, wurden Synonyme aller drei Schlüsselwörter zum Suchstring hinzugefügt, z. B. (“Requirement Engineering” oder RE oder Requirement) und der Suchstring wurde am Ende des Prozesses mit dem Betreuer besprochen. Um die Bedrohung durch die Datenbankauswahl zu verringern, wurden andere erfolgreich durchgeführte SLRs als Orientierungspunkt betrachtet, um fünf Datenbanken auszuwählen, sodass die Bedrohung durch fehlende Artikel verringert wird [2, 13, 26].

4.5.2 Interne Validität

Einschluss- und Ausschlusskriterien wurden formuliert, um interne Inkonsistenzen zu verringern. Außerdem wurde die durchgeführte Datenanalyse von einem Betreuer überprüft, um die Verzerrung der Forschung zu verringern.

4.5.3 Schlussfolgerungsvalidität

Die Validität der Schlussfolgerung hängt von den Ergebnissen ab, die sich aus den gesammelten Arbeiten ergeben.

4.5.4 Externe Validität

Um externe Validität zu erreichen, müssen alle Arbeiten zum gleichen Forschungsthema gesammelt werden. Es gibt jedoch keine Garantie, dass alle Artikel im SLR-Prozess gefunden wurden. Um diese Gefahr zu mindern, wurde das Snowballing-Verfahren durchgeführt.

Tabelle 4.3: Ein Überblick über die analysierten Arbeiten.

N.	Name des Artikels	Autor	Jahr
01	Agile Quality Requirements Engineering Challenges: First Results from a Case Study.	Alsaqaf, W.; Daneva, M.; Wieringa, R.	2017
02	Quality Requirements in Large-Scale Distributed Agile Projects – A Systematic Literature Review	Alsaqaf, Wasim; Daneva, Maya; Wieringa, Roel	2017
03	Understanding Challenging Situations in Agile Quality Requirements Engineering and Their Solution strategies: Insights from a Case Study	Alsaqaf, Wasim; Daneva, Maya; Wieringa, Roel	2018
04	Quality requirements challenges in the context of large-scale distributed agile: An empirical study	Alsaqaf, Wasim; Daneva, Maya; Wieringa, Roel	2019
06	A Case Study on Benefits and Side-Effects of Agile Practices in Large-Scale Requirements Engineering	Bjarnason, Elizabeth; Wnuk, Krzysztof; Regnell, Bjorn	2010
07	Requirements Engineering and Software Testing in Agile Methodologies	Coutinho, Jarbele C. S.; Andrade, Wilkerson L.; Machado, Patrícia D. L.	2019
08	Review on Agile requirements engineering challenges	Elghariani, Kaiss; Kama, Nazri	2016
09	Requirement Engineering in Agile.	F. Tazeen; M. Waqas	2019
10	Challenges of Aligning Requirements Engineering and System Testing in Large-Scale Agile: A Multiple Case Study	Gomes De Oliveira Neto, Francis-co; Horkoff, Jennifer; Knauss, Eric; Kasauli, Rashidah; Liebel, Grischa	2017
11	A Mapping Study on Requirements Engineering in Agile Software Development	Heikkila, Ville T.; Damian, Dani-ela; Lassenius, Casper; Paasivaara, Maria	2015
12	A Reflection on Agile Requirements Engineering: Solutions Brought and Challenges Posed	I., Irum; M., Lauriane; D., Maya; S., Siti Salwah	2015
13	A systematic literature review on agile requirements engineering practices and challenges.	Inayat, Irum; Salim, Siti Salwah; Marczak, Sabrina; Daneva, Maya; Shamshirband, Shahaboddin	2015
14	Requirements Engineering in Agile Projects: A Systematic Mapping based in Evidences of Industry.	Medeiros, Juliana D. R. V.; Vasconcelos, Daniela Alves Alexandre; Silva, Carla; Wanderley, Eduardo	2015
15	A systematic review of Requirements Engineering practices in agile model	Okesola, J.; Adebiyi, M.; Okokpujie, K.; Odepitan, D.;	2019
17	Agile Requirements Engineering with prototyping: A case study	Kapyaho, Maria; Kauppinen, Marjo	2015
18	Requirements Engineering Challenges in Large-Scale Agile System Development	Kasauli, R.; Liebel, G.; Knauss, E.; Gopakumar, S.; Kanagwa, B.	2017
19	Agile Requirements Engineering Practices: An Empirical Study	Cao, Lan; Ramesh, Balasubramaniam	2008
20	Evaluation of efficient Requirement Engineering techniques in agile software development	Malik, M. Usman; Chaudhry, Nadeem Majeed; Malik, Khurram Shahzad	2013

N.	Name des Artikels	Autor	Jahr
22	A Research Study on Critical Challenges in Agile Requirements Engineering	Batra, Mona; Bhatnagar, Archana	2019
23	Scrum Requirements Engineering Practices and Challenges in Offshore Software Development	Vithana, Nipunika	2015
24	Agile requirements engineering practices and challenges: an empirical study.	Cao, Lan; Ramesh, Balasubramaniam ; Baskerville, Richard ,	2010
25	Requirements Engineering Practice and Problems in Agile Projects: Results from an International Survey	Wagner, Stefan; Méndez Fernández, Daniel; Kalinowski, Marcos; Felderer, Michael	2017
26	A Systematic Literature Review of Challenges and Critical Success Factors in Agile Requirement Engineering	Saleh, Mohammed; Baharom, Fauziah; Mohamed, Shafinah F. P.; Ahmad, Mazida	2018
27	Agile Requirements Engineering: A systematic literature review	Schön, Eva-Maria; Thomaschew-ski, Jörg; Escalona, María José	2017
28	Key Challenges in Agile Requirements Engineering.	Schön, Eva-Maria; Winter, Dominique; Escalona, María José; Thomaschewski, Jörg	2017
29	Exploring Issues in Agile Requirements Engineering in the South African Software Industry.	Sebega, Y.; Mnkandla, E.	2017
30	Impact and Challenges of Requirement Engineering in Agile Methodologies: A Systematic Review.	Alam, Sehrish; Bhatti, Shahid Nazir; Shah, S. Asim Ali; Jadi,Amr Mohsen	2017
31	Agile Requirements Engineering: An Empirical Analysis and Evidence from a Tertiary Education Context.	Thomas, Meetu; Senapathi, Mali	2019
32	Identifying and Structuring Challenges in Large-Scale Agile Development Based on a Structured Literature Review.	Uludag, Omer; Kleehaus, Martin; Caprano, Christoph; Matthes, Florian	2018
33	A Case Study in Requirements Engineering in Context of Agile.	Gaikwad, V., & Joeg, P.	2017
34	Agile Requirements Engineering in Practice: Status Quo and Critical Problems.	Wagner, Stefan; Méndez Fernández, Daniel; Kalinowski, Marcos; Felderer, Michael.+	2018

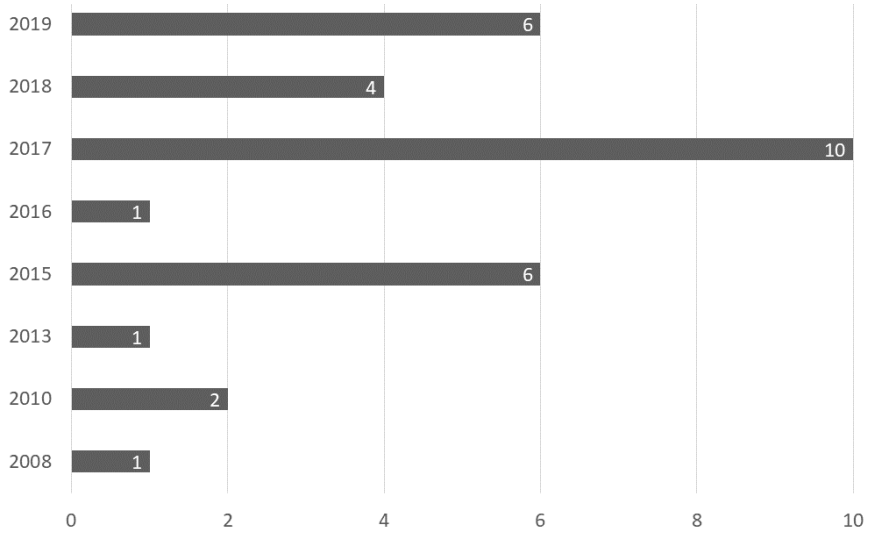


Abbildung 4.3: Anteil der veröffentlichten Arbeiten pro Jahr.

Kapitel 5

Ergebnisse der systematischen Literaturrecherche

In diesem Kapitel werden die Ergebnisse der systematischen Literaturrecherche (SLR) zur ersten und zweiten Forschungsfrage (RQ1 & RQ2) präsentiert. RQ1 & RQ2 werden zusammen beantwortet, indem immer erst die Challenge beschrieben wird und dann die - möglicherweise schon vorhandene - Lösung. Anschließend werden die Ergebnisse diskutiert.

5.1 Herausforderungen & Lösungen für ARE

In diesem Abschnitt werden die Herausforderungen im ARE unter 13 Over-Challenges zusammengefasst und beschrieben. Tabelle 5.1 enthält diese Over-Challenges sowie deren Quellen und wie häufig diese Probleme in der Literatur erwähnt wurden. Anschließend werden diese Over-Challenges mit dazugehörigen Unter-Challenges detailliert beschrieben und Lösungsvorschläge aus der Literatur zugeordnet.

Tabelle 5.1: Erkannte Over-Challenges im agilen Requirements Engineering.

ID	Over-Challenge	Quelle	Σ n
1	Dokumentations- und User Stories-bezogene Probleme	1,2,4,7,8,10,11,12,13,14,15,17,18,19, 22,23,24,25,26,28,29,32,33,34	24
2	Kunden-, Nutzer-, Stakeholder-bezogene Probleme	2,3,4,7,8,9,11,12,13,14,15,17,18,19, 22,23,24,25,26,28,29,31,33,34	23
3	Planungs- und Aufwandsschätzungsbezogene Probleme	7,8,10,11,12,13,15,19,22,23,24,26,30, 31	14
4	Qualitätsanforderungsbezogene Probleme	1,2,3,4,8,12,13,15,17,19,20,23,26,30	14
5	Anforderungsmanagement-bezogene Probleme	7,8,10,13,15,18,25,26,27,28,29,30,34	13
6	Anforderungspriorisierung-bezogene Probleme	2,4,11,12,19,23,24,26,29,31,33	11
7	Qualitätssicherung der Anforderungen (FR & NFR)	1,2,3,4,7,14,19,22,24,28,32	11
8	Teambezogene Probleme	1,2,4,6,11,14,17,22,28,31	10
9	Tracing bezogener Probleme	3,10,12,18,25,28,29,30,31,34	10
10	Product Owner-bezogene Probleme	1,2,3,4,31,32	6
11	Koordinations- und Kommunikationsprobleme	7,18,22,25,34	5
12	Das Gesamtbild der Anforderungen zu haben	2,4,17,28	4
13	Umfang Management-bezogener Probleme	6,9	2

5.1.1 Dokumentationsbezogene Probleme

Es gibt fünf Herausforderungen im Zusammenhang mit der Dokumentation im ARE: (1) Minimale Anforderungsdokumentation; (2) Insuffizienz der User Stories; (3) Zerlegung von Anforderungen und Features; (4) Anforderungsdokumente aktuell halten; (5) Unzureichend spezifizierte Anforderungen. Diese Herausforderungen sind in Tabelle 5.2 mit weiteren Details aufgelistet.

Tabelle 5.2: Erkannte Herausforderungen im Zusammenhang mit der Dokumentation.

Challenge	Paper	Lösung	Paper
Minimale Anforderungs-dokumentation	1,4,8, 12,13, 15,19, 22,24, 26,29	(1) Konsistente und häufige direkte Kommunikation zwischen den Teams; (2) Zusammengestellte Teams und Kunde vor Ort; (3) die Verwendung von User Stories, ergänzt durch Delivery Stories bzw. detaillierte Artefakte; (4) Delivery Stories; (5) Prototypen.	12,13, 15,17, 22
Insuffizienz der User Stories	1,2,4, 7, 11, 14,18, 22,23, 33	(1) Delivery-stories, die User Stories mit funktionalen Spezifikationen, High-Level-Design und Testszenarien erweitern; (2) ein hierarchisches Anforderungsmodell und explizite Produkt- und Portfoliomanagementprozesse; (3) Ein aspektorientierter Anforderungsansatz; (4) FDD-basierter Anforderungsanalyseprozess; (5) Nach der anfänglichen Anforderungserhebung bei den Benutzern sollten die User Stories mit Hilfe des traditionellen Anforderungsanalyse- und Spezifik-ationsprozesses in vollständige, eindeutige und überprüfbare Anforderungen umgewandelt werden; (6) traditionelle RE-Praktiken, wie z. B. das Management der Traceability von Anforderungen, sollten bei der agilen Transformation beibehalten werden, anstatt sich ausschließlich auf User Stories zu verlassen; (7) Verwendung unterschiedlicher Ansätze zur Dokumentation von QRs z. B. Wiki der Annahmen, mehrere PBs, Regeln in Überwachungswerkzeugen usw.	1, 2, 4, 11
Zerlegung von Anforderungen und Features	18,32	-	-
Anforderungs-dokumente aktuell halten	10,28	Eine Kombination von Praktiken (Änderungsmanagement, Anforderungsmanagement und Akzeptanztests).	10
Unzureichend spezifizierte Anforderungen	25,34	-	-

Minimale Anforderungsdokumentation

Agile Methoden konzentrieren sich weniger auf die Dokumentation, die durch informelle und häufige Kommunikation und Zusammenarbeit ersetzt wird. Stattdessen werden die Anforderungen in User Stories festgehalten [8, 19, 24, 29]. Es ist charakteristisch für agile Methoden, dass die Dokumentation minimal ist. Der Kunde kann jedoch Schwierigkeiten haben, dem agilen Prozess zu vertrauen, ohne konkrete Anforderungen zu haben, die er schriftlich vorliegen hat und nachlesen kann, um anhand dessen die Tätigkeiten der Entwickler nachzuvollziehen. Fehlende Dokumentation kann auch zu einem Problem bei einem Personalwechsel während des Projekts oder bei der Übergabe der Software an die Wartung werden [17]. User Stories sind die einzige Dokumentation, die im agilen Requirements Engineering verwendet wird. Dies kann zu Problemen beim Tracing dynamischer Anforderungen führen [22, 26], z.B. wenn es zu einer Fehlkommunikation mit den Teammitgliedern kommt und diese nicht vor Ort sind. In solchen Fällen ist eine

Dokumentation erforderlich, um unerwartete Veränderungen der Anforderungen verfolgen zu können [8]. Jedoch sind User Stories extrem kurze Beschreibungen der Funktion des Systems, die das Entwicklungsteam implementiert. Wenn man den iterativen Aspekt agiler Methoden berücksichtigt (iterative Priorisierung von Anforderungen), könnten alle anfänglichen User Stories aufgrund der Anpassungsfähigkeit der agilen Prinzipien überflüssig oder irrelevant werden. Der Zugang zum tatsächlichen Kunden macht die Dokumentation von Spezifikationen weniger wichtig und eine enge Zusammenarbeit, ebenso macht die Nähe zu anderen Mitgliedern des Entwicklungsteams eine aufwendige Konstruktionsdokumentation oft überflüssig [19, 24]. Wenn allerdings die Kommunikation aufgrund einer Vielzahl von Problemen erschwert wird, u. a. bei rascher Personalumstellung, schnellen Änderungen der Anforderungen, Nichtverfügbarkeit geeigneter Kundenvertreter und wachsender Komplexität der Anwendung, kann das Fehlen einer solchen Dokumentation zu einer Vielzahl von Problemen führen.
Dazu gehören die Unfähigkeit, (1) die Software zu skalieren; (2) die Anwendung im Laufe der Zeit weiterzuentwickeln und (3) neue Mitglieder in das Entwicklungsteam aufzunehmen [19, 24]. Aufgrund der minimalen Dokumentation und des isolierten Wissens kann also das Wissen über frühere Architekturentscheidungen und die Rechtfertigung von QR-Kompromissen verloren gehen, was wiederum jede weitere Änderung an der Software zu einer schwierigen Aufgabe macht. Bei vielen Änderungen in der Software-Architektur kann dies auch zu einer Fragmentierung des Architekturwissens führen [1,4]. Wenn die Anforderungen mit Kunden an verteilten geographischen Standorten kommuniziert werden sollen und die Kunden nicht vor Ort sind, wird es umständlich, eine solche Situation mit wenig oder keiner Dokumentation anzugehen [13, 15]. Eine minimale Dokumentation erschwert auch das Software-Debugging für das Entwicklungsteam [15].

Lösungen: (1) Eine konsistente und häufige persönliche Kommunikation zwischen den Teammitgliedern kann dazu beitragen, die Anforderungsdokumentation zu verbessern, da die erstere für die letztere unerlässlich ist [12, 15, 22]; **(2)** Zusammengestellte Teams sind funktionelle Teams, die an einem Projekt arbeiten und sich denselben Arbeitsbereich teilen. Zusammengestellte Teams haben Vorteile wie reibungslose Kommunikation, Vertrauen unter Kollegen, eine interaktive Umgebung, Koordination und Teamarbeit. Daher können sie die Anforderungsdokumentation verbessern [12, 60]; **(3)** Kunde vor Ort ist eine der Praktiken der Methode Extreme Programmierung. In dieser Praktik wird vorgeschlagen, dass der Kunde während des gesamten Projekts dem Entwicklungsteam zur Verfügung stehen sollte, z. B. um Fragen zu beantworten und Probleme zu lösen. Der Kunde ist die Person, die mit dem Projektteam zusammensitzt, User stories generiert und priorisiert, Akzeptanztests für jedes Release durchführt und die endgültigen Geschäftsentscheidungen trifft. Daher liefert der Kunde vor Ort die Anforderungen und repräsentiert das gesamte Wissen, das dem Entwicklungsteam zur Verfügung stehen muss. „Kunde vor Ort“ kann die minimale Dokumentationsherausforderung lösen, wenn eine effiziente Verfügbarkeit sichergestellt werden kann [12, 60]; **(4)** Delivery Stories werden erstellt, indem User Stories um eine funktionale Spezifikation, ein High-Level-Design und Testszenarien erweitert werden. Auf diese Weise erhält der Lieferant eine Einschätzung der zur Anforderungserfüllung notwendigen Aufgaben. Somit helfen Delivery Stories den Entwicklern, die richtigen Implementierungsentscheidungen zu treffen

und die Kodierungsphase richtig zu dokumentieren [13, 15, 61]; **(5)** Prototypen können als umfassende, aber wartungsarme Dokumentation für die Pläne einer Iteration dienen. Zudem erleichtern Prototypen auch das Schreiben von Akzeptanztests, da sie detaillierte Spezifikationen für die einzelnen Features liefern [17].

Insuffizienz der User Stories

Die Insuffizienz der User Stories ist eine wichtige Herausforderung, da User Stories nur sichtbare funktionale Anforderungen dokumentieren können, jedoch nicht in der Lage sind, QRs und Abhängigkeiten zu dokumentieren [1, 7, 14, 22]. Eigenschaften wie Einheitlichkeit und Verifizierbarkeit von User Stories sind schwer zu validieren, besonders bei großen, komplexen oder von der Hardware abhängigen Softwaresystemen, da dann Informationen für eine ausreichende Dokumentation in der User Story fehlen [11, 18]. In solchen Fällen sind separate System- und Subsystemanforderungen erforderlich. Weitere Probleme entstehen aus dem direkten Bezug der User Stories auf die Feature-Implementierung, weshalb sie nicht immer systematisch aus bestehenden Anforderungen abgeleitet werden. Folglich ist ein direktes Tracing aus bestehenden Anforderungen nicht immer möglich [18]. Des Weiteren können Tracing-Probleme im Falle von zeitlich veränderlichen Anforderungen, Code und Tests entstehen, weil User Stories und agile RE-Praktiken die Traceability von Anforderungen nicht explizit unterstützen [11]. Darüber hinaus muss zwischen dem Konzept der User Stories einerseits, welches die Wünsche des Kunden ausdrückt, und dem Konzept der Anforderungen andererseits unterschieden werden [4]. User Stories werden im Produkt-Backlog auf einem relativ hohen technischen Niveau gehalten, was beim Beitritt neuer Projektteilnehmer zu Problemen bei der Weiterentwicklung der Anforderung führen kann [14, 23].

Lösungen: **(1)** Delivery Stories können einige Probleme mit dem Format der User Stories minimieren, da sie das Verständnis der Anforderungen verbessern und bei der Gestaltung des Systems helfen [11, 61]; **(2)** Die Klarheit der Anforderungen kann durch die Verwendung eines hierarchischen Anforderungsmodells und expliziter Produkt- und Portfoliomanagement-Prozesse verbessert werden. Dadurch werden die Anforderungen für die Entwickler in ausreichend detaillierter Weise beschrieben [11]; **(3)** Eine weitere Lösung ist ein aspektorientierter Ansatz für agile Anforderungen, der aspektorientierte Konzepte in die agile Software-Entwicklung auf der Anforderungsebene integriert. Basisfunktionalitäten werden zu Szenarien verfeinert und aus diesen Szenarien werden übergreifende Szenarien identifiziert, die anschließend zur vollständigen Beschreibung des Verhaltens der Systemfunktionalitäten zusammengefügt werden. Dieser Ansatz kann verwendet werden, um übergreifende Anforderungen zu identifizieren und zu verwalten [11, 62]; **(4)** Basierend auf einer Studie von Ge et al. [63] über die agile Entwicklung von sicheren Webanwendungen, kann ein Anforderungsanalyseprozess verwendet werden, der Feature Driven und Development-basiert ist, um explizit auf Sicherheitsanforderungen einzugehen. In [63] werden allgemeine Methoden zur Entwicklung von Informationssystemen wie z. B. das eben erwähnte Feature-Driven Development (FDD) und ausgereifte Sicherheitsmethoden wie die Risikoanalyse empfohlen, die dann zur Entwicklung sicherer Web-Anwendungen kombiniert werden [11]. **(5)** Die Dokumentation der Anforderungen sollte vollständig, eindeutig und überprüfbar

sein. Dies kann mit einer zu Beginn der Erhebungsphase erstellten Sammlung der Anforderungen aller Benutzern und einer anschließenden Umwandlung der User Stories in vollständige, eindeutige und überprüfbare Anforderungen erreicht werden, z. B. mit Hilfe des traditionellen Anforderungsanalyse- und Spezifikationsprozesses [11]. Firesmith [64] hat Beispiele und Anleitungen dafür geliefert, wie Stories in vollständige, eindeutige und überprüfbare Textanforderungen umgewandelt werden können; **(6)** Bei der Entwicklung großer und komplexer oder Hardware-abhängiger Systeme sollten zusätzlich zu User Stories unbedingt die wichtigsten traditionellen RE-Praktiken wie z. B. das Management der Traceability von Anforderungen eingehalten werden [11]; **(7)** Agile Teams in [2,4] verwenden unterschiedliche Ansätze, um QRs zu dokumentieren, z. B. ein Wiki der Annahme, mehrere Product Backlogs und Regeln in Monitoring-Tools. Diese Arbeiten zeigen, dass agile Teams kreative Lösungen finden, um mit der Unzulänglichkeit von User Stories zur Dokumentation von QRs umzugehen [1,2,4].

Zerlegung von Anforderungen und Features

Die Zerlegung von großen und komplexen Anforderungen in kleinere und umsetzbare Anforderungen ist in der agilen Entwicklung eine Herausforderung [32], da die Zerlegung einer Feature-Anfrage in kleine Pakete, die sowohl einen Kundennutzen haben als auch in kleinen Iterationen geliefert werden können, schwierig ist. Agile Werte und Praktiken wie die kontinuierliche Lieferung hängen von einer guten Wertvorstellung ab. Allerdings ist dies insbesondere bei der Entwicklung von großen Systemen wegen der teils unklaren Rolle des Kunden und der unklaren Skalierbarkeit schwer umzusetzen [18].

Anforderungsdokumente aktuell halten

Ein weiteres Problem liegt in der Nachvollziehbarkeit von Änderungen, insbesondere bei Verbindungen zwischen verschiedenen User Stories. Ursachen hierfür sind u. a. fehlende Informationen in den Anforderungsspezifikationen. Dies behindert Entwickler und Tester beim Entwerfen und Testen der Software. Das Ziel und zugleich die Herausforderung ist die Erstellung von testbaren und leicht zu aktualisierenden Anforderungsdokumenten unter Beachtung der Änderungen an den Testartefakten [10, 28]. Dabei ist insbesondere die Anpassung an veränderte Umgebungsfaktoren schwierig zu realisieren [28].
Lösungen: Agile RE-Praktiken, die helfen können sind die Kombination von Praktiken wie Änderungsmanagement, Anforderungsmanagement und Akzeptanztests. Anforderungs- und Änderungsmanagement, sodass den Stakeholdern bei der Auswahl von Features geholfen wird. Dies minimiert zugleich die Auswirkungen von Änderungen und verlagert die Pflege der Verbindungen zwischen Tests- und Anforderungsinformationen von Verantwortlichkeiten von Personen an Werkzeuge [10].

Unzureichend spezifizierte Anforderungen

Unterspezifizierte Anforderungen sind Anforderungen, die zu abstrakt sind und verschiedene Interpretationen zulassen. Ursachen können die Fähigkeiten der beteiligten Personen entweder im Entwicklungsteam oder auf der Kundenseite darstellen, z.B. im Entwicklungsteam ein Mangel an Kompetenz, Anforderungen zu schreiben und Kundenbedürfnisse zu analysieren oder auf der Seite der Kunden ein Mangel an Wissen von den Funktionen des Systems. Darüber hinaus ist es bei agilen Projekten oft

problematisch, schnell zu definieren, was getan werden muss, sodass nicht genug Zeit für die Definition im erforderlichen Detaillierungsgrad bleibt [25,34].

Tabelle 5.3: Herausforderungen bei Kunden, Benutzern, Stakeholdern.

Challenge	Paper	Lösung	Paper
Verfügbarkeit und Beteiligung von Kunden, Nutzern und Stakeholdern	3,8, 11,12, 13,15, 19,22, 23,24, 26,29	(1) Anforderungsingenieur vor Ort; (2) Definition der Rollen von Domaininhabern und Geschäftsanalysten; (3) Ethnographie; (4) Scrum Erweiterung durch zielorientierte RE Prozesse; (5) Zusätzliche Anforderungs-dokumentation; (6) Mind-mapping; (7) Erweiterung der TDD auf Anforderungserhebung und -analyse; Validierung anhand Story-Tests; (8) ATDD; (9) Ersatz-/ Proxy-Kunden.	8,11, 12,13, 15
Fehlendes Wissen bei Kunden, Nutzern und Stakeholders	12,19, 23,25, 28,29, 33,34	(1) Ernennung eines Kundenvertreters; (2) kontinuierliche Koordination und Präsentation möglicher Lösungen für Stakeholder; (3) Transparenz über Gründe für Entscheidungen schaffen; Entscheidungsfindungskompetenez der Product Owner stärken; (4) Stakeholder über Folgen einer Einmischung in detaillierte Entscheidungen informieren; (5) Bereitstellung alternativer Lösungen für Anforderungen; (6) Kunden über Notwendigkeit regelmäßiger Interaktion im Projektverlauf aufklären	12, 28, 33
Kommunikat-ionsprobleme	4,7, 14,17, 19,23,29	Prototyping unterstützt Kommunikation durch ausreichende: (1) Kundenpräsenz, 2) Kundenkompetenz; (3) Konsens zwischen Kundengruppen;	17
Fehlende Einbeziehung von Kunden und Stakeholdern	4,17, 19,23, 28,31	(1) Verantwortlichkeiten der Kunden festlegen; (2) Schulung des Kunden in agilen Methoden und über die Rolle des Projektinhabers; (3) Prototyping; (4) Einsatz von Qualitätssicherungspersonal; (5) Interviews; (6) Benutzer-Beobachtung mit Think-aloud-Methode; (7) A/B-Tests; (8) UX-Labs; (9) Analyse des Nutzungsverhaltens; (10) Benutzerfreundliche Tests; (11) Alpha/Beta/Silent-Starts; (12) veröffentlichte Versionen kontinuierlich verbessern; (13) UX-Boards verwenden; (14) Methodenanzahl minimieren; (15) Auswertung im Team ohne Berichtserstellung; (16) Finanzielle Einschränkungen für Nutzereinbindung minimieren; (17) Zugang zu realen Nutzern durch Panels/vorherige Rekrutierung; (18) Beteiligung von Stakeholdern an regelmäßigen Iterationen; (19) Ziele vorschlagen statt Lösungen vorschreiben; (20) zu Beginn alle Stakeholder einbeziehen; Personenzahl im Zeitverlauf reduzieren; (21) vorab Klärung von Zweck/Bedeutung der Treffen	17, 19,28
Fehlendes Kunden-/ Stakeholder-Feedback	18	-	-

5.1.2 Kunden-, Nutzer-, Stakeholder-bezogene Probleme

Es gibt fünf große Herausforderungen in Bezug auf Kunden, Nutzer und Interessenvertreter im ARE: (1) Verfügbarkeit und Beteiligung von Kunden, Nutzern und Stakeholdern; (2) fehlendes Wissen bei Kunden, Nutzern und Stakeholders; (3) Kommunikationsprobleme; (4) fehlende Einbeziehung von Kunden und Stakeholdern; (5) fehlendes Kunden-/ Stakeholder-Feedback. Diese Herausforderungen sind in Tabelle 5.3 mit weiteren Details aufgelistet.

Verfügbarkeit und Beteiligung von Kunden, Nutzern und Stakeholdern

Eine hohe Kundenverfügbarkeit und der Zugang der Kunden zur Spezifizierung der Anforderungen und zum Feedback wird durch agile Methoden vorausgesetzt und befürwortet,

was jedoch oft unrealistisch ist, wie empirische Studien bestätigen [3, 8, 11, 12, 13, 15, 22, 26, 29]. Ursachen sind u. a. geschäftliche Faktoren wie Zeit, Kosten und Arbeitslast des Kundenvertreters [8, 12, 13, 15, 22]. Im äußersten Fall kann die Unerreichbarkeit des Kundenvertreters zum Scheitern des Projekts führen, wohingegen agile Prinzipien die Anwesenheit der Kunden für die direkte und unkomplizierte Zusammenarbeit und Kommunikation der Entwickler mit den Kunden verlangen [29]. Indirekte Verbindungen zwischen den Kunden und den Entwicklern über Vermittler sind weniger effektiv als direkte Verbindungen, da bei der Verwendung eines Ersatzkunden die tatsächlichen Anforderungen nicht vollständig an das Entwicklungsteam übermittelt werden könnten [23]. Eine Kundenvertretung vor Ort ist jedoch schwer zu erreichen [19, 24], insbesondere bei großen Projekten mit vielen Kunden. In diesen Fällen ersetzt ein Kundenvertreter, z. B. der Product Owner in Scrum oder ein Produktmanager den Kunden, muss jedoch seine Aufmerksamkeit zwischen den Kunden und Entwicklern aufteilen. Wenn es solche Kundenvertreter nicht gibt, wird die Bewertung und Priorisierung der Anforderungen von den Entwicklern vorgenommen, die möglicherweise ein zu geringes Marktverständnis besitzen [11].

Lösungen: Die folgenden Lösungen wurden von den Autoren vorgeschlagen: **(1)** Ein Anforderungsingenieur kann den Kunden vor Ort begleiten und dadurch RE-bezogene Aufgaben für den Kunden vor Ort durchführen, um die Arbeitslast und die Qualifikationsanforderungen des Kunden vor Ort zu reduzieren [11]; **(2)** Zusätzliche Rollen wie der Domänenbesitzer und der Business Analyst können vorteilhaft sein, wenn in großen Organisationen agiles RE zum besseren Verständnis der Anforderungen durchgeführt wird. Die Aufgabe eines Business Analyst ist die Dokumentation des Geschäftsprozesses und der Datenanforderungen in Form von User Stories, während der Domänenbesitzer für die Sammlung von Wissen über die Geschäftsdomäne des Kunden, deren Verteilung unter den Teammitgliedern sowie der wiederverwendbaren Dokumentation verantwortlich ist [11, 61]; **(3)** Ethnographie kann zur Anforderungserhebung und -analyse in einem agilen Softwareentwicklungsprojekt verwendet werden, um die Kundenbedürfnisse besser zu verstehen [11]; **(4)** Die Verbesserung von Scrum durch zielorientiertes Requirements Engineering und IT-Governance-Prozesse kann Probleme mit dem Modell des Single Product Owners in groß angelegten Scrum-Entwicklungskontexten vermindern [11].
(5) Zur Verminderung des Overheads aufgrund von persönlicher Kommunikation zwischen Kunden oder Kundenvertretern und den Entwicklungsteams kann eine zusätzliche Dokumentation der Anforderungen in großen agilen Entwicklungsprojekten von Vorteil sein [11]; **(6)** Daneben ist Mind-mapping eine Lösung für die Erhebung von Anforderungen in Scrum, falls das Team mehrere Kunden hat. Denn Kunden, die in der Anforderungserhebung mit Hilfe von mind-map geschult werden, produzieren mehr Anforderungen als die Kunden ohne eine solche Schulung [11]; **(7)** Die Idee der testgetriebenen Entwicklung (TDD) kann auf die Anforderungserhebung und -analyse und auf die Systemvalidierung erweitert werden, indem die Kunden die Anforderungen als Story-Tests schreiben. Das Lösungssystem kann dann automatisch anhand der Story-Tests validiert werden, was den Arbeitsaufwand für den Kunden verringert [11]; **(8)** Die akzeptanztestgetriebene Entwicklung Methode (ATDD) bietet einen Kompromiss zwischen traditioneller RE und agiler RE, indem sie eine detaillierte Dokumentation aller Anforderungen liefert,

jedoch auf eine iterative Weise. Diese Dokumentation ist wie lauffähige Tests aufgebaut und basiert auf einer engen Kommunikation mit dem Kunden, sodass sie als aktuelle Dokumentation für Entwickler und Kunden dient und eine automatisierte Überprüfung der Geschäftsanforderungen während der gesamten Projektlaufzeit ermöglicht wird. Folglich kann für die Entwicklung von persönlichen Aspekten mehr Zeit verwendet werden, da die Zeit für das Testen der automatisierten Teile eingespart wird. Allerdings erfordert ATDD von den Entwicklern und Kunden mehr Vorarbeit und Disziplin als reine agile RE [11, 65]; **(9)** In der Praxis verwenden die meisten agilen Teams einen Ersatz- oder Proxy-Kunden, die die Rolle eines echten Kunden übernehmen, z. B. einen Product Owner, wenn echte Kunden nicht zur Verfügung stehen und Inkonsistenzen bei Anforderungsänderungen während der Evaluation vermieden werden sollen. Andere Organisationen setzen die Praxis des "Entwicklers vor Ort" durch die Versetzung eines Entwicklers an den Standort des Kunden [8, 12, 13, 15].

Fehlendes Wissen bei Kunden, Nutzern und Stakeholdern

Fehlende Wissen bezieht sich auf die zwei Bereiche des Wissens über agile Methoden und über Domänen. Der Kunde sollte einerseits über agile Methoden gut informiert sein und ihnen vertrauen, da es sonst schwierig ist, mit dem Kunden sinnvoll zu kommunizieren, Kundenbedürfnissen oder internen Geschäftsinformationen zu sammeln und eine konstruktive Rückmeldung zu erhalten [23, 25, 34]. Andererseits fehlt Kunden, die nur mit traditionellen Entwicklungsprozessen vertraut sind, das Verständnis und Vertrauen für agile RE-Prozesse, da dieser keine ausführlichen Anforderungsdokumentationen produziert [19]. Zudem ist die unabhängige Entscheidungskompetenz des Entwicklungsteams bei der Anwendung agiler Methoden häufig Grund für Unverständnis [28]. Solcher Mangel an agilen Kenntnissen vom Kunden kann das agile Entwicklungsteam herausfordern [33], hinzu kommen häufig fehlende Domänenkenntnisse beim Kunden, was ernsthafte Probleme bei der Durchführung des Projekts verursachen kann [25, 34, 29]. **Lösungen: (1)** Angemessene Ernennung eines Kundenvertreters mit ausreichender Domänenkenntnis von Kundenseite. [12]; **(2)** Beim ersten Treffen mit dem Kunden erklären, dass die Interaktion im Verlauf des Projekts regelmäßig stattfinden und der Kunde durch regelmäßige Diskussionen über Anforderungen und inkrementelle Rückmeldungen zum Produkt wie Produktdemos einbezogen wird [33]. Die folgenden Techniken wurden in [28] vorgeschlagen, um den Stakeholdern zu verdeutlichen, dass das Entwicklungsteam unabhängige Entscheidungen treffen kann: **(3)** kontinuierliche Koordination und Präsentation von möglichen Lösungen vor den Stakeholdern; **(4)** Gewährleistung von Transparenz über die Gründe für Entscheidungen und die Stärkung des Product Owners durch Verleihung von Kompetenz für Entscheidungsfindung; **(5)** den Stakeholdern die Konsequenzen seiner Einmischung in detaillierte Entscheidungen erklären; **(6)** Bereitstellung alternativer Lösungen für eine Anforderung und erklären, dass die von einem Stakeholder vorgeschlagene Lösung nur eine von vielen Alternativen ist [28];

Kommunikationsprobleme bei Kunden, Nutzern und Stakeholdern

Agile Entwicklung setzt auf direkte Kommunikation, allerdings können Kommunikationsprobleme trotz der starken Konzentration auf direkte Kommunikation auch bei agilen

Projekten immer noch auftreten. Dies liegt zum Teil daran, dass in Ermangelung einer umfassenden Dokumentation ein viel größerer Bedarf an qualitativ hochwertiger und intensiver Kommunikation und Interaktion während des gesamten Projekts besteht [17], andererseits ist auch Wissen über agile Methoden und Vertrauen dazu auf Kundenseite erforderlich. Falls der Kunde nicht ausreichende technische Kompetenz besitzt, besteht die Gefahr unzureichender Übermittlung der Anforderungen an das Entwicklungsteam [23]. Bei Projekten, die eine hochwertige iterative Interaktion nicht erreichen oder beibehalten können, birgt die direkte Kommunikation Risiken in sich, wie z. B. unzureichend entwickelte Anforderungen oder schlimmstenfalls falsche Anforderungen [7, 19, 17, 29, 14]. Wenn mehrere Kundengruppen und Kundenvertreter mit widersprüchlichen Bedürfnissen und Meinungen zur Verfügung stehen, ist die Konsensbildung in den kurzen Entwicklungszyklen für das Entwicklungsteam schwierig. Dies kann beim Management von Stakeholdern ein Problem darstellen [17, 19, 23].

Lösung: Kapyaho und Kauppinen [17] zeigen in ihrer Arbeit, dass Prototyping beim Aufbau einer ausreichend guten Kommunikation helfen kann, indem die folgenden Herausforderungen bewältigt werden: **(i)** Erreichen einer ausreichenden Kundenpräsenz im Projekt; **(ii)** Sicherstellung einer ausreichenden Kundenkompetenz; **(iii)** Erreichen eines Konsenses zwischen mehr als einer Kundengruppe. Zur Entwicklung von Prototypen in den Anfangsphasen des Entwicklungszyklus gibt es mehrere schnelle und kostengünstige Möglichkeiten. Die Konstruktion von UI-Prototypen ist ein effizienter Weg betrachtet werden, um einen Konsens über die geplanten Schritte zu erreichen, indem es eine Basis für die Kommunikation bereitstellt [17].

Fehlende Einbeziehung von Kunden und Stakeholdern

Agile Methoden setzen die Einbeziehung aller Interessengruppen voraus, um die Anforderungen iterativ zu erfassen [4]. Dies birgt jedoch die Herausforderung, die Beteiligten während des gesamten Entwicklungsprozesses in regelmäßigen Iterationen einzubeziehen, damit die Produktentwicklung erfolgreich verläuft. Die Erarbeitung von funktionalen Anforderungen und denjenigen an die Benutzerfreundlichkeit in Zusammenarbeit mit den direkten Endnutzern des Produkts stellen eine weitere Herausforderung dar [28]. Regelmäßige direkte Feedbacksitzungen ermöglichen die Einholung der Feedbacks von Stakeholdern über die umgesetzten Anforderungen und von neuen Anforderungen von Stakeholdern. Zur Erfassung aller Anforderungen sollten jedoch zunächst alle Stakeholder identifiziert werden, welche die verschiedenen Standpunkte des Systems repräsentieren. Ohne einen ausführlichen Identifikationsprozess könnten agile Teams wichtige Interessengruppen übersehen, was wiederum zu fehlenden Anforderungen und damit zu erhöhten Projektkosten führt [4]. Wenn zu Beginn des Projekts nicht alle Stakeholder richtig identifiziert und ihre Perspektiven bei der Sammlung, Verhandlung und Priorisierung von QRs nicht berücksichtigt worden sind, kann dies zu verschiedenen Problemen führen, z. B. zur Vernachlässigung von QRs einiger Stakeholder gegenüber anderen [2,4] oder zum Übersehen von Quellen für QRs [4]. Außerdem verursacht dies Probleme beim Stakeholder-Management. Optimalerweise sollten die Entwickler bei jedem Sprint-Review und anderen Sprint-Meetings Feedback von Stakeholdern erhalten, da diese für die erfolgreiche Fortsetzung des Projekts wichtig sind. Jedoch ist der Kunde manchmal

nicht während des gesamten Projekts erreichbar, insbesondere bei ausländischen Kunden. Agile Entwicklungsunternehmen versuchen durch die Integration des Kunden in den Entwicklungsprozess Vertrauen aufzubauen. Wenn die Kundenbeteiligung jedoch zu gering ist, gibt es kaum eine Basis, auf der Vertrauen aufgebaut werden kann [17]. Cao und Ramesh [19] erwähnen, dass die Anwendung von Praktiken wie Review-Meetings und Akzeptanztests im agilen Requirements Engineering eine Herausforderung in Bezug auf die Einbeziehung von Kunden und Stakeholdern darstellen kann. Zum Beispiel finden mehrere Organisationen trotz der Betonung der Verwendung von Akzeptanztests als agile Praktiken die Implementierung solcher Tests schwierig, weil sie keinen Zugang zu den Kunden haben, die diese Tests entwickeln [19]. Zudem kann es passieren, dass die Kunden, die diese Tests entwickeln, nicht direkt involviert sind [31]. Auch wenn der verfügbare Kundenvertreter nicht die Entscheidungsautorität über die Priorisierung und Verfeinerung von Anforderungen besitzt, führt dies zu problematischen Situationen [23].

Lösungen: **(1)** Empfohlen wird, von Anfang an die Verantwortlichkeiten der Kunden klarer festzulegen [17]; **(2)** den Kunden über agile Methoden und über die Rolle des Projekt-Owners zu informieren [17]; **(3)** Prototypen können zudem helfen, indem dadurch der Kunden in den Entwicklungsprozess integriert wird, was die meisten Vertrauensprobleme lösen und auch zu besserer Kommunikation und schnellerem gegenseitigem Verständnis beitragen kann [17]; **(4)** Viele Organisationen nutzen eigens ausgebildetes Qualitätssicherungspersonal (QA), um Kunden bei der Entwicklung dieser Tests zu unterstützen [19]; **(5)** Es wird empfohlen, die folgenden Techniken einzusetzen: Prototypen, Interviews, Beobachtung der Benutzer mit der Think-aloud-Methode, A/B-Tests, UX-Labs, Analyse des Benutzerverhaltens, freundliche Benutzertests, Alpha/Beta/Silent-Starts, kontinuierliche Verbesserung einer freigegebenen Version, Verwendung eines UX-Boards zur Wiedergabe von Benutzererkenntnissen und Testen von Hypothesen mit realen Benutzern. Darüber hinaus kann eine Anpassung der Nutzerforschung an ASD durch Reduktion der Methoden auf das Minimum helfen, daneben eine Evaluierung im Team ohne Berichtserstellung, eine Verringerung der finanziellen Einschränkungen für die Nutzereinbindung sowie der Zugangsbeschränkungen zu realen Nutzern durch Panels oder eine vorherige Rekrutierung [28]; **(6)** Die Definition von Stakeholdern und ihre Einbeziehung in regelmäßige Iterationen sollte zu Beginn vorgenommen werden; Ziele sind vorzulegen, statt Lösungen vorzuschreiben; alle möglichen Stakeholder sollen zu Beginn einbezogen und die Anzahl der Personen im Laufe der Zeit reduziert werden; der Zweck der Treffen und die Bedeutung der zu diskutierenden Ergebnisse sind schon im Vorfeld zu klären [28].

Fehlendes Kunden-/ Stakeholder-Feedback

Fehlendes Feedback ist eine weitere Herausforderung in Bezug auf die Beziehung zwischen Kunden und Stakeholder. Eines der möglichen Probleme bilden lange oder komplizierte Feedbackzyklen, auf der anderen Seite birgt fehlendes Feedback hohe Risiken. Ein Beispiel stellen Tests der Software zusammen mit tatsächlicher Hardware dar, sodass das Feedback zur Funktionalität der Software verschoben werden muss, bis die Hardware fertig ist. Wenn Kunden viel Zeit zum Testen neuer Features und deren Genehmigung benötigen, erreichen Feedbacks die Entwicklungsteams zu spät, weil die Entwickler womöglich bereits

an einem anderen Teil des Produkts arbeiten und nicht mehr genau wissen, worum es sich bei den Feedbacks handelt. Ein Grund dafür ist, dass Kunden teils keinen Wert darin sehen, schnell und häufig Feedback zu geben, selbst bei kleineren Inkrementen. Häufig entsteht dieses Problem dann, wenn das neu entwickelte System in ein größeres System integriert werden soll [18]. Darüber hinaus gibt es meist aufgrund der Komplexität und des Umfangs eines zu entwickelnden Produkts in großen Unternehmen eine Vielzahl von externen und internen Stakeholdern. Zusätzlich zu deren Anforderungen kommen aus vielen verschiedenen Quellen Anforderungen für das Projekt hinzu, zum Beispiel von Kunden, Behörden, Standardisierungsorganisationen usw. Folglich ergibt sich ein hoher Bedarf an Klärung und Diskussion der Anforderungen mit anderen Stakeholdern innerhalb oder außerhalb der Organisation. Dies verursacht Verzögerungen beim Feedbackprozess. Die durch Feedback gewonnenen Erkenntnisse sollten effektiv verwaltet werden und zusätzlich sollte der Prozess des Rückmeldens erleichtert werden [18].

5.1.3 Planungs- und Aufwand-Schätzungsbezogene Problem

Es gibt fünf Herausforderungen im Zusammenhang mit der Planung und Aufwandsschätzung im ARE: (1) Aufwandsschätzung (Budget und Zeitplan); (2) Vertragliche Probleme und die Volatilität der Anforderungen im ARE; (3) Fehlen eines formalen RE-Prozesses für die Aufwandsabschätzung; (4) Zeit- und Ressourcenverfügbarkeit; (5) Genauigkeit der Aufwandsschätzungen. Diese Herausforderungen sind in Tabelle 5.4 mit weiteren Details aufgelistet.

Tabelle 5.4: Planung und Aufwandsschätzung bezogene Herausforderungen im ARE

Challenge	Paper	Lösung	Paper
Aufwandsschätzung (Budget und Zeitplan)	7,8, 11,13, 15,22, 23,24,30	(1) regelmäßige Kommunikation und die Priorisierung von User Stories; (2) den Ansatz der anfänglichen Projektbewertung übernehmen; (3) Die ständige Planung und Aktualisierung der Zeitplan.	13,15, 22
Vertragliche Probleme und die Volatilität der Anforderungen im ARE	12,13, 15,26	(1) feste Zahlung pro Release; (2) Festlegung von rechtlichen Maßnahmen.	12,13, 15
Fehlen eines formalen RE-Prozesses für die Aufwandsabschätzung	19,31	-	-
Zeit- und Ressourcenverfügbarkeit	10	Priorisierung der Anforderungen, so dass Anforderungen automatisch ausgewählt werden können.	10
Genauigkeit der Aufwandsschätzungen	26	Aufwandsvorhersage mit einem Bayesschen Netzwerkmodell	26

Aufwandsschätzung (Budget und Zeitplan)

In vielen agil entwickelnden Softwareprojekten stellt die als notwendig eingeschätzte Projektbudget- und Zeitschätzung eine Schwierigkeit dar [7, 8, 13, 15]. Der Umfang eines Projekts basiert auf bekannten User Stories, wovon einige in kommenden Iterationen verworfen oder neue hinzugefügt werden [15]. Ursachen hierfür ist die notwendige Projektweiterentwicklung, was insbesondere bei Projekten mit vielen Features zu Änderungen bisheriger Schätzungen führen kann [13, 15, 23, 24]. Denn durch die iterativen Methoden im agilen RE werden im Verlauf der Entwicklungszyklen genauere Schätzungen erreicht.

Die Abschätzung von Kosten und Zeitplänen für das gesamte Projekt ist sehr schwer, denn das erfordert eine umfangreiche Anforderungsanalyse und -spezifikation, insbesondere bei einem großen Projekt [11, 24, 30]. Upfront-Abschätzungen sind hingegen im Falle instabiler (dynamischer) Anforderungen nicht möglich [8, 13, 22].

Lösungen: (1) Zur Erfüllung von Budget- und Zeiteinschränkungen werden regelmäßige Kommunikation und die Priorisierung von User Stories empfohlen [13, 22]; **(2)** Zudem wird die Übernahme anfänglicher Projektbewertungen vorgeschlagen, sodass den Entwicklern die richtigen Anforderungen und Planungen für eine angemessene Kostenabschätzung vorliegen [15]. **(3)** Die Planung aktualisiert den Zeitplan und die Kostenschätzung regelmäßig und fortlaufend, basierend auf der gemessenen Produktivität, sodass die Zeitplanschätzung mit der Zeit immer genauer wird [24].

Vertragliche Probleme und die Volatilität der Anforderungen im ARE

Der Vertrag ist rechtlich bindend, da Änderungen in der Regel aus Angst vor Projektmisserfolg und Budgetüberschreitung nicht erlaubt werden. Diese vertraglichen Einschränkungen können trotz möglicher Anpassungen oder sogar Nacharbeiten unter Einbeziehung des Kunden eine Herausforderung darstellen [12, 13, 15, 26].

Lösungen: (1) Eine feste Zahlung pro Release schützt die Investition des Kunden, da man nur für eine Iteration Zahlungen festlegen und entsprechend einplanen muss. Zusätzlich kann dieses Vorgehen auch eine hohe Volatilität der Anforderungen verhindern. Verträge mit festen Zahlungen pro Release reduzieren zudem projektspezifische Risiken auf Lieferantenebene bei agilen, ausgelagerten Projekten [12, 13]; **(2)** Rechtliche Maßnahmen zur Gewährleistung einer strategischen Distanz zu vertragliche Problemen und der adaptiven Idee der koordinierten RE werden empfohlen [15].

Fehlen eines formalen RE-Prozesses für die Aufwandsabschätzung

In vielen agilen Projekten werden iterative RE-Praktiken eingesetzt, da diese eine Zerlegung grober Arbeiten in kleinere, feinere Details ermöglichen, sodass die voraussichtliche Annäherung an die Gesamtarbeitszeit präziser geschätzt werden kann. Anfängliche Aufwandsschätzungen basieren in agilen Projekten jedoch meist nur auf User Stories, da Projekte nicht immer einem formalen RE-Prozess folgen [31]. Viele dieser User Stories werden möglicherweise später verworfen und viele weitere können während der Entwicklung hinzugefügt werden. Daher müssen die ursprünglichen Schätzungen während der Entwicklung angepasst werden. Da sich der Projektumfang ebenfalls ändern kann, ist es schwierig und manchmal unmöglich, genaue Kosten- und Zeitschätzungen für das gesamte Projekt zu erstellen. Daraus folgt als zusätzliche Hürde die auf derlei Schätzungen basierende Unterstützung des Managements für ein Projekt zu gewinnen und zu erhalten [19, 31].

Zeit und Ressourcenverfügbarkeit

Eine der größten Schwierigkeiten ergibt sich aus den für ein Projekt zur Verfügung stehenden begrenzten Ressourcen, da die erforderliche Funktionalität zusammen in der erwarteten Qualität, in einem vorgeschriebenen Zeitintervall mit einem begrenzten

Budget geliefert werden muss [10]. **Lösung:** Die Stakeholder müssen eine Priorisierung der Anforderungen vornehmen, sodass Anforderungen automatisch ausgewählt werden können [10].

Genauigkeit der Aufwandsschätzungen

Die Genauigkeit der Aufwandschätzung ist einer der wichtigsten Faktoren für den Erfolg des Projekts [26]. Negative Beispiele sind Fehlkalkulationen bei der Projektlaufzeit, wodurch das Projektbudget überschritten werden und das Projekt scheitern kann. Agile Methoden vermeiden den Formalismus des traditionellen Requirements Engineering, weshalb die Projektplanung in agilen Projekten oft unter Mangel an detaillierten Spezifikationen und stabilen Anforderungen leidet. Dieses Problem erschwert die Vorabschätzungen [26] und folglich auch genaue Aufwandsabschätzungen.

Lösung: Empfohlen wird eine Aufwandsvorhersage mit einem Bayesschen Netzwerkmodell, das in agilen Ansätzen hilfreich und effektiv ist [26]. Das Modell ist relativ klein, einfach und alle Eingabedaten sind leicht zu erheben, sodass die Auswirkungen auf die Agilität minimal bleiben. Das Modell sagt den Aufgabenaufwand basierend auf einer mathematischen Berechnung voraus und ist unabhängig von den verwendeten agilen Methoden, sodass es auch für den Einsatz in der frühen Projektphase geeignet ist. Das Modell wird anhand einer Datenbank mit 160 Aufgaben aus realen agilen Projekten validiert. Die Vorhersagegenauigkeit wird durch den Prozentsatz der richtigen über alle Vorhersagen gemessen [57].

5.1.4 Qualitätsanforderungsbezogene Probleme

Es gibt drei Herausforderungen im Zusammenhang mit den Qualitätsanforderungen im ARE: (1) Vernachlässigung von QRs; (2) Qualitätsanforderungen und Erhebungsherausforderungen; (3) Kommunikations- und Interaktionsprobleme im Zusammenhang mit QR. Diese Herausforderungen sind in Tabelle 5.5 mit weiteren Details aufgelistet.

Tabelle 5.5: Herausforderungen in Bezug auf Qualitätsanforderungen in ARE

Challenge	Paper	Lösung	Paper
Vernachlässigung von QRs	1,2, 3,4, 8,12, 13,15, 17,19, 20,23, 26,30	(1) Attribut-getriebenes SCRUM; (2) Das Artefakt des Security Backlog und die Rolle des Security Master in SCRUM; (3) Ein leichtgewichtiges Framework zur Identifizierung, Modellierung und Planung von NFRs und deren Verknüpfung mit funktionalen Anforderungen in agilen Prozessen; (4) Ein Java-basiertes Simulationswerkzeug zur Modellierung nicht-funktionaler Anforderungen für halbautomatische agile Prozesse.	3,12, 13,15, 26
Qualitätsanforderungen, Erhebungsherausforderungen	1,2,4	(1) Teams sind um bestimmte Komponenten des Systems herum organisiert (Komponenten Teams); (2) Aufstellung eines Vorbereitungsteams zur Erfassung der Anforderungen; (3) Reservieren eines Teils des Sprints, der für die Durchführung wichtiger QRs verwendet werden soll. **(Einige Probleme bestehen noch (siehe Tabelle 5.6)**.	2,4
Kommunikations und Interaktionsprobleme im Zusammenhang mit QR	1,4	Spätes Erkennen der Undurchführbarkeit von QRs: -	-
		Schlechte Organisation der Teams: (1) Teams sind um bestimmte Komponenten des Systems herum organisiert, z.B. um eine Datenbank, eine Benutzeroberfläche usw.; (2) Teams sind um bestimmte Produktfeatures herum organisiert, wie z. B. Login, Log-Verarbeitung, etc.; (3) Teams sind um eine einzelne Entwicklungsfunktion herum organisiert.	4

Vernachlässigung von QRs

Qualitätsanforderungen werden auch als nicht-funktionale Anforderungen (NFR) bezeichnet. Nicht-funktionale Anforderungen (NRFs) konzentrieren sich auf die Systemqualität einschließlich Wartbarkeit, Testbarkeit, Nutzbarkeit und Sicherheit. Diese Anforderungen werden jedoch im agilen Modell oft bis zu einem späteren Zeitpunkt vernachlässigt [26], da sich die Kunden oft auf die Kernfunktionalität wie wichtige Geschäftsfunktionalitäten konzentrieren. Dadurch ergeben sich große Herausforderungen, die zu massiven Überarbeitungen führen können [8, 12, 13, 15, 17, 19, 23]. Eine häufige Ausnahme ist die Konzentration auf die Bedienungsfreundlichkeit, insbesondere dann, wenn die Kunden aktiv in das sich entwickelnde System eingebunden sind und ständig Feedbacks geben [19]. Wenn die Qualitätsanforderungen (NFRs) bei der Planungssitzung nicht erfasst werden, kann dies zu einem System führen, dass wichtige NFRs nicht erfüllt [23]. Die Vernachlässigung von NFRs kann verschiedene Probleme verursachen, wie z. B: Überschreitung des Budgets, Mängel bei der Nutzbarkeit, Sicherheit und Testbarkeit [8, 12, 13, 15, 30]. Wenn nicht-funktionale Anforderungen erst spät erkannt werden, kann es sehr schwierig werden, sie in das bereits entwickelte System zu integrieren [23]. Die Vernachlässigung von NFRs kann folglich zu Systemen führen, die die Erwartungen der Benutzer nicht erfüllen. In kleinen zusammengesetzten Projekten ist meist eine schnelle Lösung dieser Probleme durch Anpassung der Anforderungen bei der nächsten Iteration und Änderung der bereits gelieferten Teile des Produkts möglich [1, 20]. In großen verteilten agilen Projekten (ALSD) wird dieses Vorgehen erschwert, da die Teams über mehrere Standorte verteilt sind und es kaum Möglichkeiten zur Ad-hoc-Koordination und Kommunikation zwischen den Teammitgliedern und mit den Kunden gibt [1]. Die folgenden Szenarien können zur Vernachlässigung der QRs führen:
(I) Problematisch ist die Behandlung der QRs als Teil der FRs, da bei niedriger Priorität einiger FRs auch die dazugehörigen QRs vernachlässigt werden [1, 4]; **(II)** Der Product Owner (PO) kennt nicht alle Domänen gleich gut und kann eine Voreingenommenheit in die Priorisierung einbringen, indem er die QRs der von ihm vertretenen Perspektive höher priorisiert als andere Perspektive, welche so vernachlässigt werden. QRs, die nach Meinung des PO keinen Geschäftswert haben, könnten leichter vernachlässigt werden. Die Vernachlässigung dieser QRs kann zu einem unflexiblen und schwer zu wartenden System führen [2, 4]; **(III)** Beim Priorisieren von Anforderungen gibt es eine Tendenz dazu, die nicht-funktionalen Anforderungen zu übersehen [23], da Anforderungen hauptsächlich auf Grundlage des Geschäftswerts für den Kunden priorisiert werden und somit nicht-funktionale Anforderungen wie Sicherheit und Skalierbarkeit zu Beginn häufig nicht berücksichtigt werden [23]. Die Verwendung des Geschäftswerts als Hauptpriorisierungskriterium kann langfristig zu ernsthaften Qualitätskompromissen führe, insbesondere da meist die Zeit bis zur Markteinführung den wichtigste Geschäftswert für die Priorisierung darstellt [17]. **(IV)** Die Vernachlässigung von QRs kann durch das Übersehen wichtiger Stakeholdern und Quellen geschehen [2, 4]; **(V)** Eine Vernachlässigung interner QRs, d. h. von Qualitätsanforderungen, die nicht sichtbar sind, kann durch Druck von Firmen auf agile Teams zur Lieferung von Funktionalitäten entstehen, zusammen mit mangelndem Interesse des Kunden an internen QRs [2, 4]; **(VI)** Die Prioritäten im Zusammenhang mit den User Stories könnten verschoben werden, wenn die Fristen näher rücken und die Notwendigkeit zur Lieferung von FRs größer wird als diejenige von QRs [4]; **(VII)** Die

Vernachlässigung nichtfunktionaler Anforderungen kann entstehen, indem in User Stories nur funktionale Anforderungen aufgezeichnet werden [8] und User Stories nur System- bzw. Produktfeatures erfüllen [13].

Lösungen: (1) Eine wichtige Methode zur Vermeidung der Vernachlässigung von QRs ist die Verwendung von Attributgetriebenem SCRUM (ACRUM), einer Scrum-Erweiterung, die durch Qualitätsattribute getrieben wird [3]. Drei Praktiken sind im ACRUM zentral: die eingebettete Analyse von Qualitätsattributen (AQUA), die Relation Assoziation Matrix (RAM) und die Validierung von Qualitätsattributen (VAQ) in SCRUM, um Qualitätsattribute in einem System zu erreichen. Das Hauptmerkmal von ACRUM ist, dass es von den SCRUM Werten und Praktiken abgeleitet ist, um mit dem SCRUM Prozess kompatibel zu sein und dessen Agilität zu erhalten [58]; **(2)** Ein weiterer Ansatz ist die Verwendung eines Security Backlog Artefakts und einer Security Master Rolle wie im SCRUM [3]. Einerseits kann ein Security Backlog als Analyse und Implementierung von Sicherheitsfeatures in Scrum-Phasen verwendet werden. Andererseits ist die Security Master Rolle verantwortlich für den Sicherheitsbereich mit folgenden Aufgaben:Identifizieren der Funktionen, die ein Sicherheitsrisiko basierend auf dem Sicherheitsprinzip darstellen, Dokumentieren des Risikos im Security Backlog und Durchführen von Sicherheitstests für ausgewählte Features [60]; **(3)** Eine weitere Lösung ist die Verwendung der NORMAP-Methodik und eines Frameworks zur Identifizierung, Modellierung und Planung von NFRs und deren Verknüpfung mit funktionalen Anforderungen in agilen Prozessen [13, 15, 26]. Diese Methodik verwendet einen risikobasierten Ansatz zur Berechnung und Visualisierung einer verbesserten Anforderungsimplementierungssequenz, die auf neu vorgeschlagenen Metriken für Anforderungsqualität und Projektmanagement basiert. NORMAP schlägt eine agile Anforderungs-Taxonomie vor, die Anforderungen entweder als funktional (Agile Use Case - AUC)) oder nicht-funktional (Agile Loose Case - ALC) klassifiziert [53]. Ein Agile Choose Case (ACC) [53] ist ebenfalls Teil der vorgeschlagenen Taxonomie und repräsentiert potenzielle Lösungen von ALCs (NFRs) gemäß dem Chungs NFR Framework [70] (siehe auch [66, 68]); **(4)** Zusätzlich kann das NORMATIC-Werkzeug helfen, ein Java-basiertes Simulationswerkzeug zur Modellierung nicht-funktionaler Anforderungen für halbautomatische agile Prozesse [12, 13, 26]. NORMATIC unterstützt NORMAP, die allgemeinere Methodik zur Modellierung nicht-funktionaler Anforderungen für agile Prozesse.

Tabelle 5.6: Herausforderungen bei der Erhebung von Qualitätsanforderungen im ARE

Challenge	Paper	Lösung	Paper
Übersehen von QR-Quellen	1, 4	-	-
Mangelnde Sichtbarkeit von QRs	1,4	(1) In großen Projekten können verteilte Teams um bestimmte Komponenten herum organisiert werden; (2) Aufstellung eines Vorbereitungsteams zur Erfassung der Anforderungen; (3) Reservieren eines Teils des Sprints, der für die Durchführung wichtiger QRs verwendet werden soll.	2, 4
Mehrdeutiger QRs Kommunikationsprozess	4	-	-
Fehlen einer weithin akzeptierten Technik zur Erfassung der QRs	1, 3	-	-

Qualitätsanforderungen Erhebungsherausforderungen

Diese Herausforderungen beziehen sich auf verschiedene Aspekte der Ermittlung der richtigen QRs von den richtigen Stakeholdern (siehe Tabelle 5.6).

Übersehen von QR-Quellen. Agile RE hängt von der Einbeziehung der Stakeholder ab, um die Anforderungen iterativ zu erfassen. QRs sind von Natur aus Querschnittsanforderungen, d. h. QRs können andere Anforderungen aus unterschiedlichen Perspektiven beeinflussen. Daher ist vor allem eine Richtlinie für eine gründliche Analyse der Stakeholder erforderlich, insbesondere bei großen Projekten. Hierbei helfen direkte Feedbacksitzungen mit Feedback der Beteiligten zu den umgesetzten Anforderungen und Besprechen neuer Anforderungen der Beteiligten an das Projekt. Zum Zusammentragen dieser Anforderungen sollten jedoch alle Stakeholder ermittelt werden, die die verschiedenen Perspektiven des Systems vertreten. Fehlt ein Prozess zur Identifizierung des Stakeholders, können agile Teams wichtige Stakeholder vergessen, was zu fehlenden Anforderungen und damit zu steigenden Projektkosten führt [1, 4].

Mangelnde Sichtbarkeit von QRs. QRs werden in externe und interne Anforderungen kategorisiert. Externe QRs sind für die Stakeholdern sichtbar und beschreiben, wie das System die gewünschte Funktion erfüllen sollte, um von ausreichender Qualität zu sein. Dazu geören z. B. Sicherheit, Leistung, Verfügbarkeit. Interne QRs hingegen beschreiben die einfache Verständlichkeit, Wartung und Erweiterung des Systems, z. B. Wartbarkeit, Modifizierbarkeit, Erweiterbarkeit und sind im Gegensatz zu externen QRs für die Stakeholder zunächst kaum sichtbar [4].

Lösungen: (1) Eine Lösung ist die Organisation verteiler Teams in großen verteilten agilen Projekten um bestimmte zentrale Komponenten herum. Diese Komponententeams entwickeln ein Eigentumsgefühl für die Komponenten, für die sie verantwortlich sind, wodurch die interne Qualität der einzelnen Komponente und damit die Qualität des gesamten Systems gesteigert wird; **(2)** Weitere Lösungen sind die Einrichtung eines Vorbereitungsteams, um Anforderungen zu sammeln und den PO davon zu befreien, die einzige Quelle für die Anforderungen zu sein sowie **(3)** das Reservieren eines Teils des Sprints extra für die Implementierung wichtiger QRs [2, 4].

Mehrdeutiger QRs-Kommunikationsprozess. Die Festlegung eines klaren QRs-Kommunikationsprozesses ist im verteilten agilen Kontext eine Herausforderung, die zu Missverständnissen bei den Qualitätsanforderungen führen kann. Ein Beispiel, das diese Herausforderung verursachen kann, ist von Alsaqaf et al. [4] wie folgt beschrieben: Die Teams waren nicht immer in der Lage, die benötigten QRs eindeutig zu kommunizieren. Die Ursache war der Umstieg von einem Anforderungsdokument einschließlich Beschreibung der QRs durch PO und Softwarearchitekt auf ein Feature-Dokument mit der Beschreibung eines bestimmten Features. In dem Feature-Dokument gibt es ein Kapitel extra für QRs, dasjedoch immer nur die folgende Aussage enthielt:– „Nicht anwendbar“. Als die Entwickler den Software-Architekten dazu befragten, war dessen Antwort, dass die allgemeinen QRs immer noch gültig seien und verwies die Teams auf ein altes Dokument. Das war absolut nicht klar, insbesondere da die Teams einen Testmanager besaßen, der Vorgaben für QRs zur Erfüllung der Tests machte [4].

Fehlen einer weithin akzeptierten Technik zur Erfassung der QRs. Die Erfassung der Qualitätsanforderungen kann auf keine etablierte agile Entwicklungsmethode zurückgreifen [1, 3].

Kommunikations- und Interaktionsprobleme im Zusammenhang mit QR

Innerhalb eines großen agilen Projektteams, z. B. in einer Scrum-of-Scrums-Situation oder zwischen den Entwicklungsteams und der Organisation des Kunden ist die gemeinsame Nutzung von Informationsressourcen häufig problematisch und kann zu einem zu späten Erkennen der Undurchführbarkeit von QRs und schlechter Organisation der Teams führen [1, 4].

Spätes Erkennen der Undurchführbarkeit von QRs. Ursachen für zu spätes Erkennen von undurchführbaren QRs können eine ungeeignete Arbeitskoordination und unzureichende Kommunikation zwischen verteilten Teams sein. Folgen sind eine teuren Refaktorierung der Software-Architektur und Neuimplementierung der gelieferten Funktionen sein [1,4], insbesondere da QRs selten in einem einzigen Code-Stück implementiert werden, sondern sich über das gesamte System erstrecken. Daher liegen sie in der Regel in der Verantwortung verschiedener Teams. Diese Teams sollten einen eindeutigen Interaktionsprozess, zur Gewährleistung der richtigen Umsetzung derjenigen QRs einrichten, für die sie gemeinsam verantwortlich sind [1, 4].

Schlechte Organisation der Teams. Große agile Projekte, die mehrere Teams umfassen, stehen vor der Herausforderung, diese Teams rund um Product Backlog Items zu organisieren. Ein Product Backlog (PB) ist eine Liste von Arbeitselementen, die PBIs genannt werden und z. B. User Stories, ausstehende Bugs oder verschiedene Aufgaben umfassen. PBIs werden von Software-Teams verwendet zur Koordination der einzelnen Arbeitsschritte verwendet [72]. Das PB ist die einzige Quelle der Anforderungen für alle Änderungen, die an dem Produkt vorgenommen werden können, weshalb sich Probleme bei der Arbeitsorganisation direkt auf die Projektumsetzung niederschlagen [4].

Lösungen: Hierfür wurden verschiedene Ansätze vorgeschlagen, die je nach Kontext und zu implementierendem System kombiniert werden können: **(1)** Komponententeams sollen um bestimmte Komponenten des Systems herum organisiert werden, z. B. um eine Datenbank, eine Benutzeroberfläche usw.; **(2)** Feature-/Szenario-Teams sollen um bestimmte Produktfeatures herum organisiert werden, wie z. B. Login, Log-Verarbeitung, usw.; **(3)** Funktionale Teams sollen um eine einzelne Entwicklungsfunktion herum organisiert werden, z. B. ein Testteam oder ein Architekturteam; Jede dieser Praktiken birgt Vor- und Nachteile, sodass eine unbegründete Kombination die Qualitätsattribute des Systems negativ beeinflussen kann [4].

5.1.5 Anforderungsmanagement bezogene Probleme

Es gibt vier Herausforderungen im Zusammenhang mit dem Änderungsmanagement: (1) Kontinuierliche Verwaltung von Anforderungen und Anforderungsänderungen; (2) Das Problem mit dem bewegten Ziel; (3) Erstellen und Verwalten von Dokumenten; (4) angemessene agile Werkzeugketten. Diese Herausforderungen sind in Tabelle 5.7 mit weiteren Details aufgelistet.

Kontinuierliche Verwaltung von Anforderungen und Anforderungsänderungen

Die Kontinuierliche Verwaltung von Anforderungen in der agilen Softwareentwicklung ist eine Herausforderung, da nicht alle Anforderungen zu Beginn fixiert sind, sondern sich im Projektverlauf ändern [28]. Deshalb wird die Anforderungsänderung und der

Umgang mit kontinuierlichen Anforderungsänderungen in mehreren Arbeiten als wichtiger Aspekt agiler Methoden genannt. Die flexible Natur agiler Methoden begrüßt Änderungen, allerdings können Nacharbeiten zu Arbeitsverzögerungen führen und bei der Evaluierung der Folgen dieser Änderungen Probleme verursachen [8, 13, 15, 30]. Der Änderungsprozess ist im gesamten ARE-Prozess präsent, weshalb ein Werkzeugzur Nachvollziehbarkeit der Änderungen an den Artefakten sehr wichtig ist. Die große Anzahl an verfügbaren Versionskontroll- oder Konfigurationsmanagement-Tools stellt jedoch eine Herausforderung bei der Auswahl geeigneter Tools für eine bestimmte Aktivität dar [29].

Lösungen: Die folgenden Techniken, Methoden und Werkzeuge werden vorgeschlagen: **(1)** Neil et al. [76] schlagen einen systematischen Ansatz für die agile Anforderungsevolution vor, bei dem man mit wenig Aufwand Anforderungen ändern und die Konsequenzen dieser Änderungen automatisch bewerten kann. Zudem ist eine Verschiebung der Entscheidung über widersprüchliche Anforderungen möglich, bis mehr Informationen zur Verfügung stehen. Eine existierende Methode ist die Verwendung von RE-KOMBINE, einem Framework zur Kontrolle von Anforderungen für spätere Umsetzung, das sowohl formal als auch flexibel ist [8, 15, 26]. **(2)** Ein weiterer Ansatz ist, Spike Stories oder Ticketing-Tools wie z. B. JIRA zu nutzen, um Unsicherheiten in den Anforderungen zu bewerten [28]. JIRA wird auch von [8, 13, 15, 30] als agiles RE-Werkzeug für den Einsatz in komplexen Projekten empfohlen; **(3)** Weitere Lösungsansätze sind zudem eine enge Zusammenarbeit mit dem anfragenden Stakeholder, **(4)** Eine regelmäßige Kommunikation innerhalb des Teams, **(5)** Das Product Backlog kontinuierlich zu verbessern und zu priorisieren, **(6)** Eine detaillierte Beschreibung der Anforderungen im Sprint Backlog, **(7)** Eine regelmäßige Überprüfung der Ergebnisse, **(8)** Den Reifegrad einer Anforderung mit dem Team zu diskutieren, **(9)** Die Gruppierung von User Stories zu Epen, **(10)** Die Verwendung der Kano-Analyse, **(11)** Das Screening und die Bewertung des Themas, **(12)** Eine relative Gewichtung [28].

Tabelle 5.7: Herausforderungen im Zusammenhang mit dem Änderungsmanagement

Challenge	Paper	Lösung	Paper
Kontinuierliche Verwaltung von Anforderungen und Anforderungsänderungen	8,13, 15,28, 29,30	(1) Ein Framework, um Anforderungen formal aber dennoch so flexibel auszudrücken, dass eine spätere Umsetzung möglich ist. (2) Spike Stories nutzen, um Unsicherheiten in den Anforderungen zu bewerten und Ticketing-Tools wie z. B. JIRA verwenden; (3) Enge Zusammenarbeit mit dem anfragenden Stakeholder; (4) regelmäßige Kommunikation innerhalb des Teams; (5) das Product Backlog kontinuierlich verbessern und priorisieren; (6) detaillierte Beschreibung der Anforderungen im Sprint Backlog; (7) regelmäßige Überprüfung der Ergebnisse; (8) den Reifegrad einer Anforderung mit dem Team diskutieren; (9) Gruppierung von User Stories zu Epen; (10) Verwendung der Kano-Analyse; (11) Screening und Bewertung des Themas; (12) relative Gewichtung.	8,13, 15,26, 28,30
Das Problem mit dem bewegten Ziel	25,34	Festlegung von Anforderungen während eines Sprints	25,34
Erstellen und Verwalten von Dokumenten	7,10, 27	-	-
Angemessene agile Werkzeugkette	18	-	-

Das Problem mit dem bewegten Ziel

Das Problem mit dem bewegten Ziel wird durch sich ändernde Prioritäten aufgrund von z. B. Änderungen im Management oder Weiterentwicklung des Unternehmens verursacht. Dies führt dazu, dass bereits spezifizierte Anforderungen veraltet sein können, da Veränderungen und Instabilität, besonders in der Kundenorganisation, nicht vom Projektprozess isoliert sind, sondern Probleme in andere Bereich durchdringen [25, 34].

Lösung: Zur Behebung dieser Probleme ist das Fixieren von Anforderungen während eines Sprints wichtig, wie es u. a. in Scrum betont wird [25, 34].

Erstellen und Verwalten von Dokumenten

Die Erstellung und Pflege der generierten Artefakte in der Anforderungsphase und die Pflege der Dokumentation stellen eine Herausforderung dar, besonders im Falle von großen agilen Projekten, in denen Requirements Engineering und Systemtest aufeinander abgestimmt werden sollen [7]. Wenn die Dokumente nicht richtig verwaltet werden, werden die Informationen in Anforderungs- oder Testdatenbanken redundant, inkonsistent oder sogar widersprüchlich mit hohen Risiken für den Projekterfolg [10]. Konkrete Richtlinien für das Anforderungsmanagement innerhalb der agilen Softwareentwicklung fehlen aktuell, weshalb die Frage des praktischen Anforderungsmanagement zukünftig empirisch evaluiert werden sollte [27].

Angemessene agile Werkzeugkette

Laut Kasauli et al. [18] werden aktuell für das Anforderungsmanagement traditionelle Werkzeuge und für die eigentliche Entwicklung auf agile Entwicklung ausgerichtete Tools wie z. B. JIRA verwendet. Diese Werkzeuge sind jedoch weitgehend voneinander getrennt und sind nicht miteinander verknüpft. Um das Requirements Engineering agiler durchführen zu können, müssen passende Werkzeuge entwickelt und etabliert werden. Darüber hinaus ist der derzeitige Prozess der Aktualisierung der Systemanforderungen zu langsam und umständlich. Auch dieses Problem kann durch ein effizienteres Werkzeug, das den agilen Informationsfluss besser unterstützt, behoben werden. Dadurch können Entwickler möglicherweise zur Durchführung von Änderungen an den Anforderungen motiviert werden, sodass die Distanz zwischen agilen User Stories und Anforderungen verringert wird [18].

5.1.6 Anforderungspriorisierungsbezogene Probleme

Es gibt zwei Herausforderungen im Zusammenhang mit der Priorisierung von Anforderungen im ARE: (1) Falsche Priorisierung von Anforderungen uund(2) Kontinuierliche Repriorisierung. Diese Herausforderungen sind in Tabelle 5.8 mit weiteren Details aufgelistet.

Tabelle 5.8: Anforderungspriorisierungsbezogene Herausforderungen im ARE.

Challenge	Paper	Lösung	Paper
Falsche Priorisierung von Anforderungen	2, 4, 11, 12, 23, 24, 26, 29, 31	-	-
Kontinuierliche Repriorisierung	19, 33	-	-

Falsche Priorisierung von Anforderungen

Eine falsche Priorisierung von Anforderungen kann aufgrund von vielen Problemen erfolgen. Ursachen kann eine Priorisierung der User Stories durch den Kunden basierend auf dem Geschäftswert sein, da dies von geschäftlichen Anforderungen und Prioritäten des Kunden abhängt. Wird diese Priorisierung bei der Überprüfung nicht korrigiert, bevor sie in die Entwicklung gehen, kann die falsche Priorisierung von Anforderungen zu erheblichen Zeit- und Geldverlusten und Nacharbeiten in der Entwicklung führen [12]. Bei der traditionellen Entwicklung fließen in die Priorisierung viele Faktoren ein, darunter der Geschäftswert, Risiken, Kosten, Implementierungsabhängigkeiten. Im Gegensatz dazu wird der vom Kunden definierte Geschäftswert bei der agilen RE meist als einziger oder primärer Faktor zur Priorisierung verwendet. Aufgrund der starken Einbindung der Kunden in den Entwicklungsprozess können geschäftliche Hintergründe zu jedem Entwicklungszyklus und für die einzelnen Anforderungen angegeben werden, sodass die Entwickler ein klares Verständnis der Prioritäten des Kunden entwickeln und dadurch die Kundenanforderungen besser erfüllen können [24].
Ein solcher geschäftswertorientierter Ansatz zur Anforderungsanalyse führt zu einem besseren Verständnis der Geschäftsdomäne, des Systems, der Priorisierung der Anforderungen, der Änderungen am Design und der Beseitigung von überflüssigen Funktionalitäten. Dies wiederum trägt zur Ausrichtung des Systems an den Geschäftsanforderungen des Kunden bei [24]. Nachteile der Verwendung des Geschäftswerts - oft mit Fokus auf die Markteinführungszeit - als einziges oder primäres Kriterium für die Priorisierung von Anforderungen sind Probleme bei der Sicherheit oder Effizient [19, 24]. Zudem können Mängel bei der Skalierbarkeit und Stabilität entstehen, wenn das System mit dem Hauptaugenmerk auf den Geschäftswert entworfen wird [11, 23, 24, 29]. Derlei Probleme können zwar für den Kunden zunächst als zweitrangig erscheinen, sind jedoch für den betrieblichen Erfolg entscheidend [11, 19, 24].
Unterschiedliche Kunden können widersprüchliche Bedürfnisse haben und es kann schwierig sein, einen Konsens über die Prioritäten der Anforderungen zu erreichen. Außerdem sollte die Priorisierung der Anforderungen für jede Iteration so vorgenommen werden, dass die Kundenzufriedenheit erreicht wird [26]. Darüber hinaus kann der Kunde Schwierigkeiten bei der Priorisierung der Anforderungen haben [11], u. a. aufgrund von mangelnder Autorität des Kundenvertreters in Entscheidungen über die Priorisierung und Verbesserung von Anforderungen zu treffen [23]. Agile Teams erhalten ihre priorisierten Anforderungen vom Product Owner, dessen Aufgabe die Vertretung der an einem Projekt beteiligten Benutzern aus verschiedenen Domänen ist. Verzerrungen können deshalb leicht entstehen, wenn der Product Owner nicht alle Domänen gleich gut kennt, sodass die von ihm vertretenden Perspektive(n) in die Priorisierung stärker einfließen und folglich höher priorisiert werden als die anderen Perspektiven [2, 4].

Kontinuierliche Repriorisierung

Repriorisierungen können einerseits in jeder Iteration aufgrund der agilen Natur von Projekten und der Tatsache, dass Änderungen im agilen Projekt in jeder Phase des Projekts willkommen sind, auftreten. Andererseits kann eine neue Priorisierung von Anforderungen auch noch erforderlich sein, nachdem die Lieferung an den Endkunden zugesagt wurde. Ursachen für diese Notwendigkeit können Komplexitäts- oder Tech-

nologieeinschränkungen sein, die eine Lieferung innerhalb des zugesagten Zeitrahmens unmöglich machen [33]. Problematisch ist zudem, dass Repriorisierungen zu Instabilität des Systems führen können [19].

5.1.7 Qualitätssicherung der Anforderungen (FR & NFR)

Die Herausforderungen der Qualitätssicherung betreffen die Aktivitäten der Verifizierung und Validierung der Anforderungen [1, 3, 4]. Es gibt zwei Herausforderungen im Zusammenhang mit der Qualitätssicherung von Anforderungen: (1) fehlende formale Modellierung für detaillierte Anforderungserhebung und Anforderungsspezifikation und (2) unzureichende Anforderungsverifizierung/Testspezifikation (FR & NFR). Diese Herausforderungen sind in Tabelle 5.9 mit weiteren Details aufgelistet.

Tabelle 5.9: Herausforderungen im Zusammenhang mit der Qualitätssicherung

Challenge	Paper	Lösung	Paper
fehlende formale Modellierung für detaillierte Anforderungserhebung und Anforderungsspezifikation	1,3,4, 14,24, 28,32	-	-
Unzureichende Anforderungsverifizierung/Testspezifikation (FR & NFR)	1,3,4, 7,22, 24	Für QR-Verifizierung: (1) automatisierte Überwachungswerkzeuge Verwenden (2) Teams aus QR-Spezialisten; (3) Innovation und Planung Iteration (IP)	2, 4
		Wegen Kundenbeteiligung siehe 5.1.2	-
		Wegen Mangel an formaler Modellierung und klarer oder vollständiger Spezifikation [1, 3, 4, 19, 22, 24]	
		Wegen fehlender formaler Inspektionen [19, 24]	
		Kommunikation ins Teams [7]	

Fehlende formale Modellierung für detaillierte Anforderungserhebung und Anforderungsspezifikation

Bei agilen Entwicklungsmethoden fehlt die formale Modellierung detaillierter Anforderungen [1, 3, 4, 24]. Die Erfassung von Anforderungen zur Ableitung detaillierter Testfälle sowie klarer und verständlicher Anforderungen unter Vermeidung von Unsicherheiten in der Entwicklung stellt eine große Herausforderung dar [28]. Zudem sind Anforderungsspezifikationen für das Entwicklungsteam erforderlich, die präzise die für Implementierung, Wartung und Schulung notwendigen Anforderungen vorgeben sollte häufig problematisch [14, 32].

Unzureichende Anforderungsverifizierung/ Testspezifikation (FR & QR)

Das Fehlen einer formalen Modellierung der detaillierten Anforderungen in agilen Entwicklungsmethoden erschwertden Prozess der Verifikation der Anforderungen [1, 3, 4, 22, 24]. Einerseits ist die Modellierung von QRs schwierig und daher ist die Identifizierung und Gestaltung von Akzeptanztests problematisch [1, 3, 4]. Fehlinterpretationen und eine unzureichende Überprüfung der Anforderungen durch den Analysten können zu notwendigen Nacharbeiten führen [22]. Bei der Verifizierung kann eine schlechte Kommunikation mit den Testern über Anforderungsänderungen zu Softwareproblemen durch fehlende Verifizierungen neue Anforderungen oder falscher Verifizierungen alter, ungültiger Anforderungen führen [7]. Auch Konsistenzkontrollen

oder formale Inspektionen werden beim agilen RE nur selten durchgeführt. Eine detailliertere Überprüfung der Anforderungen in den Prozess sollte jedoch in agiler RE unbedingt durchgeführt werden. Dabei ist eine klare, vollständige Spezifikation notwendig, da sonst insbesondere bei kritischen Anforderungen die Entwicklung erschwert wird. Agile Praktiken legen großen Wert auf kontinuierliches Testen, was eine Kernpraxis in agilen Methoden wie XP darstellt. Der Akzeptanztest in enger Zusammenarbeit mit den Kunden ist in agilen Praktiken von hoher Bedeutung, ist jedoch im Falle eines schwierigen Zugriffs auf Kundenschwierig zu implementieren. Häufig übernimmt in solchen Fällen die Qualitätssicherung (QS) die Kundenrolle [19, 24].

Lösungen: (1) Eine Lösung ist die Verwendung automatisierter Überwachungsinstrumente wie z. B. SONAR [70], um die Qualität der in Entwicklung befindlichen Software zu überwachen. Dabei werden QRs, die sich auf die interne Funktionsweise der Software beziehen, als Regeln im Monitor-Tool implementiert, das bei der Verletzung der definierten Regeln warnt [2,4]; **(2)** Eine Hilfe stellen Teams aus QR-Spezialisten mit fundiertem Wissen über den jeweiligen QR dar, die in ALSD-Projekten die Verantwortung für wichtige QRs tragen. Im Falle von Sicherheit als wichtiger QR für ein beispielhaftes Projekt wird ein Team mit Sicherheitsspezialisten zusammengestellt und die Verantwortung für die Umsetzung der Sicherheitsanforderungen in allen verteilten Teams wird diesem Sicherheitsream zugewiesen [2,4]; **(3)** Die Methode der Innovation und Planungs-Iteration (IP) bietet einen weiteren Lösungsansatz. IP ist ein Begriff aus dem SAFe [71] und bezeichnet eine Zeitperiode entsprechend einem Sprint. Ein agiles Team kann ein IP anfordern, um an anderen Aktivitäten als der Lieferung von User Stories mit einem Geschäftswert zu arbeiten, z. B. an QRs, die sonst womöglich vernachlässigt werden [2,4].

Tabelle 5.10: Herausforderungen im Zusammenhang mit agilen Teams

Challenge	Paper	Lösung	Paper
Probleme mit der Teamreifegrad und Vertrauen auf implizites Wissen	1,2, 4,11, 14,22	(1) Anfänglicher Team-Reifegrad-Test; (2) eine Führungsrolle zur Koordinierung der Zusammenarbeit; (3) das Feature Team. (4) Zusätzliche Anforderungsdokumentation	2, 11
Motivation und Widerstand gegen Veränderungen	6,14, 17,31	(1) Dokumentation detaillierter Anforderungen als Akzeptanztestkriterien; (2) Prototyping	6, 17
Schwierigkeiten mit verteilten Teams	14,28	Verwendung von skalierten Frameworks und Techniken. Frameworks wie (1) LeSS, (2) SAFe oder (3) Scrum of Scrum und Techniken wie: (1) ein gemeinsames Verständnis unter allen zu schaffen; (2) die Verbesserung der kontinuierlichen Kommunikation und Zusammenarbeit; (3) das Training der Fähigkeit zur Lösung von Abhängigkeiten; (4) wöchentliche Koordinationsmeetings durchzuführen; (5) Teams im Matrixmanagement zu organisieren; (6) Communities von Praktiken für übergreifende Themen aufzubauen; (7) Release-Planung in SAFe; (8) teamübergreifende Verfügbarkeit von Produkt- und Sprint-Backlogs; (9) Einbindung von temporären Stellvertretern in anderen Teams; (10) Durchsetzung von kontinuierlicher Integration; (11) Verbesserung der API-getriebenen Entwicklung; (12) Microservices.	28

5.1.8 Teambezogene Probleme

Es gibt drei Herausforderungen im Zusammenhang mit agilen Teams: (1) Schwierigkeiten mit verteilten Teams; (2) Motivation und Widerstand gegen Veränderungen; (3) Probleme mit der Teamreifegrad und Vertrauen auf implizites Wissen. Diese Herausforderungen sind in Tabelle 5.10 mit weiteren Details aufgelistet.

Probleme mit der Teamreifegrad und Vertrauen auf implizites Wissen

Agile RE erfordert hoch qualifizierte Mitarbeiter. Der Erfolg agiler Projekte hängt vom impliziten Wissen ab, das in den Teams eingebettet ist. Problematisch kann dies bei vorhandener Fachkompetenz, aber mangelnder Erklärungs- und Vermittlungskompetenz einiger Entwickler sein [1, 4]. Zudem ist die Fluktuation von Mitarbeitern aufgrund des impliziten Anforderungswissens ebenfalls problematisch [11]. Architekturentscheidungen können erfahrene Entwickler in der Regel besser als Nachwuchsentwickler treffen, weshalb agile Teams meist aus einer Mischung aus erfahrenen und jungen Entwicklern gebildet werden. Dadurch stehen diese Teams vor der Herausforderung, das Wissen von den erfahreneren auf die weniger erfahrenen Teammitglieder zu übertragen, sodass beide Seiten das gleiche Wissen über das System teilen und die Gesamt Qualität des Systems verbessern können, indem sie die korrekten QRs einsetzen [1, 4]. Die Reife der beteiligten Teams ist für den Aufbau und die Führung ihrer eigenen Kommunikationskanäle von entscheidender Bedeutung, weshalb bei mangelnder Kommunikation zwischen den oder mangelnder Reife Probleme entstehen [2]. Mögliche Folgen sind eine Überschreitung des Projektzeitplans [22].

Lösungen: (1) Die Reife des Teams möglichst im Voraus geprüft werden und darauf aufbauend sollte ein geeignetes Koordinationsmodell gewählt werden; **(2)** Bei mangelnder Reife innerhalb der Komponententeams ist eine höhere Führungsrolle erforderlich, um die Zusammenarbeit zwischen den Teams zu koordinieren, während ausgereifte agile Teams selbstorganisierter arbeiten können und weniger produktiv wären, wenn eine Führungsrolle zur Überwachung ihrer Arbeit hinzugezogen würde; **(3)** Feature Teams sind ein weiterer Ansatz zur Organisation verteilter agiler Teams [2]. Feature-/Szenario-Teams werden um bestimmte Produktfeatures herum organisiert, wie z. B. Login, Log-Verarbeitung, usw.; **(4)** Die Dokumentation zusätzlicher Anforderungen kann den durch Personalfluktuation verursachten Wissensverlust in großen agilen Entwicklungsprojekten mindern [11].

Motivation und Widerstand gegen Veränderungen

In vielen Teams gibt es Schwierigkeiten beim permanenten Aktualisieren der Dokumentation [17]. Ständige Änderungen der Anforderungen und Überarbeitungen können zu einem unmotivierten Team [6, 14, 17] und Widerstand gegen Veränderungen führen [31].

Lösungen: (1) Laut Bjarnason et al. [6] wird durch die Dokumentation von detaillierten Anforderungen als Akzeptanztestkriterien die Motivation der Entwickler erhöht, mit diesen Anforderungen zu arbeiten. Wie ein Interviewpartner in [6] ausgedrückt hat: “Code zu schreiben macht mehr Spaß als Anforderungen, weil sie automatische Testfälle verwenden” [6]. Kapyaho und Kauppinen [17] betrachten generell auch automatisierte Akzeptanztests als eine Motivation zum Schreiben, da sie aus Code und nicht aus reiner

Dokumentation bestehen [17]. **(2)** Prototyping und die Aktualisierung eines Prototyps kann motivierender sein als das Schreiben von Anforderungsdokumenten, da es einfach ist, die visuellen Ergebnisse sowohl mit dem Kunden als auch mit dem Entwicklungsteam durchzugehen [17].

Schwierigkeiten mit verteilten Teams

Funktionale oder technische Abhängigkeiten zwischen verschiedenen Teams sind in der agilen RE aufgrund des erheblichen Koordinationsaufwands problematisch [28]. Es ist zudem schwierig, Projektwissen auf ein großes Team zu übertragen [14].

Lösungen: Der Einsatz von skalierten Frameworks wie **(1)** LeSS, **(2)** SAFe oder **(3)** Scrum of Scrum wird empfohlen. Außerdem wird die Verwendung von den im Folgenden beschriebenen Techniken vorgeschlagen [28]: **(1)** Ein gemeinsames Verständnis unter allen zu schaffen; **(2)** Die Verbesserung der kontinuierlichen Kommunikation und Zusammenarbeit; **(3)** Das Training der Fähigkeit zur Lösung von Abhängigkeiten; **(4)** wöchentliche Koordinationsmeetings durchzuführen; **(5)** Teams im Matrixmanagement zu organisieren. Das Matrixmanagement ist eine gemischte Organisationsform mit von lateraler Autorität, Einfluss oder Kommunikation anstelle einer klassischen Hierarchie. In einer Matrix gibt es in der Regel zwei Befehlsketten, jeweils eine entlang von Funktions- und Projektlinien [72]. Weitere Lösungsansätze sind **(6)** Der Aufbau von Sammlungen von Praktiken für übergreifende Themen; **(7)** Release-Planung in SAFe; **(8)** Teamübergreifende Verfügbarkeit von Produkt- und Sprint-Backlogs; **(9)** Einbindung von temporären Stellvertretern in anderen Teams; **(10)** Durchsetzung von kontinuierlicher Integration; **(11)** Verbesserung der API-getriebenen Entwicklung; **(12)** Microservices. Die Verwendung von Microservices ermöglicht Entwicklern, Produkte in Bausteine zu zerlegen, die nur eine Funktion sehr gut erfüllen. Dies erfordert eine Anpassung von Standards und Praktiken durch Hinzufügen eines zusätzlichen Overheads in der Designphase [28, 73].

5.1.9 Tracing bezogener Probleme

Es gibt zwei Herausforderungen im Zusammenhang mit Tracing: (1) Fehlende Traceability von Anforderungen; (2) Erstellen und Pflegen von Traces. Diese Herausforderungen sind in Tabelle 5.11 mit weiteren Details aufgelistet.

Tabelle 5.11: Herausforderungen im Zusammenhang mit der Tracing.

Challenge	Paper	Lösung	Paper
Fehlende Traceability von Anforderungen	3,10, 12,25, 29,30, 34	(1) Methoden zur korrekten Erfassung und Speicherung der Anforderungen, um eine unvollständige Anforderungserhebung zu beheben und Traceability in späteren Phasen zu ermöglichen (vgl. [54-56]). (2) Verwendung mehrerer Product Backlogs zur Einbeziehung von Qualität Anforderungen aus unterschiedlichen Perspektiven. (3) Kontinuierliche Planungs- und Überprüfungssitzungen	4,10, 12,23
Erstellen und Pflegen von Traces	18,28, 30,31	-	-

Fehlende Traceability von Anforderungen

Die Verfolgung der Veränderungen einer Anforderung, die Traceability, stellt eine wichtige Herausforderung in agilen Methoden dar [12, 25, 29, 34]. Ursachen für Probleme bei der Traceability kann eine minimale Dokumentation sein (siehe Abschnitt 5.1.1). Zudem ist aufgrund der direkten Kommunikation die Nachverfolgung von Änderungen an Anforderungen schwierig [30]. Weitere Ursachen für mangelnde Traceability sind, dass die Dokumentation nicht auf dem neuesten Stand gehalten wird [25, 34] oder wenn sich die Anforderungen ändern [30]. Besonders bei Qualitätsanforderungen fehlt ein Mechanismus zur Traceability [3]. Zwar verursachen die Erstellung und Pflege von Verbindungen für eine gute Traceability von Anforderungen und Testfällen, verursacht aber zusätzliche Kosten. Allerdings kompensieren diese meist die zusätzlichen Kosten, die durch die Probleme durch eine fehlende Traceability entstehen [10].

Lösungen: (1) Es werden einige Methoden vorgeschlagen, um die Anforderungen zu sammeln und zu speichern, um unvollständige Anforderungserhebungen zu beheben [53] und die Traceability in späteren Phasen zu ermöglichen, z. B. [54,55,56] (Irum et al. [12]); **(2)** Eine Lösung ist die Verwendung mehrerer Product Backlogs, um Anforderungen aus verschiedenen Perspektiven aufzunehmen. Durch die Verwendung unterschiedlicher Perspektiven mit unterschiedlichen Interessen und QRs kann die Traceability zwischen den einzelnen QRs und ihrem jeweiligen Perspektiv und den Interessen der Stakeholder gewährleistet werden [4]; **(3)** Das Führen einer Anforderungs-Traceability-Matrix (RTM) u. a. in Excel kann ebenfalls helfen [23]. Die Anforderungs-Traceability-Matrix erfasst die kompletten Benutzer- und Systemanforderungen für das System und hilft bei dem Tracing von der Anforderung bis zum Testen zur Überprüfung, ob die Anforderung erfüllt ist [74]. Zwei Beispiele für RTM in einer agilen Umgebung sind in [74] und [75] beschrieben. Bei einer fehlenden Aktualisierung der RTM entstehen Probleme bei Änderungswünschen an den Anforderungen [23]. **(4)** Kontinuierliche Planungs- und Überprüfungssitzungen können die Bemühungen um die Aufrechterhaltung von Verbindungen zur Traceability unterstützen, weil Entwickler und Tester bei jeder Rückschau auf ihren Arbeitsstatus zurückgreifen müssen [10].

Erstellen und Pflegen von Traces

Die Erstellung und Pflege von Traces im agilen Requirements Engineering ist eine Herausforderung [18,30,31]. Zwar wurde die Notwendigkeit und der Nutzen von Tracing festgestellt, weshalb sogar Standards hierfür gefordert werden. Jedoch gibt es im Moment nicht genügend Anreize für agile Entwickler, Traces zu erstellen [18]. Kasauli et al. [18] fordern deshalb einen Anreiz oder direkt sichtbaren Nutzen für die Entwickler und eine Vereinfachung der Trace-Erstellung. Wenn die Änderungen in einer Word-Datei oder einem Lösungsdokument unter Verwendung von Team Foundation Server notiert werden ist eine Verfolgung und Pflege der Änderungshistorie nicht möglich. Dies ist nicht gut mit dem Testen und der Qualitätssicherung vereinbar und fehleranfällig [31]. Ein weiteres Problem ist die Analyse von Anforderungen in Bezug auf die vergangene Entwicklung zur Vermeidung von Nebenwirkungen [28].

Tabelle 5.12: Herausforderungen im Zusammenhang mit dem Product Owner

Challenge	Paper	Lösung	Paper
Anwesenheit eines Vermittlers wie z. B. Product Owner	31,32	-	-
Herausforderungen der Qualitätsanforderungen bezüglich der Rolle des POs	1,2,3,4	-	-

5.1.10 Product Owner bezogene Probleme

Es gibt zwei Herausforderungen im Zusammenhang mit dem Product Owner: (1) Die Anwesenheit eines Vermittlers wie z. B. des PO; (2) Herausforderungen der Qualitätsanforderungen bezüglich der Rolle des Product Owners. Diese Herausforderungen sind in Tabelle 5.12 mit weiteren Details aufgelistet.

Anwesenheit eines Vermittlers wie z.B. der Product Owner

Die Einbeziehung eines Vermittlers - meistens des Product Owners - wirkt sich auf den Erhebungsprozess der Anforderung aus [31], da bei der Kommunkation des Product Owners mit den Entwicklungsteams über die Geschäftsanforderungen Probleme entstehen können [32]. Häufig kommt es zum Verlust kritischer Informationen bei der, wenn es keine direkte Kommunikation zwischen dem Entwicklungsteam und den unterschiedlichen Kundengruppen gibt. Folglich benötigen die Entwickler zusätzliche Zeit und Arbeit für einen Konsens über die Anforderungen, bevor eine endgültige Entscheidung getroffen werden kann [31].

Herausforderungen der Qualitätsanforderungen bezüglich der Rolle des POs

Einige Herausforderungen der Qualitätsanforderungen ergeben sich aus der Rolle des Product Owners, z.B. fehlende Kenntnisse des Product Owners, die Abhängigkeit zum Product Owner als dem einzigen Ansprechpartner für die Sammlung der Anforderungen [1,3] und der Fokus des PO auf QRs aus einer bestimmten Perspektive unter Vernachlässigung der QRs aus anderen Perspektiven [2,4].

5.1.11 Koordinations- und Kommunikationsprobleme

Es gibt drei Herausforderungen im Zusammenhang mit der Koordination und Kommunikation: (1) Kommunikationsfehler; (2) Informieren und Synchronisieren zwischen Teams; (3) unterschiedliche geografische Standorte. Diese Herausforderungen sind in Tabelle 5.13 mit weiteren Details aufgelistet.

Tabelle 5.13: Herausforderungen in Bezug auf Koordination und Kommunikation

Challenge	Paper	Lösung	Paper
Kommunikationsfehler	7,25,34	-	-
Information und Synchronisierung zwischen Teams	18	-	-
Herausforderung unterschiedlicher geografischer Standorte im Agilen Requirements Engineering	22	-	-

Kommunikationsfehler

Zu den zehn wichtigsten Problemen des ARE gehören laut Wagner et al. [25,34] Kommunikationsfehler sowohl innerhalb des Projektentwicklungsteams als auch zwischen Entwicklern und Kunden. Häufigste Ursache sei fehlende Zeit, meist aufgrund des

Versuchs, Zeit in der Entwicklung zu sparen, was zu einer hohen Dichte an Sitzungen und Dokumentation in kurzer Zeit führt. Kommunikationsfehler können zudem aufgrund von allgemeineren Kommunikationsproblemen entstehen, z.B. durch einen Mangel an offenem Dialog im Team oder fehlender Kenntnis auf Seiten der Entwickler bezüglich der Abläufe bei Fragen an andere Teams. Diese Kommunikationsprobleme können zu unnötiger Arbeit und Risiken führen. Zum Beispiel kann Zeit damit verschwendet werden, neue Funktionen zu entwickeln, die zuvor bereits von anderen Teams entwickelt worden sind [25,34]. Eine weitere Herausforderung entsteht bei fehlender Kommunikation zwischen den Experten für Dokumentation von Anforderungen und Akzeptanztests über die Dokumentation bei der Kombination von Requirements Engineering mit Softwaretests (REST) [7].

Information und Synchronisierung zwischen Teams

In großen Projekten müssen Teams Informationen mit anderen Teams austauschen, um den Entwicklungsprozess zu synchronisieren. Dieser Informationsaustausch birgt Schwierigkeiten und hohen Zeitaufwand, was die Arbeitsgeschwindigkeit und die Agilität des Teams einschränkt. Die Aufteilung von Anforderungen im Projekt und die Synchronisierung von Projektinformationen kann auf zwei verschiedene Arten erfolgen. Zum einen können Anforderungen von einer externen Rolle oder einem externen Team aufgeteilt werden, indem sie an verschiedene agile Teams weitergegeben und deren Arbeitsschritte synchronisiert werden. Da dies Risiken birgt, sollte diese Methode nur im Falle von Interaktionen oder Abhängigkeiten von Features von anderen Teams gewählt werden. Zum anderen können Anforderungen innerhalb jedes Teams aufgeteilt und implementiert werden. Beide Fälle sind in kleinen und mittleren Projekten machbar, jedoch besteht das Problem in der Synchronisierung von Teams in großen Projekten in beiden Fällen [18].

Die Herausforderung unterschiedlicher geografischer Standorte im ARE

Im agilen Requirements Engineering befinden sich häufig Teammitglieder an unterschiedlichen geografischen Standorten und in verschiedenen Zeitzonen, was zu Schwierigkeiten bei der Koordination zwischen den Mitgliedern und zu einem Mangel an Kommunikation führen kann, da zusätzliche Kommunikation vonnöten ist [22].

Tabelle 5.14: Herausforderungen im Zusammenhang mit dem Gesamtbild der Anforderung

Challenge	Paper	Lösung	Paper
Fehlendes Gesamtbild	2,4,17	Gesamtbild in regelmäßigen Abständen überprüfen.	17
		Ein gemeinsames Verständnis über die Bedeutung des Gesamtbildes mittels einer Produktvision schaffen.	28
Probleme beim Verwalten des Gesamt-bildes	2,4,28	Gesamtbild in regelmäßigen Abständen überprüfen.	17
		(1) Definition von Epen oder Unterzielen zu Beginn des Projekts; (2) Verwaltung des Gesamtbildes durch den Product Owner;(3) Transparenz der Änderungen unter allen; (4) Story-Mapping; (5) kontinuierliche Teilnahme der Benutzer; (6) Identifizierung einer zentralen Kontaktperson für jedes Thema und Verwendung von Visualisierung.	28

5.1.12 Gesamtbild der Anforderungen

Es gibt zwei Herausforderungen im Zusammenhang mit dem Gesamtbild der Anforderungen: (1) fehlendes Gesamtbild; (2) Probleme beim Verwalten des Gesamtbildes. Diese Herausforderungen sind in Tabelle 5.14 mit weiteren Details aufgelistet.

Fehlendes Gesamtbild

Minimale Vorplanung führt dazu, dass zu Beginn der Entwicklung im agilen RE ein klares Anforderungsbild fehlt [2,4,17] und später erhebliche Nacharbeiten nötig werden mit der Folge großer Frustration. Auch ein rascher Wechsel des Schwerpunkts im Projekt kann ein Anzeichen für ein fehlendes Gesamtbild sein.
Lösung: Probleme bei der Einhaltung des Gesamtbildes der Anforderungen können durch eine regelmäßige Überprüfung des Gesamtbildes am wirksamsten minimiert werden [17]. Eine Produktvision hilft ebenfalls bei einem gemeinsamen Verständnis über die Bedeutung des Gesamtbildes [28].

Probleme beim Verwalten des Gesamtbildes

In der agilen Softwareentwicklung ist es eine Herausforderung, das Gesamtbild der Anforderungen zum Zeitpunkt der Implementierung von komplexen Anforderungen nicht aus dem Blick zu verlieren [28]. Agile Teams, die für die Implementierung spezifischer Komponenten verantwortlich sind, übernehmen zwar die volle Verantwortung für die jeweiligen Komponenten, können jedoch dabei die Gesamtcharakteristiken des Systems vernachlässigen. Diese Konzentration auf die eigene Komponente kann dazu führen, dass das Gesamtbild der Anforderungen verloren geht [2,4].

Lösungen: Die folgenden Techniken werden empfohlen: (1) eine regelmäßige Überprüfung des Gesamtbildes [17]; (2) eine Definition von Epen oder Unterzielen zu Beginn des Projekts; (3) die Verwaltung des Gesamtbildes als Verantwortung des Product Owners; (4) die Schaffung von Transparenz bezüglich der Veränderungen unter allen; (5) das Verstehen von Verbindungen zwischen User Stories mit Hilfe von Story Mapping; (6) die Visualisierung der Kundenreise am Anfang; (7) die kontinuierliche Einbeziehung der Nutzer, um sich auf das zu lösende Problem zu konzentrieren; (8) die Identifizierung einer zentralen Kontaktperson für verwandte Themen, um eine zügige Koordinierung zu ermöglichen; (9) die Verwendung der Visualisierung mittels Roadmaps; (10) das Skizzieren des Systems und der Prozesse; (11) die Darstellung der Wertflüsse [28].

5.1.13 Umfang Managementbezogener Probleme

Es gibt eine Herausforderungen im Zusammenhang mit dem Umfang des Managements, das Overscoping (siehe Tabelle 5.15).

Tabelle 5.15: Herausforderungen im Zusammenhang mit dem Scope Management

Challenge	Paper	Lösung	Paper
Overscoping	6, 9	“One Continuous Scope Flow”-verfahren	6

Overscoping

Overscoping ist eines der wichtigsten Themen bei agilen Projekten und tritt heutzutage tendenziell immer häufiger auf. Overscoping liegt vor, wenn der Umfang variiert und mehr Features bzw. Funktionalitäten hinzugefügt werden, als implementiert werden können. Dies kann daran liegen, dass unrealistisch große Mengen an Funktionen mit niedriger Priorität angefordert werden und die Entwicklungsteams sich zur Implementierung verpflichten, ohne späteren Änderungen mit einzuplanen. Problematisch ist auch die

häufige Unterschätzung des für die Entwicklung erforderlichen Aufwands [6]. Folglich wird das Projekt nicht in der geschätzten Zeit abgeschlossen [9, 6].

Lösung: Bjarnason et al. [6] erwähnen die "One Continuous Scope Flow"-Praxis, um der Herausforderung des Overscoping zu begegnen. Diese Praxis dokumentiert den Zufluss von Anforderungen an die Entwickler in einer Liste von Features, die kontinuierlich neu diskutiert und priorisiert werden. Dies bietet die Vorteile einer kurzen Vorbereitungszeit für jedes Feature, eines geringen Aufwands und dadurch geringerer Enttäuschung der Entwickler bei der Streichung von Features. Dadurch kann die Kommunikation über die Planung und die Koordination zwischen Management und Entwicklern verbessert werden. Durch eine kontinuierlichen Veränderung des Projektumfangs ist zwar noch Overscoping möglich, jedoch ist das Streichen von Features einfacher, da nicht zu viel Vorarbeit geleistet worden ist. Ein Nachteil dieser Praxis ist jedoch, dass das System erst am Ende des Lebenszyklus vollständig ist, mit dem Risiko, dass Probleme auf Systemebene erst am Ende des Projekts aufgedeckt werden [6].

5.2 Diskussion

Die Ergebnisse der systematischen Literaturrecherche zeigen, dass für die meisten Herausforderungen im agilen Requirements Engineering bereits verschiedene Lösungen entwickelt wurden, um den Projekterfolg sicherzustellen. Dennoch bleiben viele Herausforderungen bestehen. In diesem Abschnitt wird die Häufigkeit der Erwähnung von Over-Challenges erläutert und anschließend werden die verbleibenden Herausforderungen diskutiert, die in dieser Arbeit identifiziert wurden. Die Ergebnisse der systematischen Literaturrecherche zeigen, dass Over-Challenges in der Literatur mit der in Tabelle 5.16 dargestellter Häufigkeit erwähnt wurden.

Tabelle 5.16: Over-Challenges und Häufigkeit der Erwähnung.

ID	Over-Challenges	Häufigkeit der Erwähnung (Artikel-Anzahl)
1	Dokumentations- und User Stories bezogene Probleme	24
2	Kunden-, Nutzer-, Stakeholder-bezogene Probleme	23
3	Planungs- und Aufwand-Schätzungsbezogene Probleme	14
4	Qualitätsanforderungsbezogene Probleme	14
5	Anforderungsmanagement bezogene Probleme	13
6	Anforderungspriorisierung bezogene Probleme	11
7	Qualitätssicherung der Anforderungen (FR & NFR)	11
8	Teambezogene Probleme	10
9	Tracing bezogener Probleme	10
10	Product Owner bezogene Probleme	6
11	Koordinations- und Kommunikationsprobleme	5
12	Gesamtbild der Anforderungen im Blick zu behalten	4
13	Umfang Management-bezogene Probleme	2

Dokumentations-, Kunden-, Benutzer-, Stakeholder-bezogene Probleme wurden in der Literatur nahezu gleich häufig diskutiert und erwähnt. Eine Ursache hierfür sind die in agilen Methoden zentrale direkte Kommunikation und leichtgewichtige Dokumentation in

Form von User Stories. Einerseits hilft dies bei der flexiblen und schnellen Reaktion auf Kundenwünsche, birgt jedoch andererseits viele Schwierigkeiten und neue Herausforderungen, die in Abschnitt 5.1.1 und 5.1.2 diskutiert wurden. Für diese zwei Kategorien von Herausforderungen wurden bereits zahlreiche Lösungen vorgeschlagen und die wenigen verbleibenden Probleme sind in Tabelle 5.17 aufgelistet (siehe 1.1, 1.2 und 2.1 in Tabelle 5.17). Probleme im Zusammenhang mit der Planung, den Qualitätsanforderungen und dem Anforderungsmanagement werden in der Literatur ungefähr gleich häufig behandelt, aber sie stehen mit etwa 13 Zitierungen deutlich weniger im Fokus als die ersten beiden Probleme. Für planungs- und aufwandschätzungsbezogene Probleme wurden einige Lösungen im SLR vorgeschlagen und nur für ein Problem wurde keine Lösung präsentiert (siehe 3.1 in Tabelle 5.17).
Für Probleme im Zusammenhang mit Qualitätsanforderungen wurden verschiedene Lösungen vorgeschlagen und nur für vier Probleme sind keine Lösungen in den SLR-Ergebnissen vorhanden (siehe 4.1, 4.2, 4.3 und 4.4 in Tabelle 5.17). Für das Änderungsmanagement werden verschiedene Lösungen vorgeschlagen, aber ob diese Lösungen nur die Probleme des Anforderungs- und Änderungsmanagements in spezifischen Projekten lösen können oder universelle Ansätze sind, benötigt weitere Forschung. Es gibt jedoch noch einige zu lösende Probleme (siehe 5.1 und 5.2 in Tabelle 5.17). Priorisierungsprobleme, Probleme bei der Qualitätssicherung von Anforderungen, teambezogene Probleme und Traceability-bezogene Probleme werden mit einer relativ ähnlichen Häufigkeit von etwa zehn Zitierungen in der systematischen Literaturrecherche erwähnt. Die ersten beiden Herausforderungen, die Priorisierung und die Qualitätssicherung, scheinen kritisch zu sein, da in der Literatur keine Lösungen für die diskutierten Herausforderungen im Zusammenhang mit der Anforderungspriorisierung erwähnt wurden, und es gibt nur wenige Lösungen die Probleme im Zusammenhang mit der Anforderungssicherung. Das Problem mit der Priorisierung von Anforderungen wurde insgesamt in elf Arbeiten diskutiert, wovon der Großteil Probleme mit falscher Anforderungspriorisierung oder geschäftswertbasierter Priorisierung direkt oder indirekt als herausfordernde Situation beschreibt. Allerdings wurde in keiner Arbeit eine entsprechende Lösung für diese Situationen vorgeschlagen. Probleme im Zusammenhang mit der Priorisierung sind in Tabelle 5.15 aufgelistet (siehe 6.1, 6.2 und 6.3 in Tabelle 5.17).
Die Priorisierung ist ein sehr wichtiger Prozess im Requirements Engineering und eine schlechte Priorisierung von Anforderungen kann zu diversen Problemen im Projekt und sogar zum Scheitern des Projekts aufgrund von Budget- oder Zeitüberschreitungen führen. Dies kann sich auch auf die Kundenzufriedenheit auswirken und sogar die Motivation des Teams an ihrer Projektmitarbeit beeinträchtigen. Für Probleme im Zusammenhang mit der Qualitätssicherung von Anforderungen wurden nur einige Lösungen für eines von zwei Hauptproblemen vorgeschlagen (siehe Abschnitt 5.1.7). Die Probleme ohne Lösungsvorschlag aus SLR sind in Tabelle 5.17 aufgeführt (siehe 7.1 und 7.2 in Tabelle 5.17). Da die Qualitätssicherung ein sehr wichtiger Teil des Requirements Engineering und Software Engineering ist und diese Herausforderung stark vernachlässigt worden ist, sind weitere Forschungen für zuverlässige Praktiken notwendig. Im Gegensatz zur Anforderungspriorisierung und Qualitätssicherung wurden für teambezogene Probleme verschiedene Lösungen vorgeschlagen, jedoch gibt es auch hier nicht für alle erwähnten Herausforderungen in SLR Lösungen. Für ein Problem im Zusammenhang mit dem

Tracing wurde keine Lösung präsentiert (siehe 9.1 in Tabelle 5.17). Tracing ist eine wichtige Aufgabe und wird aus verschiedenen Gründen benötigt, u.a. um nachzuvollziehen, warum eine Anforderung zum Projekt hinzugefügt worden ist oder wer ein Feature verlangt hat. Dies ist besonders bei großen Projekten mit vielen Anforderungen von verschiedenen Stakeholdern wichtig, um einen nachvollziehbaren Workflow und eine Dokumentation zu erstellen. Product Owner-bezogene und Koordinations- und Kommunikationsprobleme werden mit etwa 6 Zitierungen in verschiedenen Arbeiten erwähnt. Die erste Herausforderung bezieht sich hauptsächlich auf die Rolle der Stakeholder im Projekt und wie sie im Projekt Probleme verursachen können. Allerdings wurden beim SLR keine Lösungen für diesen Fall vorgeschlagen, weshalb dieses verbleibende Problem unter 10.1 und 10.2 in Tabelle 5.17 aufgeführt ist. Im Zusammenhang mit Koordinations- und Kommunikationsproblemen werden drei Probleme erwähnt, für die keine Lösungen vorgeschlagen werden (siehe 11.1, 11.2 und 11.3 in Tabelle 5.17).
Diese drei Probleme können teilweise kritisch sein, da sie hauptsächlich mit der Kommunikation zwischen den Teams, der Informationsübertragung und der Synchronisierung der Arbeit im Projekt zusammenhängen. Eine schlechte Kommunikation kann verschiedene Probleme wie fehlende Anforderungen oder den Verlust anderer kritischer Informationen verursachen. Das Gesamtbild der Anforderungen und die mit dem Umfangsmanagement verbundenen Probleme werden in der Literatur am wenigsten erwähnt und scheinen deshalb nicht sehr problematisch zu sein. Darüber hinaus wurden bereits verschiedene Lösungen für diese beiden Herausforderungen vorgeschlagen. Tabelle 5.17 zeigt alle Herausforderungen, für die in den vorhandenen Literaturen keine Lösungen vorgeschlagen sind. Diese Herausforderungen sind alle den entsprechenden Over-Challenges (OC) zugeordnet und wird mit der Anzahl der Zitierungen angegeben, um die Probleme zu priorisieren. Probleme im Zusammenhang mit der Priorisierung von Anforderungen (OC: 6) werden von elf Arbeiten erwähnt und haben in der Liste der Herausforderungen höchste Priorität. Herausforderungen im Zusammenhang mit der Qualitätssicherung von Anforderungen (OC: 7) werden in zehn Arbeiten erwähnt und sind die zweitwichtigste Herausforderung. Probleme im Zusammenhang mit dem Product Owner (OC: 10) werden von sechs Arbeiten erwähnt und sind die dritte große Herausforderung.
Koordinations- und Kommunikationsprobleme werden von fünf Arbeiten erwähnt (OC: 11) und sind die viertgrößte Herausforderung in dieser Liste. Diesen Herausforderungen folgen Probleme im Zusammenhang mit dem Anforderungsmanagement (OC: 5), dievon vier Arbeiten erwähnt werden; Probleme im Zusammenhang mit der Traceability (OC: 9), ebenfalls von Arbeiten erwähnt; Probleme im Zusammenhang mit der Dokumentation und den User Stories (OC: 1), die vonebenfalls von vier Arbeiten erwähnt werden; Probleme im Zusammenhang mit den Qualitätsanforderungen (OC: 4), von drei Arbeiten erwähnt; Probleme im Zusammenhang mit der Planung und Aufwandsschätzung (OC: 3), die von zwei Arbeiten erwähnt werden; und schließlich Probleme im Zusammenhang mit Kunden, Benutzern und Stakeholdern (OC: 2), von einer Arbeit erwähnt. Fehlendes Feedback von Kunden/Stakeholdern wird nicht von vielen Autoren erwähnt, ist aber eine sehr wichtige Herausforderung, denn ohne Feedback von Kunden ist eine korrekte Spezifikation von Anforderungen unmöglich. Dies beeinflusst auch die Qualitätssicherung von Anforderungen. Außerdem sind Qualitätsanforderungen und die Qualitätssicherung von Anforderungen miteinander verbunden und ohne Qualitätsanforderungen ist die

Qualitätssicherung von Anforderungen schwer erreichbar. Qualitätsanforderungen sind relativ wenig erforscht und werden eher vernachlässigt. Diese Anforderungen umfassen wichtige Aspekte des Systems wie Sicherheit, Benutzerfreundlichkeit usw., daher sollten sie stärker berücksichtigt werden.

Tabelle 5.17: Herausforderungen ohne Lösung (OC: Over-Challenges, N: Die Erwähnung).

OC	Bestehendes Problem ohne Lösung in SLR	Paper	N	ΣN
1	1.1. Zerlegung von Anforderungen und Features	18,32	2	4
	1.2. Unzureichend spezifizierte Anforderungen	25,34	2	
2	2.1. Fehlendes Feedback von Kunden /Stakeholdern	18	1	1
3	3.1. Fehlen eines formalen RE-Prozesses für die Aufwandsabschätzung	19,31	2	2
4	4.1. Fehlen einer weithin akzeptierten Technik zur Erfassung der QRs	1,3	2	3
	4.2. Übersehen von QR-Quellen	1,4	2	
	4.3. Mehrdeutiger QRs-Kommunikationsprozess	4	1	
	4.4. Spätes Erkennen der Undurchführbarkeit von QRs	1,4	2	
5	5.1. Erstellen und Verwalten von Dokumenten	7,10,27	3	4
	5.2. Angemessene agile Werkzeugkette	18	1	
6	6.1. Falsche Priorisierung von Anforderungen	2,4,11,12,19, 23,24,26,31	10	11
	6.2. Kontinuierliche Repriorisierung	19,33	2	
7	7.1. fehlende formale Modellierung für detaillierte Anforderungserhebung und Anforderungsspezifikation	1,3,4,14,24, 28,32	7	10
	7.2. Unzureichende Anforderungsverifizierung / Testspezifikation	1,3,4,7,19, 22,24	7	
8	-	-	-	-
9	9.1. Erstellen und Pflegen von Traces	18,28,30,31	4	4
10	10.1. Die Anwesenheit eines Vermittlers, z. B. PO	31,32	2	6
	10.2. Herausforderungen der Qualitätsanforderungen bezüglich der Rolle des Product Owners	1,2,3,4	4	
11	11.1. Kommunikationsfehler	7,25,34	3	5
	11.2. Informieren und Synchronisieren zwischen Teams	18	1	
	11.3. Herausforderung unterschiedlicher geografischer Standorte im Agilen Requirements Engineering	22	1	
12	-	-	-	-
13	-	-	-	-

Kapitel 6

Lösungen für die Herausforderungen des ARE

In diesem Kapitel werden Lösungen für Herausforderungen vorgestellt, welche in der systematischen Literaturrecherche keiner Lösung zugeordnet wurden. Diese Lösungen werden unter den Abschnitten 6.1 bis 6.10 vorgestellt.

6.1 Dokumentations- und User Stories bezogene Probleme

Für die in Tabelle 6.1 dargestellten Probleme wurden im SLR-Prozess keine Lösungen vorgeschlagen. Deshalb werden sie hier behandelt.

Tabelle 6.1: Dokumentations- und User Stories bezogene Probleme.

ID	Problem	Paper
1	Zerlegung von Anforderungen und Features	18, 32
2	Unzureichend spezifizierte Anforderungen	25, 34

Zerlegung von Anforderungen und Features

Die folgenden Techniken werden von Kannan et al. [77] vorgeschlagen, um das Problem der Zerlegung der Anforderungen zu lösen:
(1). User Stories passen häufig bei einer detaillierten Ausarbeitung für eine bestimmte Entwicklungsiteration nicht mehr vollständig in eine Iteration und müssen in zwei oder mehr untergeordnete Stories aufgeteilt werden. Zur besseren Planung und Abschätzung der Größe einer Story und des Aufwands ihrer Umsetzung können Entwickler Story Points zur Messung der Größe einer User Story oder eines Features verwenden, indem jeder User Story ein Punkt zugewiesen wird bzw. zwei Punkte, wenn sie doppelt so viel Aufwand erfordert. Ein Story Point entspricht dabei dem Aufwand, der Komplexität oder dem Risiko, das mit der Entwicklung eines Features verbunden ist. Die Gesamtgröße jeder Iteration wird durch die durchschnittlich pro Iteration abgeschlossenen Story Points (Velocity) gemessen. So lassen sich genauere Voraussagen bezüglich der Auslieferung zukünftiger Features treffen [77]. Die Methode der Story Points erleichtert somit eine Aufwandsschätzung für die Entwicklung zukünftiger User Story im Vergleich zur traditionellen Schätzung mittels persönlicher Erfahrung und historischen Daten [78].
(2). Kannan et al. [77] empfehlen, dass Stories, die zu groß für eine einzige Iteration sind, nach Akzeptanzkriterien oder nach den Dimensionen in SPIDR aufgeteilt werden können (Spikes, Pfade, Interfaces, Daten und Regeln):

Tabelle 6.2: Dimensionen in SPIDR nach Kannan et al. [77].

Dimension	Beschreibung
Spikes	Spikes sind kleine, prototypische Implementierungen einer Funktionalität und helfen bei der Evaluierung der Machbarkeit neuer Systeme.
Pfade	Wenn es mehrere alternative Pfade in einer User Story gibt, können einzelne Pfade zu separaten User Stories geschrieben werden.
Schnittstellen	Stories werden aufgeteilt nach Art des Interface-Geräts, der zu unterstützenden Plattform (z. B. iOS vs. Android) oder den verfügbaren Features auf der Benutzeroberfläche (UI). Daten Stories können nach einem Teilbereich der relevanten Daten aufgeteilt werden.
Regeln	Stories werden nach der Geschäftsregel aufgeteilt, zunächst eine allgemeine Funktion zu liefern, um den Anwendern frühzeitig einen Nutzen zu bieten und diese dann durch Hinzufügen engerer Geschäftsregeln zu verfeinern.

(3). Die Skalierung von User Stories für größere Projekte [77] wird durch folgende Methoden erreicht: Für einige Projekte wird eine Excel-Tabelle zur übersichtlichen Darstellung mehrerer User Stories mit einer Zeile pro User Story und drei Spaltenüberschriften für die Komponente einer User Story (Als..., Ich will..., damit...) verwendet. Weitere nützliche Spalten sind ein kurzer Titel jeder User Story, die geschätzte Größe in Story Points und die Priorität einer Story. Tabellenkalkulationswerkzeuge ermöglichen ein einfaches Sortieren, Filtern oder Priorisieren [77]. Bei größeren Projekten wird ein Use Case-Diagramm verwendet, in dem jede User Story als ovaler Use Case dargestellt wird innerhalb eines Rechtecks, das den Gesamtumfang des Projekts darstellt. Jedes Oval wird durch eine Linie mit einer Stick-Figure für die Rolle(n) verbunden, die mit dieser User Story interagieren. Das Endergebnis ist ein einseitiges Diagramm, das den Wert aus Sicht der potenziellen Benutzer des Systems zusammenfasst. In einigen großen Projekten wird eine Feature Breakdown Struktur (FBS) eingesetzt - ähnlich einem Organigramm bzw. Arbeitsaufteilungsplan im TRE - zur Zerlegung des Projekts in die einzelnen Komponenten und zur Organisation der zu liefernden User Stories [77].

Außerdem wird die folgende Technik der INVEST-Checkliste von Mike Cohns [79] und Bill Wake [80] vorgeschlagen, um das Problem der Zerlegung von Anforderungen zu lösen:

(4). Die INVEST-Checkliste zur schnellen Evaluierung von User Stories hilft bei der Diskussion und Zerlegung von User Stories und dient der Erfüllung der sechs Merkmale guter Stories [80]. Diese Merkmale bilden das Akronym "INVEST": Unabhängig (I), Verhandlungsfähig (N), Wertvoll (V), Einschätzbar (E), Klein (S), Testbar (T).
Unabhängig: Stories sind am einfachsten zu bearbeiten, wenn sie unabhängig sind. Denn wenn sich Stories konzeptionell nicht überschneiden, können sie in beliebiger Reihenfolge geplant und umgesetzt werden. Eine Unabhängigkeit der Stories ist jedoch nicht immer möglich [80]. **Verhandlungsfähig:** Eine gute Story ist verhandelbar und somit kein expliziter Vertrag für Features, sondern die Details werden während der Entwicklung von Kunde und Entwickler gemeinsam erstellt. Deshalb besteht eine gute Story nicht aus Details, sondern aus einer Kernaussage. Im Laufe der Zeit kann die Story um Notizen, Testideen usw. erweitert werden, die jedoch nicht zur Planung oder Priorisierung benötigt werden [80]. **Wertvoll:** Eine Story muss für den Kunden wertvoll sein. Häufig wird eine Story in mehrere Schichten aufgeteilt, z.B. in eine Netzwerkschicht, eine Persistenzschicht, eine Logikschicht und eine Präsentationsschicht. Dies ist aus Sicht des Entwicklers sinnvoll, jedoch bieten einzelne Schichten für den Kunden keinen Wert. Deshalb sollte

eine Story die Essenz benennen, als würde vertikal durch die Schichten geschnitten. Andererseits kann jede Schicht für den Kunden wertvoll sein, wie in der „Pay-as-you-go“ Haltung von XP in Bezug auf die Infrastruktur [83]. **Einschätzbar:** Eine gute Story kann geschätzt werden. Anstelle einer exakten Schätzung reicht eine, die die Einordnung und Planung für den Kunden ermöglicht. Größere Stories sind schwieriger zu schätzen, was für eine Aufteilung in kleinere Stories spricht (siehe Abschnitt 3 mit Aufteilungen gemäß [18]. Ein wichtiger Faktor ist hierbei Erfahrung des Teams [80]. **Klein:** Gute Stories sind in der Regel klein, da sie u. a. genauere Schätzungen ermöglichen [80]. **Testbar:** Eine gute Story ist testbar. Testbarkeit war schon traditionell ein Merkmal von guten Anforderungen. Zur Erreichung der Testbarkeit hilft eine frühe Erstellung der Tests. Ein Indiz für eine unklar formulierte oder nicht wertvolle Story ist, wenn ein Kunde nicht weiß, wie er etwas testen soll [80].

Unzureichend spezifizierte Anforderungen

Unzureichend spezifizierte Anforderungen werden von Wagner et al. [25,34] als Folge unzureichender Fähigkeiten der beteiligten Personen entweder im Entwicklungsteam oder auf der Kundenseite beschrieben. Zur Vermeidung werden die folgenden Maßnahmen empfohlen: **(1).** Eine anfängliche Schulung der Entwickler zu erforderlichen Techniken im Projekt und in der Arbeitsdomäne des Kunden hilft beim Verfassen und der Analyse von Anforderungen [17]; **(2).** Die Information des Kunden über agile Methoden und über die Rolle des Projekt-Owners ist wichtig [17]; **(3).** Die Praxis des “Entwicklers vor Ort“ hilft dabei, dem Kunden das benötigte Wissen zur Verfügung zu stellen [8, 12, 13, 15]; **(4).** Die Methode der Innovation und Planungs-Iteration (IP) kann den Entwicklern helfen, den erforderlichen Detaillierungsgrad von Anforderungen zu erreichen [2, 4] (siehe Kapitel 5.1.7).

6.2 Kunden-, Nutzer-, Stakeholder-bezogene Probleme

Für die in Tabelle 6.3 dargestellten Probleme wurden im SLR-Prozess keine Lösungen vorgeschlagen, weshalb sie hier behandelt werden.

Tabelle 6.3: Kunden-, Nutzer-, Stakeholder-bezogene Probleme

ID	Problem	Paper
1	Fehlendes Feedback von Kunden /Stakeholdern	18

Fehlendes Feedback von Kunden /Stakeholdern

In der Literatur wird fehlendes Feedback von Kunden/Stakeholdern mit folgenden Problemen in Verbindung gebracht: **(1)** Lange oder komplizierte Feedback-Zyklen; **(2)** Uneffektive Verwaltung der Erkenntnisse aus Feedback-Sitzungen.
Für die Behebung des zweiten Problems helfen Anforderungsmanagement-Tools, die in der Abschnitt 6.5 behandelt werden. Das erste Problem bei Abhängigkeiten von entwickelten Systemen mit einem anderen Teil des Systems auftreten, was Tests einzelner Teile unmöglich macht. Zur Vermeidung sollte bereits bei der Planung auf Abhängigkeiten geachtet und Subsysteme möglichst unabhängig geplant werden, um Tests unabhängig von anderen Komponenten des Systems zu ermöglichen. Wenn dies nicht möglich ist,

ist eine genaue Zeitabschätzung und Verwaltung von Feedbackzeiten vonnöten. Wenn z. B. ein Hardware-Prototyp zwei Wochen braucht, bevor er zur Verfügung steht, können zwei Iterationen für die Software-Entwicklung bzw. eine große Iteration für die Software-Entwicklung geplant werden, um mit der ersten Hardware-Version zu testen und Feedback für den entwickelten Code zu erhalten.

Bei einer Verzögerung von Feedback durch User oder Stakeholder kann anhand der folgenden Praktiken aus SLR-Verfahren die Verzögerung minimiert werden:

(1) Von Anfang an die Verantwortlichkeiten der Kunden klarer festlegen [17]; **(2)** Den Kunden über agile Methoden und über die Rolle des Projekt-Owners informieren [17]; **(3)** Die Verwendung eines Prototyps kann helfen, indem er den Kunden in den Entwicklungsprozess integriert wird und damit die meisten Vertrauens- und Kommunikationsprobleme gelöst werden sowie ein schnelles gegenseitiges erreicht wird [17]; **(4)** Alle möglichen Stakeholder sollen zu Beginn einbezogen werden und die Anzahl der Personen im Laufe der Zeit reduziert werden;
(5) Der Zweck der Treffen und die Bedeutung der zu diskutierenden Ergebnisse soll im Vorfeld geklärt werden [28]; **(6)** In der Einleitungssitzung wird dem Kunden erklärt, dass eine regelmäßige Interaktion, Feedback und Diskussionen über Anforderungen während des gesamten Projekts nötig ist [33]; **(7)** Hilfreich ist zudem die Gewährleistung von Transparenz über die Gründe für die Entscheidungen und die Stärkung des Product Owners mit Kompetenz in der Entscheidungsfindung [28]; **(8)** A/B-Tests helfen ebenfalls [28]; **(9)** Die Methode der testgetriebenen Entwicklung (TDD) hilft auch bei der Anforderungserhebung und -analyse und der Systemvalidierung, indem die Kunden die Anforderungen als Story-Tests schreiben. Dies ermöglicht eine automatische Validierung anhand der Story-Tests, was den Arbeitsaufwand für den Kunden verringert [11].

Bei traditionellen Methoden werden auch Prototyping, Interviews, TDD, Checklisten, Akzeptanztests usw. verwendet, um die Anforderungen in der Erhebungsphase zu validieren und später während der "Negotiation" Feedback vom Kunden zu erhalten. Die Feedback-Sitzungen finden jedoch im Gegensatz zur ARE nicht regelmäßig sowie iterativ statt und der entwickelte Code wird erst nach dem Entwicklungsprozess der Hardware auf die Spezifikation getestet und verifiziert.

6.3 Planungs- und Aufwand-Schätzungsbezogene Probleme

Für das folgenden in Tabelle 6.4 dargestellte Probleme wurde im SLR-Prozess keine Lösungen vorgeschlagen.

Tabelle 6.4: Planungs- und Aufwand-Schätzungsbezogene Probleme.

ID	Problem	Paper
1	Fehlen eines formalen RE-Prozesses für die Aufwandsabschätzung	19, 31

Fehlen eines formalen RE-Prozesses für die Aufwandsabschätzung

Die Vorhersage der Attribute eines agilen Projekts ist aufgrund der agilen Charakteristika selbst eine besondere Herausforderung. Denn aufgrund der zu Projektbeginn meist unzureichenden Detailinformationen ist eine Schätzung schwierig bis unmöglich. Meist

ändern sich zudem während des Entwicklungsprozesses die Kundenanforderungen [81, 82]. Die folgenden Methoden zur Aufwandsschätzung sind zwar aus dem traditionellen Requirements Engineering bekannt, sind jedoch in der agilen Gemeinschaft für die Größen- und Aufwandsschätzung aufgrund grundlegender Unterschiede im Ansatz und der Philosophie aktuell nicht etabliert [81].
(1) Source Lines of Code (SLOC). Die einfachste Möglichkeit zur Messung der Größe eines Programms ist die Methode der Zählung der Code-Zeilen, genannt Source Lines of Code. Zusammen mit der Produktivität des Projektteams werden so der Aufwand und der Zeitplan berechnet. Vorteile des Line of Code sind ihre Automatisierbarkeit, Einfachheit und Sichtbarkeit . Der Aufwand korreliert stark mit der SLOC: Ein Projekt mit höherer SLOC erfordert mehr Zeit und Aufwand für die Entwicklung. Aber Funktionalität ist nicht effektiv mit SLOC korreliert [81].
(2) Funktion Point Analysis (FPA)-Technik. Die FPA schätzt die Metriken der zu liefernden Funktionalität anhand von fünf Hauptkomponenten: externe Eingaben, externe Ausgaben, externe Abfragen, interne logische Dateien und externe Schnittstellendateien. Das Funktionspunkt-Maß ist unabhängig von Sprache, Entwicklungsmethode und verwendeter Hardware bzw. Plattform, sodass es für organisationsübergreifendes Benchmarking geeignet ist. Weitere Vorteil sind die Möglichkeit zur Ableitung effektiver empirischer Formeln und Verbesserungsmöglichkeiten. Ein Nachteil ist hingegen die manuelle Zählweise, die nicht automatisiert werden kann. Zudem sind für eine Funktion Point Analysis detaillierte Informationen über Eingaben, Ausgaben, Screens, Datenbanktabellen sowie logische Datensätze und Felder erforderlich, um eine [81].
(3) Die Use Case Points-Methode. Eine Erweiterung der FPA für Projekte mit objektorientierten Methoden ist die Use Case Points-Methode . Für die Identifizierung von Akteuren oder einen Anwendungsfall sind keine detaillierten Richtlinien definiert. Ein Nachteil dieser Schätzungsmethode besteht darin, dass sie eine detaillierte Analyse erfordert und nicht in der frühen Projektphase durchgeführt werden kann, sondern erst nach Erstellung des Use Case-Dokuments nach der Unterzeichnung des Projekts [81].
(4) Wideband Delphi-Methode. Eine weitere Technik ist eine Gruppenschätzungsmethode, die Breitband-Delphi-Methode. Ein Vorteil ist die breite analytische Perspektive. Sie eliminiert die extremen Positionen und erzwingt einen Konsens in der Mitte. Nachteile sind ihre oft einseitigen Ergebnisse und den benötigte Aufwand an Zeit und Engagement [81].
(5) COCOMO II (konstruktives Kostenmodell) ist die am meisten akzeptierte Methode. Sie ermöglicht die Wiederverwendung und das Reengineering von Software und berücksichtigt auch die Volatilität der Anforderungen [81]. COCOMO II beinhaltet drei Teilmodelle: Applikationskomposition, Early Design und Post-Architecture. Zur Schätzung des Aufwands und des Zeitplans bei Projekten wird die Applikationskomposition verwendet mit ihren integrierten computergestützten Softwareentwicklungswerkzeugen für eine schnelle Anwendungsentwicklung. Die Modelle Early Design und Post-Architecture werden zur Schätzung von Aufwand und Zeitplan bei Softwareinfrastruktur-, größeren Anwendungs- und eingebetteten Softwareprojekten verwendet. Für eine grobe Schätzung auf der Grundlage einer unvollständigen Projekt- und Produktanalyse eignet sich besonders das Modell Early Design. Das Post-Architecture-Modell wird verwendet, wenn das Top-Level-Design vollständig ist und detaillierte Informationen über das Projekt

bekannt sind [82].

Die folgende agile Aufwandsschätzungsmethode ist ebenfalls bekannt:
(6) Story Point: Eine weitere Aufwandsabschätzungsmethode namens Story Point ist aus dem agilen Requirements Engineering bekannt und wurde bereits in Abschnitt 6.1 zur Aufteilung von Stories beschrieben. Sie bietet vor allem Vorteile zur agilen quantitativen Abschätzung des Aufwands anhand von User Stories, die im ARE zentral sind (Zu den Details siehe Abschnitt 6.1).

COCOMO II kann für den iterativen Einsatz modifiziert werden, indem jede Iteration als ein einzelnes Projekt ohne Abhängigkeiten zu anderen Iterationen abgeschätzt wird. Die folgende Methode zur Aufwandsabschätzung ist als hybride Methode für die agile webbasierte Projektabschätzung bekannt:
(7) AgileMOW ist ein aus der Erweiterung des webMO- und Cocomo II Early Design-Modells entwcikelter Ansatz zur Abschätzung von Aufwand und Kosten der Software-Entwicklung für webbasierte agile Softwareprojekte. Die Bewertung erfolgt anhand einer Reihe von Einschränkungen durch Stakeholder und Umweltmerkmale mit Hilfe des COCOMO II-Modells. Ein Vorteil von AgileMOW ist der hohe Grad an Sichtbarkeit während der Planungsphase [82]. Das Architekturmodell AgileMOW ist in Abbildung 6.1 mit den dazugehörigen Einflussfaktoren dargestellt.

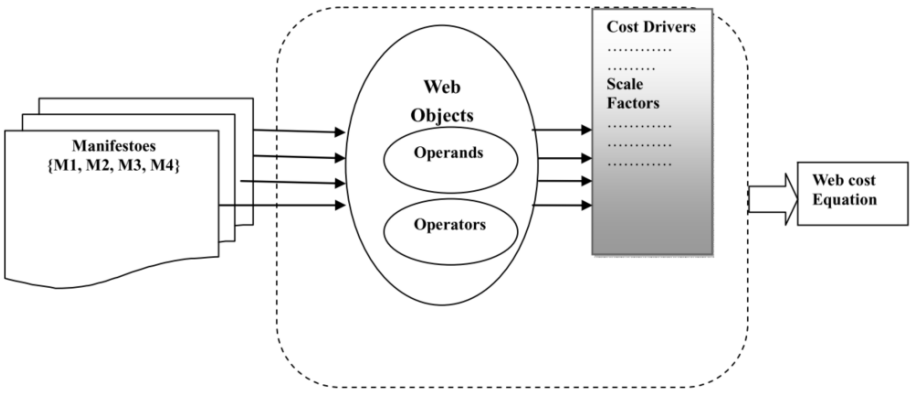


Abbildung 6.1: Vorgeschlagene Architektur von AgileMOW [82].

(8) Raslan und Ramadan [84] stellen das auf dem Constructive Cost Model (COCOMO II) dem Ansatz der Story Points und der Fuzzy-Logik basierende Frühzeitige Aufwandsschätzungsmodell (EEEM) zur frühen Aufwandsschätzung vor. Das EEEM beginnt mit einer Envisioning-Phase, die in der ersten Woche eines Projekts durchgeführt wird und während derer Umfang sowie Architektur des Projekts in Übereinstimmung mit den vorhandenen Anforderungen identifiziert werden. Dadurch entsteht eine vorläufige Schätzung des Projektzeitplans und -budgets ohne hohe Kosten für eine Dokumentation. Die Abschätzung geschieht anhand von COCOM Early Design Faktoren, die in fünf Skalierungsfaktoren und sieben Kostentreiber unterteilt sind. Zu den Skalierungsfaktoren gehören Precedentedness (PREC), Process Flexibility (FLEX), Risk Resolution (RESL), Team Cohesion (TEAM) und Process Maturity (PMAT) [84]. Vorteile der EEEM sind die Erhaltung der Integrität der agilen Prinzipien sowie eine Erhöhung des Vorhersagewerts auf 80 % erzielt [84].

Im Gegensatz zur traditionellen RE ist die agile RE auf Veränderungen eingestellt. Die daraus resultierende Dynamik der Anforderungen macht eine genaue Aufwandsabschätzung schwierig, sodass meist eine Experteneinschätzung vorgenommen wird, die jedoch uneffektiv, voreingenommen, arbeitsintensiv und ungenau sein kann. Daher wird eine systematische und dennoch leichtgewichtige Methodik zur Aufwandsschätzung zur Verbesserung der Experteneinschätzung benötigt [85]. Eine hybride Methode, wie sie in den (7) und (8) vorgestellt wird, kann dieses Problem lösen, indem formale Methoden mit einer Experteneinschätzung kombiniert werden.

6.4 Qualitätsanforderungsbezogene Probleme

Für die folgenden in Tabelle 6.5 dargestellten Probleme wurden im SLR-Prozess keine Lösungen vorgeschlagen, weshalb sie hier behandelt werden.

Tabelle 6.5: Planungs- und Aufwand-Schätzungsbezogene Probleme.

ID	Problem	Paper
1	Fehlen einer weithin akzeptierten Technik zur Erfassung der QRs	1, 3
2	Übersehen von QR-Quellen	1, 4
3	Mehrdeutiger QRs-Kommunikationsprozess	4
4	Spätes Erkennen der Undurchführbarkeit von QRs	1, 4

Fehlen einer weithin akzeptierten Technik zur Erfassung der QRs

Alsaqaf et al. [1,2,3,4] erwähnen, dass bei agilem RE eine allgemein etablierte Technik zur Erfassung der Qualität von Anforderungen fehlt. Diese Herausforderung wird wahrscheinlich ohne Antwort bleiben, bis experimentelle Studien die Techniken zur Erhebung von Qualitätsanforderungen in verschiedenen Bereichen der Industrie bewerten und eine Standard-Lösung unter den vorgeschlagenen Lösungen finden.

Übersehen von QR-Quellen

Chris Rupp und die SOPHISTen [88] beschreiben in ihrer Arbeit die Quellen von Anforderungen in traditionellen und agilen Umgebungen wie folgt: Anforderungsquellen bestehen aus Dokumenten, Systemen in Betrieb und zu befragenden Menschen wie Stakeholder. Die Ergebnisse werden in einer ständig zu aktualisierenden Liste gesammelt, die entweder separat oder als Anhang der Dokumentation der Stakeholder geführt wird. Bei der Arbeit mit Scrum haben Anforderungsquellen ebenfalls eine hohe Bedeutung, einerseits aufgrund des Produkt Owners, zu dessen Aufgaben die Pflege der Listen mit den Anforderungsquellen gehört [88].
Andererseits sollten auch die Entwickler Zugang zu den Listen haben, da eine Kommunikation ausschließlich über den Bottleneck des Product-Owners mit den Stakeholdern höchst uneffektiv ist und ein direkter Kontakt zwischen Entwickler und Stakeholder spätestens beim Sprint-Review-Meeting stattfindet und dann Detailinformationen erwartet werden [88]. Eine Dokumenten- und Systemliste sollten dem Development-Team ebenfalls zur Verfügung stehen, was zu Beginn der Zielfindung allerdings kaum möglich ist, da erst die anschließende Analyse des Systems die zur Erstellung der Anforderungsquellen nötigen

Informationen zur Abgrenzung und Einbindung von Betroffenen liefert. Folglich wird die Suche nach Anforderungsquellen und nach Stakeholdern über die gesamte Projektlaufzeit fortgesetzt, was zu einer zentralen Aufgabe eines Requirements-Engineers gehört. Sind in einem Bereich keine Anforderungsquellen oder Vertreter relevanter Stakeholder-Rollen vorhanden, muss der Requirements Engineer dies der Projektleitung kommunizieren und auf einer Lösung beharren [88].

Die Identifizierung von Stakeholdern ist auch für das Übersehen von QR-Quellen eine Hauptursache und kann durch folgende Methoden des traditionellen Requirements Engineering verbessert werden.
(1) Ballejos und Montagna [89] schlagen eine Methode zur Durchführung der Stakeholder-Identifikation vor, die auch bei firmenübergreifenden Projekten funktioniert und in organisatorische, interorganisationale und externe Dimensionen unterteilt ist. Dadurch werden systematisch alle Personen, Gruppen und Organisationen bestimmt, deren Interessen und Bedürfnisse das zu entwickelnde System beeinflussen können oder von ihm betroffen sind. Die Methode ist ein erster Ansatz und analysiert sowohl die Mitgliedsorganisationen von interorganisationalen Netzwerken und deren Beziehungen untereinander als auch die Erwartungen und Bedürfnisse externer Organisationen, die durch die Implementierung des Systems betroffen sind. Ein großer Vorteil ist die Berücksichtigung von Stakeholdern aus den verschiedenen Dimensionen sowie die Assoziation zwischen ihren Typen und Rollen, was das Risiko des Vergessens von Stakeholdern minimiert und die Anzahl der zu behandelnden und mit einem bestimmten Stakeholder zu assoziierenden Rollen deutlich einschränkt. Außerdem liefert die Methode eine Abschätzung des Einflusses und Interesses jedes Stakeholders auf das Projekt. Die Methode ist flexibel für die Erweiterung um Informationen und Wissen über die beteiligten Dimensionen und um neue Kriterien für die Auswahl. Auch Rollen können der Liste der zu berücksichtigenden Rollen hinzugefügt werden (vgl. [86], Seite 1-17).
(2) John M. Bryson [89] stellt in seinem Werk 15 Techniken zur Identifizierung und Analyse von Stakeholdern vor. Die Techniken sind in vier Kategorien eingeteilt: Organisieren der Beteiligung; Erstellen von Ideen für strategische Interventionen; Aufbau einer gewinnenden Koalition rund um die Entwicklung, Überprüfung und Annahme von Vorschlägen; Implementieren, Überwachen und Evaluieren von strategischen Interventionen. Er behauptet, dass alle beschriebenen Techniken vom Konzept her einfach sind und sich auf standardmäßige Moderationsmaterialien wie Flipcharts, Markierungsstifte, Klebeband, farbige Klebepunkte usw. stützen. Fünf Techniken zur Identifizierung und Analyse von Stakeholdern sind für die Organisation der Beteiligung besonders relevant: ein Verfahren zur Auswahl von Teilnehmern für die Stakeholder-Analyse, die grundlegende Technik der Stakeholder-Analyse, Macht- und Interessenraster, Stakeholder-Einflussdiagramme und die Matrix für die Beteiligungsplanung [89].
Sechs weitere Techniken sind besonders relevant, um Ideen für strategische Interventionen zu entwickeln: Machtbasis- und Interessenrichtungsdiagramme; das Finden des gemeinsamen Wohls und der Struktur eines überzeugenden Arguments; Erschließung individueller Stakeholder-Interessen zur Verfolgung des gemeinsamen Wohls; Stakeholder-Issue-Interrelationship-Diagramme; Problem-Frame-Stakeholder-Maps; ethische Analyseraster. Drei Techniken für die Überprüfung und Annahme von Vorschlägen werden hier

betrachtet: Raster für Zustimmung und Ablehnung der Stakeholder, Rollenspiele für die Stakeholder und Raster für Attraktivität der Richtlinien gegenüber den Fähigkeiten der Stakeholder (vgl. [89]).

Die folgende Methode zur Identifizierung von Stakeholdern ist für das agile Requirements Engineering geeignet.
(3) Ein Softwaresystem hat Ken Power zufolge ebenso wie eine Organisation eine statische und eine dynamische Struktur [87]. Die Anwendung der Stakeholder-Theorie kann den Managern einer Organisation beim Verständnis der statischen Struktur und ihrer Beziehung zu den verschiedenen Stakeholder helfen. In diesem Artikel wird auch erklärt, wie traditionelle Rollen in einer Organisation auf agile Rollen abgebildet werden, indem ein Modell zur Abbildung von Stakeholdern in Stakeholder-Gruppen und zur Quantifizierung des Einflusses von Stakeholdern vorgestellt wird [87].

Mehrdeutiger QRs-Kommunikationsprozess

Im verteilten agilen Kontext stellt die mehrdeutige Kommunikation von QRs eine große Herausforderung dar, da Teams z. B. in SAFe kein Anforderungsdokument vom Product Owner und Softwarearchitekten erhalten, sondern ein Feature-Dokument (FD) dieses ersetzt. Unter Umständen gibt es jedoch Differenzen zwischen dem FD und den Aussagen des Softwarearchitekten [4]. Die Lösung in solchen Situationen kann die Definition einer Rolle sein, die die Feature-Dokumente ständig aktualisiert und Informationen zwischen den Teams austauscht. In Abschnitt 5.1.8 werden einige Lösungen für eine bessere Kommunikation zwischen den Teams vorgestellt. In Abschnitt 6.10 werden diese Methoden ebenfalls näher thematisiert.

Spätes Erkennen der Undurchführbarkeit von QRs

Wie in Abschnitt 5.1.4 beschrieben, behaupten Alsaqaf et al. [1, 4], dass eine unangemessene Arbeitskoordination und unzureichende Kommunikation zwischen verteilten Teams zu einer verspäteten Erkenntnis führen kann, dass eine Qualitätsanforderung nicht realisierbar ist. Ursache ist meist die Verteilung von QRs über verschiedene Codestück und damit über die Verantwortung verschiedener Teams. Diese Teams sollten einen klaren Interaktionsprozess etablieren, um die ordnungsgemäße Umsetzung der QRs, für die sie gemeinsam verantwortlich sind, sicherzustellen [1, 4]. Schwierigkeiten bei verteilten Teams wurden bereits in Abschnitt 5.1.8 diskutiert und einige Lösungen vorgeschlagen. Ähnliche Probleme wie hier im Fall der Qualitätsanforderungen können auch für funktionale Anforderungen passieren, aber mit einem geringeren Effekt auf die Kommunikation zwischen Teams, da FRs nicht so stark übergreifende Verantwortungen betreffen.

6.5 Anforderungsmanagement-bezogene Probleme

Für die folgenden in Tabelle 6.6 dargestellten Probleme in Tabelle 6.5 wurden im SLR-Prozess keine Lösungen vorgeschlagen, weshalb sie hier behandelt werden.

Tabelle 6.6: Planungs- und aufwandsschätzungsbezogene Probleme.

ID	Problem	Paper
1	Erstellen und Verwalten von Dokumenten	7,10,27
2	Angemessene agile Werkzeugkette	18

Erstellen und Verwalten von Dokumenten

Verschiedene Anforderungsmanagement-Tools und -Leitfäden je nach Ziel des Projekts und verwendeter Methodik werden als Lösung für diese Herausforderung im Folgenden vorgestellt [90]. Siddiqui et al. [91] beschreiben sieben webbasierte Anforderungsmanagement-Tools mit zentralisierten Datenbanken, Einfache Traceability und Aktualisierungsmöglichkeit sowie Kompatibilität allen Softwareentwicklungsprozessen einschließlich Wasserfall, Spirale und agilen Methoden.
Konkret werden folgende Anforderungsmanagement-Tools beschrieben: IRIS Business Architect, ACCOMPA, JAMA Contour, Gatherspace, AgileSpecs, Blueprint und CASE Spec. Wichtige Features sind die Möglichkeit zur Kollaboration, Erstellung einer Historie, Tracking und Kommentare für Anforderungen, Statusberichte, Traceability und zentralisierte Datenbank, Import/Export von Daten, Zusammenfassungsberichte und benutzerdefinierte Attribute. Im Folgenden werden diese Tools kurz beschrieben.

Zu den traditionellen Werkzeugen gehören:

IRIS Business Architekt: Der IRIS Business Architekt ist ein Management-Tool, das alle Standards befolgt, und Anforderungsänderungen dynamisch, effektiv und kontinuierlich verarbeitet. IRIS wirbt mit Benutzerfreundlichkeit, Sicherheit und guter Dokumentation sowie der Möglichkeit, die IT mit dem Geschäftsbetrieb zu verbinden. Eine Testversion ist verfügbar und es unterstützt Cloud, SaaS, Mac, Windows und Web [91].
ACCOMPA: Accompa ist ein Cloud-basiertes Anforderungsmanagement-Tool, das anpassbar ist und mit Cloud und SaaS modelliert wurde. Accompa wird als eines der besten, einfach zu bedienenden und preiswerten cloudbasierten Anforderungsmanagement-Tools bezeichnet [91].
JAMA: In [91] wird das Jama-Tool als das passendste Anforderungsmanagement-Tool für Projekte genannt, die mit dem Wasserfallmodell arbeiten oder wenn die gesamten Anforderungen zu Beginn erfasst werden oder sich Anforderungen wiederholen. Eine Testversion ist verfügbar und es unterstützt Windows für die Installation und kann in der Cloud, SaaS und im Web eingesetzt werden [91].
Gatherspace: Gatherspace ist ein webbasiertes Anforderungsmanagement-Tool zur Verwaltung, Organisation und zum Austausch von Anforderungen im Team einschließlich einer Analyse von Geschäftsprozessen und der Möglichkeit zur einfachen Anwendung durch nicht-technische Mitarbeiter [91].
CASE Spec: CASE Spec dient der Spezifikation von Produkten mittels Trace-Graph oder speziellem Spezifikationseditor und ist ein kombiniertes Framework für Spezifikation und Traceability einschließlich Trace-Graphen, mehrstufigen Trace-Rastern und Impact-Graphen. Die Analyse der Traceability wird vollständig grafisch und tabellarisch erreicht und ist damit benutzerfreundich und leicht visualisierbar für Präsentation auch für nicht-technische Stakeholder. Es verwaltet auch Trace-Beziehungen und Ansichten, lässt

sich leicht mit Attributen anpassen, mit einem zentralen Repository zusammenarbeiten und mit vordefinierten Vorlagen [91].

Zu den agilen Werkzeugen gehören:

AgileSpecs: AgileSpecs ist ein webbasiertes agiles Anforderungsmanagement-Tool, das für jede Größe von Organisationen und Projekten entwickelt worden ist und ein komplettes Kommunikations- und Kollaborationszentrum, ein Entscheidungsunterstützungssystem und ein Tool zur Problemverfolgung sowie zur Ausrichtung und Planung beinhaltet [91].

Agile und Traditionelle Werkzeuge:

Blueprint: Blueprint ist ein webbasiertes Anforderungsmanagement-Tool für die Verwaltung von Modellen, Anforderungen und Dokumentationen für die Cloud und SaaS und wird von fast 100 namhaften Unternehmen zur Verbesserung der Softwarequalität, der Beschleunigung der Lieferung und der Erfüllung der Budgetgrenzen eingesetzt. Vorteile sind die Kompatibilität mit allen Methoden einschließlich Wasserfall- und agiler Softwareentwicklungsmodelle. Es kann auf Mac und Windows installiert werden [91].

In [92] werden die folgenden Methoden für das Anforderungsmanagement erwähnt. Requirements-Management-Tools auf dem Markt umfassen Rational DOORS und Rational DOORS Next Generation von IBM; Polarion von Siemens; Rational RequisitePro von IBM; Integrity von PTC; Atego Requirements Synchronize von Atego; Team Foundation Server von Microsoft; HP Application Lifecycle Management (HP ALM) von HP Software Division of Hewlett Packard Enterprise; Caliber von MICRO FOCUS; MediaWiki von Wikimedia Foundation; Jama von Jama Software; Jira von Atlassian in Kombination mit Confluence von Atlassian [92]. Zur Modellierung von Anforderungen werden Innovator von MID; Visio von Microsoft; Rational Rose Enterprise von IBM; TAU G2 von IBM; ARIS von Software AG; Yed von yWorks; Enterprise Architect von Sparx Systems; MagicDraw von No Magic Inc.; PowerDesigner von SYBASE; Bizagi Modeler BPMN von Bizagi; Balsamiq Mockups von Balsamiq Studios; Silk Together von Micro Focus; Rational Rhapsody von IBM empfohlen [92].

In [93] werden verschiedene Werkzeuge für das Anforderungsmanagement mit ihren Features beschrieben. Unter diesen Werkzeugen werden einige explizit mit Unterstützung für agile Projekte erwähnt.

Modern Requirements: Modern Requirements ist ein Anforderungsmanagement-Tool zur Verbesserung der Traceability, Prozessautomatisierung inklusive Online-Dokumentenerstellung und Visualisierungssupport. Es wurde für Azure DevOps, TFS oder VSTS von Microsoft entwickelt und soll die Entwicklungszeit verkürzen. Das Tool beinhaltet folgende Features: KI-inspirierte BA-Assistentin Alice; anpassbare Dokumentenerstellung; drei Visualisierungstypen: Diagramme, Mock-ups, Use Cases; Online-Überprüfung und -Genehmigung; Basisauskleidung, Wiederverwendung von Anforderungen und Folgenabschätzung; Test Case-Automatisierung und User Story-Erstellung; Online-Dokumentenerstellung in vordefinierten Vorlagen - Smart Docs; Planungsboards und anpassbare Dashboards; Unterstützung von BABOK, BABOK Agile, Scrum und Hybrid-Taxonomien [92].

Visure von Visure Solutions, Inc. [92,93]: Dieses Tool bietet Features wie die Unterstützung verschiedener Entwicklungsprozesse, Agile Methoden, V-Modell,

Wasserfallmodell sowie Round-Trip-Integration mit MS Word/Excel, kollaborative rollenbasierte Freigabe für den Review- und Genehmigungsprozess, vollständige Traceability von Systemanforderungen bis zum Test. Zudem soll das Tool das Risikomanagement, Testmanagement und die Fehlerverfolgung verbessern sowie die Überprüfung der Anforderungsqualität mithilfe von Natural Language Processing und semantischer Analyse erleichtern. Für die Einhaltung von Standards bietet es anpassbare Vorlagen wie für die Normen ISO26262, IEC62304, IEC61508, CENELEC50128, DO178/C, FMEA, SPICE, CMMI usw. Die Kompatibilität ist aufgrund der offenen Umgebung hoch und das Tool u.a. in DOORS, Jama, Siemens Polarion, PTC, Perforce, JIRA, Enterprise Architect, HP ALM, Microfocus ALM, PTC, TFS, Word, Excel, Test RT, RTRT, VectorCAST, LDRA integriert werden [93].

SpiraTeam: SpiraTeam ist eine integrierte Anforderungs- und Qualitätsmanagementlösung von Inflectra mit Schwerpunkt auf die Erfüllung der Auflagen von Audit-Prüfungen wie Traceability. Dafür wird die Verwaltung von Projektanforderungen und Testcases, die Agile Softwareentwicklungsplanung und -verwaltung und ein Projektportfolio-Management mit Executive Dashboards für die Berichterstattung und Analyse in Echtzeit bereitgestellt sowie ein Baselining für robustes Konfigurationsmanagement und Versionskontrolle von Artefakten. Zudem beinhaltet das Tool die Möglichkeit zum Bug-, Issue- und Task-Tracking sowie zur besseren Teamarbeit elektronische Signaturen und Dokumentenzusammenarbeit mit Quellcode- und IDE-Integration für Entwickler. Es gibt Kompatibilität mit u. a. JIRA, Visual Studio, IBM Doors, Jama, Perforce, Excel usw; Das Tool kann vor Ort oder in der Cloud gehostet werden (AWS) [93].

Agile Manager: Das Tool ist eine betont agile Projektmanagement-Lösung zum Planen, Verfolgen und Ausführen agiler Projekte, zur Beseitigung von von Latenzen und der Einführung kontinuierlicher Verbesserungen durch SaaS- oder On-Premise-Bereitstellung. Dieses Tool beinhaltet folgende Features: Drag-and-Drop-Oberfläche zur einfachen Aufgabenzuweisung und Kapazitätsverwaltung; Aufgaben- und Release-Planungsboards zum schnellen Einblick in das gesamte Projekt; erweiterte Entwicklungsanalysen für den gesamten Quellcode; Anwendungsänderungen-Verfolgung. Ziele sind die Ausrichtung und Förderung von agilen Teams mit den Unternehmenszielen und die Verbesserung derQualität des Codes [93].

CA Agile Central: CA Agile Central ist eine Plattform der Enterprise-Klasse und dient der Skalierung agiler Entwicklungspraktiken und der Überprüfung von Produktivität, Vorhersagbarkeit und Reaktionsfähigkeit anhand von Echtzeit-Performance-Metriken. Das Tool bietet folgende Features: eine kollaborative SaaS-Plattform für die agile Softwareentwicklung; Statusverfolgung in Echtzeit; anpassbare Dashboards und eine hohe Skalierbarkeit [93].

Agile Designer: CA Agile Designer ist ein Werkzeug zur Anforderungserfassung, Testautomatisierung und zum Entwurf von Testfällen, das den manuellen Testaufwand reduzieren soll. Folgende Funktionen sind enthalten: Hilfe bei der Entwicklung einer engen Zusammenarbeit zwischen Benutzern, Geschäftsbetrieb und IT; Senkung der Kosten, Unterstützung der Testung, Verfolgung sich ändernder Benutzeranforderungen; Hilfe bei der Verbesserung der Testqualität mit CA Agile Requirements Designer [92].

CodeBeamer [92,93]: CodeBeamer ist eine Software für das Anforderungsmanagement und kann mit Quellcode, Aufgaben, Bugs, Tests, Releases und allen anderen Artefakten

verknüpft werden. Zentrales Feature ist eine lückenlose Traceability über den gesamten Lebenszyklus. Daneben beinhaltet es die Möglichkeit zur Integration mit MS Office, JIRA, IBM DOORS, Enterprise Architekt über eine REST-API; erweiterte Anforderungs-Workflows und Prozessdurchsetzung; effiziente Zusammenarbeit durch Dokumentenmanagement-Funktionen des CodeBeamers; ein agiles Planungsboard zur einfachen Planung und Verwaltung von Releases [93].
TargetProcess: Targetprocess ist eine visuelle Softwarelösung für agiles Anforderungsmanagement und Projektmanagement. Dieses Tool bietet die folgenden Features: ein Targetprozess für einen schnellen Überblick über die Projekte mit verschiedenen Hierarchieebene und Detailgraden sowie hoher Skalierbarkeit. Es ist für Anwendung in Agile at scale-Frameworks wie SAFe, NEXUS, LeSS und SOS konzipiert und hilft bei der Planung und Ausführung von High-Level-Strategien besonders in großen Entwicklerteams durch die gemeinsame Nutzung von Boards mit beliebigen Personen zum Informationsaustausch. Zudem ermöglicht es das Erstellen von benutzerdefinierten Ansichten und die Integration mit anderen Tools, um eine zentrale Drehscheibe für die Zusammenarbeit und das Arbeitsmanagement zu schaffen sowie die Auswahl mehrerer Karten, sodass der Benutzer sie an eine beliebige Stelle verschieben kann [93].

Wie oben dargestellt, gibt es viele Anforderungsmanagement-Werkzeuge, die im agilen und traditionellen Requirements Engineering Prozess eingesetzt werden können. Einige dieser Tools ermöglichen es sogar, sowohl in agilen als auch in traditionellen Umgebungen zu arbeiten. Es ist nun wichtig, in der Planungsphase die richtigen Werkzeuge für das Projekt entsprechend den Projektbedingungen auszuwählen.

Angemessene agile Werkzeugkette

In 5.1.5 erwähnen Kasauli et al. [18], dass die derzeit in der agilen Entwicklung eingesetzten Werkzeuge weitgehend voneinander getrennt sind und keine sinnvolle Verbindung zueinander haben. Deshalb ist es notwendig, entsprechende Werkzeuge zu etablieren. Außerdem kritisieren sie, dass der aktuelle Prozess der Aktualisierung von Systemanforderungen zu langsam und umständlich ist und eine effizientere Werkzeuglösung benötigt wird, die den agilen Informationsfluss besser unterstützt. Wie bereits oben im Abschnitt „Erstellen und Verwalten von Dokumenten“ dargestellt, gibt es eine große Auswahl an Anforderungsmanagement-Werkzeugen, die eine gute Werkzeugkette im agilen Bereich bieten. Aber die Effektivität dieser Werkzeuge muss noch untersucht werden.

6.6 Anforderungspriorisierung-bezogene Probleme

Für die folgenden in Tabelle 6.7 dargestellten Probleme wurden im SLR-Prozess keine Lösungen vorgeschlagen, weshalb sie hier behandelt werden.

Tabelle 6.7: Planungs- und Aufwandschätzungsbezogene Probleme.

ID	Problem	Paper
1	Falsche Priorisierung von Anforderungen	2,4,11,12,19,23,24,26,29,31
2	Kontinuierliche Repriorisierung	19,33

Falsche Priorisierung von Anforderungen

Wie in Abschnitt 5.6 beschrieben, kann ein falsches Priorisierungsproblem durch fünf Gründe verursacht werden: 1) Verwendung des Geschäftswerts als einziger Parameter für die Priorisierung von Anforderungen; 2) Kunden können keine einheitliche Entscheidung treffen; 3) Kunden fehlt das Wissen, um Anforderungen zu priorisieren; 4) Kunden haben nicht genug Autorität; 5) Product Owner fehlt das Fachwissen über die Domäne.
Vier dieser Probleme beziehen sich auf Personen und die Kommunikation zwischen ihnen. Diese Probleme konnten mit den folgenden Lösungen aus Abschnitt 5.2 von SLR-Verfahren gelöst werden:

(1) Ein Anforderungsingenieur, der den Kunden vor Ort begleitet (gemäß dem Prinzip des Entwicklers vor Ort), kann RE-bezogene Aufgaben für den Kunden vor Ort durchführen, sodass die Arbeitslast des Kunden vor Ort reduziert wird [8, 11, 12, 13, 15]; **(2)** der Entwickler vor Ort kann die Kunden zudem beraten und ihnen helfen, das benötigte Wissen zu erlernen [11]; **(3)** Rollen wie der Domänenbesitzer und der Businessanalyst können im agilen RE vorteilhaft sein, insbesondere in großen Organisationen, um das Verständnis der Anforderungen zu verbessern. Der Business-Analyst einerseits dokumentiert den Geschäftsprozess und die Datenanforderungen in Form von User Stories. Der Domänenbesitzer andererseits ist dafür verantwortlich, Wissen über die Geschäftsdomäne des Kunden zu sammeln, es unter den Teammitgliedern zu teilen und es in einer Form zu verpacken, die in zukünftigen Projekten wiederverwendbar ist [11, 61]. Der Project Owner kann auf der Grundlage der Informationen, die diese Rollen ermitteln, eine Priorisierung für die Wichtigkeit der Kunden und Stakeholder vornehmen und diesen eine Priorität zuweisen. Darauf basierend wird anschließend den verschiedenen Stakeholdern ein passender Grad an Autorität bei der Entscheidungsfindung im Projekt verliehen. Diese Rollen können auch das Domänenwissen der Stakeholder verbessern, indem sie die notwendigen Informationen über die Geschäftsdomäne bereitstellen.
(4) Kapyaho und Kauppinen [17] zeigen in ihrer Arbeit, dass Prototyping dabei helfen kann, eines Konsenses zwischen mehreren Kundengruppen zu erreichen; **(5)** A/B-Tests [28]: Beim A/B-Testing werden zwei unterschiedliche Varianten A und B derselben Anwendung in Live-Experimenten verglichen und die bessere ausgewählt. Live-Benutzer werden zufällig einer der beiden Varianten zugewiesen und einige Metriken von Interesse werden gesammelt und verglichen, z. B. die Wahrscheinlichkeit dafür, dass ein Benutzer in einer E-Commerce-Webanwendung kauft. Dieses Vorgehen ermöglicht eine Maximierung erwünschter Metriken durch die iterative Entwicklung von Varianten und deren vergleichende Bewertung durch Live-Experimente [94]. Zudem erleichtern A/B-Tests in Verbindung mit Prototyping die Entscheidungsfindung.

Im Zusammenhang mit dem Problem, den Geschäftswert als einzigen Parameter für die Priorisierung von Anforderungen zu verwenden, werden neben dem Geschäftswert weitere Parameter aus dem traditionellen Requirements Engineering empfohlen. Folgende Lösungen sind in der Literatur beschrieben:

(1) Achimugu et al. [96] haben eine systematische Literaturrecherche über Techniken der Anforderungspriorisierung in der agilen Softwareentwicklung durchgeführt und einen umfassenden Überblick über die Anforderungspriorisierung gegeben, indem sie die Stärken

und Schwächen der vorhandenen Anforderungspriorisierung-Techniken untersucht haben. Einige Priorisierungstechniken mit ihren jeweiligen Stärken und Schwächen sind in Tabelle 6.8 aufgeführt [96]. Zudem ist in Tabelle 6.9 ein Vergleich dieser Anforderungspriorisierungstechniken dargestellt. Anhand dessen soll die Auswahl einer oder mehrerer geeigneter Priorisierungstechniken je nach Bedeutung von Kundenwichtigkeit, Kundenpräferenz, strategische Wichtigkeit, Beurteilungen über die Erfahrungen der Teilnehmer, Geschäftswert, Risiko und Strafe getroffen werden [96].

Tabelle 6.8: Techniken der Anforderungspriorisierung in agiler Softwareentwicklung [96].

Priorisierungstechniken	Beschreibung	Stärke (+) und Schwäche (-)
Analytischer Hierarchieprozess (AHP) [95, 96]	Die AHP-Technik berechnet die relative Wichtigkeit der einzelnen Anforderungen mithilfe einer paarweisen Vergleichsmatrix [96].	(+) Fähigkeit, Zielkonflikte aufzulösen; (+) zuverlässiges Ergebnis; (-) Zeitaufwand bei höherer Anzahl von Anforderungen; (-) Nicht skalierbar, daher problematisch für größere Projekte [96].
Binärer Suchbaum (BST)	BST ordnet die Anforderungen in einer hierarchischen Reihenfolge der Parent-Child-Beziehung. Zuerst werden alle erhobenen Anforderungen analysiert und dann in eine Rangfolge gebracht [96].	(+) BST kann problemlos bis zu tausenden Anforderungen skalieren und ist dennoch schnell; (-) BST vergibt keine Prioritätswerte, sondern nur eine einfache Rangfolge der Anforderungen; (-) BST zeigt, welche Anforderung günstiger ist, aber das Ausmaß, in dem die Anforderung wichtig ist, kann nicht bekannt sein und daher ist der Vergleich nur ordinal [96].
Kosten-Wert-Ranking (CV)	CV bewertet die Kosten und den Wert jeder Anforderung aus einer Implementierungsperspektive. Die Kosten-Wert-Rangliste priorisiert die Anforderungen auf der Basis der Implementierungskosten und ihres wahrgenommenen Werts [96].	(+) Fähigkeit, die Bewertungen von Kosten und Wert von Anforderungen für die Implementierung zu kombinieren; (-) Zeitaufwendig und nicht skalierbar; (-) Die Komplexität der Anforderungsberechnung steigt bei der Verwaltung von Abhängigkeiten mit zunehmender Anzahl von Anforderungen [96].
Kumulative Abstimmung [95,96]	Der Kostenwert ist ein verhältnisorientiertes RP-Verfahren. Kunden und Stakeholder erhalten eine feste Anzahl von Einheiten, die für die Priorisierung von Anforderungen verwendet werden, wobei den Anforderungen auf der Grundlage der Kunden- oder Stakeholder-Präferenz eine Stimme gegeben wird [96].	(+) Die Einfachheit des Ansatzes; (-) Nicht geeignet für eine große Anzahl von Anforderungen; (-) Erlaubt keine Bewertung des relativen Prioritätsunterschieds zwischen den Anforderungen [96].
Kano Model [95,96]	Kano ist ein Vergleich der Kundenzufriedenheit. Das Kano-Modell teilt sie in fünf Kategorien ein: Muss-Qualität, Zufriedenheits-Qualität, Attraktivitäts-Qualität, Indifferenz-Qualität und Rückwärts-Qualität [96].	(+) Kano befasst sich mehr mit den Kundenpräferenzen für Vertrauenswürdigkeit; (+) Die Kano-Methode ist der schnellste Weg, um Anforderungen zu priorisieren; (-) Es kann nur für die Analyse der Auswirkungen verwendet werden; (-) Es ist nicht geeignet für Vorschläge zu neuen Produktfeatures[96].
MoSCoW	Priorisierung von Anforderungen auf der Basis des unmittelbarsten Geschäftsnutzens zu einem frühen Zeitpunkt. MoSCoW steht für: M: Muss-Anforderung, S: Soll-Anforderung, C: Könnte-Anforderung, W: Wird nicht-Anforderung. M steht dabei für die höchste und W für die niedrigste Anforderung [96].	(+) Es ist konsistent, einfach, schnellund kann mit vielen Alternativen umgehen; (+) Es ist leicht skalierbar, da es sowohl für kleine als auch für große Anzahlen von Anforderungen geeignet ist; (-) Nachteil ist die fehlende Abstufung innerhalb der Kategorien. Es ist schwierig zu erkennen, welche SOLLEN- oder KÖNNEN-Anforderungen wichtiger sind als andere. Besser geeignet für Produkte mit weniger Kunden [96].

Priorisierungstechniken	Beschreibung	Stärke (+) und Schwäche (-)
Planungsspiele	Im Planungsspiel kategorisieren die Kunden die Anforderungen in die Kategorien wesentlich, bedingt und optional. Der gesamte Prozess basiert auf zwei Kriterien: dem technischen Risiko, das von den Entwicklern bestimmt wird, und dem Geschäftswert, der von den Kunden bestimmt wird [96].	(+) Das Planspiel hat eine bessere Modifikation der numerischen Berechnung; (+) Der Priorisierungsprozess ist einfach und schnell zu erledigen; (-) Problematisch bei großer Anzahl von Anforderungen [96]. Paarweise Analyse Anforderungen werden paarweise verglichen, um eine Rangfolge zu erstellen, bis die wichtigsten Anforderungen ganz oben auf dem Stapel erscheinen. Der paarweise Vergleich von Anforderungen basiert auf der Wichtigkeit [96]. (+) Die Kriterien für den Vergleich der Optionen können informell bleiben, so dass die Beurteilung auf den Erfahrungen der Teilnehmer basiert. (-) Langwierig, kompliziert und liefert unzuverlässige Ergebnisse; (-) ignoriert den Detailgrad; (-) beschränkte Skalierbarkeit [96].
Quality Functional Deployment (QFD)	Die QFD-Technik basiert auf Matrizen, in denen die Erwartungen des Kunden in chronologischer Reihenfolge dargestellt werden, um zu bestimmen, wie diese Erwartungen von den Entwicklern erfüllt werden sollen. Die Matrix wird als Haus der Qualität erstellt, das sowohl das Was (Kundenbedürfnisse) als auch das Wie (Entwicklerbedürfnisse) widerspiegelt [96].	(+) QFD ist eine strukturierte Methodik für Kundenbedürfnisse in Form der Stimme des Kunden; (-) Hauptsächlich in kleinen Systemen angewendet; (-) Einschränkung bei Inkonsistenzen und Skalierbarkeit [96].
Wertorientierte Priorisierung (VOP) [95,96]	Der VOP basiert auf den Bewertungen der Stakeholder durch Verknüpfung mit den identifizierten Geschäftswerten. Die Anforderungen werden entsprechend ihrer Auswirkungen auf die anerkannten Geschäftswerte der Organisation priorisiert [96].	(+) Der Geschäftswert der Organisation wird bei der Priorisierung berücksichtigt; (-) Ignoriert Anforderungsabhängigkeiten; (-) Nicht geeignet für größere Projekte [96].

Tabelle 6.9: Vergleich der Techniken zur Anforderungspriorisierung anhand verschiedener Faktoren [96].

Priorisierung-Techniken	Skalierbar?	Komplexität	Dauer	Aspekt
Analytischer Hierarchieprozess	Nein	Sehr kompliziert.	Sehr langsam	strategische Wichtigkeit, Strafe
Binärer Suchbaum (BST)	Ja	Einfach	Langsam	-
Kosten-Wert-Ranking (CV)	Nein	Kompliziert	Schnell	Kundenwichtigkeit
Kumulative Abstimmung	Nein	Einfach	Schnell	Kundenwichtigkeit
Kano Model	Nein	Einfach	Schnell	Kundenpräferenz
MoSCoW	Ja	Sehr einfach	Schnell	Geschäftlicher Vorteil
Planunsspiele	Nein	Einfach	Schnell	Geschäftswert, Technisches Risiko
Paarweise Analyse	Nein	Kompliziert	Schnell	Urteile über Erfahrungen der Teilnehmer
Quality Funktional Deployment	Nein	Kompliziert	Langsam	Meinung des Kunden
Wert-orientierte Priorisierung	Nein	kompliziert	Schnell	Geschäftswert

(2) Noor et al. [95] haben eine systematische Literaturübersicht über Techniken der Anforderungspriorisierung mit Fokus auf die agile Softwareentwicklung durchgeführt und eine Reihe von Priorisierungsmethoden gefunden, die in agilen Umgebungen üblich sind. Einige dieser Arbeiten verwenden kombinierte und hybride Techniken. Diese sind in Tabelle 6.10 aufgeführt.

Tabelle 6.10: Hybride Techniken in [95].

Nr.	Technik
1	Analytischer Hierarchieprozess (AHP) [95,96]
2	Kumulative Abstimmung [95,96]
3	Kano Model [95,96]
4	Wert-orientierte Priorisierung (VOP) [95,96]
kombinierte und hybride Techniken	
5	Adaptive Fuzzy Hierarchical Cumulative Voting (AFHCV) [95]
6	hybride Gamified-Methode mit Analytischer Hierarchieprozess [95]
7	FuzzyHCV -Hybrid auf Hierarchical Cumulative Voting und Fuzzy-Expertensystem [95]
8	Interaktiver Genetischer Algorithmus (IGA) [95]
9	suchbasierte Methode [95]
10	die Kombination von Planung Spiele und Analytischer Hierarchieprozess [95].

Noor et al. [95] erwähnen, dass einige Forscher für bessere Ergebnissen und effizientere Prozesse die Verwendung mehr als eine Technik zur Anforderungspriorisierung empfehlen. [95]. Durch die Kombination von Methoden wie z. B. Analytic Hierarchy Process (AHP) und Kumulativer Stimmabgabe (CV) mit anderen Techniken können die Vorteile der Zuverlässigkeit und hohen Verbreitung genutzt und Nachteile wie Skalierbarkeitsproblemen und hoher Zeitaufwand ausgeglichen werden [95]. Zudem wird empfohlen nicht den Wert-Faktor oder Risiko-Faktor zu implementieren, wie heutzutage am häufigsten der Fall sei, sondern auch Faktoren für Benutzer, Entwickler und Tester für die Priorisierung der Anforderungen berücksichtigt werden sollen. Dies werde zusammen mit den kombinierten Methoden die schnelle und hochqaulitative Lieferung von Softwareprodukten fördern.

(3) Rahim et al. [97] schlagen eine Technik zur Priorisierung von Anforderungen in der agilen Entwicklung (RIZE) vor, bei der durch eine einfache mathematische Gleichung (siehe [97], Abschnitt III, Seite 3, Gleichung 1) eine Punktzahl ermittelt zur Bestimmung der Priorität von Anforderungen verwendet wird. Die Methode beinhaltet folgende Schritte: die Identifizierung von verschiedenen Anforderungsaspekten, das Zuweisen einer Gewichtung für jeden identifizierten Aspekt, das Zuweisen der Punktzahl der Aspekte und Berechnen der Prioritätspunktzahl für jede Anforderung. Im letzten Schritt wird für die verschiedenen Aspekte der Priorisierung für jede Anforderung ein numerischer Wert zwischen 1 und 10 zugewiesen, der zum Sortieren der Anforderungen verwendet werden kann. Ein höherer Wert steht für eine höhere Wichtigkeit. Vorteile seien die einfache Anwendung, hohe Anpassbarkeit, Zeitersparnis und Skalierbarkeit.
Daneben werden einige neue Methoden vorgeschlagen deren Hauptmerkmale in Tabelle 6.11 aufgelistet sind (zur vollständigen Zusammenfassung siehe [97], Seite 5, Tabelle I). Die meisten Priorisierungsmethoden verwenden nicht nur den Geschäftswert als

Parameter für die Priorisierung von Anforderungen. Wenn in der Planungsphase die richtige Methode gewählt wird, wird somit das Problem des Geschäftswerts als einzige Anforderung behoben. Im Allgemeinen müssen diese Methoden allerdings in der Praxis erforscht werden, um ihre Wirksamkeit in verschiedenen Projekten zu überprüfen.

Tabelle 6.11: Zusammenfassung der verschiedenen Priorisierungstechniken [97].

Techniken	Starvation-Szenarios?	Einfach zu verstehen?	Dauer	Anpassbar?	Skalierbar?	Einfach zu bedienen?	Beteiligte Aspekte
Theorie W [97]	Nein	Nein	Hoch	Ja	Nein	Ja	größtenteils
Top Ten Anforderungen [97]	Nein	Ja	Niedrig	Nein	Nein	Ja	größtenteils
Wiegers Priorisierung [97]	Nein	Ja	Durchschnitt	Ja	Ja	Ja	Geschäftswert, Kosten, Risiko
Numerische Belegung [97]	Nein	Ja	Niedrig	Nein	Nein	Ja	größtenteils

Kontinuierliche Repriorisierung

Es wurden keine direkten Forschungen in Bezug auf diese Herausforderung identifiziert. Wenn aber in jeder Iteration die Anforderungen gut priorisiert worden sind, sollte eine Re-Priorisierung nicht oft vorkommen. Die Priorisierungsmethoden sind der Schlüssel zur Lösung dieser Probleme und eine gute Priorisierungsmethode kann diese Probleme größtenteils verhindern.

6.7 Qualitätssicherung der Anforderungen (FR und NFR)

Für die in Tabelle 6.12 dargestellten Probleme wurden im SLR-Prozess keine Lösungen vorgeschlagen, weshalb sie hier behandelt werden.

Tabelle 6.12: Qualitätssicherung der Anforderungen (FR und NFR) Probleme.

ID	Problem	Paper
1	Fehlende formale Modellierung für detaillierte Anforderungserhebung und Anforderungsspezifikation	1,3,4,14,24,28,32
2	Unzureichende Anforderungsverifizierung/ Testspezifikation	1,3,4,7,22,24

Fehlende formale Modellierung für detaillierte Anforderungserhebung und Anforderungsspezifikation

Im Zusammenhang mit der fehlenden formalen Modellierung für die detaillierte Anforderungserhebung und Anforderungsspezifikation treten folgende Probleme auf: **(1)** Klare und verständliche Anforderungen Formulierung; **(2)** Erstellung präziser Anforderungsspezifikationen und Ableitung detaillierter Testfälle. Es wurden hier keine Lösungen für eine formale Anforderungsdokumentation gefunden, jedoch wird im Folgenden eine Arbeit zu einer hybriden Methode zur besseren Dokumentation und zum besseren Informationsaustausch innerhalb des Teams vorgestellt.

Portela et al. [98] schlagen die hybride Methode Scrumconix vor, einen leichtgewichtigen

Ansatz zum Dokumentieren in agilen globalen Software-Entwicklungsumgebungen mit dem Ziel der Minimierung von Auswirkungen von sprachlichen und kulturellen Distanzen. Dabei liefert Scrumconix ein Domänenmodell, das zur Visualisierung des Projektumfangs nützlich ist. Zudem beinhaltet es Use-Case-Diagramme zur Identifikation von Software-Funktionalitäten und Robustheitsdiagramme für eine klare und gemeinsame Sprache. Hiermit sei eine genaue Schätzung durchführbar und Testfälle seien leicht zu erstellen. Die Verwendung von Diagrammen verbessere ebenfalls die Kommunikation zwischen lokalen und ausländischen Teammitgliedern. Scrumconix verwendet das Projektmanagement-Framework von Scrum sowie einen Teil des Software-Engineering-Leitfadens von ICONIX und besteht aus zwei Teilen: Sprint Zero und Sprint One bis N.
Sprint Zero liefert ein Gesamtverständnis des Projekts mittels folgender Phasen: **(1)** Sammeln der Gesamtanforderungen; **(2)** Domänen- und Use Case-Modellierung; **(3)** Referenz-Software-Architektur; **(4)** Gesamtprojektschätzung; **(5)** Verfeinerung der GUIs; und **(6)** Sprint-Planung (vgl. [98], Seite 1). Sprint Eins bis N bestehen aus **(1)** Codierung; **(2)** Verbesserung der GUI-Storyboards; **(3)** Entwicklung von Testfällen; **(4)** Testen/Fehlerbehebung; **(5)** Sprint-Planung (vgl. [98], Seite 1).

Ein webbasiertes Informationsmanagement-Tool kann zudem automatisch eine formale Dokumentation aus eingegebenen User Stories und Testfällen und anderen Informationen, die durch das Projekt kommen, generieren und damit Standards zu erfüllen helfen. Darüber hinaus kann jeder Rolle eine definierte Aufgabe zugewiesen werden, um wichtige Informationen in ihrer Arbeit in eine Datenbank einzutragen, die dann das Werkzeug auswertet und sie in formaler Form zur Spezifikation hinzufügt. Weitere Forschung ist erforderlich, um die Möglichkeit zu untersuchen, ein solches Werkzeug mit den richtigen Parametern zu entwerfen, die in einem agilen Projekt benötigt werden.

Unzureichende Anforderungsverifizierung/ Testspezifikation

Im Zusammenhang mit dem Problem einer unzureichenden Anforderungsverifizierung/ Testspezifikation werden folgende Teilprobleme erwähnt: (1) fehlende formaler Modellierung und Fehlen einer klaren oder vollständigen Spezifikation [1, 3, 4, 19, 22, 24]; (2) fehlende formale Inspektion [19, 24]; (3) Kommunikationsprobleme in Teams [7]. Das erste Problem wurde oben im ersten Teil breit diskutiert und das dritte Problem wird in Abschnitt 6.10 behandelt. Für das Fehlen von formalen Inspektionen in der agilen Methodik wurde keine Forschung gefunden und es besteht ein Bedarf, dies im agilen Requirements Engineering zu untersuchen.

6.8 Tracing-bezogene Probleme

Für das in Tabelle 6.13 dargestellte Problem wurde im SLR-Prozess keine Lösungen vorgeschlagen, weshalb es hier behandelt wird.

Tabelle 6.13: Tracing-bezogenes Problem.

ID	Problem	Paper
1	Erstellen und Pflegen von Traces	18,28,30,31

Erstellen und Pflegen von Traces

Die Erstellung und Pflege von Traces ist mit Hilfe von Anforderungsmanagement-Tools und Prozessautomatisierung möglich. In Abschnitt 6.5. werden im Unterabschnitt zum Erstellen und Verwalten von Dokumenten viele Anforderungsmanagement-Tools vorgestellt, die das Tracing unterstützen. CASE Spec [91], Modern Requirements [93], Visure [92, 93], usw. (vgl. Abschnitt 6.5 für weitere Details). Die folgende Arbeit von Pablo Oliveira et al. [99] bietet mit der Technik TraceMan - Traceability Manager Approach ebenfalls einen passenden Mechanismus zur Sicherstellung der Traceability zwischen User Stories, traditionellen Anforderungsdokumenten, Testspezifikationen, Architekturdesign und Source Code. TraceMan ist ein werkzeugbasierter Ansatz zur Verwaltung der Traceability zwischen Artefakten, die in verschiedenen Werkzeugen erstellt wurden. Es erfordert keine komplette Änderung des Werkzeugsatzes der Entwicklungsumgebung, sondern es dient der zentralen Verwaltung und zur Erstellung von Trace-Links zwischen den Artefakten (vgl. [99], Seite 3-4).
Traceability für große Softwaresysteme ist aufgrund der exponentiell steigenden Anzahl an Traceability-Verbindungen nur mittels automatisierter Ansätze realisierbar, insbesondere in der industriellen Softwareentwicklung. Als solches automatisierte Traceability-Tool ist TraceMan als Add-in für das in der Industrie weit verbreitete Werkzeug Enterprise Architekt von Sparx Systems implementiert. TraceMan basiert auf in Rally dokumentierten User Stories und Spezifikationsdokumenten in Form von Textdokumenten mit der Endung .doc und ist somit z. B. mit Microsoft Word oder Oracle Open Office verknüpfbar (siehe Abbildung 6.2 und vgl. [99], Seite 5, B. User Stories und Test cases). Beide Artefakte können mittels TraceMan in einer einzigen Umgebung - Enterprise Architect - kombiniert werden (vgl. [99], Seite 4, A. Requirements Documentation and User Stories).
TraceMan setzt eine kontinuierlich erarbeitete Architektur Design Spezifikation voraus, die dem Entwicklungsteam als Leitfaden dienen und ebenfalls mit Enterprise Architect erstellt werden kann, jedoch ist auch ein Import aus anderen Tools möglich. Zudem können Verknüpfungen zwischen den User Stories und den Architekturelementen, die diese adressieren, erstellt werden, da die User Stories bereits in Enterprise Architect vorhanden sind (vgl. [99], Seite 6).
Das agile Manifest legt den Fokus auf ein funktionierendes System anstelle einer umfangreichen Dokumentation zu haben. Der Aufwand für eine gute Traceability soll sicherstellen, dass der Quellcode am Ende das System hervorbringt, das idealisiert und entworfen wurde. Folglich kann der Quellcode nicht verworfen werden, da die Traceability zwischen Entwicklungsartefakten im agilen Bereich im Fokus liegt. TraceMan nutzt die Möglichkeiten des Fraunhofer-Werkzeugs SAVE für die Traceability des Quellcodes, darunter eine Verifikationsmöglichkeit der Übereinstimmung zwischen geplanter und tatsächlicher Architektur, die aus dem Quellcode extrahiert wird. Da SAVE auf Eclipse läuft, erfordert der Vergleich zwischen Quellcode und Architektur einen speziellen Mechanismus (vgl. [99], Seite 6, D. Architecture and Source Code).

Abbildung 6.2: TraceMan importiert User Stories aus Rally in Enterprise Architect [99].

6.9 Product Owner bezogene Probleme

Für die folgenden in Tabelle 6.14 dargestellten Probleme in Tabelle 6.14 wurden im SLR-Prozess keine Lösungen vorgeschlagen, weshalb sie hier behandelt werden.

Tabelle 6.14: Tracing bezogener Probleme.

ID	Problem	Paper
1	Die Anwesenheit eines Vermittlers wie z. B. der PO	31,32
2	Herausforderungen der Qualitätsanforderungen bezüglich der Rolle des Product Owners	1,2,3,4

Die Anwesenheit eines Vermittlers wie z. B. der PO

In Abschnitt 5.1.10 wurde erwähnt, dass Product Owner Probleme bei der Kommunikation der Geschäftsanforderungen mit den Entwicklungsteams verursachen und dass kritische Informationen während der Übertragung von Informationen verloren gehen können. Bei mangelnder Kompetenz des Product Owners zu sicherer Kommunikation und Management kann dies zu großen Problemen im Projekt führen. Daher ist eine ausreichende Qualifikation der Person in der Rolle des Product Owners entscheidend.

Herausforderungen der Qualitätsanforderungen bezüglich der Rolle des Product Owners

Das Problem der Stakeholder-Qualifikation und des Wissensstandes kann auch dazu führen, dass QRs übersehen werden oder eine falsche Priorisierung von QRs erfolgt. Wie bereits in Abschnitt 6.6 erwähnt, können falsche Priorisierungen von Anforderungen durch die Definition der Rollen des Domain Owners und des Business Analysten gelöst werden. Außerdem muss der Product Owner über gute Kommunikationsfähigkeiten verfügen.

6.10 Koordinations- und Kommunikationsprobleme

Für die folgenden in Tabelle 6.15 dargestellten Probleme wurden im SLR-Prozess keine Lösungen vorgeschlagen. Deshalb werden sie hier behandelt.

Tabelle 6.15: Tracing bezogener Probleme.

ID	Problem	Paper
1	Kommunikationsfehler	7,25,34
2	Informieren und Synchronisieren zwischen Teams	18
3	Die Herausforderung unterschiedlicher geografischer Standorte im Agilen Requirements Engineering	22

Kommunikationsfehler

In Abschnitt 5.1.11 wurde erwähnt, dass die Kommunikationsausfälle hauptsächlich durch einen hohen Bedarf an Besprechungen und Dokumentation im Verhältnis zur verfügbaren Zeit verursacht werden. Kommunikationsfehler können aber auch durch allgemeinere Kommunikationsprobleme wie z. B. mangelnden offenen Dialog im Team oder mangelndes Wissen der Entwickler über Kommunikationsabläufe entstehen [7, 25, 34]. Probleme im Zusammenhang mit dem Zeitmanagement könnten möglicherweise mit den Methoden aus Abschnitt 5.1.3. und 6.3. gelöst werden. Probleme im Zusammenhang mit Wissen könnten mit den Lösungen aus Abschnitt 5.1.2 gelöst werden. Probleme, die mit dem Wissen der Entwickler über Kommunikationsprozesse zusammenhängen, könnten mit einer Schulung des Entwicklerteams in den Kommunikationsmethoden, die im Projekt verwendet werden, behoben werden.

Information und Synchronisierung zwischen Teams

In Abschnitt 5.1.11 wurden zwei Probleme in Bezug auf diese Herausforderung erwähnt: **(1)** Der Prozess der Information und Synchronisierung zwischen Teams ist schwierig und zeitaufwändig; **(2)** Bei großen Projekten ist die Synchronisierung trotz einer Aufteilung der Anforderungen im Projekt entweder durch eine externe Rolle (Team) oder lokal und innerhalb jedes Teams problematisch [18]. Zur Verbesserung der Synchronisation der Arbeitsabläufe sollten aufgrund der Komplexität der Kommunikation in großen Projekten Werkzeuge eingesetzt werden . Dadurch kann zudem eine Verbindung zwischen den beteiligten Teams erreicht werden, was wiederum die Abhängigkeiten innerhalb verschiedener Subsysteme bewerkstelligt und Informationen effizient ausgetauscht werden. Werkzeuge können somit bei der Fehlervermeidung helfen und darüber hinaus auch die Zeitplanung deutlich verbessern. Im Folgenden werden Arbeiten vorgestellt, die mögliche Lösungen beinhalten.

(1) Alqhtani et al. [100] stellen eine Lösung für eine effektive Kommunikation zwischen verteilten Entwicklungsteams und damit einen erfolgreiche Projektabschluss vor, indem sie administrative und technische Abläufe integrieren. Der Ansatz beinhaltet zwei Teile: **(I)** Einerseits wird zum Management von verteilten Teams eine Struktur aus einem Teamleiter, einem Kommunikationskoordinator, einem Mitglied des technischen Supports und fünf bis sieben Entwicklern in jedem Land vorgeschlagen. Dabei ist der Teamleiter für die Steuerung des Entwicklungsprozesses verantwortlich. Der Kommunikations-Koordinator regelt die Planung von Besprechungen zwischen den Teams, indem er einen geeigneten Zeitpunkt der Besprechung und geeignete Kommunikationsmedien auswählt. Das technische Support-Mitglied ist für die Durchführung erfolgreicher Meetings verantwortlich. Die Entwicklungsmitglieder sind für die Implementierung der Software verantwortlich. **(II)** Andererseits werden drei Kommunikations-Tools zur Verwaltung von Online-Meetings, Problemlösung und technischem Support sowie Messaging zwischen verteilten Teams vorgeschlagen und erläutert. Dazu gehören die Programme WebEx, TeamViewer und Gruppenmailing. Bei Meetings über WebEx können sich verteilte Entwicklungsteams treffen, zusammenzuarbeiten und effektiv austauschen, da alles, was sie benötigen, an einem Ort vereint ist. TeamViewer ermöglicht es einem Mitglied, sich in wenigen Sekunden mit einem Remote-Server oder PC auf der ganzen Welt zu verbinden,

sodass direkter Support unabhängig vom Standort möglich ist und die Probleme verteilter Teams gelöst werden können. Teammitglieder können auch das Gruppenmailsystem nutzen, um eine Nachricht an mehrere Personen zu senden und so die Kommunikation zu verbessern. Zudem werden regelmäßige Besprechungen der Teamleiter untereinander und mit ihren Teams während der Entwicklung empfohlen, um einen gemeinsamen Umfang Ziele festzulegen und somit verteilte Projekte erfolgreich abzuschließen [100].

(2) Schneider et al. [99] stellen einen Ansatz vor, der die FLOW-Distanz als Maß für die Kommunikationsintensität und die genutzten Kommunikationskanäle nutzt, um einen angemessenen Informationsfluss in Softwareentwicklungsteams und damit den Projekterfolg zu fördern. Ziel ist eine Verbesserung der teaminternen Kommunikation. Die FLOW-distanz ist ein einfach zu berechnendes Maß, das eine Erfassung jeder einzelnen Interaktion zwischen zwei oder mehreren Teammitgliedern überflüssig macht. Außerdem kann auf Basis der FLOW-Distanz die FLOW-Zentralisierung berechnet werden, die auf Methoden der sozialen Netzwerkanalyse zur Erkennung von teaminternen Strukturen angewendet werden kann. Mittels der FLOW-Distanz wird dem Projektleiter der Überblick über eventuell vorhandene soziale Konflikte und Stimmungen erleichtert, die den Informationsfluss und damit den Projekterfolg beeinflussen. Die Berücksichtigung der FLOW-Distanz kann den Projekterfolg in Bezug auf verschiedene Metriken unterstützen, u. a. auf die Anforderungserfüllung, Kundenzufriedenheit und die Stimmung im Team, indem unnötige Arbeit durch einen ausreichenden Informationsfluss vermieden wird.

Die Herausforderung unterschiedlicher geografischer Standorte im Agilen Requirements Engineering

In Abschnitt 5.1.11 wird erwähnt, dass die Teammitglieder aufgrund unterschiedlicher geografischer Standorte und funktionsübergreifender Teams immer zusätzliche Kommunikation benötigen (Batra und Bhatnagar [22]). Dieses Problem kann mit der im letzten Unterabschnitt zur Information und Synchronisierung zwischen Teams erwähnten Methode gelöst werden, indem Werkzeuge zur Unterstützung der Kommunikation und des Synchronisationsprozesses verwendet werden. Der Einsatz von Scrumconix (Portela et al. [98]), wie in Abschnitt 6.7 beschrieben wurde, könnte ebenfalls helfen, diese Herausforderung zu lösen.

Kapitel 7

Zusammenfassung und Ausblick

In diesem Kapitel werden die Vorgehensweise und Ergebnisse der Masterarbeit zusammengefasst.

7.1 Zusammenfassung

Der Fokus dieser Arbeit besteht in der Untersuchung der Herausforderungen des agilen Requirements Engineering (ARE) und deren Lösungen mittels agiler und traditioneller Methoden. In dieser Masterarbeit sollten die folgenden Forschungsfragen beantwortet werden:

(RQ1) Was sind die typischen Probleme des ARE und wie wirken sie sich auf den gesamten Prozess des RE aus?
(RQ2) Welche Lösungen in der Literatur sind bereits für die Probleme des ARE vorgeschlagen worden?

Um den Ist-Stand der Forschungen zu den Herausforderungen des agilen Requirements Engineering zu erörtern, wurde zu Beginn eine Systematischer Literaturrecherche (SLR) durchgeführt - wie von Kitchen et al. [46] empfohlen. Für die Optimierung des SLR-Prozesses wurde zusätzlich die Snowballing-Technik von Wohlin et al. [46] eingesetzt. Insgesamt wurden 31 Arbeiten im SLR-Prozess identifiziert, die sich auf Probleme und Herausforderungen des agilen Requirements Engineering beziehen. Das Ergebnis dieses SLR-Prozesses war eine Liste von 13 Herausforderungen, die als Over-Challenges bezeichnet werden. Jede dieser Over-Challenges klassifiziert ähnliche Herausforderungen unter sich. Darüber hinaus wurden bestehende Lösungen aus SLR-Prozessen zusammen mit diesen 13 Over-Challenges identifiziert. Nach der Analyse des Ist-Zustand aus bestehenden Forschungsarbeiten wurden in Kapitel 6 für bestehende Herausforderungen Lösungen aus agilen und traditionellen RE-Arbeiten identifiziert. Allerdings wurden für einige Herausforderungen in ARE keine Lösungen gefunden.
Am häufigsten wurden Probleme für Kunden, Benutzern und Stakeholdern in Zusammenhang mit der Dokumentation in der Literatur erwähnt und diskutiert. Eine Ursache hierfür ist die Betonung von direkter Kommunikation und einfacher Dokumentation in agilen Methoden mit Konsequenzen für das Verständnis und die Nachverfolgbarkeit. Viele Lösungen für diese Herausforderungen wurden jedoch im SLR-Prozess identifiziert und sind somit nicht als kritisch zu betrachten. Das Ergebnis des SLR-Prozesses zeigt, dass die meisten Herausforderungen bereits adressiert wurden und einige Lösungen für

diese Herausforderungen auch vorgeschlagen wurden. Einige Herausforderungen blieben allerdings bisher unbehandelt, darunter wurden am häufigsten Priorisierungsprobleme und die Qualitätssicherung von Anforderungen genannt und deshalb als kritisch identifiziert, da in der Literatur keine Lösungen für die Priorisierung von Anforderungen gefunden wurden und es nur wenige Lösungen für die Probleme im Zusammenhang mit der Qualitätssicherung von Anforderungen gab. Außerdem ist die Qualitätssicherung ein sehr wichtiger Teil sowohl des Requirements Engineering als auch des gesamten Software Engineering. Nach der Untersuchung für geeignete Lösungen in Bezug auf die Priorisierung wurden einige Lösungen identifiziert, jedoch muss eine praxisorientierte Untersuchung durchgeführt werden, um diese Lösungen zu bewerten. Außerdem wurde keine Lösung für die Re-Priorisierung von Anforderungen identifiziert. Die meisten Herausforderungen im Zusammenhang mit der Qualitätssicherung von Anforderungen beziehen sich auf die Verwendung formaler Prozesse im agilen Requirements Engineering. Für Herausforderungen im Zusammenhang mit dem Product Owner und der Koordination und Kommunikation im Team wurden ebenfalls keine Lösungen in SLR-Prozess identifiziert, jedoch scheinen diese nicht kritisch zu sein aufgrund ihrer seltenen Erwähnung in der Literatur.

Zudem wurden einige Lösungen für Probleme im Zusammenhang mit der Koordination und Kommunikation im Team vorgeschlagen. Ebenso gibt es für Probleme im Zusammenhang mit dem Product Owner einige Lösungen, die ursprünglich zur Überwindung anderer Herausforderungen empfohlen wurden, aber auch auf diese Herausforderung übertragbar sind. Qualitätsanforderungen waren ein weiteres Problem, das kritisch erscheint. Zwar wurden auch hier einige Lösungen aus dem SLR-Prozess erhalten, aber die meisten davon stammen von einem einzigen Autor, weshalb diese Herausforderung bisher erst wenig untersucht worden ist. Daher muss erforscht werden, ob andere Herausforderungen daraus entstehen können und wie weitreichend die Problematik in der Praxis ist. Im Zusammenhang mit Problemen des Anforderungsmanagements wurden mehrere Werkzeuge vorgestellt, aber die Effektivität dieser Werkzeuge sollte ebenfalls noch stärker in der Praxis erforscht werden. Diese Werkzeuge können auch eine Lösung für Tracing-Probleme sein, weshalb sie sehr vielversprechend sind. Für Probleme, die mit der Aufwandsschätzung zusammenhängen, werden auch einige Lösungen in der SLR vorgeschlagen, außerdem wurden einige Methoden außerhalb der SLR identifiziert, jedoch besteht noch Bedarf an Bewertungen dieser Lösungen. Einige Lösungen, die in Kapitel 6 identifiziert wurden, bestehen in hybriden Einsätzen von agilen und traditionellen Methoden, was effektiv gegen Kernprobleme des agilen Requirements Engineering helfen kann. Aber um eine effektive hybride Methode zu entwickeln, ist zusätzliche praxisorientierte Forschung und Evaluation notwendig.

7.2 Ausblick

Der nächste Schritt bildet die Untersuchung bestehender ungelöster Probleme, z. B. dem Fehlen eines formalen RE-Prozesses für die Aufwandsabschätzung, da ohne gute Zeit- und Budget-Abschätzungen der Projekterfolg riskiert wird und Kunden eine Vision für ihren Aufwand an Zeit und Geld benötigen. Kontinuierliche Repriorisierung sollte auch in Forschungen in Betracht gezogen werden, da dies zeitliche und betriebswirtschaftliche

Probleme verursachen kann. Fehlende formale Modellierung für detaillierte Anforderungserhebung und Anforderungsspezifikation ist auch wichtig und sollte ebenfalls erforscht werden. Dieses Problem hängt auch mit Herausforderung minimaler Dokumentation zusammen. Qualitätssicherung und Priorisierung von Anforderungen erfordern ebenfalls weitere Forschung. Die meisten Herausforderungen im Zusammenhang mit der Qualitätssicherung von Anforderungen beziehen sich auf die Verwendung formaler Prozesse im agilen Requirements Engineering, z. B. unzureichender Anforderungsverifizierung bzw. Testspezifikation wegen fehlender formalen Inspektionen in der agilen Methodik. Außerdem sollten praxisorientierte Arbeiten in Bezug auf die Unterstützung durch Werkzeuge und deren Wirksamkeit bei identifizierten Herausforderungen durchgeführt werden.

Literaturverzeichnis

[1] Alsaqaf, Wasim; Daneva, Maya; Wieringa, Roel (2017): Agile Quality Requirements Engineering Challenges: First Results from a Case Study, in: ESEM, S. 454–459.
[2] Alsaqaf, Wasim; Daneva, Maya; Wieringa, Roel (2018): Understanding Challenging Situations in Agile Quality Requirements Engineering and Their Solution strategies: Insights from a Case Study, in: IREC, S. 274–285.
[3] Alsaqaf, Wasim; Daneva, Maya; Wieringa, Roel (2017): Quality Requirements in Large-Scale Distributed Agile Projects – A Systematic Literature Review, in: International Working Conference on Requirements Engineering: Foundation for Software Quality, S. 219-234.
[4] Alsaqaf, Wasim; Daneva, Maya; Wieringa, Roel (2019): Quality requirements challenges in the context of large-scale distributed agile: An empirical study, in: Information and software technology, 2019, 110. Jg., S. 39-55.
[5] Batool, Asma; Motla, Yasir Hafeez; Hamid, Bushra; Asghar, Sohail; Riaz, Muhammad; Mukhtar, Mehwish; Ahmed, Mehmood (2013): Comparative study of traditional requirement engineering and Agile requirement engineering, in: 2013 15th International Conference on Advanced Communications Technology (ICACT), S. 1006-1014.
[6] Bjarnason, Elizabeth; Wnuk, Krzysztof; Regnell, Bjorn (2011): A Case Study on Benefits and Side-Effects of Agile Practices in Large-Scale Requirements Engineering, in: IWSPM, S. 30–39.
[7] Coutinho, Jarbele C. S.; Andrade, Wilkerson L.; Machado, Patrícia D. L. (2019): Requirements Engineering and Software Testing in Agile Methodologies, in: SBES 2019, S. 322–331.
[8] Elghariani, Kaiss; Kama, Nazri (2016): Review on Agile requirements engineering challenges, in: ICCOINS 2016, S. 507–512.
[9] Fatima, Tazeen; Mahmood, Waqas (2019): Requirement Engineering in Agile, in: IJEME 9 (4), S. 20–33.
[10] Neto, Francisco Gomes De Oliveira; ; Horkoff, Jennifer; Knauss, Eric; Kasauli, Rashidah; Liebel, Grischa (2017): Challenges of Aligning Requirements Engineering and System Testing in Large-Scale Agile: A Multiple Case Study, in: REW, S. 315–322.
[11] Heikkila, Ville T.; Damian, Daniela; Lassenius, Casper; Paasivaara, Maria (2015): A Mapping Study on Requirements Engineering in Agile Software Development, in: SEAA, S. 199–207.
[12] Inayat, Irum; Moraes, Lauriane; Daneva, Maya; Salim, Siti Salwah (2015): A Reflection on Agile Requirements Engineering: Solutions Brought and Challenges Posed, in: Scientific Workshop Proceedings of the XP2015, S. 1–7.
[13] Inayat, Irum; Salim, Siti Salwah; Marczak, Sabrina; Daneva, Maya; Shamshirband, Shahaboddin (2015): A systematic literature review on agile requirements engineering practices and challenges, in: Computers in Human Behavior 51, S. 885–929.
[14] Medeiros, Juliana D. R. V.; Alves, Daniela C. P.; Vasconcelos, Alexandre; Silva, Carla; Wanderley, Eduardo (2015): Requirements Engineering in Agile Projects: A Systematic Mapping based in Evidences of Industry, in: CibSE 2015, S. 460.
[15] Julius Okesola, Marion Adebiyi, Kennedy Okokpujie, David Odepitan, RowlandGoddy-Worlu, Olamma Iheanetu, Zacchaeus Omogbadegun and Ariyo Adebiyi (2019): A systematic review of requirement engineering practices in agile model, in: International Journal of Mechanical

Engineering and Technology (IJMET), 2019, 10. Jg., Nr. 2, S. 671-687.
[16] Camoglu, Kadir; Kandemir, Rembiye (2020): Comparison of requirements engineering knowledge areas in terms of traditional and agile software methods, in: Trakya Üniversitesi Mühendislik Bilimleri Dergisi, 2020, 20. Jg., Nr. 2, S. 79-91.
[17] Kapyaho, Marja; Kauppinen, Marjo (2015): Agile requirements engineering with prototyping: A case study, in: RE 2015, S. 334–343.
[18] Kasauli, Rashidah; Liebel, Grischa; Knauss, Eric; Gopakumar, Swathi; Kanagwa, Benjamin (2017): Requirements Engineering Challenges in Large-Scale Agile System Development, in: RE 2017, S. 352–361.
[19] Cao, Lan; Ramesh, Balasubramaniam (2008): Agile requirements engineering practices: An empirical study, in: IEEE Software, vol. 25, no. 1, S. 60-67. **[20]** Malik, M. Usman; Chaudhry, Nadeem Majeed; Malik, Khurram Shahzad (2013): Evaluation of Efficient Requirement Engineering Techniques in Agile Software Development, in: International Journal of Computer Applications, 83, S. 24-29.
[21] Fritzsche, M. (2008): Agile Methods and Requirements Engineering in Change Intensive Projects, in: ENASE 2008, S. 81-88.
[22] Batra, Mona, Bhatnagar, Archana (2019): A Research Study on Critical Challenges in Agile Requirements Engineering, in: Advances in Computer, Bd. 80, S. 1–44.
[23] Vithana, Nipunika (2015): Scrum Requirements Engineering Practices and Challenges in Offshore Software Development, in: IJCA 116, S. 43-49.
[24] Ramesh, Balasubramaniam; Lan Cao; Baskerville, Richard (2010): Agile requirements engineering practices and challenges: an empirical study, in: Information Systems Journal, 20: S. 449-480.
[25] Wagner, Stefan; Fernández, Daniel M.; Felderer, Michael; Kalinowski, Marcos (2017), Requirements Engineering Practice and Problems in Agile Projects: Results from an International Survey. ArXiv, abs/1703.08360.
[26] Saleh, Mohammed; Baharom, Fauziah; Mohamed, Shafinah F. P.; Ahmad, Mazida (2018): A Systematic Literature Review of Challenges and Critical Success Factors in Agile Requirement Engineering, in: 9th Knowl. Manag. Int. Conf. 2018, S. 248-254.
[27] Schön, Eva-Maria; Thomaschewski, Jörg; Escalona, María José (2017), Agile Requirements Engineering: A systematic literature review. Computer Standards und Interfaces 49, S. 79–91. DOI: 10.1016/j.csi.2016.08.011.
[28] Schön, Eva-Maria; Winter, Dominique; Escalona, María José; Thomaschewski, Jörg (2017), Key Challenges in Agile Requirements Engineering, in: Agile Processes in Software Engineering and Extreme Programming, Bd. 283, S. 37–51.
[29] Sebega, Y.; Mnkandla, E. (2017): Exploring Issues in Agile Requirements Engineering in the South African Software Industry, in: The Electronic Journal of Information Systems in Developing Countries, 81. Jg., Nr. 1, S. 1-18.
[30] Alam, Sehrish; Bhatti, Shahid N.; Shah, Asim A.; Jadi, Amr M. (2017): Impact and Challenges of Requirement Engineering in Agile Methodologies: A Systematic Review, in: IJACSA, 8. Jg., Nr. 4, S. 411-418.
[31] Thomas, Meetu; Senapathi, Mali (2019): Agile Requirements Engineering: An Empirical Analysis and Evidence from a Tertiary Education Context, in: IISIT 16, S. 97–112.
[32] Uludag, Omer; Kleehaus, Martin; Caprano, Christoph; Matthes, Florian (2018): Identifying and Structuring Challenges in Large-Scale Agile Development Based on a Structured Literature Review, in: 2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC). IEEE, 2018. S. 191-197.
[33] Gaikwad, Vandana; Joeg, Prasanna (2017): A case study in requirements engineering in context of agile, in: International Journal of Applied Engineering Research, 2017, 12. Jg., Nr. 8, S. 1697-1702.

[34] Wagner, Stefan; Méndez Fernández, Daniel; Kalinowski, Marcos; Felderer, Michael (2018): Agile Requirements Engineering in Practice: Status Quo and Critical Problems, in: CLEI Electronic Journal, 21. Jg., Nr. 1, S. 15.
[35] Agile Business Consortium, what is business Agility, https://www.agilebusiness.org/page/WhatisBusinessAgility, Stand 19.07.2020.
[36] Abid, Muhammad Ali; Din, Zia Ud; Khan, Muhammad Ijaz; Naeem, Tariq (2020): Factors affecting requirements engineering in agile software development: a systematic analysis, in: International Transaction Journal of Engineering, Management, Applied Sciences & Technologies,Vol. 11, Nr. 9.
[37] Kotonya, Gerald; Sommerville, Ian (1998): Requirements engineering: processes and techniques, John Wiley & Sons, Inc.
[38] Nuseibeh, Bashar; Easterbrook, Steve (2000): Requirements engineering: a roadmap, in: Proceedings of the Conference on the Future of Software Engineering, S. 35-46.
[39] Paetsch, Frauke; Eberlein, Armin; Maurer (2003): Requirements Engineering and Agile Software Development, in: Twelfth IEEE International workshop 2003, S. 308-313.
[40] Curcio, Karina; Navarro, Tiago; Malucelli, Andreia; Reinehr, Sheila (2018), Requirements engineering: A systematic mapping study in agile software development, in: Journal of Systems and Software.
[41] Zakari, Abubakar; Lawan, Ahmad;;Bekaroo, Girish (2017): A hybrid three-phased approach in requirement elicitation, in: ORCID, International Conference on Emerging Trends in Electrical, Electronic and Communications Engineering. Springer, Cham, 2016. S. 331-340.
[42] Elshandidy, Heba, and Sherif Mazen(2013): Agile and traditional requirements engineering: A survey, in: International Journal of Scientific &Engineering Research 4.9, S. 473-482.
[43] Rupp, Chirs;Pohl, Klaus (2015): Basiswissen Requirements Engineering, Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering: Foundation Level nach IREB-Standard. 4. überarbeitete Auflage, Heidelberg: dpunkt Verlag.
[44] Kitchen, Barbara; Charters, Hamand S. (2007): Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001. Keele University and Durham University Joint Report.
[45] SLR in Software Engineering, https://userpages.uni-koblenz.de/ laemmel/esecourse/slides/slr.pdf , Stand 19.07.2020.
[46] Wohlin, Claes (2014): Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th international conference on evaluation and assessment in software engineering. 2014. S. 1-10.
[47] Ochodek, Miroslaw; Kopczyńska, Sylwia; Nawrocki, Jerzy (2020): A Case Study on a Hybrid Approach to Assessing the Maturity of Requirements Engineering Practices in Agile Projects (REMMA), in: International Conference on Current Trends in Theory and Practice of Informatics. Springer, Cham, 2020, S. 689-698.
[48] Kumar, Manoj; Shukla, Manish; Agarwal, Sonali (2013): A Hybrid Approach of Requirement Engineering in Agile Software Development, in: 2013 International Conference on Machine Intelligence and Research Advancement. IEEE, 2013, S. 515-519.
[49] Ali, Haad; Ahmad, Abbas; Ahmed, Sheeraz: Hybrid Approach for Requirement Prioritization, in: International Journal of Scientific &Engineering Research Volume 9, Issue 12, S. 100-105.
[50] Shahmoradpour, Ardashir; Afraz, Soheil: A new approach in aspect- oriented requirement engineering for agile software development, in: International journal of research in computer applications and robotics, Vol.6, Issue 1, S. 20-26.
[51] Prenner, Nils; Unger-Windeler, Carolin; Schneider, Kurt (2020): How are Hybrid Development Approaches Organized?, in: ICSSP 2020 Proceedings of the International Conference on Software and System Processes, S.145–154.

[52] Wohlin, Claes; Runeson, Per; Höst, Martin; Ohlsson, Magnus C.; Regnell, Bjöorn; Wesslén, Anders (2000): Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, Norwell, MA, USA.
[53] Farid, Weam M.; Mitropoulos, Frank J. (2012): Novel lightweight engineering artifacts for modeling non-functional requirements in agile processes, in: 2012 Proceedings of IEEE Southeastcon, Orlando, FL, 2012, S. 1-7.
[54] Leffingwell, Dean (2011): Agile Software Requirements Lean Requirements Practices for Teams, Programs, and the Enterprise, Addison Wesley.
[55] De Lucia, Andrea, and Abdallah Qusef (2010): Requirements engineering in agile software development, in: Journal of emerging technologies in web intelligence 2.3, S. 212-220.
[56] Emmerich, Wolfgang (2011): Working with User Stories, in: Workshop on Agile Requirements Engineering, Lancaster, United Kingdom, July 25-29 2011.
[57] Dragicevic, Sridana; Celar, Stipe; Turic, Mili (2017): Bayesian network model for task effort estimation in agile software development, in: Journal of Systems and Software, 127, S. 109-119.
[58] Cigniti Technologies, https://www.cigniti.com/blog/benefits-of-colocated-teams-for-agile-software-testing/ (stand 08.12.2020)
[59] Sistla, Karthik; Sherry, Ashlin Paul; Manjula, R. (2016): A comparative study of collocated and distributed agile software development, in: IJAR 4 (10), S. 1142–1148.
[60] Koskela, Juha; Abrahamsson, Pekka (2004): On-Site Customer in an XP Project: Empirical Results from a Case Study, in: Software Process Improvement 3281, S. 1-11.
[61] Daneva, Maya; van der Veen, Egbert; Amrit, Chintan; Ghaisas, Smita; Sikkel, Klaas; Kumar, Ramesh et al. (2013): Agile requirements prioritization in large-scale outsourced system projects: An empirical study, in: Journal of Systems and Software 86 (5), S. 1333–1353.
[62] Araujo, Joao, and Joao Carlos Ribeiro (2005): , Towards an aspect-oriented agile requirements approach, Eighth International Workshop on Principles of Software Evolution (IWPSE'05). IEEE 2005. S. 140-143.
[63] Ge, Xiaocheng; Paige, Richard; Polack, Fiona; Chivers, Howard; Brooke, Phillip (2006): Agile development of secure web applications, in: Proceedings of the 6th international conference on Web engineering 2006, S. 305-312.
[64] Firesmith, Donald (2004): Generating Complete, Unambiguous, and Verifiable Requirements from Stories, Scenarios, and Use Cases, in: Journal of Object Technology 2005, Vol. 3, S. 27-40.
[65] Haugset, Borge; Stalhane, Tor (2012): Automated acceptance testing as an agile requirements engineering practice, in: 2012 45th Hawaii International Conference on System Sciences. IEEE, 2012, S. 5289-5298.
[66] Farid, Weam M. (2012): The NORMAP Methodology: Lightweight Engineering of Non-functional Requirements for Agile Processes, in: 2012 19th APSEC., S. 322–325.
[67] Chung, Lawrence; Nixon, Brian A.; Yu, Eric; Mylopoulos, John (2012): Non-functional requirements in software engineering (Vol. 5). Springer Science & Business Media.
[68] Farid, Weam M.; Mitropoulos, John (2012): NORMATIC: A visual tool for modeling Non-Functional Requirements in agile processes, in: 2012 Proceedings of IEEE Southeastcon, Orlando, FL, 2012, S. 1-8.
[69] Sedano, Todd; Paul Ralph; Cécile Péraire (2019): The product backlog, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), , S. 200-211.
[70] SONAR, an automated monitoring tool. https://www.sonarqube.org/, Stand 19.07.2020.
[71] Knaster, Richard; Leffingwell, Dean (2017): SAFe 4.0 distilled: applying the Scaled Agile Framework for Lean software and systems engineering. Addison-Wesley Professional, 2017.
[72] Larson, Erik W.;Gobeli, David H. (1987): Matrix Management: Contradictions and Insights, California Management Review, 29(4), S. 126–138.
[73] Taibi, Davide; Lenarduzzi, Valentina; Pahl, Claus; Janes, Andrea (2017): Microservices in agile software development, in: Proceedings of the XP2017 Scientific Workshops. 2017. S. 1-5.

[74] Duraisamy, Gunavathi; Atan, Rodziah (2013): Requirement traceability matrix through documentation for scrum methodology, in: Journal of Theoretical & Applied Information Technology, 2013, 52. Jg., Nr. 2, S. 154-159.
[75] Jeong, Serin; Cho, Heetae; Lee, Seonah (2018): Agile requirement traceability matrix, in: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings (ICSE '18), S. 187-188.
[76] Neil, Ernst; Borgida, Alexander; Jureta, Ivan; Mylopoulos, John (2014): Agile requirements engineering via paraconsistent reasoning, in: Information Systems 43, S. 100–116.
[77] Kannan, Vaishnavi; Basit, Mujeeb A.; Bajaj, Puneet; Carrington, Angela R.; Donahue, Irma B.; Flahaven, Emily L. et al. (2019): User stories as lightweight requirements for agile clinical decision support development, in: JAMIA 26, S. 1344–1354.
[78] Coelho, Evita; Basu, Anirban (2012): Effort Estimation in Agile Software Development using Story Points, in: International Journal of Applied Information Systems (IJAIS), 2012, 3. Jg., Nr. 7.
[79] Cohns, Mike (2004): User Stories Applied: For Agile Software Development. Addison Wesley., USA.
[80] Wake, Bill (2003): INVEST in Good Stories, and SMART Tasks. https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/ (Stand 21.12.2020).
[81] Coelho, Evita; Basu, Anirban (2012): Effort estimation in agile software development using story points, in: International Journal of Applied Information Systems (IJAIS), 2012, 3. Jg., .S. 7-10.
[82] Devnani-chulani, Sunita; Clark, Brad; Boehm, Barry (2000): Calibrating the COCOMO II post-architecture model, In: Proceedings of the 20th international conference on Software engineering. IEEE, 1998. S. 477-480.
[83] Litoriya, Ratnesh; Kothari, Abhay (2013): An Efficient Approach for Agile Web Based Project Estimation: AgileMOW, in: JSEA 06 (06), S. 297–303.
[84] Raslan, Atef; Ramadan, Nagy (2018): An Enhanced Framework for Effort Estimation of Agile Projects, in: International Journal of Intelligent Engineering and Systems 2018, 11. Jg., Nr. 3, S. 205-214.
[85] Tanveer, Binish (2016): Hybrid Effort Estimation of Changes in Agile Software Development, In: International Conference on Agile Software Development. Springer, Cham, 2016. S. 316-320.
[86] Ballejos, Luciana C.; Montagna, Jorge M. (2008): Method for stakeholder identification in interorganizational environments, in: Requirements Engineering 13 (4), S. 281–297.
[87] Power, Ken (2010): Stakeholder Identification in Agile Software Product Development Organizations: A Model for Understanding Who and What Really Counts, in: AGILE Conference, S. 87-94.
[88] Rupp, Chris; die SOPHISTen (2014): Requirements-Engineering und -Management. Aus der Praxis - von klassisch bis agil, 6. Auflage, Carl Hanser Verlag München.
[89] Bryson, John M. (2004): What to do when Stakeholders matter, in: Public Management Review 6 (1), S. 21–53.
[90] Die besten Anforderungsmanagement Tools 2020, https://thedigitalprojectmanager.com/de/anforderungsmanagement-tools/ (Stand 25.12.2020).
[91] Siddiqui, Shams; Bokhari, Mohammad; Shuaib, Mohammed (2018): Web Based Requirements Management Tools for Software Development: A Study, in: Proceedings of the 12th International Conference on "Computing for Sustainable Global Development", S. 4049-4053.
[92] Sophist Website, Requirements-Management, Requirements-Management-Tool, https://www.sophist.de/unsere-themen/requirements-management/ (Stand 26.12.2020)
[93] Guru99, 30 Best Requirements Management Tools in 2020, https://www.guru99.com/requirement-management-tools.html (stand 26.12.2020)

[94] Tamburrelli, Giordano; Margara, Alessandro (2014): Towards Automated A/B Testing, in: International Symposium on Search Based Software Engineering. Springer, Cham, 2014. S. 184-198.
[95] Borhan, Noor; Zulzalil, Hazura; Mohd Ali, Norhayati; Hassan, Sa'adah (2019): Requirements Prioritization Techniques Focusing on Agile Software Development: A Systematic Literature Review, in: International Journal of Scientific & Technology Research 8, S. 2118-2125.
[96] Achimugu, Philip; Selamat, Ali; Ibrahim, Roliana; Mahrin, Mohd Naz'ri (2014): A systematic literature review of software requirements prioritization research, in: Information and Software Technology 56 (6), S. 568–585.
[97] Rahim, Md Shamsur; Chowdhury, AZM Ehtesham; Das, Shovra (2017): Rize: A proposed requirements prioritization technique for agile development, in: 2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC), S. 634-637.
[98] Portela, Luis Tadeo; Borrego, Gilberto (2016): Scrumconix: Agile and Documented Method to AGSD, in: ICGSE 2016, S. 195–196.
[99] Antonino, Pablo Oliveira; Keuler, Thorsten; Germann, Nicolas; Cronauer, Brian (2014): A Non-invasive Approach to Trace Architecture Design, Requirements Specification and Agile Artifacts, in : ASWEC 23rd, S. 220–229.
[100] Alqhtani, Mashael Saeed; Qureshi, M. Rizwan Jameel (2014): A Proposal to Improve Communication between Distributed Development Teams, in IJISA 6 (12), S. 34–39.
[101] Klünder, Jil; Schneider, Kurt; Kortum, Fabian; Straube, Julia; Handke, Lisa; Kauffeld, Simone (2016): Communication in Teams - An Expression of Social Conflicts, in: Bogdan, Christian et al. (Hrsg.): Human-Centered and Error-Resilient Systems Development, Vol. 9856. Cham: Springer International Publishing, S. 111–129.

Abkürzungsverzeichnis

ASD	Agile Software-Entwicklung
ASE	Agile Software Engineering
ARE	Agile Requirement Engineering
ATDD	akzeptanztestgetriebene Entwicklung Methode
AHP	Analytischer Hierarchieprozess
ACRUM	Attributgetriebenem SCRUM
ALSD	großen verteilten agilen Projekten
AQUA	Analyze von Qualitätsattributen
AUC	Agile Use Case
ALC	Agile Loose Case
ALM	HP Application Lifecycle Management
ACC	Agile Choose Case
AK	Ausschluss Kriterien
BST	Binärer Suchbaum
CV	Kumulativer Stimmabgabe
DSDM	Dynamic Systems Development Method
DoD	Definition of Done
DFD	Data flow diagrams
ERD	Entity relationship diagram
EK	Einschluss Kriterien
EEEM	Frühzeitige Aufwandsschätzungsmodell
FD	Feature-Dokument
FDD	Feature-Driven Development
FBS	Feature Breakdown Struktur
FR	Funktionale Anforderungen
FLEX	Process Flexibility
FPA	Funktion Point Analysis
IP	Planungs-Iteration
JAD	Joint Application Development
OC	Over-Challenges
PB	Product Backlog
PBI	Product Backlog Item
PO	Product Owner
PREC	Skalierungsfaktoren gehören Precedentedness
QA	Qualität Sicherheit
QFD	Quality Funktional Deployment
RE	Requirement Engineering
RTM	Anforderungs-Traceability-Matrix
REST	Kombi nation von Requirements Engineering mit Software tests
RUP	Rational Unified Process
RQ	Forschungsfragen
RAM	Relation Assoziation Matrix
RESL	Risk Resolution
SE	Software Engineering
SAFe	Scaled Agile Framework
SLR	Systematic Literature Review
SLOC	Source Lines of Code
TRE	traditionelle Requirements Engineering
UI	User Interface
VOP	Wertorientierte Priorisierung

Abbildungsverzeichnis

Tabellenverzeichnis