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

Analyse der Balance von agilem und traditionellem Requirements Engineering in einer Fallstudie

Analysis of the balance of agile and traditional requirements engineering in a case study

Bachelorarbeit

im Studiengang Informatik

von

Ssonja Mieke

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

Hannover, 27.08.2020

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit 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 27.08.2020

Ssonja Mieke

Zusammenfassung

Der Prozess des Requirements Engineering stellt einen kritischen Faktor für den Erfolg eines jeden Projekts dar. Wie beim Erheben und Verwalten von Anforderungen vorgegangen wird, ist essentiell dafür, ob das Produkt am Ende der Entwicklung den Kundenvorstellungen entspricht. Dabei gehen Unternehmen selten nach Lehrbuch vor. Um effiziente und effektive Softwareentwicklung betreiben zu können, kombinieren sie traditionelle, d.h. planbasierte und agile Verfahren in hybriden Software-Entwicklungsverfahren. Ziel ist es, schnell und flexibel auf Änderungen reagieren zu können und gleichzeitig die Möglichkeit zu haben, langfristig zu planen. In dieser Arbeit wird untersucht, wie sich die Kombination von traditionellen und agilen Verfahren der Software-Entwicklung auf das Requirements Engineering auswirkt. Zu diesem Zweck wird eine Fallstudie durchgeführt, in der exemplarisch ein hybrides Projekt in einem Unternehmen untersucht wird. Die Ergebnisse zeigen, dass Anforderungen sowohl zu Projektbeginn erhoben werden, als auch im Verlauf des Projekts. Auf diese Weise kann innerhalb des Projekts ein gewisses Maß an Planungssicherheit erreicht werden und gleichzeitig besteht genug Flexibilität, um auf Veränderungen zu reagieren. Probleme ergeben sich dabei vor allem im Bereich der Dokumentation der Anforderungen, da Anforderungsdokumente kontinuierlich über den Projektverlauf wachsen. Dies erfordert viel Disziplin bei der Pflege der Dokumentation. Die hier durchgeführte Fallstudie trägt dazu bei, ein tieferes Verständnis für das Requirements Engineering in hybriden Software-Entwicklungsverfahren zu erlangen.

Abstract

The requirements engineering process is a critical factor for the success of any project. How to proceed in requirements elicitation and management is essential for the product meeting the customer's expectations in the end. Although hardly any company implements the used processes by the book. To develop software efficiently and effectively they combine traditional or plan-based and agile methods into hybrid software development approaches. The goal of these approaches is to respond to change fast and flexible, while having the possibility of long-term planning. This thesis explores how combining traditional and agile methods affects the requirements engineering process. For this, a case study is conducted, in which a hybrid project in a company is explored. The results show that requirements are both elicited in the beginning of a project and during the project itself. This way it is possible to gain planning reliability while having enough flexibility to respond to change. The most problems are found in the documentation of requirements since documents grow over the whole time of the project. It requires a lot of discipline to maintain these documentations. This case study contributes to acquiring a deeper understanding of requirements engineering in hybrid software development approaches.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das Erheben und Verwalten von Anforderungen in der Software-Entwicklung ist nicht nur notwendig, sondern stellt auch einen kritischen Faktor für den Erfolg eines jeden Projekts dar. So hängt das Scheitern von Projekten zu einem großen Teil davon ab, wie Anforderungen erhoben und verwaltet werden [30]. Dabei kann nach unterschiedlichen Modellen vorgegangen werden. Man unterscheidet in diesem Zusammenhang zwischen *traditionellen*, d.h. plan-basierten und *agilen* Verfahren.
Im *traditionellen Vorgehensmodell* wird viel Wert auf detaillierte Planung und Dokumentation gelegt. Es wird mit festen Anforderungen gearbeitet, die zu Beginn des Projekts ausführlich erfasst werden. Auf diese Weise ist schon zu Projektbeginn ersichtlich, welches Produkt entwickelt werden soll und weitere Schritte können dementsprechend geplant und durchgeführt werden. Dabei wird jeder Aktivitätsschritt im Software-Entwicklungsprozess dokumentiert. Es handelt sich also um ein phasen-orientiertes Vorgehensmodell, bei dem die einzelnen Schritte sequentiell aufeinander folgen. Ein Beispiel für ein solches Modell ist das Wasserfallmodell nach Royce, in dem die einzelnen Phasen der Software-Entwicklung nach und nach durchlaufen werden. [2]
Durch detaillierte Spezifikation zu Beginn des Entwicklungsprozesses kann so in der traditionellen Software-Entwicklung ein hohes Maß an Planungssicherheit erreicht werden. Anforderungen werden ausführlich erhoben, validiert und verifiziert. Allerdings ist die schnelle Lieferung von Software, d.h. bereits zur Laufzeit des Projekts, bei einem solch schwergewichtigen Ansatz nicht vorgesehen. Der Kunde wird zu Beginn des Projekts in die Erhebung der Anforderungen einbezogen und erhält am Projektende die fertig entwickelte Software. Ein konstantes Einbeziehen des Kunden während der Projektlaufzeit ist in der traditionellen Software-Entwicklung normalerweise nicht vorgesehen. [22]
Daraus können sich verschiedene Nachteile ergeben. Wurden zum Beispiel Anforderungen zu Projektbeginn falsch verstanden und werden dementsprechend nicht nach den Wünschen des Kunden umgesetzt, so wird dies erst

sehr spät, am Projektende, bemerkt. Je später ein solcher Fehler gefunden wird, desto teurer wird er [8]. Auch Änderungen in den Wünschen des Kunden, werden zur Projektlaufzeit in der Regel nicht bemerkt, da die traditionelle Software-Entwicklung keine Routinen vorsieht, um sich mit dem Kunden über den Stand der Entwicklungen auszutauschen. Diese Nachteile bestehen bei der *agilen Software-Entwicklung* nicht. In diesem Vorgehensmodell werden inkrementelle und iterative Verfahren genutzt, die den Kunden eng in die Entwicklung der Software miteinbeziehen. Grundlage ist das „Agile Manifesto“, das Werte und Prinzipien der agilen Software-Entwicklung festsetzt. In iterativen Zyklen erhält der Kunde die bereits entwickelte Software. Er kann so leicht entscheiden, ob das entwickelte Produkt seine Anforderungen erfüllt. Anforderungen können und dürfen sich also ändern und werden im Verlauf des Projekts für die jeweilige Iteration erhoben. [2, 22]
Die agile Software-Entwicklung bietet allerdings nicht nur Vorteile. So identifizieren Ramesh et al. [27] in ihrer empirischen Studie mehrere kritische Herausforderugen für die agile Software-Entwicklung. Darunter befinden sich Probleme wie etwa die Schätzung von Kosten und Aufwand oder das Vernachlässigen nicht-funktionaler Anforderungen. Diese werden in agilen Projekten oft wenig beachtet und in frühen Iterationszyklen meist ignoriert. Es ergeben sich also sowohl bei der traditionellen, als auch bei der agilen Herangehensweise spezifische Probleme.

1.1 Problemstellung

In der Praxis verbinden viele Unternehmen den Ansatz der traditionellen und der agilen Software-Entwicklung miteinander, um von den Vorteilen beider Verfahren zu profitieren. Aus dieser Verbindung entsteht der Ansatz der *hybriden Software-Entwicklung*. Er wird von Kuhrmann et al. [19, S.1 f.] wie folgt definiert:

> *„A hybrid software development approach is any combination of agile and traditional (plan-driven or rich) approaches that an organizational unit adopts and customizes to its own context needs (e.g., application domain, culture, processes, project, organizational structure, techniques, technologies, etc.).“*

Es handelt sich also um eine Kombination von agilen und traditionellen, d.h. plan-basierten Verfahren, die ein Unternehmen verwendet und so anpasst, dass das Verfahren auf die eigenen Bedürfnisse zugeschnitten ist.
Charakteristisch ist dabei, dass nicht nur Praktiken aus traditionellen und agilen Kontexten miteinander kombiniert werden, sondern auch Rollen aus beiden Bereichen existieren können. So kann beispielsweise ein klassischer Projektmanager neben einem Scrum Master vorhanden sein. [5]

Tell et al. [36] gehen davon aus, dass die Verwendung hybrider Modelle unabhängig von Unternehmensgröße und Industrie-Sektor mittlerweile die Norm geworden ist. Auch die Forschung anderer Autoren wie Kuhrmann et al. [19] stützt diese Aussage.
Obwohl viele Unternehmen agile und traditionelle Ansätze miteinander kombinieren, gibt es bisher nur wenig Forschung dazu, wie dies konkret geschieht und welche Kombinationen von Frameworks, Methoden und Praktiken eingesetzt werden [36]. Theocharis et al. [37] haben in diesem Zusammenhang eine Lücke in der aktuellen Literatur aufgedeckt. Um ein besseres Verständnis darüber zu gewinnen, wie hybride Ansätze in der Software-Entwicklung eingesetzt werden, ist das Sammeln weiterer Daten nach Meinung der Forscher unerlässlich. Viele Unternehmen, die nach einem hybriden Ansatz arbeiten, verfügen über einen stark individualisierten Prozess. Dieser ist in der Regel evolutionär, angepasst auf die Bedürfnisse des Unternehmens, entstanden. Die Unternehmen müssen also im Entstehungsprozess des hybriden Vorgehensmodells immer wieder entscheiden, welche Methoden für sie die richtigen sind und wie traditionelle und agile Ansätze miteinander kombiniert werden sollen. Dies stellt eine große Herausforderung dar. [25] Auch ist wenig darüber bekannt, warum hybride Ansätze der Software-Entwicklung in manchen Fällen zum Erfolg führen und in anderen Fällen scheitern [5]. Es ist also noch offen, was die spezifischen Probleme sind, die hybride Projekte scheitern lassen und was Faktoren dafür sind, dass ein solches Projekt zum Erfolg führt. Gerade im Bereich der Organisation hybrider Projekte gibt es erst wenige Fallstudien und Untersuchungen. Auch steht in diesen Studien oft nicht der gesamte Entwicklungsprozess im Mittelpunkt, sondern der Fokus liegt mehr auf dem Team. Es fehlt also an detaillierter Forschung auf diesem Gebiet, die Einblicke darin gewährt, wie hybride Entwicklungsmodelle umgesetzt werden. [25]

1.2 Lösungsansatz

Um weitere Einblicke zu gewinnen, wie agile und traditionelle Ansätze in hybriden Modellen miteinander kombiniert werden, wird in dieser Arbeit eine Fallstudie durchgeführt. Der Fokus liegt dabei auf dem Requirements Engineering, da das Erheben und Verwalten von Anforderungen ein kritischer Faktor für den Erfolg eines jeden Projekts ist [30].
In der Fallstudie wird der Entwicklungsprozess in einem hybriden Projekt in einem Unternehmen untersucht. Daraus werden Erkenntnisse abgeleitet, wie innerhalb des Projekts vorgegangen wird. Dabei steht im Fokus, welche Auswirkungen das hybride Vorgehen auf den Requirements Engineering Prozess hat. Es wird untersucht, warum sich das Unternehmen konkret für dieses Vorgehen entschieden hat und welches die spezifischen Probleme und Herausforderungen sind, die sich im Requirements Engineering durch den

gewählten Prozess ergeben.
Innerhalb der Fallstudie wird dabei nach der Grounded Theory vorgegangen (siehe Kapitel 3.3). Dieses Verfahren eignet sich vor allem für Bereiche, die noch nicht umfassend erforscht wurden, da es theoriegenerierend ist [33]. Somit ist es gut für eine Fallstudie im Bereich der hybriden Software-Entwicklung geeignet, da noch nicht umfassend geklärt ist, wie traditionelle und agile Verfahren kombiniert werden und warum manche Projekte scheitern und andere nicht [36, 5]. Die Daten werden in dieser Fallstudie durch Interviews mit den Projektbeteiligten gewonnen. Dafür wird das semi-strukturierten Interview eingesetzt. Diese Interview-Form bietet durch vorher festgelegte Fragen einen klaren Rahmen für das Interview, erlaubt es aber auch, flexibel auf das Gehörte einzugehen. Auf diese Weise können reichhaltige und umfassende Daten erhoben werden. [15]
Durch den hier beschriebenen Ansatz ist es möglich, einen Einblick darin zu gewinnen, wie Anforderungen in hybriden Projekten erhoben und verwaltet werden. Nur durch weitere Einblicke in hybride Projekte ist es möglich, ein umfassendes Verständnis dafür aufzubauen, was die konkreten Herausforderungen und Probleme solcher Projekte sind und welche Faktoren zum Erfolg eines hybriden Projekts führen.

1.3 Struktur der Arbeit

Diese Arbeit ist wie folgt strukturiert. In Kapitel 2 werden zunächst traditionelle, agile und hybride Software-Entwicklung voneinander abgegrenzt und der jeweilige Requirements Engineering Prozess beschrieben. In Kapitel 3 werden die in dieser Arbeit verwendeten Forschungsmethoden vorgestellt. In diesem Rahmen wird erläutert, was unter einer Fallstudie zu verstehen ist, was beim Durchführen von Interviews zu beachten ist und es wird die Grounded Theory Methodologie vorgestellt. Kapitel 4 beschreibt die Durchführung der Fallstudie. Zu diesem Zweck wird zunächst eine Fallbeschreibung gegeben. Daran schließen sich Einblicke in Datensammlung und -analyse an. In Kapitel 5 werden die Ergebnisse der Arbeit zusammengefasst und erläutert. Daran schließt sich in Kapitel 6 die Disskussion der Forschungsergebnisse an. In Kapitel 7 wird die Arbeit noch einmal in Bezug zu anderen Arbeiten gesetzt. Abschließend wird in Kapitel 8 eine Zusammenfassung und ein Ausblick gegeben.

Kapitel 2

Grundlagen

Das Erheben, Analysieren, Dokumentieren und Validieren von Anforderungen ist eine wichtige Tätigkeit innerhalb des Software-Entwicklungsprozesses, unabhängig davon nach welcher Entwicklungsmethode vorgegangen wird [12, 32]. Die im Requirements Engineering Prozess erhobenen Anforderungen stellen in diesem Zusammenhang einen entscheidenden Faktor für die Qualität der entstehenden Software dar. Empirische Studien zeigen, dass Fehler im Bereich der Anforderungserhebung innerhalb des Software Lebenszyklus im Vergleich zu anderen Fehlern relativ häufig vorkommen. Auch sind diese Fehler oft teuer und aufwändig in der Korrektur. [3, 32]
Ziel des Requirements Engineerings ist das Identifizieren, Modellieren, Kommunizieren und Dokumentieren von Anforderungen an ein System und den Kontext, in dem das System verwendet wird [23]. Eine Anforderung ist nach IEEE Standard [1] wie folgt definiert:

(1) Eine Bedingung oder eine Fähigkeit, die ein Benutzer zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt.

(2) Eine Bedingung oder Fähigkeit, die ein System oder Teilsystem erfüllen muss, um einen Vertrag, einen Standard, eine Spezifikation oder andere formell vorgegebene Dokumente zu erfüllen.

(3) Eine dokumentierte Repräsentation einer Bedingung oder Eigenschaft gemäß 1. oder 2.

Anforderungen können auf vielfältige Weise klassifiziert werden. Eine Möglichkeit zur Klassifizierung ist die Unterscheidung von funktionalen und nicht-funktionalen Anforderungen. Eine funktionale Anforderung beschreibt das Ergebnis des Verhaltens, das von einer Funktion des Systems bereit gestellt wird, d.h. es wird beschrieben, was das System tun soll. Nichtfunktionale Anforderungen schränken den Lösungsraum ein, indem sie zum Beispiel Vorgaben in Bezug auf Performanz oder Sicherheit des Systems machen. [3, 1]

Der Requirements Engineering Prozess sollte aus strukturierten, wiederholbaren Aktivitäten bestehen und sowohl den Engineering-, als auch den Management-Aspekt von Anforderungen berücksichtigen [3]. Typische Tätigkeiten im Requirements Engineering sind das Erheben, Analysieren und Abstimmen, Dokumentieren und Validieren von Anforderungen, sowie das Anforderungsmanagement [23].

2.1 Traditionelle Software-Entwicklung

Die traditionelle Software-Entwicklung umfasst eine Vielzahl von Vorgehensmodellen, wie beispielsweise das Wasserfall-Modell nach Royce oder das V-Modell. Bei diesen Modellen unterteilt sich der Lebenszyklus der Software-Entwicklung in der Regel in verschiedene Phasen, die sequentiell durchlaufen werden. Dabei nehmen Planung und Dokumentation eine zentrale Rolle ein. [20, 2]

2.1.1 Traditionelles Requirements Engineering

In der traditionellen Software-Entwicklung werden die Anforderungen vollständig zu Projektbeginn erhoben und dokumentiert. Ziel ist es eine umfassende Anforderungsspezifikation zu erstellen, die als Basis für gemeinsames Wissen dient. Auf diesem Dokument bauen alle weiteren Schritte der Software-Entwicklung auf. [12]
Es wird also bereits vor dem Systemdesign und der Implementierung eine fest definierte Menge an Anforderungen benötigt [23]. Da die Anforderungen in der traditionellen Software-Entwicklung nur zu Beginn des Projekts erhoben werden, ist dieses Vorgehensmodell nicht geeignet, wenn sich Anforderungen häufig ändern. Kumar et al. [20] geben weiterhin zu bedenken, dass das vollständige und detaillierte Erfassen aller Anforderungen zu Projektbeginn oft unrealistisch ist.

Innerhalb des traditionellen Requirements Engineering werden fünf Hauptaktivitäten unterschieden (siehe Abbildung 2.1): das Erheben, das Analysieren und Abstimmen, die Dokumentation, die Validierung und das Verwalten von Anforderungen. Die *Anforderungserhebung* dient dazu

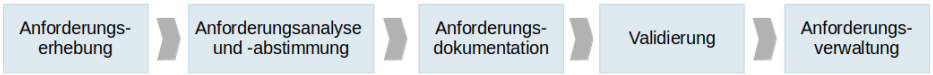

Abbildung 2.1: Phasen des Requirements Engineering

Anforderungen, die Systemgrenze und den Systemkontext zu identifizieren. Zu diesem Zweck ist ein enges Einbinden aller relevanten Stakeholder

notwendig. In der Phase der *Anforderungsanalyse und -Abstimmung* werden die Anforderungen auf Notwendigkeit, Konsistenz, Vollständigkeit und Realisierbarkeit untersucht. Konflikte und Inkonsistenzen zwischen den Anforderungen werden durch Abstimmung mit den beteiligten Stakeholdern aufgelöst. Die *Anforderungsdokumentation* dient dazu Anforderungen zwischen Stakeholdern und Entwicklern zu kommunizieren. Sie bildet die Basis für alle folgenden Schritte in der Software-Entwicklung und sollte eindeutig, vollständig, korrekt, verständlich, konsistent, präzise und realisierbar sein. Die *Validierung der Anforderungen* dient dazu sicher zu stellen, dass die Anforderungen eine akzeptable Beschreibung des zu implementierenden Systems darstellen. Die *Anforderungsverwaltung* schließt alle Aktivitäten in den Bereichen Änderungsmanagement, Versionierung und Rückverfolgbarkeit ein. Die Rückverfolgbarkeit von Anforderungen dient dazu, Beziehungen zwischen Anforderung, Design und Implementierung eines Systems zu identifizieren. [23]

2.1.2 Vor- und Nachteile

Das traditionelle Vorgehen innerhalb des Requirements Engineerings bringt sowohl Vor- als auch Nachteile mit sich. Zunächst einmal ist die traditionelle Software-Entwicklung gut für hochgradig kritische Projekte geeignet, in denen sich die Anforderungen nicht oder nur begrenzt ändern [20]. Bereits zu Projektbeginn ist eine langfristige und detaillierte Projektplanung möglich [25]. Weiterhin ist das traditionelle Vorgehen gut für große Projekte geeignet, in denen viele Entwickler beteiligt sind [20].

Oft ist es schwierig bereits zu Projektbeginn alle Anforderungen klar zu definieren. Dies kann verschiedene Gründe haben. Zum Beispiel kann es sein, dass Kunden nicht in der Lage sind, schon zu Projektbeginn genau zu spezifizieren, welche Eigenschaften das benötigte Produkt haben soll. [12]
Dies hat zur Folge, dass sich Anforderungen im Laufe des Projekts ändern. In traditionellen Software-Entwicklungsmodellen ist es schwieriger auf ständige Änderungen in den Kundenanforderungen zu reagieren. Auch sind Änderungen, wenn sie in einer späten Phase der Entwicklung auftreten, oft mit hohen Kosten verbunden. [20]
Ein weiterer Nachteil der traditionellen Software-Entwicklung ist der Mangel an schnellem Kundenfeedback. In der Regel erhält der Kunde das entwickelte Produkt erst am Projektende. Eine kontinuierliche Lieferung von Software an den Kunden verbunden mit ständigem Feedback sind nicht vorgesehen. [25]

2.2 Agile Software-Entwicklung

Im Vergleich zu traditioneller Software-Entwicklung ist die agile Software-Entwicklung weniger dokumentenzentriert und mehr Code-orientiert. Vertreter dieser Klasse von Vorgehensmodellen sind Scrum oder Extreme Programming (XP). Die agile Software-Entwicklung ist charakterisiert durch adaptive Planung und iterative Entwicklung. Auf diese Weise kann schnell und flexibel auf Veränderungen reagiert werden. Des Weiteren ist die agile Software-Entwicklung durch ein hohes Maß an direkter Kommunikation gekennzeichnet. Agile Teams arbeiten selbst-organisiert und der Mensch mit seinen spezifischen Fähigkeiten steht im Mittelpunkt. Ziel ist es, in kurzer Zeit Software von hoher Qualität zu erstellen. Um dies zu erreichen wird der Fokus weniger auf Dokumentation und mehr auf das kontinuierliche Einbeziehen des Kunden gelenkt. [20, 23]

2.2.1 Agiles Requirements Engineering

Für De Lucia et al. [12] liegt der größte Unterschied zwischen agilem und traditionellem Requirements Engineering darin, wann das Requirements Engineering stattfindet. Während die Anforderungen in traditionellen Software-Entwicklungsprozessen zu Projektbeginn erhoben werden, findet in der agilen Software-Entwicklung eine kontinuierliche Anforderungserhebung statt. Dabei gibt es kein einheitliches Vorgehen:

> *„The processes used for agile RE vary widely depending on the application domain, the people involved and the organization developing the requirements.“* [12, S. 214]

Ramesh et al. [27] identifizieren in ihrer empirischen Studie sechs agile Requirements Engineering Praktiken: (1) Zunächst einmal liegt der Fokus in der agilen Software-Entwicklung auf direkter Kommunikation. User Stories werden als Kommunikationsgrundlage zwischen Entwicklern und Kunde genutzt. Auf diese Weise wird der Fokus von geschriebener Dokumentation hin zu direkter Kommunikation gelenkt. (2) Des Weiteren werden Anforderungen in der agilen Software-Entwicklung nicht vordefiniert, sondern iterativ erhoben. Das heißt, dass die Anforderungen zu Projektbeginn nicht vollständig sind, sondern die initialen Anforderungen nur als Startpunkt für die Entwicklung dienen. In jeder Iteration werden neue Anforderungen erhoben. (3) Die Priorisierung der Anforderungen ist in der agilen Software-Entwicklung essentiell und wird in jeder Iteration neu vorgenommen. Dies ist vor allem wichtig, wenn es limitierte Ressourcen innerhalb des Projekts gibt. (4) Änderungen der Anforderungen werden durch konstante Planung ermöglicht. Dabei ist es Ziel der agilen Software-Entwicklung möglichst schnell und flexibel zu reagieren. (5) Eine weitere häufig genutzte Praktik

im Requirements Engineering ist das Prototyping. In der agilen Software-Entwicklung wird dabei das Entwickeln der wichtigsten Features in einem Prototyp, einer formalen Dokumentation vorgezogen. Der Prototyp kann als Kommunikationsgrundlage für den Austausch mit dem Kunden genutzt werden. (6) Die Validierung der Anforderungen wird in der agilen Software-Enticklung über häufige Reviews und Akzeptanztests sicher gestellt. Diese finden in der Regel am Ende einer jeden Iteration statt. [27, 13]

2.2.2 Vor- und Nachteile

Agile Software-Entwicklungsmethoden eignen sich auf Grund des iterativen Vorgehens gut für Projekte, in denen häufige Änderungen von Anforderungen auftreten [20, 4]. Durch adaptive Prozesse kann das Entwicklerteam schnell reagieren, um Änderungen auch in späten Phasen der Entwicklung zu berücksichtigen [12]. Durch das kontinuierliche Einbeziehen des Kunden wird sicher gestellt, dass die Wünsche des Kunden im Vordergrund stehen. Auch wird durch den iterativen Ansatz der agilen Vorgehensweise sicher gestellt, dass der Kunde möglichst schnell entwickelte Software erhält. [4]

Der mögliche Informationsverlust durch nicht ausreichende Dokumentation ist ein Hauptnachteil der agilen Vorgehensweise. Die einzige Form der Anforderungsdokumentation in agilen Projekten stellen User Stories oder Aufgabenbeschreibungen dar. Dies kann zu Problemen führen, wenn es beispielsweise große personelle Veränderungen innerhalb des Entwickerteams gibt, sich Anforderungen sehr schnell ändern oder die Komplexität der Anwendung sehr hoch ist. Des Weiteren kann die Kundenverfügbarkeit einen großen Nachteil für agile Requirements Engineering Praktiken bedeuten. So hängt die Effektivität der Kommunikation zwischen Kunde und Entwicklerteam stark davon ab, ob ein geeigneter Kunde verfügbar ist, ob Einigkeit unter den Kunden besteht und vom Vertrauen der Kunden in den Requirements Engineering Prozess. [27, 13]
Aufgrund der iterativ erhobenen Anforderungen ist die Budget- und Aufwandsschätzung in agilen Projekten problematisch. Zwar können die einzelnen Iterationen geschätzt werden, aber eine Schätzung des gesamten Projekts ist aufgrund des unklaren Projektrahmens nicht möglich. [27, 13, 12]
Eine weitere Herausforderung in agilen Projekten ist die Architektur der Software-Anwendung. Durch die iterativ erhobenen Anforderungen kann eine Architektur, die in einem frühen Iterationszyklus gewählt wird, unangemessen sein, wenn weitere Anforderungen hinzukommen. [27, 13]
Das Erheben nicht-funktionaler Anforderungen wie beispielsweise Sicherheit, Testbarkeit oder Benutzbarkeit wird in agilen Projekten oft vernachlässigt. Sie werden meist nicht oder unzureichend erhoben, da sie oft nicht im Bewusstsein des Kunden stehen. Die meisten nicht-funktionalen Anforderungen sollten aber zum Zeitpunkt der Entwicklung bekannt sein,

da ein spätes Aufkommen nicht-funktionaler Anforderungen zu einem erheblichen Mehraufwand führen kann. [27, 23, 13]

2.3 Hybride Software-Entwicklung

In der Praxis verbinden viele Unternehmen traditionelle und agile Verfahren miteinander um von den Vorteilen beider Ansätze zu profitieren. Der hybride Ansatz der Software-Entwicklung soll dabei sowohl Planungssicherheit für Kunden und Management bieten, als auch genug Flexibilität in Umsetzung und Entwicklung. Prozesse entstehen dabei meist evolutionär, d.h. durch Erfahrung und Anpassung über die Zeit. Welcher Prozess dabei für das jeweilige Unternehmen der richtige ist, hängt stark von den konkreten Zielen und dem Unternehmenskontext ab. [18, 36, 25]
Hybride Ansätze der Software-Entwicklung werden unabhängig von Unternehmensgröße und Industriesektor eingesetzt. Häufig werden in diesem Zusammenhang traditionelle Verfahren als Rahmenwerk genutzt, in das agile Praktiken integriert werden. Dabei werden Risiko- und Konfigurationsmanagement häufig traditionell umgesetzt, wohingegen Implementierung und Test eher agil verwirklicht werden. [19]

2.3.1 Organisation hybrider Vorgehensmodelle

Prenner et al. [26] identifizieren in ihrer systematischen Literaturrecherche drei Ansätze, nach denen hybride Software-Entwicklungsmodelle organisiert sein können: *Waterfall-Agile-Approach* (WAA), *Waterfall-Iterations-Approach* (WIA) und *Pipeline-Approach* (PA). Alle Ansätze basieren dabei auf dem Wasserfall-Modell nach Royce, unterscheiden sich aber darin, wie die einzelnen Phasen des Modells arrangiert werden und in welchem Ausmaß sie durchgeführt werden.

Der *Waterfall-Agile-Approach* (WAA) basiert auf sechs Phasen, die wie im Wasserfall-Modell nach Royce sequentiell angeordnet sind. Zunächst erfolgt die Anforderungsanalyse. In dieser Phase werden die Anforderungen für das gesamte Projekt auf high-Level Ebene erhoben. Danach folgt eine Designphase, in der die generelle Architektur festgelegt wird. Da die Anforderungen i.d.R. zunächst in verschiedenen Artefakten erfasst werden, erfolgt als nächster Schritt die Backlog Erstellung. Daraufhin findet die iterative Entwicklung des Produkts statt. In dieser Phase werden agile Entwicklungsmethoden genutzt. Es werden detaillierte Entscheidungen in Bezug auf die zuvor erhobenen Anforderungen und das Design getroffen. Auf die Entwicklung folgt die Testphase, in der das Produkt umfassend getestet wird, bevor es schließlich in der letzten Phase in die operative Nutzung übergeht.

Im *Waterfall-Iterations-Approach* (WIA), der oft mit dem WAA kombiniert wird, wird ein iterativer Ansatz als Prozessgrundlage verwendet. Innerhalb der einzelnen Iterationen wird das Wasserfall-Modell genutzt. Die einzelnen Phasen unterscheiden sich dabei nur dadurch von den Phasen des WAA, dass die Backlog Erstellung entfällt. Das Backlog existiert in diesem Ansatz bereits und wird in jeder Iteration ergänzt.

Auch der *Pipeline-Approach* (PA) folgt dem klassischen Wasserfall-Modell, allerdings werden die einzelnen Phasen in diesem Ansatz nicht sequentiell durchlaufen, sondern parallel für unterschiedliche Inkremente. Das bedeutet, dass sich ein Inkrement beispielsweise in der Designphase befindet, während ein anderes Inkrement zur gleichen Zeit entwickelt wird. Auch beim PA entfällt die Phase der Backlog Erstellung, da das Backlog wie im WIA von Iteration zu Iteration ergänzt wird. Die Entwicklungsphase kann innerhalb des PA in verschiedene kürzere Iterationen unterteilt werden.

2.3.2 Hybrides Requirements Engineering

Während es sich bei Anforderungen in der agilen Software-Entwicklung i.d.R. um informelle, priorisierte User Stories handelt, die unvorhersehbaren Veränderungen unterliegen, werden Anforderungen in der traditionellen Software-Entwicklung geplant zu Projektbeginn erhoben [6]. Die Unterschiede im Vorgehen können Schwierigkeiten dabei verursachen, wenn agile und traditionelle Ansätze miteinander kombiniert werden. So kann die Verwendung agiler Methoden im Requirements Engineering beispielsweise dazu führen, dass Anforderungen nicht ausreichend validiert und verifiziert werden können, wie es in der traditionellen Software-Entwicklung üblich ist. Boehm und Turner [7] schlagen in diesem Zusammenhang vor agile Anforderungen zu stärken, in dem sie mit weiteren Informationen versehen werden. In diesen weiteren Informationen können zum Beispiel nicht-funktionale Anforderungen wie Zuverlässigkeit oder Sicherheit berücksichtigt werden. [7]

Des Weiteren gibt es keinen einheitlichen Zeitpunkt zu dem Anforderungen in hybriden Vorgehensmodellen erhoben werden. Je nach spezifischem Ansatz (siehe Kapitel 2.3.1) können Anforderungen entweder zu Projektbeginn erhoben werden, oder iterativ im Verlauf des Projekts. Dabei verfügen die meisten hybriden Vorgehensmodelle über eine initiale Anforderungsanalyse und eine initiale Designphase. Diese werden mit einer kontinuierlichen Anforderungsanalyse und kontinuierlichem Design in der Implementierungsphase verbunden. Unterschiede bestehen zwischen verschiedenen hybriden Ansätzen im Ausmaß der Detaillierung und der Häufigkeit der Phasen innerhalb des Modells.
In der Phase der initialen Anforderungsanalyse werden dabei i.d.R. generelle

und allgemeine Anforderungen für das Projekt erhoben. Viele Projekte, in denen nach einem hybriden Vorgehensmodell gearbeitet wird, verfügen über einen Product Owner oder ein Product Owner Team, das für den Requirements Engineering Prozess verantwortlich ist. Auch nutzt die Mehrheit der bisher untersuchten Projekte User Stories oder Use Cases um ein Backlog zu erstellen. Es konnte aber auch beobachtet werden, dass Anforderungen zunächst in einer Anforderungsspezifikation erfasst wurden und zu einem späteren Zeitpunkt in User Stories umgewandelt wurden. Die Implementierung der Anforderungen wird i.d.R. agil vorgenommen. Dabei werden für jede Iteration Anforderungen ausgewählt, die umgesetzt werden sollen. Falls nötig wird für diese Anforderungen nochmals eine Anforderungsanalyse vorgenommen. [26]

Kapitel 3

Vorstellung der verwendeten Forschungsmethoden

In diesem Kapitel werden die in dieser Arbeit verwendeten Forschungsmethoden näher erläutert. Zunächst wird dargestellt, was unter einer Fallstudie zu verstehen ist. Danach wird beschrieben, wie innerhalb der Fallstudie vorgegangen wird. Zu diesem Zweck wird zunächst ausgeführt, wie mittels Interviews Daten gewonnen werden. Danach wird die *Grounded Theory*-Methodologie vorgestellt und erörtert.

3.1 Fallstudie

Die Fallstudie ist eine wichtige Forschungsmethode im Bereich des Software Engineering und auf viele Fragestellungen in diesem Gebiet anwendbar [16]. Dabei handelt es sich bei diesem Ansatz um eine empirische Forschungsmethode mit dem Ziel, zeitgenössische Phämomene und deren Kontext zu untersuchen. [29]

3.1.1 Ziel der Fallstudie

Die Fallstudie dient in der Regel einem explorativen Zweck, d.h. es werden Daten in einer realen Umgebug gesammelt und analysiert. Diese Daten sollen es ermöglichen, neue Einblicke zu gewinnen und weitergehende Ideen und Hypothesen zu formen. Dabei handelt es sich meist, aber nicht ausschließlich, um qualitative Daten, da diese reichhaltiger und weitergehender sind als quantitative Daten. Ziel der qualitativen Forschung ist also das Sammeln und Analysieren nicht-numerischer Daten mit dem Ziel tiefere Einblicke zu gewinnen.
Eine Fallstudie beinhaltet drei Hauptcharakteristiken: (1) Es handelt sich zunächst um einen flexiblen Ansatz, der es ermöglicht komplexe und dynamische Sachverhalte der realen Welt zu analysieren. Durch diese

Anpassungsfähigkeit ist es möglich, eine Fallstudie in unterschiedlichen Bereichen einzusetzen. (2) Des Weiteren sollen Schlussfolgerungen auf Basis klarer Beweisketten gezogen werden. Grundlage dafür sind Daten, die in einer geplanten und konsistenten Weise aus verschiedenen Quellen gesammelt wurden. (3) Die Fallstudie basiert dabei entweder auf einer bereits existierenden Theorie oder generiert neue Theorien. [29]

3.1.2 Vorgehen innerhalb der Fallstudie

Innerhalb der Fallstudie gibt es kein fest vorgeschriebenes Vorgehen zum Sammeln und Analysieren von Daten. Höst et al. [16] stellen in diesem Zusammenhang fest, dass es gerade im Bereich des Software Engineering keine spezifischen Anleitungen gibt, wie eine Fallstudie in diesem Bereich durchzuführen ist. Es gibt generelle methodische Fachliteratur, die aber auf den Bereich des Software Engineering angepasst werden muss. Die Autoren schlagen aus diesem Grund eine Reihe von Checklisten vor, mit denen Fallstudien im Bereich Software Engineering durchgeführt und evaluiert werden sollten.
Bei einer Fallstudie sollte darauf geachtet werden, dass unterschiedliche Datenquellen herangezogen werden. Dies können zum Beipiel verschiedene Interviewpartner, Dokumente oder Feldbeobachtungen sein. Auf diese Weise wird sichergestellt, dass der zu untersuchende Gegenstand möglichst vollständig erfasst wird. Auch können so verschiedene Sichtweisen auf ein bestimmtes Problem betrachtet werden. In der hier durchgeführten Fallstudie sollen die Erkenntnisse hauptsächlich über Interviews gewonnen werden. [29]

Da die Fallstudie lediglich ein Rahmenwerk bildet und kein festes Vorgehen vorgibt, wie genau Daten erhoben und analysiert werden sollen, ist es im Rahmen dieser Arbeit nötig, eine Forschungsmethode zu finden, die innerhalb der Fallstudie angewendet werden kann. Es wird die *Grounded Theory* verwendet. Dieser Ansatz ist aus verschiedenen Gründen gut geeignet. Zunächst einmal handelt es sich bei der Grounded Theory um ein theoriegenerierendes Verfahren, d.h. der Fokus liegt nicht auf der Validierung bereits existierender Theorien, sondern es sollen neue Theorien und Konzepte entdeckt und entwickelt werden [33]. Das Verfahren eignet sich also vor allem für Bereiche, die noch nicht umfassend erforscht wurden. Dies ist gerade für den Bereich der hybriden Software-Entwicklung der Fall, da bisher nur wenig darüber bekannt ist, wie traditionelle und agile Verfahren miteinander kombiniert werden und warum manche Projekte scheitern und andere nicht [36, 5].
Das Requirements Engineering ist, wie die Software-Entwicklung selbst, ein

interaktiver Prozess. Es kann mit Hilfe der Grounded Theory untersucht werden, wie Menschen in der Software-Entwicklung zusammenarbeiten, da die Grounded Theory gut dazu geeignet ist soziale Interaktion und soziales Verhalten zu erforschen. [24, 14]
Weiterhin bietet die Grounded Theory ein klar definiertes Vorgehen zur Datensammlung und Datenanalyse. Es werden beispielsweise Verfahren, wie das *theoretical sampling* (siehe Kapitel 3.3.3), das die Grundlage für die Datenerhebung und -Analyse bildet oder verschiedene Coding-Verfahren eingesetzt. Auf diese Weise stellt die Grounded Theory eine systematische Forschungsstrategie dar, bei der es möglich ist, qualitative Forschung auf ein sicheres Fundament zu stellen. [21]

Die Grounded Theory eignet sich also aufgrund ihres explorativen Charakters zur Anwendung in einer Fallstudie und bietet gleichzeitig die Möglichkeit auf eine definierte Weise Daten zu sammeln und umfassend zu analysieren. Die theoriegenerierende Eigenschaft der Methode bietet einen zusätzlichen Vorteil für das in dieser Arbeit betrachtete Forschungsgebiet.

3.2 Interview

Interviews sind eine häufig verwendete Form der Datensammlung innerhalb der qualitativen Forschung. Im Forschungsgebiet des Software Engineering werden sie eingesetzt, um Daten zu erheben, die quantitativ nicht oder nur schwer gemessen werden können. Dabei handelt es sich in der Regel um Meinungen, Gedanken, Gefühle und Eindrücke der interviewten Probanden. Die Interviews geben dabei einen Einblick in die subjektive Erlebenswelt der Befragten. [15]

3.2.1 Varianten von Interviews

Man unterscheidet drei Arten von Interviews: *strukturierte Interviews*, *unstrukturierte Interviews* und *semi-strukturierte Interviews*. Im strukturierten Interview wird mit den gestellten Fragen ein spezifisches Ziel verfolgt. Dementsprechend werden in dieser Form des Interviews vorwiegend geschlossene Fragen gestellt. In Tabelle 3.1 wird beispielhaft gezeigt, wie geschlossene und offene Fragen formuliert werden können. Die Fragen werden im strukturierten Interview vorab geplant und in einer zuvor festgelegten Reihenfolge während des Interviews abgearbeitet. Je strukturierter ein Interview ist, desto quantifizierbarer sind die daraus gewonnenen Daten. In qualitativen Studien muss das Interview jedoch flexibel genug sein, um auch auf unvorhergesehene Informationen eingehen zu können.
Dies ist beim unstrukturierten Interview der Fall. In dieser Interviewform wird der Befragte als Quelle von Antworten und Fragen gleichermaßen genutzt. Fragen werden so offen wie möglich formuliert. Im Extremfall

Interviewfragen	
geschlossene Frage	Wie viele Jahre arbeiten Sie bereits in diesem Unternehmen?
offene Frage	Wenn Sie eine Anforderung erhalten und bearbeiten sollen, wie verfahren Sie dann?

Tabelle 3.1: verschiedene Möglichkeiten der Fragestellung

werden gar keine Fragen gestellt, sondern der Proband bekommt lediglich das Thema des Interviews mitgeteilt. Auf diese Weise können vielfältige Informationen zum gewählten Themengebiet gewonnen werden. Das Interview selbst erhält durch die Art der Fragestellungen einen narrativen Charakter. Das unstrukturierte Interview ist allerdings nur schwer reproduzierbar, da die Fragen sehr offen gestellt werden.
Das semi-strukturierte Interview vereint die Vorteile von strukturiertem und unstrukturiertem Interview. Durch eine Kombination von geschlossenen und offenen Fragen können sowohl absehbare, als auch unerwartete Informationen erhoben werden. Die Interviewfragen bilden dabei einen Leitfaden für das Interview, müssen aber nicht zwangsläufig in der vorab festgelegten Reihenfolge abgearbeitet werden. Auf diese Weise bietet das semi-strukturierte Interview die Möglichkeit zur Improvisation und Exploration. [15, 31, 29]

3.2.2 Aufbau eines Interviews

Ein Interview besteht typischerweise aus verschiedenen Abschnitten. Runeson et al. [29] unterscheiden in diesem Zusammenhang vier Phasen. Das Interview beginnt mit der *Einleitung*. In diesem Abschnitt wird zunächst das Ziel des Interviews bzw. der Fallstudie vorgestellt. Dabei sollte beachtet werden, dass der Befragte weder zu viele, noch zu wenige Informationen erhält. Seaman [31] geht davon aus, dass der Interviewte das Ziel der Studie kennen sollte und zustimmen sollte, dass es ein wichtiges Ziel ist, um möglichst umfangreiche Informationen zu erhalten. Gleichzeitig besteht aber die Gefahr, dass der Befragte, wenn er zu viele Informationen vorab erhält, Informationen auslässt, weil er denkt, sie seien nicht von Interesse und seine Antworten so filtert. In der Einleitung sollte dem Befragten außerdem versichert werden, dass seine Angaben vertraulich behandelt werden. Dies dient dazu, dem Probanden die Möglichkeit zu geben offen und ehrlich zu antworten und der Gefahr vorzubeugen, dass sozial erwünscht geantwortet wird. Falls eine Audio-Aufnahme des Interviews erfolgen soll, muss der Befragte an dieser Stelle um Erlaubnis gebeten werden. [15]
Nach der Einleitung folgen *einführende Fragen*. Diese dienen wie auch die Einleitung zuvor dazu, Vertrauen zwischen Befragtem und Interviewer aufzubauen. Die einführenden Fragen sollten einfach zu beantworten sein und beschäftigen sich häufig mit dem Hintergrund des Befragten.

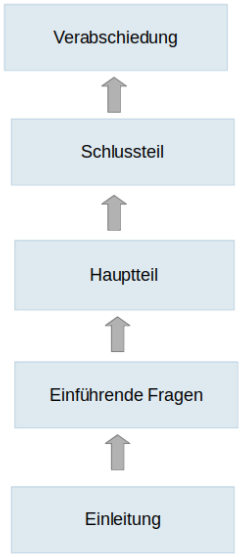

Abbildung 3.1: Interviewphasen

Im dritten Teil, dem *Hauptteil*, können auch schwierigere, persönliche oder sensible Fragen gestellt werden. Dieser Abschnitt bildet den größten Teil des Interviews. Wichtig ist es, dass dem Befragten versichert wird, dass seine Antworten vertraulich behandelt werden. Nur so kann gewährleistet werden, dass der Interviewte möglichst offen und ehrlich antwortet.
Im *Schlussteil* des Interviews können noch einmal wichtige Erkenntnisse zusammen gefasst werden. [29] In der hier vorliegenden Arbeit wird den Probanden im Schlussteil zusätzlich die Möglichkeit gegeben, wichtige Aspekte in ihren Ausführungen zu ergänzen, falls dies gewünscht ist. Abblidung 3.1 zeigt die vier beschriebenen Interviewphasen. Es wird ergänzend noch die Phase der *Verabschiedung* hinzugefügt, in der dem Interviewten für seine Mitwirkung gedankt wird.

3.2.3 Durchführung von Interviews

Bei der Durchführung von Interviews sollten verschiedenste Aspekte beachtet werden. Zunächst einmal sollten Interviewtermine möglichst frühzeitig vereinbart werden. Auf diese Weise kann flexibel auf mögliche Änderungen reagiert werden. Der Interviewende sollte weiterhin gut auf das Interview vorbereitet sein. Im besten Fall sollte ein Pilot-Interview durchgeführt werden, um die Interviewfragen zu testen. Dies gibt dem Interviewenden weiterhin die Möglichkeit, das Interview „zu üben“. Das kann vor allem sinnvoll sein, wenn der Interviewende nicht so viel Erfahrung im Führen von Interviews hat. [15]

Die Subjekte, die während des Interviews befragt werden, sollten mit großer Sorgfalt ausgewählt werden. Dabei muss darauf geachtet werden, dass die Befragten vor allem auf Basis ihrer Unterschiede gewählt werden. Es sollten möglichst verschiedene Rollen und Persönlichkeiten in die Interviews miteinbezogen werden. Auf diese Weise können Interviews ein umfassendes und ausführliches Bild liefern. [29]

Während des Interviews sollte das Vertrauen des Befragten gewonnen werden. Dies kann zum einen dadurch erreicht werden, dass dem Probanden versichert wird, dass seine Angaben vertraulich behandelt werden, zum anderen sollte dem Befragten genau erklärt werden, wie die erhobenen Daten verwendet werden. Auch sollte dem Probanden nach dem Interview für seine Mitwirkung gedankt werden. [15]

Eine Möglichkeit den Erkenntnisgewinn aus Interviews zu unterstützen sind Ton-, oder falls nötig, Videoaufnahmen. Aufnahmen vermeiden Informationsverlust. Für den Interviewenden ist es eine große Herausforderung gleichzeitig zuzuhören und das Gehörte zu notieren. In diesem Kontext sind Aufnahmen sinnvoll. Sie ermöglichen es dem Interviewenden mehr Aufmerksamkeit auf den Befragten und das Interview zu lenken. [31, 15]
Die anschließende Transkription des Interviews sollte nach Runeson et al. [29] dabei immer vom Interviewer selbst vorgenommen werden, da bei der Transkription möglicherweise neue Einblicke gewonnen werden können. Auch kann das Gehörte so besser in den Kontext des Interviews eingeordnet werden.

Die Qualität des Interviews hängt maßgeblich von den Fähigkeiten des Interviewenden ab. Dieser sollte nicht über die Antworten seines Gegenübers urteilen und einfühlsam mit dem Befragten umgehen. Dabei sollte der Proband ermutigt werden frei zu sprechen. Es sollte klargestellt werden, dass es während des Interviews keine „richtigen“ und „falschen“ Antworten gibt. Es sollte also ein respektvoller Umgang mit dem Probanden gepflegt werden. Weiterhin sollte der Interviewende gut zuhören können und in der Lage sein, seine Fragen klar auszudrücken. Der Befragte sollte nie abrupt oder unhöflich unterbrochen werden. Vielmehr sollte er, falls seine Antworten zu weit vom eigentlichen Thema des Interviews abweichen, sanft wieder zurück zum Thema geführt werden. Falls ein Befragter innerhalb des Interviews zu wenig sagt, empfiehlt Seaman [31] Fragen zu stellen, die nicht mit einem einfachen „ja“ oder „nein“ beantwortet werden können. Auch das Fragen nach bereits bekannten Details kann in diesem Zusammenhang helfen einen Redefluss herbeizuführen. [31, 15]

3.3 Grounded Theory

Bei der *Grounded Theory* handelt es sich um eine qualitative Forschungsmethode, die ursprünglich von Glaser und Strauss in ihrem Buch „The Discovery of Grounded Theory“ (1967) eingeführt wurde. Die Forschungsmethode stammt aus der Sozialforschung, findet aber mittlerweile auch in vielen weiteren Forschungsgebieten, wie beispielsweise

der Psychologie oder Anthropologie Anwendung. [21, 34]
Auch im Bereich des Software Engineering wird die Grounded Theory aufgrund ihrer induktiv Theorie-generierenden Eigenschaften immer häufiger eingesetzt [33]. So gibt es beispielsweise Arbeiten von Coleman et al. [9] oder auch Hoda et al. [14], die diesen Ansatz verwenden. Hoda et al. haben in ihrer Arbeit weiterhin gezeigt, dass die Grounded Theory als qualitative Forschungsmethode für den Bereich des Software Engineering geeignet ist und liefern eine detaillierte Beschreibung, wie das Verfahren angewendet werden kann. Die hier vorliegenden Arbeit orientiert sich sowohl an bestehenden Arbeiten zur Grounded Theory im Bereich des Software Engineering, als auch an weiterführender Literatur.

Die Grounded Theory wird von Strauss und Corbin [34, S. 273] wie folgt definiert:

> *„Grounded Theory is a general methodology for developing theory that is grounded in the data systematically gathered and analyzed. Theory envolves during actual research, and it does this through continuous interplay between analysis and data collection. “*

Das Ziel der Grounded Theory ist es also auf Grundlage von gesammelten Daten eine Reihe von zusammenhängenden Hypothesen zu bilden, aus denen eine Theorie generiert wird. Dabei enthält die Grounded Theory die folgenden drei Schlüsselelemente: (1) theoretical sampling, (2) verschiedene Kodierungsverfahren und (3) memoing. Das *theoretical sampling* bildet die Grundlage für die Datenerhebung und -Analyse. Innerhalb der *constant comparison*-Methode werden verschiedene coding-Verfahren eingesetzt, um Daten auf steigenden Abstraktionsleveln miteinander zu vergleichen. Das Verfassen von *Memos* dient dazu, den Reflextionsprozess zu unterstützen. [21] In Kapitel 3.3.3 werden die Schlüsselelemente der Grounded Theory näher erläutert.

3.3.1 Risiken der Grounded Theory

Die Anzahl der Studien, in denen als Forschungsmethode die Grounded Theory verwendet wird, wächst stetig. Auch im Bereich des Software Engineering erfreut sie sich zunehmend größerer Beliebtheit. [33]
Damit gehen jedoch auch Risiken einher. Lueger [21] sieht die Gefahr in der Anwendung der Methode darin, dass sie in Studien als „methodisches Gütesiegel “ verwendet wird, ohne dass die zugrunde liegenden Verfahrensweisen genug Beachtung finden. Er formuliert seine Sorge dabei wie folgt:

> *„Gerade weil die Grounded Theory nicht als technisches Regelwerk konzipiert ist, sondern als flexible Forschungsstrategie, in der die konsequente Analyse systematisch die Datenerhebung*

> *antreibt, ist die Versuchung groß, die einer solchen Methodologie zugrunde liegenden Verfahrensweisen der Beliebigkeit anheim fallen zu lassen“* [21, S. 191]

Andere Autoren wie Stol et al. [33] teilen diese Bedenken und merken weiterhin an, dass in den bereits bestehenden Studien im Bereich Software Engineering, in denen die Grounded Theory verwendet wurde, oft eine Diskrepanz zwischen dem, was die Forscher in ihrer Studie tun und der Grounded Theory Methodologie besteht. So fehlen oftmals Informationen, wie genau die Methoden und Verfahren der Grounded Theory angewendet wurden. Auf Basis dieser Problematik geben Stol et al. [33] eine Vielzahl von Empfehlungen, was beim Durchführen und Berichten einer Grounded Theory Studie beachtet werden sollte. Zunächst sollte der Forscher sich mit der Grounded Theory beschäftigen bevor er mit der eigentlichen Studie beginnt. Dabei ist es nach Meinung der Autoren notwendig, sich detailliert mit den einzelnen Verfahren, aber auch den Versionen innerhalb der Grounded Theory auseinander zu setzen. Außerdem sollte der Forscher innerhalb seiner Studie berichten, wie genau seine Implementierung der Methoden und Verfahren der Grounded Theory ist. Aus diesem Grund werden in den Folgenden Kapiteln immer wieder Beispiele gegeben, wie die Verfahren der Grounded Theory in der hier vorliegenden Arbeit angewendet wurden. Dies soll das Vorgehen nachvollziehbar und transparent machen.

3.3.2 Versionen der Grounded Theory

Seit der Entstehung der Grounded Theory hat diese sich stetig weiterentwickelt und verändert. Im Zuge dieses Prozesses haben sich viele verschiedene Strömungen ausgebildet [21]. Auch Strauss und Glaser, die die Methode ursprünglich gemeinsam entwickelten, haben sich immer weiter voneinander entfernt. Dies resultierte darin, dass jeder der beiden Forscher seine eigene Version der Grounded Theory hervorbrachte.[9]
Die Grounded Theory nach Glaser gilt allgemein als „klassische“ Grounded Theory. Diese Version hat einen starken Fokus auf dem Emergenzprinzip, d.h. die Theorien und Konzepte müssen von selbst aus den Daten aufkommen. [33]
Strauss, der seine Version der Grounded Theory zusammen mit Juliet Corbin entwickelte, legt den Fokus innerhalb des Vorgehens auf die praktische Umsetzung. Das Kodierungsverfahren nimmt dabei einen zentralen Stellenwert ein. [21]

Die Grounded Theory nach Strauss bietet in der Praxis einige entscheidende Vorteile, weshalb sie als Grundlage für diese Arbeit verwendet wird. Zunächst einmal ist es in dieser Version der Methodologie möglich Forschungsfragen vorab zu definieren. Diese Fragen helfen Grenzen zu etablieren, was genau

in der jeweiligen Studie untersucht werden soll. Strauss und Corbin [10] beschreiben diese Eigenschaft von Forschugsfragen wie folgt:

> *„Another important aspect of the research question is that it helps to establish the boundaries of what will be studied. It is impossible for any investigator to cover all aspects of a problem.“* [10, S. 25]

Während Glaser davon ausgeht, dass der Wissenschaftler kein Vorwissen über das zu untersuchende Feld besitzt und dementsprechend unvoreingenommen in den Prozess der Datensammlung und -Analyse gehen kann, verfolgen Strauss und Corbin auch hier einen pragmatischeren Ansatz [9]. Sie gehen davon aus, dass Objektivität in der qualitativen Forschung ein Mythos ist [10]. Der Forscher darf Vorwissen über das zu untersuchende Feld besitzen und dieses auch nutzen. Allerdings spielt der Begriff der Sensitivität (*sensitivity*) eine große Rolle. Unter Sensitivität verstehen Strauss und Corbin *„the ability to pick up on subtle nuances and cues in the data that infer or point to meaning“* [10, S. 19]. Der Wissenschaftler sollte also sensibel gegenüber Nuancen und Hinweisen in den Daten sein und diese interpretieren und deuten können. Ein weiterer Grund warum die Grounded Theory nach Strauss für diese Arbeit gewählt wurde, ist die Tatsache, dass diese Version das konsultieren von Literatur während des Prozesses erlaubt. Es ist also möglich, zu Beginn eine Literaturrecherche durchzuführen, deren Ziel es z.B. ist, Fragen für die Datensammlung zu erhalten. In der klassischen Grounded Theory sollte eine initiale Literaturrecherche zum untersuchten Thema vermieden werden, um im Vorfeld keine Annahmen zu treffen. [9, 33]

3.3.3 Komponenten der Grounded Theory

In diesem Kapitel wird erläutert, aus welchen Bestandteilen die Grounded Theory sich zusammensetzt und wie die einzelnen Komponenten der Methodologie in Zusammenhang stehen.

Theoretical Sampling

Beim *theoretical sampling* handelt es sich um eine Methode zum identifizieren weiterer Datenquellen basierend auf Lücken in der entstehenden Theorie oder weiter zu erforschender Konzepte, die noch nicht ausreichend untersucht wurden [33]. Corbin und Strauss [10, S. 143] definieren den Begriff wie folgt:

> *„A method of data collection based on concepts/themes derived from data. The purpose of theoretical sampling is to collect data from places, people, and events, that will maximize opportunities to develop concepts in terms of their properties and dimensions, uncover variations, and identify relationships between concepts.“*

Das theoretical sampling beginnt dabei mit der Datensammlung und den ersten Analyseergebnissen. Diese geben einen konzeptuellen Bezugsrahmen zum Sammeln weiterer Daten [21].

Abbildung 3.2 zeigt den Prozess des theoretical sampling. Zunächst werden Daten gesammelt und anschließend analysiert. Die Analyse erfolgt dabei sobald ausreichend Daten vorhanden sind. In diesem Punkt unterscheidet sich das theoretical sampling von konventionellen Sampling-Methoden, bei denen bereits vor der Analyse alle Daten erhoben werden. Aus der ersten Analyse ergeben sich im theoretical sampling Konzepte. Diese werfen jedoch noch Fragen auf, da sie in der Regel während der ersten Datensammlung und -Analyse nicht vollständig untersucht wurden. Dies führt dazu, dass weitere Daten erhoben werden, um die noch offenen Fragen zu klären. Die Datenerhebung knüpft somit unmittelbar an aufkommende Fragestellungen an. Dadurch wird das theoretical sampling zu einem offenen und flexiblen Prozess bei dem Erhebung und Analyse der Daten Hand in Hand gehen. [10]

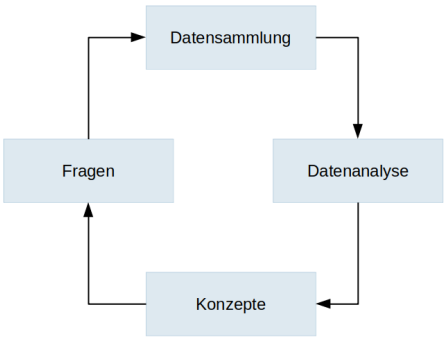

Abbildung 3.2: Ablauf des theoretical sampling

Den Kern des Prozesses bilden dabei meist Interviews. Zu Beginn der Datensammlung wird initial geplant, welche Fragen gestellt werden und welche Quellen für die Datenerhebung genutzt werden. Basierend auf den Ergebnissen der Analyse können dann weitere Interviews geführt, Interviewfragen geändert oder ergänzt werden. Auch können auf Basis der bereits vorhandenen Daten weitere Quellen für die Datenerhebung ausfindig gemacht werden. Die Folge dieses Vorgehens ist ein stetiges Vergleichen zwischen analysierten Daten und entstehender Theorie. [9]
Dieser Prozess wird fortgeführt, bis *theoretische Sättigung* erreicht wurde. Dies ist der Fall, wenn nicht mehr zu erwarten ist, dass die Untersuchung neuer Daten noch zu einer Weiterentwicklung der Kategorien bzw. der Theorien beiträgt. [21]

Kodierungsverfahren und Constant Comparison

Die Datenanalyse und Interpretation findet innerhalb der Grounded Theory über verschiedene Kodierungsstufen hinweg statt. Bei der Kodierung handelt es sich um einen induktiven Prozess, d.h. aus den empirischen Daten werden sukzessive theoretische Konzepte abgeleitet. [21]

Die Kodierung der Daten kann dabei Zeilenweise, Satz- oder Absatzweise erfolgen [33]. Man unterscheidet in der Grounded Theory nach Strauss drei verschiedene Arten von Kodierung: (1) open coding, (2) axial coding und (3) selective coding [21, 11].

Unter *Open coding* versteht man die erste Datenanalyse zur Entwicklung provisorischer Konzepte und Dimensionen [21]. Ziel des Verfahrens ist es, dem Analysierenden neue Einblicke in die Daten zu geben. Zu diesem Zweck werden die Daten analytisch aufgebrochen und durch Konzepte beschrieben. [11]
Die verwendeten Codes können dabei entweder von außen festgelegt werden, oder aus den Daten selbst stammen [9].

Die entstandenen Codes werden im folgenden fortwährend miteinander verglichen. Dieser Prozess des Vergleichens verschiedener Teile von Daten auf Gemeinsamkeiten und Unterschiede nennt man *constant comparison*. Die Methode dient dazu, ein höheres Abstraktionslevel zu erreicht. So werden mit dieser Methode beispielsweise konzeptuell gleiche Codes zu einem Konzept gruppiert. Bei einem Konzept handelt es sich um Worte, die für eine in den Daten enthaltene Idee stehen. Konzepte sind also Interpretationen, d.h. das Produkt von Analyse. Die constant comparision Methode erlaubt dem Forscher verschiedene Themen voneinander abzugrenzen. Für diese Themen können dann spezifische Eigenschaften und Dimensionen identifiziert werden. Unter einer Dimension versteht man in diesem Zusammenhang die in einem bestimmten Umfang bestehende Variation einer Eigenschaft. Die constant comparison Methode kann dabei z.B. auf Daten, Memos, Codes oder Kategorien angewendet werden. [10, 33]

Die zweite Kodierungsart, das *axial coding*, dient dazu Kategorien mit ihren Subkategorien zu verknüpfen [11]. Bei Kategorien handelt es sich um Konzepte höherer Ordnung unter denen Konzepte niedrigerer Ordnung auf Basis gemeinsamer Eigenschaften gruppiert werden. Kategorien werden auch Themen genannt. Sie repräsentieren relevante Phänomene und befähigen den Analysierenden Daten miteinander zu kombinieren [10]. Die Kodierung findet im axial coding „um die Achse“ einer Kategorie statt und es werden Beziehungen zwischen den Kategorien identifiziert. Das axial coding dient der weiter Entwicklung der Kategorien. Diese werden mit Eigenschaften und Dimensionen ausgefüllt. [9, 11, 33]

Die dritte Form des Coding in der Grounded Theory nach Strauss ist das *selective coding*. Diese Form der Kodierung wird verwendet, sobald die Kernkategorie identifiziert wurde. Bei der Kernkategorie handelt es sich um eine Kategorie, die die zentralen Phänomene der Studie repräsentiert. Sie kann aus bereits existierenden Kategorien aufkommen, oder es kann ein

abstrakterer Begriff nötig sein, um das Hauptphämomen zu erklären. Alle anderen Kategorien stehen in Beziehung zur Kernkategorie und die zentrale Kategorie sollte häufig in den Daten vorkommen. [11, 9]
Im selective coding werden werden alle Kategorien systematisch in Bezug zur Kernkategorie gesetzt. Kategorien, die noch weiterer Erklärung bedürfen, werden in dieser Phase der Kodierung weiter ausdetailliert. [11]
Der Prozess des selektiven Kodierens dient der theoretischen Integration [21]. Darunter verstehen Strauss und Corbin [10, S. 263] folgendes:

> *„The process of linking categories around a core category and refining and trimming the resulting theoretical construction.“*

Bei der theoretischen Integration handelt es sich um den letzten Schritt der Datenanalyse. Ziel ist die Theoriegenerierung. In diesem Arbeitsschritt können unterschiedlichste Techniken, wie das erneute Lesen von Memos, das Erstellen von Diagrammen oder einfaches Nachdenken eingesetzt werden. Unter einer Theorie versteht man eine Reihe von ausreichend entwickelten Kategorien, die systematisch in Beziehung zueinander stehen und einen theoretischen Rahmen bilden, um ein Phänomen zu erklären [10].

Memoing

Das *Memoing* stellt einen wichtigen Bestandteil der Forschungsarbeit dar. Während der Datensammlung und -Analyse werden vom Forschenden Memos verfasst. Dabei handelt es sich um Notizen, Ideen, Fragen, Diagramme oder auch Zeichnungen. Ziel des Memoing ist es, den aktuellen Forschungsstand zu reflektieren und festzuhalten. Dabei können Memos Beschreibungen vorläufiger Eigenschaften und Beziehungen zwischen Kategorien enthalten, Lücken identifizieren oder Fragen und Ideen festhalten, die während des Forschungsprozesses nicht verloren gehen sollen. [33, 21, 9]

Kapitel 4

Durchführung der Fallstudie

In diesem Kapitel wird der konkrete Fall, der innerhalb der Fallstudie untersucht wird, näher beschrieben. Dabei wird darauf eingegangen, auf welchen Voraussetzungen die Fallstudie aufbaut, welches Projekt untersucht wird, wer die Projektbeteiligten sind und wie Daten erhoben und analysiert werden.

Die Fallstudie eignet sich aus verschiedenen Gründen zur Anwendung in der Forschung im Bereich des Software Engineering bzw. des Requirements Engineering, das eine Teildisziplin des Software Engineering ist. Bei diesem Gebiet handelt es sich nicht um ein rein technisches Forschungsgebiet. Es müssen Individuen, Gruppen oder ganze Organisationen in den Forschungsprozess miteinbezogen werden. Auch müssen innerhalb der Fragestellungen dieses Forschungsgebiets oft psychologische, soziale oder kulturelle Phänomene mitberücksichtigt werden. Software Engineering muss also als interdisziplinäres Forschungsgebiet verstanden werden, das jene Gebiete mit einschließt, in denen Fallstudien normalerweise durchgeführt werden wie Soziologie oder Psychologie. [9, 29, 16]

4.1 Literaturrecherche und Forschungsfragen

Zu Beginn dieser Arbeit wurde eine initiale Literaturrecherche durchgeführt. Dabei lag der Fokus darauf, erste Ideen, Fragen und Konzepte für das theoretical sampling zu erhalten. Die erste Literaturrecherche bildet weiterhin die Grundlage für die in dieser Arbeit formulierten Forschungsfragen.

Strauss und Corbin [10] gehen davon aus, dass eine oder mehrere Forschungsfragen benötigt werden, die den Forscher innerhalb der Datensammlung führen. Die Forschungsfragen sollten dabei genug Flexibilität und Freiheit bieten, um ein Thema in einer gewissen Tiefe zu

erforschen. Die Fragen sollten das Themengebiet, das untersucht werden soll, identifizieren und festlegen, was innerhalb dieses Themengebiets von Interesse für die Forschung ist. [10]

In dieser Arbeit werden die folgenden drei Forschungsfragen untersucht:

RQ1 Welche Auswirkungen hat die Kombination von agilen und traditionellen, d.h. plan-basierten Verfahren in der Software-Entwicklung auf das Requirements Engineering?

RQ2 Was sind Gründe dafür, dass sich Unternehmen dazu entschließen nach einem hybriden Modell zu arbeiten?

RQ3 Welche Probleme und Herausforderungen ergeben sich in hybriden Software-Entwicklungsmodellen in Bezug auf das Requirements Engineering?

Zunächst einmal wird untersucht, wie agile und traditionelle Verfahren der Software-Entwicklung im konkreten Fall miteinander kombiniert und in Balance gebracht werden. Dabei stehen die Auswirkungen auf das Requirements Engineering im Vordergrund. Die zweite Forschungsfrage wurde gewählt, um herauszufinden, was die konkreten Voraussetzungen dafür sind, dass sich das Unternehmen für einen hybriden Prozess entschieden hat. Die dritte Forschungsfrage bezieht sich sowohl auf mögliche Probleme und Herausforderungen bei der Sammlung, Analyse und Erstellung von Anforderungen, als auch auf Probleme bei der Umsetzung durch die Entwickler.

4.2 Fallbeschreibung

Das Unternehmen, in dem die Fallstudie durchgeführt wird, wurde 2001 gegründet und umfasst im Jahr 2020 etwa 130 Mitarbeiter*innen, die an verschiedenen Standorten in ganz Deutschland zusammenarbeiten. Kerngeschäft des Unternehmens ist das Consulting, welches vor allem, aber nicht ausschließlich, in den Bereichen Banken und Versicherungen stattfindet. Neben dem Consulting verfügt das Unternehmen über einen Bereich, in dem eigene Produkte entwickelt werden. Dabei handelt es sich hauptsächlich um Produkte zur Überwachung von Datenbanken.

4.2.1 Projekt

Bei dem Projekt, welches in dieser Fallstudie untersucht wird, handelt es sich um eine firmeninterne Produktentwicklung. Ziel des Projekts ist die Entwicklung eines Datenbanken-Monitoring-Tools, welches es ermöglichen soll, verschiedene Datenbanktypen, sowie das Betriebssystem unter dem die

Datenbank verwendet wird, zu überwachen. Zum Zeitpunkt der Fallstudie läuft das Projekt seit ca. einem Jahr.

An dem Projekt sind vornehmlich zwei Bereiche des Unternehmens beteiligt. Der erste Bereich ist dabei für die Entwicklung des Produkts zuständig. Dieser Bereich umfasst zum Zeitpunkt der Fallstudie sechs Mitarbeiter*innen, von denen fünf im Projekt mitwirken. Unter den Mitwirkenden gibt es die Rollen des Projektleiters, des Softwareentwicklers und des Softwarearchitekten. Die Mitarbeiter dieses Bereichs arbeiten üblicherweise zusammen an einem Ort.
Der zweite Bereich ist dafür zuständig, die Anforderungen an das Produkt zu definieren. In diesem Bereich gibt es einen Mitarbeiter, der aktiv am Projektgeschehen als Product Owner teilnimmt. Seine Aufgabe ist es dabei Anforderungen verschiedenster Art zu definieren und zu kommunizieren. Dieser Mitarbeiter befindet sich üblicherweise nicht am selben Unternehmensstandort wie die Mitarbeiter*innen, die für die Produktentwicklung zuständig sind. Es gibt keine weiteren Personen, des zweiten Bereichs, die aktiv am Projekt teilnehmen.

Das hier gewählte Projekt eignet sich für die durchgeführte Fallstudie, da es innerhalb des Projekts sowohl traditionelle, als auch agile Rollen und Verfahrensweisen gibt. So existiert beispielsweise neben dem agilen Product Owner ein traditioneller Projektleiter. Des Weiteren eignet sich das Projekt gut, da nicht zu viele Personen beteiligt sind. Würden mehr Personen innerhalb des Projekts eingesetzt, so bestünde die Gefahr, dass der Rahmen dieser Arbeit nicht ausreichen würde, um das Projekt umfassend zu untersuchen.

4.3 Datensammlung

Die Datensammlung erfolgt in dieser Arbeit über das *theoretical sampling*-Verfahren der Grounded Theory (siehe Kapitel 3.3.3). Zu diesem Zweck wurden Interviews mit den Projektbeteiligten geführt. Jedes dieser Interviews wurde umgehend transkribiert und analysiert, sodass aus jedem durchgeführten Interview Erkenntnisse gewonnen wurden, die direkt in die Fragestellungen der folgenden Interviews miteinflossen. Auf diese Weise konnten sukzessive Erkenntnisse gewonnen und Konzepte entwickelt werden.

4.3.1 Interview-Planung

Da es in dieser Arbeit darum geht, sowohl absehbare, also auch unerwartete Informationen zu erheben und zu analysieren, wird das *semi-strukturierte Interview* zur Datenerhebung verwendet. Auf diese Weise ist es möglich, während der Befragung möglichst umfangreiche Erkenntnisse zu erlangen.

Gleichzeitig können die Fragen als Grundlage verwendet werden, um mit unterschiedlichen Probanden Interviews durchzuführen, die dann verglichen und analysiert werden können (siehe Kapitel 3.2.1).

Wahl der Interviewpartner

Wie in Kapitel 3.2.3 bereits beschrieben, sollten die Interviewpartner auf Basis ihrer Unterschiede gewählt werden [29]. In der hier vorliegenden Studie wurden insgesamt fünf Interviews geführt. Dabei wurde darauf geachtet, dass jede Rolle im Projekt in mindestens einem Interview vertreten ist. Tabelle 4.1 zeigt welche Rollen in den Interviews mit jeweils wie vielen Probanden vertreten sind. Aufgrund der geringen Anzahl an Projektbeteiligten war es

Rolle	Anzahl der Teilnehmer
Projektleiter	1
Entwickler	2
Softwarearchitekt	1
Product Owner	1

Tabelle 4.1: Anzahl der Teilnehmer pro Rolle

innerhalb dieser Fallstudie nicht nötig noch weitere Personen zu befragen. Bei der einzigen Person, die nicht befragt wurde, handelt es sich um einen Software-Entwickler. Die Sicht des Software-Entwicklers ist jedoch bereits durch zwei Probanden in den Interviews vertreten, sodass nicht davon auszugehen ist, dass durch ein weiteres Interview mit der nicht interviewten Person noch neue Erkenntnisse gewonnen werden können.

Erstellung des Fragebogens

Der verwendete Fragebogen wurde auf Basis der Forschungsfragen (siehe Kapitel 4.1) erstellt. Er diente als Leitfaden für die Interviews und wurde auf Basis der gewonnenen Erkenntnisse von Interview zu Interview weiterentwickelt und ergänzt. Die initialen Interviewfragen befinden sich in Anhang A. Diese Fragen wurden so konzipiert, dass der Fragebogen wie in Kapitel 3.2.2 beschrieben aufgebaut ist. Der erste Abschnitt des Interviews enthält zwei einführende Fragen zum Hintergrund des Befragten. Diese sind für den Probanden leicht zu beantworten und dienen dazu, seine Antworten in Bezug zu seiner Rolle im Projekt zu setzen. Im weiteren Verlauf des Interviews sind die Fragen so gegliedert, dass zunächst Fragen zum Anforderungsprozess innerhalb des Projekts und dann Fragen zur persönlichen Wahrnehmung dieses Prozesses gestellt werden. Es soll dabei zunächst geklärt werden, wie im Projekt beim Erheben und Verwalten von Anforderungen vorgegangen wird. Danach sollen Erfolgsfaktoren, Probleme und Herausforderungen identifiziert werden. Die Fragen zur

subjektiven Wahrnehmung des Probanden wurden bewusst ans Ende gestellt, um dem Probanden die Möglichkeit zu geben, sich zunächst an die Interviewsituation zu gewöhnen. Es wurde generell darauf geachtet, Fragen möglichst respektvoll und eindeutig zu formulieren.

Über den Verlauf der Interviews wurden die folgenden Fragen ergänzt:

a) Wurden die nicht-funktionalen Anforderungen nur zu Projektbeginn erhoben oder werden diese auch kontinuierlich erhoben?

b) Glaubst du, ihr würdet anders arbeiten, wenn ihr nur ein Produkt betreuen würdet bzw. euch nur auf die Entwicklung neuer Produkte konzentrieren könntet? Was würde sich ändern?

c) Welche Rolle spielt Teamwork für dich beim Erarbeiten von Anforderungen?

d) Wie könnte die Qualität der Anforderungen (noch) sicher gestellt werden?

Die erste Frage wurde ergänzt, um einen tieferen Einblick in die Erhebung der nicht-funktionalen Anforderungen zu bekommen. Nicht-funktionale Anforderungen sollten zum Zeitpunkt der Entwicklung bekannt sein, da ein spätes Aufkommen einen erheblichen Mehraufwand bedeuten kann, weil diese Anforderungen weitreichende Auswirkungen, z.B. auf die Wahl von Datenbanken oder Programmiersprache, haben [23]. Ein zentrales Problem war für die interviewten Probanden die Verfügbarkeit von Mitarbeitern. So stehen die Mitarbeiter innerhalb des Projekts nicht exklusiv für die Neuentwicklung zur Verfügung, sondern müssen neben dieser Tätigkeit auch den Support bereits bestehender Produkte betreuen. Um die Auswirkungen dieses Sachverhalts auf die Wahl der Entwicklungsmethoden weiter zu untersuchen, wurde die zweite Frage ergänzt. Die dritte Frage wurde ergänzt, da im Verlauf der Interviews deutlich wurde, dass ein Teil der Anforderungen iterativ erhoben und ausdetailliert wird. Es soll mit dieser Frage untersucht werden, welche Rolle Teamwork in diesem Zusammenhang spielt. Die letzte Frage wurde ergänzt, da in den Interviews deutlich wurde, dass viele Probanden die Maßnahmen zur Qualitätssicherung der Anforderungen als nicht ausreichend betrachteten.

Um die Fragen vor den eigentlichen Interviews zu testen, wurde ein Pilot-Interview mit einem Mitarbeiter des Unternehmens durchgeführt, der nicht im untersuchten Projekt mitwirkt. Dieser Mitarbeiter wurde gebeten die Fragen auf ein beliebiges Projekt zu beziehen und Rückfragen zu stellen, falls eine Frage nicht verständlich ist. Aufgrund der so gewonnenen Erkenntnisse konnten die Fragen noch einmal angepasst und die Dauer der einzelnen Interviews vorab geschätzt werden.

4.3.2 Interview-Durchführung

Die Interviews wurden im Zeitraum vom 17.06.2020 bis 10.07.2020 durchgeführt. Aufgrund der COVID-19 Pandemie wurde allen Teilnehmern angeboten, das Interview wahlweise online durchzuführen. Vier der fünf Probanden entschieden sich für ein online Interview über Microsoft Teams. Dieses Tool wurde gewählt, da die Probanden damit vertraut sind und es innerhalb des Projektsalltags zur Kommunikation eingesetzt wird.

Die Interviews umfassten eine Dauer von jeweils 35 bis 50 Minuten. Bei allen durchgeführten Interviews wurde eine Tonaufnahme angefertigt, die als Grundlage für die anschließende Transkription diente.

In der Phase der Einleitung des Interviews (siehe Kapitel 3.2.2) wurde den Probanden versichert, dass alle Angaben, die sie während des Interviews machen, vertraulich behandelt werden. Sie wurden gebeten möglichst spontan auf die Fragen zu antworten. Dabei wurden die Probanden darauf hingewiesen, dass es weder richtige noch falsche Antworten auf die Interviewfragen gibt. Wenn die Probanden Fragen nicht verstanden haben, konnten sie jederzeit Rückfragen stellen. Auch wurde in dieser Phase das Einverständnis zur Tonaufnahme der Befragten eingeholt. Weiterhin wurde den Probanden in der Einleitung das Thema des Interviews mitgeteilt.

4.4 Datenanalyse

Zur Datenanalyse werden in dieser Arbeit verschiedene Verfahren der Grounded Theory eingesetzt. Um Klarheit darüber zu schaffen, wie diese Verfahren genau genutzt werden, werden die wichtigsten Praktiken im folgenden detailliert beschrieben und beispielhaft erläutert. Dieser Schritt stellt für Stol et al. [33] ein wichtiges Qualitätsmerkmal beim Berichten von Grounded Theory Studien dar. Nach Meinung der Forscher sollte immer auch die konkrete Umsetzung der Grounded Theory Verfahren im vorliegenden Fall beschrieben werden.

4.4.1 Überblick

Abbildung 4.1 zeigt den Prozess der Datensammlung und -Analyse der Grounded Theory, wie er in dieser Arbeit praktiziert wird. Zunächst werden mittels *theoretical sampling* iterativ Daten erhoben und anschließend analysiert. Dies geschieht in dieser Arbeit über Interviews, die zunächst transkribiert und dann abschnittsweise mit den in Kapitel 3.3.3 beschrieben Kodierungsverfahren des Open Coding und Axial Coding analysiert werden.

Während dieser Arbeitsschritte werden fortwährend Memos geschrieben (siehe Kapitel 3.3.3), um Ideen und noch offene Fragen festzuhalten. Die Erkenntnisse aus Open Coding, Axial Coding, sowie die Memos fließen dabei in der ersten Iterationsschleife direkt wieder in den Prozess der Datensammlung ein. Dieser Prozess wird so lange wiederholt, bis die zentrale Kategorie (*Core Category*) identifiziert wird. Sobald dies geschehen ist, verändert sich der Prozess der Kodierung. Es wird nun das *Selective Coding* eingesetzt und der Prozess der theoretischen Integration eingeleitet (siehe Kapitel 3.3.3). Auch in diesem Arbeitsschritt wird iterativ vorgegangen.

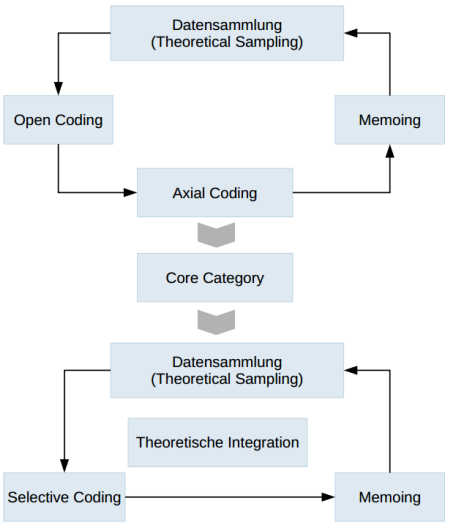

Abbildung 4.1: Gesamtprozess

Im Folgenden werden exemplarisch Beispiele gegeben, wie in den einzelnen Phasen der Datenanalyse vorgegangen wurde.

4.4.2 Open und Axial Coding

Der erste Schritt der Kodierung innerhalb der Grounded Theory bildet das *Open Coding* (siehe Kapitel 3.3.3). In dieser Arbeit wurde so vorgegangen, dass auf Basis der Interviewtranskripte zunächst die Kernaussage (*key words*) eines jeden Abschnitts festgehalten wurde. Danach wurden diesen Hauptgedanken Codes zugeordnet. Dabei handelt es sich um wenige Worte, die die Kernaussage zusammen fassen. Ein Abschnitt kann dabei einen oder auch mehrere Codes enthalten. Ein vergleichbares Vorgehen im Open Coding findet man bei Hoda et al. [14]. Dort wurde die Grounded Theory dazu genutzt, um die Selbst-Organisation agiler Teams zu untersuchen. Im folgenden soll beispielhaft gezeigt werden, wie in dieser Arbeit beim Kodieren vorgegangen wurde:

> **Interview-Zitat:** *„Und da kommt es dann auf die Größe und Tragweite der jeweiligen Anforderung an. Wenn es - so als Beispiel mal nur - heißt: „Ich brauche hier einen Knopf für Aktion X“ - dann wird das nicht zwingend in ein eigenes Konzept gegossen. Wenn es aber heißt: „Wir brauchen eine neue Schnittstelle für die Datenübertragung zwischen zwei Applikationen“, dann wird da auch natürlich eine entsprechende*

> *Planung, eine ordentliche Dokumentation gemacht, sodass das nachher auch fachgerecht umgesetzt wird.“* - P3, Product Owner
>
> **key words:** „Je komplexer die Anforderung ist, desto mehr Planung und Dokumentation ist nötig.“
>
> **Codes:** Angemessenheit der Planung, Angemessenheit der Dokumentation, höherer Aufwand für komplexere Anforderungen

Die so entstandenen Codes der jeweiligen Interviews wurden im folgenden über das *Constant Comparison*-Verfahren fortwährend mit den Codes des selben Interviews und Codes aus anderen Interviews verglichen (siehe Kapitel 3.3.3). Auf diese Weise war es möglich, die gewonnenen Informationen Schritt für Schritt auf ein höheres Abstraktionslevel zu führen.

Abbildung 4.2 zeigt die in dieser Arbeit verwendeten Abstraktionslevel bzw. Kodierungsstufen. Aus den Codes wurden Konzepte erstellt, die sich in Kategorien integrieren, die schließlich zu einer Theorie führen. Um Kategorien mit ihren Subkategorien, bzw. zugehörigen Konzepten zu verbinden, wurde das *Axial Coding* (siehe Kapitel 3.3.3) verwendet. In dieser Arbeit war der Übergang zwischen Open Coding und Axial Coding fließend. Diesen fließenden Übergang von offenem und axialem Kodieren findet man in den späteren Werken von Strauss und Corbin. Die Autoren gehen davon aus, dass offenes und axiales Codieren Hand in Hand gehen sollten, da die Trennung dieser beiden Verfahren nur einem erklärenden Zweck dient. [10]

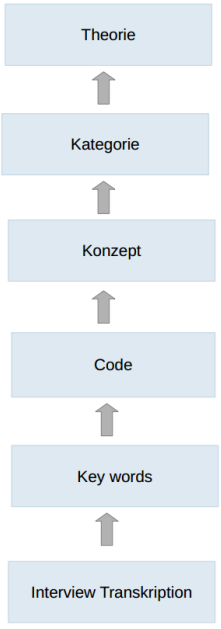

Abbildung 4.2: Stufen des Kodierungsprozesses

Das Ergebnis des axialen Kodierens waren Beschreibungen und Hypothesen bezüglich der im Fokus stehenden Kategorien und ihrer Beziehungen untereinander. Während des axialen Kodierens wurden gezielt Passagen in den geführten Interviews, die in Beziehung zueinander gesetzt werden sollten, miteinander verglichen und

analysiert. Dabei wurde darauf geachtet, dass jede Textpassage in ihrem spezifischen Kontext betrachtet wird. Für Seaman [31] spielt der Kontext eine entscheidende Rolle und sollte immer berücksichtigt werden, wenn die Daten im Rahmen des axialen Kodierens in Bezug zueinander gesetzt werden.

Während des Kodierens entstanden eine Vielzahl von Codes, Konzepten und Kategorien. Abbildung 4.3 zeigt beispielhaft aus welchen Codes bzw. Konzepten die Kategorie „Ressourcenmangel“ hervorging. Es wurde im Rahmen des Kodierens als hilfreich empfunden die Beziehungen zwischen Codes, Konzepten und Kategorien in einer graphischen Repräsentation festzuhalten. Auf diese Weise konnten bereits bestehende Verbindungen leicht identifiziert werden. Ein Code konnte dabei im Rahmen des Kodierungsprozesses je nach Kontext einem oder mehreren Konzepten zugeordnet werden.

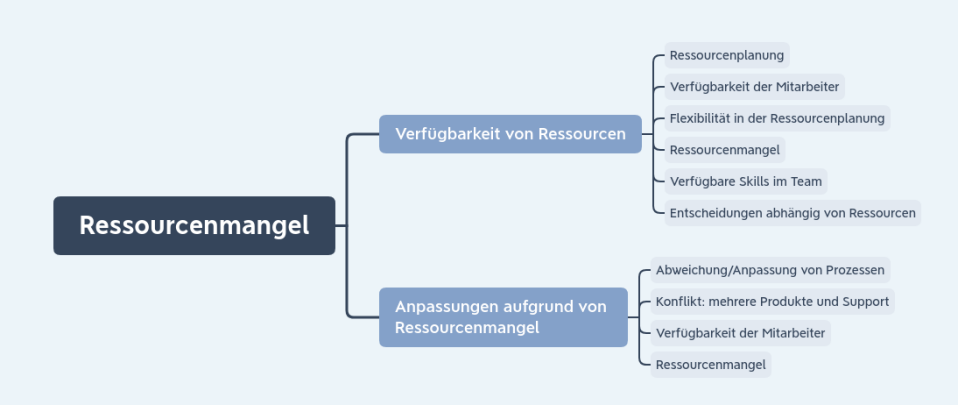

Abbildung 4.3: Aufkommen der Kategorie „Ressourcenmangel“ aus den zugrunde liegenden Konzepten und Codes

4.4.3 Kernkategorie und Selective Coding

Die erste Iterationsschleife der Datenanalyse wurde beendet, als die Kernkategorie aufkam. Für Strauss [35] gibt es verschiedene Kriterien, die bei der Wahl der Kernkategorie berücksichtigt werden sollten: Zunächst einmal sollte die Kernkategorie zentral sein, d.h. sie sollte mit möglichst vielen Kategorien und deren Eigenschaften in Verbindung stehen. Des Weiteren sollte die Kernkategorie häufig in den Daten vorkommen und im Vergleich zu anderen Kategorien lange Zeit bis zur theoretischen Sättigung benötigen. Die zentrale Kategorie sollte dabei nicht aus den Daten erzwungen werden, sondern vielmehr auf logische, konsistente Weise aus den Daten aufkommen. Sie sollte abstrakt genug sein, um als Grundlage für eine allgemeinere Theorie zu dienen. Die zentrale Kategorie sollte mehr

und mehr an Tiefe und Erklärungsgehalt gewinnen, desto mehr Kategorien mit ihr in Verbindung gebracht werden. [35, 10]
Die Kernkategorie kann aus einer Liste bereits existierender Kategorien hervorgehen, oder es kann nötig sein, eine abstraktere Beschreibung für die zentrale Kategorie zu finden. Dies ist der Fall, wenn alle im Verlauf der Datenanalyse entstandenen Kategorien gleichermaßen einen Teil zur Kernkategorie beitragen. [10]

In dieser Arbeit wurde eine neue abstrakte Kategorie als Kernkategorie definiert. Es wurde die Kernkategorie *„gemeinschaftliche Prozessoptimierung im Umfeld unvorhersehbarer Projektrisiken“* gewählt. Diese Kernkategorie steht mit allen im Rahmen dieser Arbeit entstandenen Kategorien in Verbindung und spiegelt das Hauptanliegen der Projektbeteiligten wider. Um sicher zu stellen, dass es sich bei der gefundenen Kategorie um die Kernkategorie handelt, wurde überprüft, ob die oben genannten Kriterien zum identifizieren einer Kernkategorie auf die gewählte Kategorie zutreffen.

Nach der Identifikation der Kernkategorie wurde das *Selective Coding* (siehe Kapitel 3.3.3) dazu genutzt, alle Kategorien systematisch in Bezug zur Kernkategorie zu setzen und Kategorien, die noch weiterer Erklärung bedurften, wurden ergänzen. Parallel wurden im Rahmen der theoretischen Integration (siehe Kapitel 3.3.3) noch einmal alle in der Datenanalyse entstandenen Memos untersucht und ausgewertet.

4.4.4 Memoing

Mit dem Schreiben von Memos sollte bereits während des *Open Coding* begonnen werden [21]. In dieser Arbeit wurden Memos in allen Kodierungsphasen eingesetzt, um Ideen, Beschreibungen und Hypothesen festzuhalten. Das folgende Beispiel zeigt, wie Memos verfasst wurden:

> *„Die Verfügbarkeit von Mitarbeitern scheint einen entscheidenden Einfluss darauf zu haben, wie in der Software-Entwicklung vorgegangen wird. Dadurch, dass geplante Ressourcen, wie z.B. Mitarbeiter nicht zur Verfügung stehen, muss sehr flexibel reagiert werden. Wie wirkt sich dies konkret auf den Anforderungsprozess aus?“*

Alle entstandenen Memos wurden zentral in einer Datei verwaltet. Dies wurde als hilfreich empfunden, da die Memos so nach Themen geordnet und sukzessive im Verlauf der Datensammlung und -Analyse ergänzt werden konnten. Außerdem konnten so entstehende Ideen weiter verfeinert und um zusätzliche Aspekte ergänzt werden.

Kapitel 5

Ergebnisse

Innerhalb der Fallstudie konnte eine Fülle von Informationen gewonnen werden. In diesem Kapitel werden die gewonnenen Erkenntnisse, sowie deren Bezug zu den gestellten Forschungsfragen dargestellt.

Innerhalb der Fallstudie wurden vier Kategorien identifiziert: (1) Kommunikation und Abstimmung, (2) flexible Anpassung, (3) Ressourcenmangel und (4) Angemessenheit. Diese Kategorien reflektieren die Hauptanliegen der Projektbeteiligten und fließen alle in die Kernkategorie ein.
Viele Prozesse innerhalb des untersuchten Entwicklungsprozesses basieren auf direkter *Kommunikation und Abstimmung* zwischen den Projektbeteiligten. Um dies zu unterstützen gibt es regelmäßige Treffen, die dem Austausch zu verschiedenen Themen dienen. Dabei befindet sich das Entwicklerteam im Spannungsfeld zwischen Führung durch den Projektleiter und Selbstorganisation. Auch in der agilen Software-Entwicklung findet man einen starken Fokus auf direkter Kommunikation. Anforderungen werden z.B. iterativ im direkten Austausch zwischen Kunde und Entwicklern erhoben. [27]
Die Kategorie *flexible Anpassung* umschließt sowohl Anpassungen aufgrund von Ressourcenmangel, als auch Anpassungen, die gezielt zur Prozessverbesserung vorgenommen werden. Ziel ist es im Projekt genug Flexibilität zu haben, um sowohl auf Änderungen in den Anforderungen, als auch auf eine sich ständig ändernde Mitarbeiterverfügbarkeit reagieren zu können. Der untersuchte Prozess ist dabei evolutionär durch stetige Anpassung und Veränderung gewachsen. Klünder et al. [17] haben gezeigt, dass dies der häufigste Weg ist, auf dem hybride Software-Entwicklungsverfahren entstehen. *Ressourcenmangel*, vor allem Mitarbeiterverfügbarkeit, stellt ein wichtiges Thema innerhalb der Entwicklung dar. Im untersuchten Projekt stehen die Mitarbeiter nicht exklusiv für die Neuentwicklung zur Verfügung, sondern müssen neben

dieser Tätigkeit auch den Support bereits bestehender Produkte betreuen. Durch diesen Faktor besteht ein hohes Maß an Planungsunsicherheit innerhalb des untersuchten Projekts, da die Menge der Support-Anfragen nur schwer abgeschätzt werden kann. Die vierte Kategorie, *Angemessenheit*, beinhaltet mehrere Aspekte. Zunächst einmal spielt die Angemessenheit der Dokumentation eine wichtige Rolle für die Projektbeteiligten. Dabei muss eine Balance zwischen zu viel und zu wenig Dokumentation gefunden werden. Weiterhin ist die Angemessenheit von Prozessen und die Angemessenheit des Aufwands im Requirements Engineering von Bedeutung. Im untersuchten Projekt werden die Anforderungen zu einem Teil zu Projektbeginn erhoben, zu einem anderen Teil kontinuierlich während der Projektlaufzeit.

5.1 Auswirkungen hybrider Software-Entwicklung

Die erste Forschungsfrage, die innerhalb dieser Arbeit gestellt wurde, beschäftigt sich mit den Auswirkungen der Kombination von agilen und traditionellen Verfahren der Software-Entwicklung auf das Requirements Engineering: *Welche Auswirkungen hat die Kombination von agilen und traditionellen, d.h. plan-basierten Verfahren in der Software-Entwicklung auf das Requirements Engineering?*

Im Rahmen dieser Arbeit konnten vor allem in drei Bereichen Auswirkungen festgestellt werden. Zunächst einmal gibt es innerhalb des untersuchten Projekts traditionelle und agile Rollen, die spezifische Verantwortlichkeiten in Bezug auf das Requirements Engineering haben. Weitere Einflussfaktoren finden sich in den Bereichen der Anforderungserhebung und der Dokumentation.

5.1.1 Rollen und Verantwortlichkeiten

Für Bick et al. [5] stellt das Vorhandensein agiler und traditioneller Rollen innerhalb eines Projekts eine Kerncharakteristik der hybriden Software-Entwicklung dar. Jede dieser Rollen hat bestimmte Verantwortlichkeiten und Aufgaben in Bezug auf das Requirements Engineering. Das bedeutet, dass sich das Vorhandensein der Rollen darauf auswirkt, wie die konkrete Aufgabenverteilung im Requirements Engineering vorgenommen wird.
Der Projektleiter verkörpert dabei eine traditionelle Rolle. Seine Aufgabe innerhalb des untersuchten Projekts liegt schwerpunktmäßig im administrativen Bereich. So ist er zum Beispiel zuständig für die Ressourcenplanung und die Koordination der anfallenden Aufgaben. Dies geschieht immer in enger Abstimmung mit den anderen Projektbeteiligten. In Bezug auf das Requirements Engineering besteht seine Aufgabe darin, in Abstimmung mit Entwicklern und Software-Architekt den Aufwand der Anforderungen zu

schätzen. Auch überwacht der Projektleiter die Umsetzung der Anforderungen. In diesem Zusammenhang ist er zum Beispiel dafür verantwortlich zu überprüfen, welche Anforderungen bereits umgesetzt wurden, sodass in jeder Iteration alle dafür vorgesehenen Anforderungen realisiert werden. Eine agile Rolle innerhalb des Projekts stellt der Product Owner dar. Seine Aufgabe ist es, als Schnittstelle zwischen dem Entwickler-Team und den Kundenanforderungen zu stehen. In diesem Zusammenhang stellt der Product Owner die fachlichen Anforderungen.

> *„Die Quelle ist natürlich aus unserer Sicht der Product Owner. Er schreibt die Anforderungen. Er wiederum hat natürlich die Kunden als weitere Quelle der Anforderungen. Er ist ja da sehr stark im Dialog mit den eigentlichen Endkunden [..]“* - P4, Projektleiter

Der Product Owner steht dabei in enger Abstimmung mit den Entwicklern bezüglich der detaillierten Umsetzung. Es besteht eine direkte Kommunikation zwischen Product Owner und Entwicklern. Wenn bei der Umsetzung einer Anforderung Fragen aufkommen, so können die Entwickler jederzeit auf den Product Owner zugehen. Auf diese Weise werden schnell Entscheidungen in Bezug auf die Anforderungen und ihre Umsetzung getroffen.

5.1.2 Balance von initialer und kontinuierlicher Anforderungserhebung

Das hybride Vorgehen innerhalb des Projekts wirkt sich weiterhin auf die Art und den Zeitpunkt der Anforderungserhebung aus. Dabei werden innerhalb des untersuchten Projekts zwei Phasen der Anforderungserhebung unterschieden: (1) die initiale Anforderungserhebung und (2) die kontinuierliche Anforderungserhebung.

Die Phase der initialen Anforderungserhebung erfolgt einmalig zu Projektbeginn. Im Rahmen dieses Prozesses werden sowohl funktionale, als auch nicht-funktionale Anforderungen erhoben. Gerade das Erheben nicht funktionaler Anforderungen stellt in agilen Projekten oft ein Problem dar, da diese Art von Anforderung gar nicht, oder unzureichend erhoben wird [23]. Bei den initialen Anforderungen handelt es sich in der Regel um high-Level Anforderungen. Das bedeutet, dass diese Anforderungen einen Rahmen für die weitere Projektplanung und die Entwicklung darstellen. Sie müssen allerdings im Verlauf des Projekts noch weiter ausdetailliert werden, um von den Entwicklern umgesetzt werden zu können.

> *„Wir schreiben nicht von vorn herein ein riesen Konzept, in dem quasi alles komplett beschrieben ist, sondern haben kleine einzelne Arbeitspakete definiert und auch Features definiert, die dann am*

> *Anfang ganz grob nur formuliert sind und dann im Prozess weiter ausformuliert werden, im Prozess dokumentiert werden und auch getestet und halt auch qualitätsgesichert werden. "* - P3, Product Owner

Die Ausdetaillierug der Anforderungen geschieht iterativ. Die initialen high-Level Anforderungen wurden im untersuchten Projekt als Feature-Liste festgehalten, die durch generelle Rahmenbedingungen, wie beispielsweise durch die Anwendung unterstützte Browser, ergänzt wurde. Dabei wurden alle Features dieser Liste priorisiert und einer Iteration zugeordnet, in deren Rahmen sie umgesetzt werden sollen. Jede Iteration umfasst dabei mehrere Features. Abbildung 5.1 zeigt den Prozess der initialen Anforderungserhebung. Die auf diese Weise gewonnenen Anforderungen

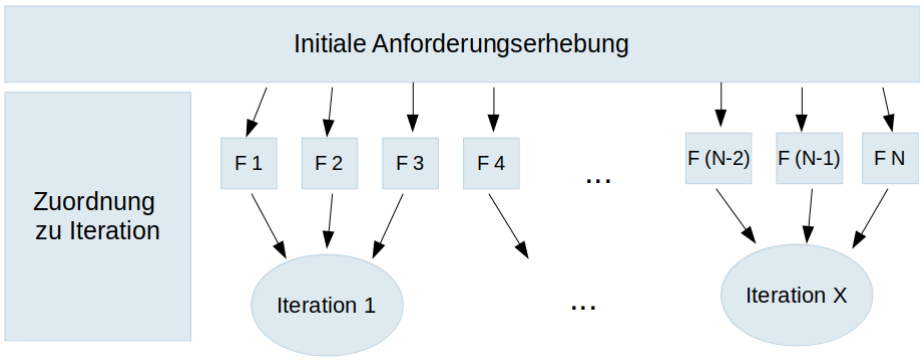

Abbildung 5.1: Prozess der initialen Anforderungserhebungserhebung

werden durch die kontinuierliche Anforderungserhebung in jeder Iteration ergänzt und konkretisiert. Beim detaillierten Erarbeiten der Anforderungen werden dabei alle Projektbeteiligten miteinbezogen. So bekommen die Entwickler zum Beispiel sowohl die Möglichkeit während der Iterations-Planung, als auch in den wöchentlichen Statusmeetings Rückfragen zu den gestellten Anforderungen zu stellen. Auf diese Weise werden innerhalb des Projekts Lücken in den erhobenen Anforderungen identifiziert oder auch Inkonsistenzen aufgedeckt. Das wichtigste Mittel um diese Lücken zu schließen ist direkte Kommunikation:

> *„[..] wir machen dann ein Planungsmeeting, wo wir uns diese ganzen Anforderungen mal anschauen und noch konkrete Fragen stellen [..]. Weil meistens ist das alles ziemlich grundlegend was wir von ihm bekommen und noch lange nicht so detailliert, wie wir das brauchen. "* - P5, Software-Architekt

Der regelmäßige direkte Austausch wird dabei von allen Projektbeteiligten als positiv wahrgenommen. Eine wichtige Rolle spielen kurze Wege, eine gute Erreichbarkeit der benötigten Ansprechpartner und schnelle Entscheidungen.

> *„Die Kontakte im Team sind sehr gut. Also, dass man den Product Owner eigentlich gut erreichen kann und diese wöchentlichen Meetings, bei denen man einen Ansprechpartner hat, wo man auch nicht ewig warten muss.“* - P1, Entwickler

Die in diesem Kapitel beschriebene Kombination von initialer und kontinuierlicher Anforderungserhebung wirkt sich weiterhin darauf aus, wie mit den einzelnen Features innerhalb einer Iteration umgegangen wird. Dabei durchläuft jedes zu entwickelnde Feature mehrere Stufen (siehe Abbildung 5.2). Die Anforderung wird zunächst vorbereitet, d.h. dass die

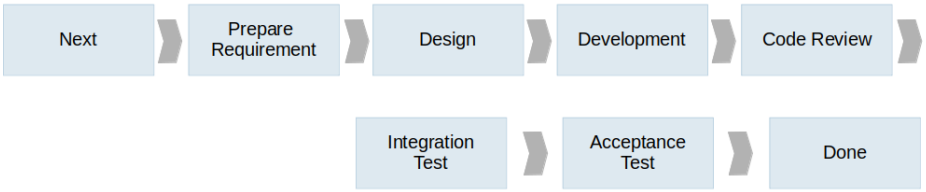

Abbildung 5.2: Kanban Prozess

Anforderungen weiter konkretisiert und präzisiert werden. Im folgenden Schritt wird daraus ein Design-Konzept erstellt. Dieser Arbeitsschritt wird immer von zwei Entwicklern gemeinsam durchgeführt, um sicher zu stellen, dass alles vollständig erfasst wurde.

> *„Und dieses Konzept wird dann später mit dem Product Owner wieder abgestimmt und dann iterativ immer weiter gearbeitet bis wir dann einen Stand erreicht haben, wo wir sagen können 'ok, jetzt können wir daraus wirklich ein Ticket erstellen'. “*- P5, Software-Architekt

Auf dieser Basis werden konkrete Aufgaben formuliert, die in Form von Tickets festgehalten und im nächsten Schritt entwickelt werden. Wenn die Entwicklung abgeschlossen ist, schließen sich noch eine Reihe von Schritten zur Qualitätssicherung an. Zunächst wird ein Code Review durch einen Entwickler durchgeführt, danach schließt sich der Integrations- und der Akzeptanztest auf Systemebene an. Der in Abbildung 5.2 beschriebene Prozess wird dabei auf einem Kanban Board abgebildet, um den Status der einzelnen Tickets verfolgen zu können.

5.1.3 Dokumentation von Anforderungen

Innerhalb des Projekts werden Anforderungen traditionell in Dokumenten festgehalten. Dabei handelt es sich in der Regel um Konzepte. Es werden jedoch nicht alle Anforderungen zu Beginn dokumentiert, wie es in der

traditionellen Software-Entwicklung üblich ist, sondern die Anforderungsdokumente wachsen agil mit dem Projekt. Das bedeutet, dass in jeder Iteration neue Anforderungen hinzukommen, die dann entsprechend nachdokumentiert werden.

> *„Das Dokument wird trotzdem agil gepflegt. Also es ist jetzt nicht so, dass wir da vorher 10 Seiten schreiben und danach werden die Use Cases aufgestellt und dann wird das 2 Monate lang programmiert, sondern die Use Cases und Features kommen schon agil rein und werden dann aber traditionell ins Dokument gepflegt.“*- P3, Product Owner

Dabei spielt die Angemessenheit der Dokumentation für die Projektbeteiligten eine große Rolle. Komplexere Anforderungen werden so z.B. ausführlicher dokumentiert. Das kann zum Beispiel bedeuten, dass komplexe Anforderungen ein eigenes Kapitel im entstehenden Konzept erhalten, in dem sie detailliert beschrieben werden, während weniger komplexe Anforderungen innerhalb eines bestehenden Kapitels erwähnt werden. Auch wird innerhalb des Projekts darauf geachtet, dass jede Information in der Dokumentation möglichst nur einmal vorhanden ist. Um dieses Ziel zu erreichen, werden einmal erstellte Dokumente in anderen Dokumenten verlinkt, falls die enthaltenen Informationen an anderer Stelle benötigt werden.

5.2 Gründe für hybride Software-Entwicklung

Die zweite Forschungsfrage beschäftigt sich damit, warum sich Unternehmen dazu entschließen nach einem hybriden Software-Entwicklungsmodell zu arbeiten: *Was sind Gründe dafür, dass sich Unternehmen dazu entschließen nach einem hybriden Modell zu arbeiten?*

Im untersuchten Projekt gibt es mehrere Gründe für den gewählten Ansatz. Zunächst einmal ist es über die gewählte Vorgehensweise möglich, Vorteile der agilen und der traditionellen Software-Entwicklung miteinander zu verbinden. Des Weiteren war es den Projektbeteiligten wichtig, einen Prozess zu definieren, der auf ihre konkreten Bedürfnisse zugeschnitten ist. Dabei war es nicht wichtig, ob eine gewählte Praktik agil oder traditionell orientiert ist, sondern das gewählte Verfahren musste für das konkrete Projekt angemessen sein. Dabei wurde der gewählte Prozess in seiner Entstehung immer wieder kritisch hinterfragt und angepasst.

> *„Bei uns wird ja fortwährend an unserem internen Prozess gearbeitet, d.h. genau das ist ein Thema, dass bei uns immer wieder kritisch hinterfragt wird und das wir auch schon häufig geändert haben.“*- P5, Software-Architekt

5.2.1 Vorteile beider Ansätze nutzen

Ein Nachteil der agilen Software-Entwicklung ist die Problematik von Kosten- und Aufwandsschätzung. Da es in agilen Ansätzen keine formale Phase der Anforderungsanalyse gibt, können diese Parameter nur schwer geschätzt werden. [27]
Im untersuchten Projekt erfolgt die Gesamtplanung des Projekts daher eher traditionell. Es wurden initial Anforderungen erhoben und analysiert, um den Umfang des geplanten Projekts abschätzen zu können (siehe Kapitel 5.1.2). Auf diese Weise kann Planungssicherheit bezüglich des benötigten Budgets und der benötigten Ressourcen erlangt werden.

> *„Das heißt da sind wir nicht so agil und sagen, 'das Budget ist uns egal [..]', sondern sind da schon – ich sag mal aufgrund der Vorkenntnisse und der Vorprodukte, weil es ja eben ein Nachfolgeprodukt ist – relativ konservativ und wissen, was auf uns zukommt.“* - P3, Product Owner

Um dennoch flexibel auf Änderungen reagieren zu können werden die initial erhobenen Anforderungen agil erweitert und ausdetailliert (siehe Kapitel 5.1.2).

5.2.2 Reaktion auf Projektrahmenbedingungen

Ein weiterer Grund, warum im untersuchten Projekt ein hybrides Vorgehen innerhalb der Software-Entwicklung gewählt wird, sind hohe Projektrisiken im Bereich der Ressourcenverfügbarkeit. So sind die Entwickler, die im Projekt beteiligt sind, neben dieser Tätigkeit auch noch für den Support weiterer bereits entwickelter Produkte zuständig.

> *„Das geht dann auch Richtung Planung: Das wir ganz schwer planen können wann wir eigentlich mit bestimmten Features fertig sind, weil eben plötzlich ein sehr hoher Support-Anteil aufkommen kann und wir das nicht abfangen bzw. nicht kompensieren können.“* - P4, Projektleiter

Dadurch, dass die Mitarbeiter nicht exklusiv für die Neuentwicklung zur Verfügung stehen, muss auch der Software-Entwicklungsprozess entsprechend angepasst werden. Die Projektbeteiligten haben sich dabei für ein iteratives Vorgehen, aber bewusst gegen einen agilen oder traditionellen Standardprozess wie Scrum oder Wasserfall entschieden.

> *„Das heißt, der Prozess richtet sich schon danach aus, wenn ein Kollege plötzlich nicht mehr verfügbar ist. Dann heißt das nicht, dass das ganze Projekt scheitert, sondern einfach nur, dass sich alles verzögert. Und das ist zwar nicht schön, aber der Prozess funktioniert.“* - P5, Software-Architekt

Durch flexible Iterationszyklen ist es möglich, ein Mehraufkommen im Support anderer Produkte aufzufangen. Das heißt, wenn es dazu kommt, dass mehr Mitarbeiter als ursprünglich geplant im Support benötigt werden, wird die Länge des Iterationszyklus der Neuentwicklung proportional zum Mehraufwand im Support verlängert. Auch werden die Features, die in der jeweiligen Iteration entwickelt werden sollen, priorisiert, sodass sicher gestellt wird, dass die wichtigsten Features am Ende der Iteration fertig gestellt sind.

> *„Das heißt, wenn das Release terminiert ist auf beispielsweise Mitte August, dann ist es so, dass Features, die mit der höchsten Priorität passieren auf jeden Fall umgesetzt sein sollten. Und die wären die einzigen, die Release-verhindernd wären.“* - P3, Product Owner

Die Iterationen haben dabei eine Länge zwischen drei und vier Monaten. Dies ist nur möglich, da es sich bei dem Produkt um eine firmeninterne Produktentwicklung handelt und es keinen fest vereinbarten Auslieferungstermin mit einem Kunden gibt.

5.3 Probleme und Herausforderungen

In der dritten hier untersuchten Forschungsfrage geht es darum, welche Probleme und Herausforderungen sich ergeben: *Welche Probleme und Herausforderungen ergeben sich in hybriden Software-Entwicklungsmodellen in Bezug auf das Requirements Engineering?*

Im untersuchten Projekt waren die meisten Projektbeteiligten grundsätzlich zufrieden mit dem etablierten Prozess. Die Vorteile des gewählten Verfahrens waren dabei für die Probanden überwiegend. Dies steht mit Sicherheit auch in Verbindung mit der Tatsache, dass die etablierten Prozesse gemeinschaftlich erarbeitet wurden.

> *„Das haben wir uns ja so ausgesucht. Wir haben das definiert. Wir haben das auch lange erarbeitet dahin zu kommen. Das hat wirklich lange gedauert. Und ich glaube das ist genau das, was wir brauchen.“* - P5, Software-Architekt

Die gemeinsame Erarbeitung trägt vermutlich dazu bei, dass die Projektbeteiligten sich stärker mit dem etablierten Prozess identifizieren, da sie Ideen und Wünsche direkt einbringen können. Dennoch konnten einige Probleme identifiziert werden, die sich in Bezug auf das Requirements Engineering in untersuchten Projekt ergeben.

5.3.1 Dokumentation

Im untersuchten Projekt spielt die Angemessenheit der Dokumentation eine große Rolle. Auf der einen Seite, soll am Ende des Projekts eine vollständige Dokumentation vorliegen, die als Grundlage für Wartung und Support dienen kann, auf der anderen Seite werden neue Anforderungen agil erfasst und umgesetzt. Dies erfordert viel Disziplin beim Nachdokumentieren von Anforderungen.

> *„Es ist oft das Problem, dass wir über etwas diskutieren. Das wird dann aber teilweise auch nicht nochmal in den Tickets festgehalten. Das vergisst man dann auch teilweise wieder. Das ist nicht unbedingt immer im Dokument niedergeschrieben.“*- P4, Projektleiter

Ziel ist es dabei, eine Information möglichst nur einmal zu erfassen. Dies soll im untersuchten Projekt dadurch sicher gestellt werden, dass einmal erfasste Informationen in neu zu erstellenden Dokumenten verlinkt werden. Dies gelingt allerdings nur teilweise.

> *„Das heißt aktuell gibt es beispielsweise auch häufig eine Doppelung der Informationen. Die sind einmal im Use Case beschrieben, der dann agil entwickelt wurde, aber auch in der ganzheitlichen Architektur-Beschreibung beispielsweise.“* - P3, Product Owner

Dadurch, dass Informationen sowohl am Ticket, als auch in den Konzepten dokumentiert werden können, ist es sehr schwierig für die Projektbeteiligten zu entscheiden, welche Information wo dokumentiert werden soll. Auch erschwert dieses Problem die einmalige Dokumentation von Informationen und Anforderungen, da nicht immer klar ist, an welcher Stelle noch Informationen zu finden sind, die mit dem Sachverhalt in Verbindung stehen, der dokumentiert werden soll.

5.3.2 Qualitätssicherung von Anforderungen

Im untersuchten Projekt gibt es keinen speziellen Prozess um Anforderungen zu validieren und zu verifizieren wie in der traditionellen Software-Entwicklung. Die Qualität und Vollständigkeit der Anforderungen kann im untersuchten Projekt daher nicht immer initial sicher gestellt werden.

> *„Ich würde sagen gefühlt ist die Qualität der Anforderungen nicht immer sicher gestellt. Weil teilweise Tickets vorhanden sind für Anforderungen mit nur einem Satz [..]“* - P2, Entwickler

Durch diesen Sachverhalt, liegt es innerhalb des Projekts in der Verantwortung von Entwicklern und Software-Architekt Rückfragen zu stellen, wenn sie

Anforderungen nicht verstehen oder diese nicht vollständig sind. In diesem Kontext wünschen sich manche Projektbeteiligten mehr Unterstützung in der Qualitätssicherung der Anforderungen. Dies könnte nach Meinung der interviewten Probanden über Templates oder Richtlinien geschehen, in denen beschrieben wird, was eine fachliche Anforderung alles enthalten sollte, um ausführlich genug zu sein.

> *„Vielleicht Templates oder ich weiß nicht Richtlinien, was in der fachlichen Anforderung enthalten sein muss, damit wir daraus auch ein vernünftiges Konzept bauen können.“* - P2, Entwickler

Des Weiteren wurden Schwächen und Probleme in der Testbarkeit von Anforderungen festgestellt. So sind nicht alle Anforderungen, die innerhalb des Projekts erhoben werden, gleichermaßen gut testbar. Da es sich bei dem entwickelten Produkt um ein Datenbanken Monitoring System handelt, stellt sich immer die Frage, ob die gesammelten Daten das gewünschte Ergebnis repräsentieren.

> *„Ja, da werden Daten gesammelt und ich kann die Daten auch anzeigen und die sehen einigermaßen plausibel aus. Das ist ein Test. Besser als gar keiner.“*- P4, Projektleiter

Dadurch dass die Testbarkeit von Anforderungen einen kritischen Faktor im untersuchten Projekt darstellt, stellt sich die Frage, ob zusätzliche Maßnahmen getroffen werden sollten, um diesem Sachverhalt Rechnung zu tragen.

Kapitel 6

Diskussion

Der in dieser Arbeit beschriebene Entwicklungsansatz ist im untersuchten Projekt durch Erfahrung über die Zeit entstanden. Dabei wurde der etablierte Prozess immer wieder von den Projektbeteiligten hinterfragt und auf die konkreten Bedürfnisse angepasst. Klünder et al. [17] haben in diesem Zusammenhang herausgefunden, dass hybride Ansätze am häufigsten durch Evolution, gefolgt von Planung als Teil der Verbesserung des Software-Entwicklungsprozesses, entstehen. Dies konnte auch im untersuchten Projekt beobachtet werden. Des Weiteren war ein starker Fokus darauf gerichtet, den Software-Entwicklungsprozess auf die eigenen Bedürfnisse anzupassen. Viele Unternehmen kombinieren agile und traditionelle Verfahren, um sie auf den eigenen Kontext zuzuschneiden [26]. Abweichungen von Prozessen sind dabei i.d.R. motiviert durch ein explizites Ziel, kontextbedingte Faktoren, Kunde, Markt, Geschäftsprozesse oder Probleme mit im Unternehmen etablierten Standardprozessen [17].
Prenner et al. [26] haben in ihrer systematischen Literaturrecherche herausgefunden, dass alle bisher beobachteten hybriden Software-Entwicklungsprozesse auf den Phasen des Wasserfall-Modells basieren. Dabei werden die folgenden Standardaktivitäten unterschieden: Anforderungsanalyse, Design, Implementierung, Test und operative Nutzung [28]. Auch in der in dieser Arbeit durchgeführten Fallstudie können diese Phasen beobachtet werden. Sie werden für jedes zu entwickelnde Feature innerhalb einer Iteration durchlaufen. Allerdings wird die Testphase des in dieser Arbeit beobachteten Prozesses noch einmal in mehrere Phasen unterteilt. Die einzelnen Phasen werden dabei im untersuchten Projekt parallel für mehrere Features durchlaufen. Das bedeutet, dass sich zum Beispiel ein Feature in der Designphase befindet, während ein weiteres Feature entwickelt wird und ein drittes Feature getestet wird. Dieses Vorgehen deutet darauf hin, dass im untersuchten Projekt nach einem Pipeline-Ansatz vorgegangen wird. Auch die Tatsache, dass die Entwicklung Feature-zentriert abläuft, stützt diese These. [26]

Im Rahmen dieses Vorgehens werden Anforderungen sowohl initial zu Projektbeginn, als auch im Verlauf des Projekts erhoben und ergänzt (siehe Kapitel 5.1.2). Dabei muss eine Balance zwischen initialer und kontinuierlicher Anforderungserhebung gefunden werden, d.h. es muss festgelegt werden, wie umfangreich und detailliert die initiale Anforderungserhebung erfolgen soll. Eine detaillierte und ausführliche Anforderungserhebung verbunden mit einem Designentwurf zu Projektbeginn können dabei unterstützen, mehr Planungssicherheit zu erlangen. Gleichzeitig wird aber das Risiko erhöht, dass es nicht mehr möglich ist, leicht auf Änderungswünsche des Kunden zu reagieren. Unternehmen, die nach hybriden Software-Entwicklungsverfahren arbeiten, müssen also den Aufwand für initiale und kontinuierliche Anforderungserhebung balancieren. [26]
Probleme haben sich im untersuchten Projekt vor allem im Bereich der Dokumentation von Anforderungen ergeben. Da nicht alle Anforderungen zu Projektbeginn erhoben werden, muss festgelegt werden, wann welche Information in welchem Umfang dokumentiert werden soll. Dies kann eine großer Herausforderung darstellen, da klare Verantwortlichkeiten festgelegt werden müssen und die Nachdokumentation von Anforderungen viel Disziplin von den Projektbeteiligten verlangen kann.

6.1 Bedrohungen der Validität

In diesem Kapitel wird die Gültigkeit der Ergebnisse dieser Fallstudie diskutiert. Es wird dabei nach den von Runeson et al. [29] vorgeschlagen Kriterien vorgegangen.

Konstruktions-Gültigkeit (Construct Validity). Eine Gefahr beim Erstellen von Fragebögen für Interviews ist die Tatsache, dass Fragen so formuliert sein können, dass sie nicht oder falsch verstanden werden. Um dem vorzubeugen, wurde der in dieser Arbeit verwendete Fragebogen im Vorfeld der Interviews getestet. Zu diesem Zweck wurde ein Pilot-Interview mit einem Mitarbeiter des Unternehmens durchgeführt, der nicht im untersuchten Projekt mitwirkt. Dieser Mitarbeiter wurde gebeten die Fragen auf ein beliebiges Projekt zu beziehen und Rückfragen zu stellen, falls eine Frage nicht verständlich ist. Die so gewonnenen Erkenntnisse konnten genutzt werden, um die Fragen noch einmal anzupassen und zu präzisieren.

Interne Gültigkeit (Internal Validity). Im Rahmen der Fallstudie wurden Interviews mit den Projektbeteiligten durchgeführt. Da der Interviewende im selben Unternehmen wie die Probanden arbeitet, besteht ein erhöhtes Risiko sozial erwünschter Antworten, d.h. es besteht die Gefahr, dass die Probanden so antworten, wie sie glauben, dass es von ihnen erwartet wird.

Dieser Faktor wurde dadurch minimiert, dass den Probanden versichert wurde, dass ihre Antworten vertraulich behandelt werden. Des Weiteren wurde den Interviewten zu Beginn des Interviews genau erklärt, wie mit den im Interview erhobenen Daten weiter verfahren wird.
Da die Fallstudie in dieser Arbeit von nur einer Person durchgeführt wurde, besteht die Gefahr, dass die Forschungsergebnisse von der Sichtweise und der Arbeit dieser Person abhängen. Um sicher zu stellen, dass die selben Forschungsergebnisse basierend auf den erhobenen Daten unabhängig von der Person des Forschenden erzielt werden können, wurde in dieser Arbeit exemplarisch ein Interview herausgenommen. Für dieses Interview wurden exemplarisch von einer zweiten Person Codes nach der in Kapitel 3.3.3 beschriebenen Vorgehensweise erstellt. Anschließend wurden die Codes beider Interviews inhaltlich miteinander verglichen. Es konnte dabei ein hohes Maß an Übereinstimmung festgestellt werden.

Extrene Gültigkeit (External Validity). Bei dieser Arbeit handelt es sich um eine Fallstudie, in der ein Einzelfall untersucht wurde, um tiefere Einblicke in ein bisher wenig erforschtes Gebiet zu gewinnen. Das in dieser Arbeit beschriebene Vorgehen ermöglicht dabei einen detaillierten und umfassenden Einblick in das Requirements Engineering eines Projekts. Die Ergebnisse, die in diesem Zusammenhang über die Grounded Theory gewonnen wurden, können allerdings nur genutzt werden, um den spezifischen Kontext zu erklären, der in der Studie untersucht wurde [14]. Der Kontext wurde in dieser Arbeit über die Forschungsfragen festgelegt, die die Grenzen des Untersuchten bestimmen [10]. Auch handelt es sich bei dem untersuchten Projekt um ein sehr kleines Projekt mit nur wenigen Teilnehmern. Aufgrund der Heterogenität hybrider Ansätze ist es nicht möglich, die Forschungsergebnisse auf jede Art von hybridem Software-Entwicklungs-Projekt zu übertragen. Die verwendeten Prozesse sind in hybriden Ansätzen oft abhängig vom individuellen Kontext und auf die Bedürfnisse des zugrunde liegenden Projekts angepasst [26].

Zuverlässigkeit (Reliability). Zwar konnten während der Fallstudie Interviews mit allen Projektbeteiligten durchgeführt werden, um Erkenntnisse zu gewinnen, allerdings war es nicht möglich, mit einem Endkunden zu sprechen. Aus diesem Grund konnte dieser Aspekt nicht in der Fallstudie berücksichtigt werden. Auch wurde der Kontext der Fallstudie durch die Wahl der Forschungsfragen festgelegt. Es ist möglich, dass es noch weitere Einflussfaktoren auf das untersuchte Gebiet gibt, die aber durch die Wahl der Forschungsfragen nicht berücksichtigt wurden.

Kapitel 7

Verwandte Arbeiten

Vijayasarathy et al. [38] haben gezeigt, dass traditionelle Ansätze wie das Wasserfall-Modell mit agilen Ansätzen kombiniert werden. Dabei ist die Wahl einer geeigneten Vorgehensweise für ein konkretes Projekt ein kritischer Faktor. In diesem Zusammenhang werden organisatorische, projektbezogene und teambezogene Charakteristiken für agile, traditionelle, iterative und hybride Vorgehensweisen identifiziert.

Boehm und Turner [6] wollen spezifische Nachteile von agiler und plan-basierter Software-Entwicklung dadurch ausgleichen, indem sie fünf Faktoren definieren, die Projektrahmenbedingungen beschreiben. Darunter befinden sich die Größe des Projekts (Anzahl der beteiligten Personen), die Kritikalität (Schaden der durch mögliche Fehler entsteht), die Dynamik (Häufigkeit mit der sich Anforderungen ändern), personelle Faktoren und Unternehmenskultur. Durch diese Faktoren soll es möglich sein, eine Balance von agilen und traditionellen Vorgehensweisen innerhalb eines Projekts zu finden, die auf das konkrete Projekt zugeschnitten sind.

Fallstudien im Bereich der hybriden Software-Entwicklung beziehen sich oft auf große Software-Entwicklungsprozesse. Bick et al. [5] haben in einer Fallstudie untersucht, welche Herausforderungen sich bei der Koordination in großangelegten Software-Entwicklungsprojekten ergeben. Dabei wurde untersucht, wie und warum die Kombination von traditioneller Planung auf teamübergreifender Ebene und agilem Entwickeln auf Teamebene in ineffektiver Koordination resultieren kann. Innerhalb der Fallstudie wurden mehrere Kriterien wie beispielsweise ein Mangel an Bewusstsein für bestehende Abhängigkeiten identifiziert.

Um zu untersuchen welche Entwicklungsansätze in der Praxis eingesetzt werden und wie diese miteinander kombiniert werden, wurde die HELENA Studie (*Hybrid dEveLopmENt Approaches in software system*

development) ins Leben gerufen. Dabei handelt es sich um ein exploratives mehrstufiges internationales Forschungsprojekt, in dessen Rahmen Daten von Teilnehmern verschiedener Länder erhoben wurden. Die *HELENA Umfrage* dient dabei als Fragebogen mit dessen Hilfe Daten zur Prozessnutzung, Prozessnutzung im Kontext von Normen und Standards, Prozessverbesserung und Erfahrung gesammelt werden. Es wurden von Mai bis November 2017 insgesamt 1467 Datenpunkte von 691 Teilnehmern erhoben. Klünder et al. [17] haben basierend auf 732 Datenpunkten der HELENA Umfrage herausgefunden, dass 76.8% der Unternehmen verschiedene Methoden zu hybriden Verfahren kombinieren. Dabei werden am häufigsten Frameworks und Methoden wie Scrum, iterative Entwicklung, Kanban, Wasserfall oder DevOps verwendet. [18, 17]
Tell et al. [36] haben auf Basis der HELENA Daten untersucht welche Frameworks und Methoden als Basis für die hybride Software-Entwicklung dienen, durch welche Praktiken sie verbunden werden und wie hybride Software-Entwicklungsmethoden charakterisiert werden können. Dabei haben sie herausgefunden, dass hybride Software-Entwicklung unabhängig von Unternehmensgröße oder Industriesektor eingesetzt wird.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Ziel dieser Arbeit ist es, einen Einblick darin zu geben, wie agile und traditionelle Verfahren der Software-Entwicklung miteinander kombiniert werden und welchen Einfluss dies auf das Requirements Engineering hat. In der in diesem Zusammenhang durchgeführten Fallstudie konnte festgestellt werden, dass im Requirements Engineering sowohl agile, als auch traditionelle Elemente genutzt werden. Anforderungen werden im untersuchten Projekt zunächst initial auf high-Level Ebene erhoben. Die so gewonnenen Anforderungen werden im Verlauf des Projekts kontinuierlich weiter verfeinert und ergänzt. Es muss also innerhalb des Projekts eine Balance zwischen initialer und kontinuierlicher Anforderungserhebung gefunden werden. Im untersuchten Projekt wird dabei so vorgegangen, dass zu Projektbeginn die wichtigsten Features der Anwendung identifiziert werden. Diese Features werden als high-Level Anforderung festgehalten, die im Verlauf des Projekts detailliert und ausführlich erarbeitet wird.
Bei der Entscheidung für einen hybriden Software-Entwicklungsansatz spielen dabei nicht nur die Nutzung von Vorteilen agiler und traditioneller Verfahren eine Rolle, sondern auch die Rahmenbedingungen des konkreten Projekts. Problematisch ist im untersuchten Projekt die Dokumentation von Anforderungen. In diesem Zusammenhang muss entschieden werden, wann, wo und in welchem Umfang Anforderungen dokumentiert werden sollen. Es wurden vor allem Probleme in Bezug auf nicht oder unzureichend dokumentierte Anforderungen oder Dopplungen von Informationen festgestellt.

8.2 Ausblick

Diese Fallstudie gibt einen Einblick in das Requirements Engineering hybrider Projekte. Viele Aspekte könnten im Rahmen weiterer Forschung noch vertieft werden. So handelt es sich bei dem untersuchten Projekt zum Beispiel um ein sehr kleines Projekt mit nur wenigen Teilnehmern. In diesem Zusammenhang könnte untersucht werden, welche Rolle die Teamgröße in Bezug auf die Art der Probleme und Herausforderungen im Requirements Engineering in hybriden Projekten spielt.
Die Dokumentation von Anforderungen scheint in hybriden Projekten eine besondere Herausforderung darzustellen. In einer weiteren Studie könnte untersucht werden, ob dieses Problem auch in anderen hybriden Projekten besteht und welche Strategien die jeweiligen Unternehmen verfolgen, um diese Herausforderung zu meistern.

Anhang A

Interviewfragen

Hintergrund

1. Was ist deine Rolle im Projekt? Beschreibe sie.
2. Welche Rollen gibt es noch im Projekt und wie stehen diese in Beziehung zu dir?

Anforderungsprozess

1. Wie wird aktuell im Projekt mit Anforderungen umgegangen? Was sind die Prozesse?
2. Was sind deine konkreten Aufgaben bezogen auf den Anforderungsprozess?
3. Gibt es innerhalb des Projekts bei der Software-Entwicklung Elemente, die du als agil einstufst?

   a) Welche Elemente sind das?

   b) Wie wirken sich diese Elemente auf das Anforderungsmanagement aus?
4. Gibt es auch traditionelle Elemente?

   a) Welche Elemente sind das?

   b) Wie wirken sich diese Elemente auf das Anforderungsmanagement aus?

5. Woher kommen die Anforderungen? Welche Quellen für Anforderungen gibt es in diesem Projekt?
6. Wie werden Anforderungen zwischen den Beteiligten kommuniziert?
7. Wie werden Änderungen der Anforderungen kommuniziert?
8. Wie werden Anforderungen dokumentiert?
9. Wie wird die Qualität der Anforderungen sichergestellt?
10. Wie werden nicht-funktionale Anforderungen (z.B. Wartbarkeit) erhoben und verwaltet?

Persönliche Wahrnehmung

1. Welche Maßnahmen helfen dir am besten deine Aufgaben im Bezug auf die Anforderungen zu erfüllen?
2. Welche Maßnahmen würden dir noch helfen?

3. Was funktioniert in Bezug auf das Anforderungsmanagement deiner Meinung nach generell gut im Projekt?
4. Wo siehst du allgemein Herausforderungen oder Probleme im aktuellen Vorgehen?
5. Was würdest du in Bezug auf das Anforderungsmanagement ändern oder noch verbessern?

6. Glaubst du, dass das hybride Vorgehen (Verbinden agiler und traditioneller Ansätze) gut für das Projekt geeignet ist?

   a) Wenn ja, welche Vorteile bringt es?

   b) Wenn nein, welche Nachteile ergeben sich? Was wäre besser geeignet?
7. Gibt es weitere Aspekte, die du ergänzen möchtest?

Literaturverzeichnis

[1] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990.

[2] A. Aitken and V. Ilango. A comparative analysis of traditional software engineering and agile software development. In *2013 46th Hawaii International Conference on System Sciences*, pages 4751–4760, 2013.

[3] A. Aurum and C. Wohlin. *Requirements Engineering: Setting the Context*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[4] J. B. Barlow, J. Giboney, M. J. Keith, D. Wilson, R. M. Schuetzler, P. B. Lowry, and A. Vance. Overview and guidance on agile development in large organizations. *Communications of the Association for Information Systems*, 29(2):25–44, 2011.

[5] S. Bick, K. Spohrer, R. Hoda, A. Scheerer, and A. Heinzl. Coordination challenges in large-scale software development: A case study of planning misalignment in hybrid settings. *IEEE Transactions on Software Engineering*, PP:1–1, 07 2017.

[6] B. Boehm and R. Turner. Using risk to balance agile and plan-driven methods. *Computer*, 36(6):57–66, 2003.

[7] B. Boehm and R. Turner. Management challenges to implementing agile processes in traditional development organizations. *IEEE Software*, 22(5):30–39, 2005.

[8] B. W. Boehm and P. N. Papaccio. Understanding and controlling software costs. *IEEE Transactions on Software Engineering*, 14(10):1462–1477, 1988.

[9] G. Coleman and R. O'Connor. Using grounded theory to understand software process improvement: A study of irish software product companies. *Information and Software Technology*, 49(6):654 – 667, 2007. Qualitative Software Engineering Research.

[10] J. Corbin and A. Strauss. *Basics of Qualitvative Research: Techniques and Procedures for Developing Grounded Theory, third ed.* Sage Publications, Thousand Oakes, 2008.

[11] J. M. Corbin and A. Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13(1):3–21, 1990.

[12] A. De Lucia and A. Qusef. Requirements engineering in agile software development. *Journal of emerging technologies in web intelligence*, 2(3):212–220, 2010.

[13] K. Elghariani and N. Kama. Review on agile requirements engineering challenges. In *2016 3rd International Conference on Computer and Information Sciences (ICCOINS)*, pages 507–512, 2016.

[14] R. Hoda, J. Noble, and S. Marshall. Developing a grounded theory to explain the practices of self-organizing agile teams. *Empirical Software Engineering*, 17(6):609–639, 2012.

[15] S. E. Hove and B. Anda. Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS'05)*, pages 10 pp.–23, 2005.

[16] M. Höst and P. Runeson. Checklists for software engineering case study research. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 479–481, 2007.

[17] J. Klünder, R. Hebig, P. Tell, M. Kuhrmann, J. Nakatumba-Nabende, R. Heldal, S. Krusche, M. Fazal-Baqaie, M. Felderer, M. F. Genero Bocco, S. Küpper, S. A. Licorish, G. Lopez, F. McCaffery, Özcan Top, C. R. Prause, R. Prikladnicki, E. Tüzün, D. Pfahl, K. Schneider, and S. G. MacDonell. Catching up with method and process practice: An industry-informed baseline for researchers. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 255–264, 2019.

[18] M. Kuhrmann, P. Diebold, J. Munch, P. Tell, K. Trektere, F. McCaffery, V. Garousi, M. Felderer, O. Linssen, E. Hanser, and C. R. Prause. Hybrid software development approaches in practice: A european perspective. *IEEE Software*, 36(4):20–31, 2019.

[19] M. Kuhrmann, P. Diebold, J. Münch, P. Tell, V. Garousi, M. Felderer, K. Trektere, F. Mccaffery, O. Linssen, E. Hanser, and C. Prause. Hybrid software and system development in practice: Waterfall, scrum, and beyond. 07 2017.

[20] G. Kumar and P. K. Bhatia. Comparative analysis of software engineering models from traditional to modern methodologies. In *2014 Fourth International Conference on Advanced Computing Communication Technologies*, pages 189–196, 2014.

[21] M. Lueger. Grounded theory. In *Qualitative Marktforschung*, pages 189–205. Springer, 2009.

[22] A. B. M. Moniruzzaman and S. A. Hossain. Comparative study on agile software development methodologies. *CoRR*, abs/1307.3356, 2013.

[23] F. Paetsch, A. Eberlein, and F. Maurer. Requirements engineering and agile software development. In *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, pages 308–313, 2003.

[24] K. W. Parry. Grounded theory and social process: A new direction for leadership research. *The leadership quarterly*, 9(1):85–105, 1998.

[25] N. Prenner. Towards improving the organization of hybrid devel-opment approaches. 03 2020.

[26] N. Prenner, C. Unger-Windeler, and K. Schneider. How are hybrid development approaches organized? -a systematic literature review. 03 2020.

[27] B. Ramesh, L. Cao, and R. Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480, 2010.

[28] W. W. Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, pages 328–338, 1987.

[29] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131, Dec 2008.

[30] G. Sandhaus, P. Knott, and B. Berg. Hybride softwareentwicklung. *Informatik-Spektrum*, 38:306–309, 2015.

[31] C. B. Seaman. *Qualitative Methods*, pages 35–62. Springer London, London, 2008.

[32] A. Sillitti and G. Succi. *Requirements Engineering for Agile Methods*, pages 309–326. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[33] K.-J. Stol, P. Ralph, and B. Fitzgerald. Grounded theory in software engineering research: A critical review and guidelines. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, page 120–131, New York, NY, USA, 2016. Association for Computing Machinery.

[34] A. Strauss and J. Corbin. Grounded theory methodology. *Handbook of qualitative research*, 17(1):273–285, 1994.

[35] A. L. Strauss. *Qualitative analysis for social scientists*. Cambridge university press, 1987.

[36] P. Tell, J. Klünder, S. Küpper, D. Raffo, S. G. MacDonell, J. Münch, D. Pfahl, O. Linssen, and M. Kuhrmann. What are hybrid development methods made of? an evidence-based characterization. In *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pages 105–114, 2019.

[37] G. Theocharis, M. Kuhrmann, J. Münch, and P. Diebold. Is water-scrum-fall reality? on the use of agile and traditional development practices. In P. Abrahamsson, L. Corral, M. Oivo, and B. Russo, editors, *Product-Focused Software Process Improvement*, pages 149–166, Cham, 2015. Springer International Publishing.

[38] L. R. Vijayasarathy and C. W. Butler. Choice of software development methodologies: Do organizational, project, and team characteristics matter? *IEEE Software*, 33(5):86–94, 2016.